KU LEUVEN

FACULTY OF ECONOMICS AND BUSINESS

FACULTY OF ECONOMICS AND BUSINESS

Social Network Analysis for Insurance Fraud Detection

Bruno Deprez
R0630718

Thesis submitted to obtain the degree of

MASTER OF ACTUARIAL AND FINANCIAL ENGINEERING

Promotor: Prof. Dr. Tim Verdonck

Work leader: Félix Vandervorst

Academic year: 2021-2022

KU LEUVEN

FACULTY OF ECONOMICS AND BUSINESS

FACULTY OF ECONOMICS AND BUSINESS

Social Network Analysis for Insurance Fraud Detection

Although the literature on fraud detection in general and fraud detection using social network analysis specifically is quite extensive, its applications have only recently found their way into the actuarial scientific research. This thesis gives an overview of the different techniques that are used to study these social networks. Two state-of-the-art algorithms are applied to a motor insurance claim dataset. Different metrics are used to evaluate the output of these models. On the one hand, the global model performance is evaluated using the classical techniques. On the other, the complementarity of these new predictions for only the top scores is studied. This research shows that social network analysis leads to novel insights that can be leveraged to uncover more illicit claims. This work adds to the literature by applying these novel network techniques in the context of insurance fraud, and by measuring in what way the network information is captured by the models. We will not touch on resampling techniques, hyper-parameter tuning etc., to enhance the models, as this thesis only serves as a proof-of-concept to see whether it is indeed worthwhile for the modeller to include these social networks.

Bruno Deprez
R0630718

Thesis submitted to obtain the degree of

MASTER OF ACTUARIAL AND FINANCIAL ENGINEERING

Promotor: Prof. Dr. Tim Verdonck

Work leader: Félix Vandervorst

Academic year: 2021-2022

Contents

Acknowledgements

The inception of this thesis is situated at the beginning of the summer holidays, almost a year ago. It marked the start of my last year at KU Leuven. The last seven years of study have now accumulated into this work. As the journey of writing this thesis has come to an end, I want to thank the people that were essential for the successful completion of this thesis.

First of all, there is my promoter prof. dr. Tim Verdonck who I want to thank for his guidance and for giving me the opportunity to pursue the research in network analysis.

Secondly, I want to thank the people at Allianz for the enabling our collaboration and providing the datasets used in my research. A special thanks goes out to Félix Vandervorst for the day-to-day coordination. His input and expertise on fraud and network analysis really brought this work to a higher level.

In addition, I want to thank prof. dr. Wouter Verbeke for the insightful discussions on the insights that can be gained from network features, and on his own research. This laid the foundation of the final chapter of this thesis, which completed the story we wanted to tell.

Next, there are my parents, whose financial and moral support enabled me to pursue a second degree in Financial and Actuarial engineering. Without their care, I would have never been the person that I am today.

Furthermore, special thanks goes out to my girlfriend, Laura, whose love and support were crucial in giving me the strength to continue my studies and research for this thesis.

Last but not least, I want to thank all members of the actuarial team at KPMG. The flexibility and understanding they had during busier periods gave me the necessary time to put into this thesis and my studies overall.

Bruno Deprez
Leuven, June 2022.

General Introduction

The underwriting of an insurance contract is based on mutual trust between the different parties. When a person takes out an insurance policy, they pay a premium and expect the insurance undertaking to pay if the insured event, e.g. a car accident, materialises. On the other hand, the insurance company must be sure that the full claim amount is legitimate, i.e. it relates to the actual insured event and the person filing the claim does not exaggerates the damages. However, there will always be people who try to game the system, and extract more money from the insurer than necessary in order to enrich themselves.

This filing of fraudulent claims is far from being a small problem. The massive impact of fraud can be observed all over the world. In the U.S., it was estimated that in 2010 the yearly total cost was over $40 billion. This cost resulted in an estimated increase of the yearly premium by $400 to $700 for an average U.S. household [3]. According to a publication of the Belgian professional association for insurers, Assuralia, 5 to 10% of claim payments for P&C insurance in Europe can be seen as insurance fraud. Looking at the Belgian market, the fraudulent claims amount to €250 million, which results in an estimated increase of €60 to €125 of the insurance premium for a Belgian family [2].

Given the high monetary impact, both the insurance undertakings as well as the different governments are motivated to tackle this problem, both by predicting and preventing fraud. In order to prevent fraud, insurance companies have put fraud detection mechanisms in place. This mostly consists of machine learning algorithms that take in different claim attributes, e.g. the location, the time of day etc. and that assign a fraud score to the claims [15]. These models are used to enhance the business rules that are in place, which are based on the experience and intuition of a domain expert [15, 6].

Fraud detection is by nature very challenging. This is because the (labelled) fraud cases are uncommon. Fraudsters try to cover their tracks by anticipating the business rules in place. In addition, they try novel techniques to commit fraud, which means that the detection techniques need to evolve constantly over time, as they are trained on historical data [6].

One new way to enhance the fraud detection algorithms is using social networks. These social networks represent the different relationships and connections between different people, as well as other objects, which will be the case in this work. First of all, non-life insurance lends itself well to the build-up of a network structure. When looking at a motor insurance, which will be the basis of study in this thesis, there are a multi-

tude of interactions that result in a network. At the underwriting of a policy, there is a connection between the person who buys the policy and the broker who sells it. They are connected via the policy to the car that it covers. In case there is an accident, a claim is filed, and in this accident, a couple of parties can be involved. First, there is the driver of the car, who can be different from the policyholder, e.g. the person's child. Next to that, another car covered by another policy can be involved as well. This then connects different people, cars and policies with each other, from which a whole network grows (see Figure 1).

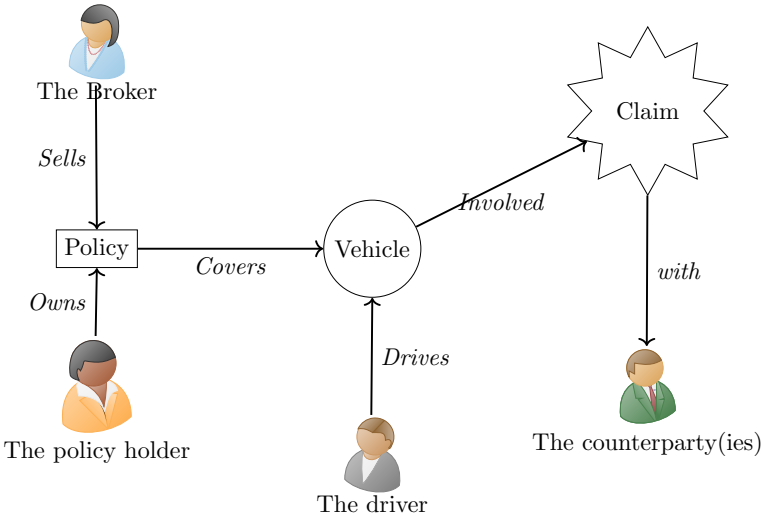


Figure 1: Example of a network structure for motor insurance claims.

The second main advantage is that it may help to build a more robust fraud detection algorithm. A network can uncover hidden patterns that can never become visible using only the claim-specific features. One of the problems is that fraudsters try to cover their tracks. In a social network context, this is much harder because the person would need to change the network structure of a large part of actors in the network in order to blend in [6]. This is because every filed claim and interaction can cause new connections in the network, which results in new patterns that may give you away.

There is, however, a large disadvantage. In the past, when one only worked with claim-specific attributes, the number of features was fixed. This makes it possible to track all claims by adding them in a table. Most machine learning algorithm are adapted to take in these data tables to first train themselves and later make new predictions. For network data, this is much harder. Even if you only keep track of a claim's direct neighbours, i.e. other objects it is connected to, you cannot keep track of this with a fixed length of features. If you take this number too small, you quickly run into the problem that some claims have too many neighbours. However, if you take it too large, you have a lot of unfilled features, which will influence the performance of your model.

Therefore, it is important to try to incorporate as much information from the network as possible, using a relatively small, and fixed, number of features. This will be an important part of the work done here.

In this thesis, we will focus on how to meet this problem and on the information that can be extracted from a network representing motor insurance data. In Chapter 1 we define more rigorously what a network is and the techniques that are used in the literature. This chapter also introduces some of the basic network features that are going to be used later on when building a model.

Chapter 2 explains the dataset we are going to use. In addition, the concepts that will be needed for the next chapter are discussed.

Afterwards, we start with the main techniques used in this thesis. The first is the BiRank algorithm which will be defined in Chapter 3. This is an extension to the famous pagerank. It tries to assign fraud scores to each claim, using the information coming from and the interconnectedness to other actors in the network, which may or may not already have a priori fraud information that can be leveraged. A second method is introduced in Chapter 4, which is called the metapath2vec algorithm. This one does not incorporate a priori fraud information, but tries to uncover the structure of the network and place of each node in it using *guided* random walks. This structure is captured in a low-dimensional representation of the network using natural language processing techniques.

Up until then, we have only used network data. Chapter 5 adds the claim specific features to the dataset. This extra data is combined with the network features to iteratively build and compare fraud detection models in Chapter 6.

Finally, Chapter 7 tries to interpret the previously obtained results. We not only look at the global performance, but also at the performance for the top scores, since this more accurately reflects the usage of these models in practice. Next to that, we study whether these different models with and without network data give complementary results, or whether they all point in the same direction. The latter would mean that incorporating social networks does not bring anything new to the table.

We note that the main purpose of this work is to analyse whether social network analysis can bring new insights for fraud detection. Here, we will not be focusing on obtaining the most performant model possible. Hence, we will not touch on resampling techniques, hyper-parameter tuning etc., to enhance the models, as this thesis only serves as a proof-of-concept to see whether it is indeed worthwhile for the modeller to include these social networks.

The accompanying code for this thesis can be found on GitHub:
`https://github.com/B-Deprez/Fraud_with_Network_Thesis`.

Chapter 1

Social Networks and Their Embeddings

There are multiple modelling techniques available today to tackle fraud detection problems. The data that is fed into these models are mostly in tabular form, and the models are specifically built for that. To train these, one often assumes that the observations $(\boldsymbol{x}_i, y_i)$ are i.i.d. (not including any social connections). When moving to networks of reported claims, it is not possible to feed these networks into the aforementioned models. The problem is that these networks can be of arbitrary size, and that their topological structure often is complex [1, 01-intro]. This makes it hard or even impossible to assign to each observation a fixed length of features that captures all information about its position in the network. In addition, since networks are build using relations between the actors in it, we can no longer justify assuming that these observations are independent.

In this thesis, we want to present the information of the social networks in such a way that these can be used in algorithms to uncover fraudulent behaviour. As will be discussed in Chapter 2, this will be done on past network data representing car insurance claims. Using this past data, we want to leverage the *social* connections in the network to uncover meaningful insights for predicting new fraud.

This first chapter introduces the key concepts of social networks, also known as graphs, and later move to the techniques used to represent the network into tabular form, which are called network embeddings. This will lay the groundwork for all other work done in the following chapters in this thesis.

The chapter is organised as follows. Section 1.1 gives the definitions of the main topic of this work, namely graphs. In addition, the different kinds of graphs/networks are introduced.

In Section 1.2, we explore the concept of network embeddings. We will see what they are and where these can be used when trying to uncover insurance fraud.

The next section, Section 1.3, covers related work in the scientific literature. It will be shown that there are different branches one can use to deal with finding fraudulent behaviour in the social network.

Finally, we give an summary of where our work fits into the literature in Section 1.4.

1.1 An Introduction to Networks

1.1.1 Basic Concepts

A fundamental understanding of networks is needed to further build towards network embeddings. As a first remark, we note that networks are also referred to as graphs in the scientific literature. In most cases, these two terms and their accompanying terminology can be used interchangeably. There is, however, a subtle difference in the usage of the terminology, as explained in [7, Chapter 2]. The terms related to graphs are used when discussing the mathematical properties of a network, where the term network indicates that one is dealing with a real system. In the definition that follows, we mention the terminology for both graphs and networks, but throughout this thesis, we try to stick to the network-related nomenclature.

Definition 1.1 is based on [6, Chapter 5] and introduces the concept of networks.

Definition 1.1. An network (graph) $G(V, E)$, consists of a set V of nodes (vertices). The set of links (edges) $E \subseteq V \times V$ expresses the connections between the nodes.

Depending on the elements of V and E, different names are given to the networks [4]. We say that the network is undirected if all edges $e_{ij} = (v_i, v_j)$ in E are unordered pairs. In this case, the direction of the connection does not matter. Mathematically, we can say that $e_{ij} = e_{ji}$ or that $e_{ij} \in E$ holds if and only if $e_{ji} \in E$ also holds. A classic example of an undirected network is one representing friendships between people. Here, the nodes are the different persons under consideration and the links represent being friends of one another. When Alice is friends with Bob, Bob is also automatically friends with Alice. There is not inherent direction associated with the links. A representation of an undirected network is shown in Figure 1.1. Coming back to the friendship example, we have that A and B, and B and C are friends, but A and C are not.

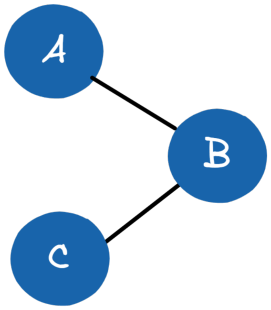


Figure 1.1: An undirected network. There is no inherent direction in the relationships.

Next to undirected networks, we can also have directed networks. Here, it is necessary to state in which node the link starts and in which it ends. Think about a large online platform where the nodes are the people, and the links represent *following* someone. When Charlie follows Dylan, it does not automatically mean that Dylan follows Charlie back. Hence, $e_{cd} \in E$, but $e_{dc} \notin E$, which means that E is not symmetric, resulting in

a directed network. It is possible to have both e_{ij} and e_{ji} in the set of links. Coming back to our example, when both Charlie and Dylan know each other well, they will most likely follow each other on the platform. A representation of a directed network is shown in Figure 1.2. Coming back to the *following* example, we have that A follows B, but B does not follow A back. This is represented by the arrow. There is also a link between B and C, but we do not use an arrow there. This means that the connection goes both ways, i.e. B and C follow each other. This could also be represented by two arrows, one starting in C and ending in B, and one the other way around.

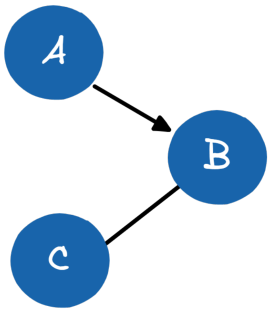


Figure 1.2: A directed network. The direction is indicated by the arrow. No arrow means that the connection goes both ways.

Both examples given above are what is called homogeneous networks. A homogeneous network is defined by imposing that all nodes are of the same type, as are all the links. When multiple types are possible for the nodes, links or both, the network is called heterogeneous. An important example is where nodes can be either an insurance company's client or a reported claim to the insurance company. The links can express the fact that the person is involved in the claim. This is actually also an example of a special kind of heterogeneous network, i.e., a bipartite network [6, 7, 28].

Definition 1.2. Suppose we have a network $G(V, E)$, where the nodes are of one of two types, i.e., where $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$. Then the network G is a bipartite network if the possible edges are only between nodes of different types. This means that $E \subseteq V_1 \times V_2$.

Depending on the context, $V_1 \times V_2$ can be either a directed or undirected set.

We go back to the insurance example of clients and filed claims. There can only be links between a client and a claim. We do not allow connections between two persons, and neither between two claims. Hence, there is a clear distinction between the two types of nodes, and only inter-group links are possible. Hence, it meets the Definition 1.2 and this network is a bipartite network.

Next to having different types of nodes, it is possible to have networks where the node types are the same, but the edges represent different kinds of relationships. Think of a network of cars where one type of edges connect cars that are part of the same household, and the other type represents cars that were involved in the same accident.

These networks are also considered as heterogeneous networks. Later on, in Chapter 2, we are going to consider a heterogeneous network with both different types of nodes and different types of edges.

Next to their types, the edges in the network can carry additional information about the strength of the connection. This is most commonly done by assigning a weight to them. An example would be that you have a social network representing friendships between people. In order to have a better representation of the strength of this friendship, you can assign the number of years two people have been friends as a weight to the corresponding edge. Even if the network would be *unweighted*, which means that you either have a connection between two nodes or not, you can still work in the weighted framework. One can represent the existence of an edge as having weight one, where the absence of an edge is represented by weight 0. This will come back when introducing the different representations of networks in the next section, and more specifically in Definition 1.3.

1.1.2 Representing Networks

When dealing with networks, it is important to have a good representation with which you can do further calculations. The representation must encompass the full network, since these will be used to construct the algorithms to build the features which are used down-stream in a typical fraud detection algorithm. Representing a network can be done in different ways [6, Chapter 5], of which we will explore two. When the dataset is very limited, one can try to draw the network and work with this to do further analysis by hand using pen and paper[1]. We already introduced this representation before in Figure 1.1, without mentioning it explicitly. There, we can quickly grasp the full extent of the network.

The previous representation is a very intuitive way of dealing with networks. When these are not too large, it can be easy to see the most important structures in the data. To come to a method that is on the one hand mathematically more pleasant and also works with larger datasets, we introduce the concept of the *adjacency matrix* [7, Chapter 2].

Definition 1.3. Let N be the number of nodes. Then the adjacency matrix A of a (weighted) network is an $N \times N$ matrix with entries

$$A_{ij} = \begin{cases} w_{ij} & \text{if there is a link between node } v_i \text{ and } v_j \\ 0 & \text{otherwise} \end{cases},$$

where w_{ij} represent the weights of the edges. When dealing with a unweighted network, all w_{ij} are set to 1.[2]

[1] Of course one may use methods other than pen and paper to do the calculations, like a blackboard, a tablet etc.

[2] Most real-life networks have far fewer connections than theoretically possible. It is to say that most nodes are connected to only a few other nodes. This results in a matrix with many 0's, called a sparse

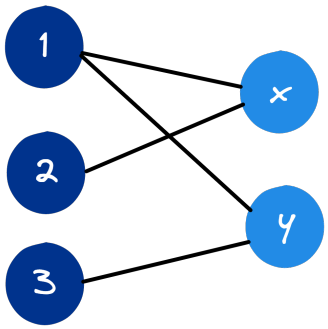


Figure 1.3: The bipartite network from Example 1.4.

Note that there is a one-to-one connection between the set of edges E and the adjacency matrix A. The rows represent the starting point of an edge, and the columns the ending point. A non-zero element denotes that a particular edge is present in E. Hence, one can construct A from E and vice versa. Also note that when we are working with undirected networks, we have that A is symmetric, so $A = A^T$. This is not necessarily the case when the network is directed.

In case of a bipartite network as defined in Definition 1.2, i.e., where we have two distinct types of nodes where only inter-group connections are possible, we can define the adjacency matrix a bit differently by using our knowledge of the network. This representation will be important later on in Chapter 3.

We know that there can only be a link between nodes of different type. If we again indicate the two groups by V_1 and V_2, and take $N_i = \#V_i$, we can take the adjacency matrix A to be an $N_1 \times N_2$ matrix with A_{ij} indicating if there is a link between $v_i \in V_1$ and $v_j \in V_2$.

The matrix A is no longer symmetric, but we gain an important property. We can easily make the projection from the bipartite network to a homogeneous network consisting of the nodes of either V_1 or V_2. This is done by making the matrix multiplication $A \cdot A^T$ or $A^T \cdot A$, respectively. This is illustrated in the next example.

Example 1.4. Suppose that two claims were reported to an insurance company, claim x and y. In total, three persons were involved in these claims, namely person 1 and 2 in claim x and person 1 and 3 in claim y. This network is represented in Figure 1.3. Note that we will opt to use undirected graphs when working with claim data. This enables us to move through the graph from one claim to the other. If we were to use directed graphs, say from a claim to the persons involved, we could start of in a claim in the network and move to a person, but there would be no way of going from this person to another claim due to the inherent direction of the connections.

matrix (see Figure 2.1 later on in Chapter 2 for an illustration of this). Saving it like this on a computer results in a lot of wasted memory and ultimately slows down your calculations. Therefore, these matrices will most often be stored in a more memory-efficient way. This falls outside the scope of this thesis, and will not be discussed further.

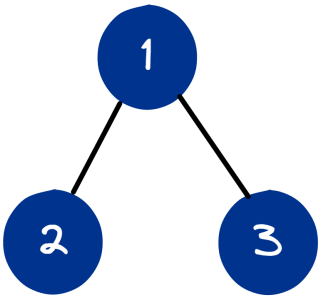


Figure 1.4: The projected network from Example 1.4.

The question that we are asked is to project this bipartite network onto a homogeneous network representing the relationships between the three claimants. It is clear from the picture that we want a connection between 1 and 2, and 1 and 3.

Mathematically, we can do this as follows. We use the alternative definition of the bipartite adjacency matrix, and construct A as

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{1.1}$$

Here, the rows represent the persons involved, and the columns correspond to the different claims.

The adjacency matrix of the projection is obtained by multiplying A with its transpose:

$$\begin{aligned} A \cdot A^T &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 2 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}. \end{aligned}$$

When translating this new adjacency matrix back to a network, and ignoring the diagonal elements, i.e. the self-connections, we obtain the one represented in Figure 1.4. This is exactly the one we expected using the gained insights from Figure 1.3.

Note that the diagonal denotes the degree of the nodes in the original network (see Section 1.3.1). When doing the projections via matrix multiplication these self-connections will always arise. Depending on the context, one can choose to ignore these diagonal elements.

1.2 Network Embedding

The previous section laid out the most important concepts that define a network. In this section, we explore the concept of network embeddings. We begin with the mathematical

definition [4], and then explain what it means and why it is an important concept to study in the context of this thesis. Each node/network embedding is defined by an embedding function.

Definition 1.5. Let $d \geq 1$ be the dimensionality of the node/network embedding. A node embedding function f is a map $f : V \to \mathbb{R}^d$ that maps each node $v \in V$ to a real-valued feature vector in $\mathbb{R}^d$, where $d << |V|$.

A network embedding is used to translate the network structure into tabular format. Each node is an entry in the dataset and the embedding tries to capture the most important information in the network and translate that into d features. It is preferred to have a d that is not too large, i.e., we want to make a low-dimensional projection of the data while losing as few information as possible [20]. These *network features* can then be used to enrich an existing dataset that already contains some *intrinsic features*, i.e. the claim-specific information used in classic methods. These intrinsic features are independent of the nodes surrounding [36, 28], e.g., the age of the policyholder or the location of the accident in the context of a car insurance.

These newly generated network features can then be used to solve the network analysis task very efficiently with classic machine learning algorithms. Since Definition 1.5 is quite broad, a multitude of different embedding functions have been constructed and researched. We will handle some of these in the next section.

1.3 Related Work

In this next section, we explore the literature on network embedding to see what information can be extracted from the network, how it can be done efficiently and how it can be used to detect insurance fraud. We explore different paths that can be taken when trying to find patterns in the data. First, we discuss some basic concepts regarding information that is present in the network. After that we look at the three main branches identified in the literature, namely anomaly detection, guilt by association, and random walks combined with natural language processing.

This section is meant to give a high-level overview. We go into more detail later for those algorithms that will be applied.

1.3.1 Basic Features

This first set of embeddings deals with calculating different metrics, namely the neighbourhood and centrality metrics, of each node. These are elementary concepts and are therefore covered in multiple textbooks about graphs [6, 7]. As we will see later, some algorithms rely on these node characteristics to uncover anomalies in the data.

An important neighbourhood metric is the degree of a node. This is nothing more than the number of connections of that node, i.e the number of edges of that node. When we are working with undirected graphs, and we use the adjacency matrix as in

Definition 1.3, the degree can be calculated by taking the row or column sum:

$$d_i = \sum_{j=1}^{N} A_{ij} = \sum_{k=1}^{N} A_{ki}. \tag{1.2}$$

In the context of fraud detection, one can also define the fraud degrees. These count the node's absolute or relative number of neighbours that are known to be fraudulent cases.

Other features that capture the importance of the nodes in the network are called centralities. These measures are based on the paths one can take in the network. So, before defining different centralities, we will first handle the paths in a network. A path is defined as follows [13, 7].

Definition 1.6. A path of length n in a network $G(V, E)$ is an ordered list of n connections:

$$(e_1 = (v_1, v_2), e_2 = (v_2, v_3), \ldots, e_n = (v_n, v_{n+1})) .$$

Here, it is possible that $v_j = v_k$ for $j \neq k$, as they are just numbered according to the step time the path goes through the node. One can pass through the same node multiple times.

A special path is the geodesic or shortest path possible between two nodes. The length of this path is called the geodesic distance between these nodes.

Using this definition, we will define the neighbourhoods of a node formally [40]

Definition 1.7. Given a node v_j from the network $G(V, E)$, we define the order n neighbourhood of v_j, $\mathcal{N}_j^n$, as the subset of the nodes V containing the nodes that have a distance to v_j equal to or lesser than n. In addition, the order n neighbourhood also contain the node v_j.

The strict order n neighbourhood contains all nodes at exactly distance n, i.e., all nodes in $\mathcal{N}_j^n \setminus \mathcal{N}_j^{n-1}$.

Going back to fraud detection, it can be of interest to know how far a node is from fraudulent nodes. If a node is close to a fraud case, then the latter can have a high influence on the former.

As said before, these geodesic path are used to calculate different centrality metrics. A first one is the closeness centrality, which measures the average distance between our node of interest e_i and all other nodes in the network:

$$c_i = \left(\frac{\sum_{j \neq i} d(v_i, v_j)}{n-1} \right)^{-1}, \tag{1.3}$$

where $d(v_i, v_j)$ denotes the (geodesic) distance, i.e., the length of the shortest path between nodes v_i and v_j. The term between brackets in Equation (1.3), the average (geodesic) distance, is called the farness. This is expressed mathematically as $f_i = c_i^{-1}$.

Another centrality measure is the betweenness. This metric expresses the number of shortest paths between any two nodes in the network that pass through a given node. If

we express the number of geodesic paths between nodes v_j and v_k as g_{jk} and the number of those paths that contain v_i as $g_{jk}(v_i)$, then we can calculate the betweenness of node v_i as:

$$b_i = \sum_{j<k:j,k\neq i} \frac{g_{jk}(v_i)}{g_{jk}}. \tag{1.4}$$

In a way, more information passes through nodes that have a higher betweenness centrality. These nodes are also more likely to be the connection between different communities in the network. This is illustrated in the next example.

Example 1.8. We take the network as in Figure 1.5. We clearly see two communities with three nodes each, and one node in the centre that connects the two. As said before, we expect this connecting node to have the highest betweenness centrality.

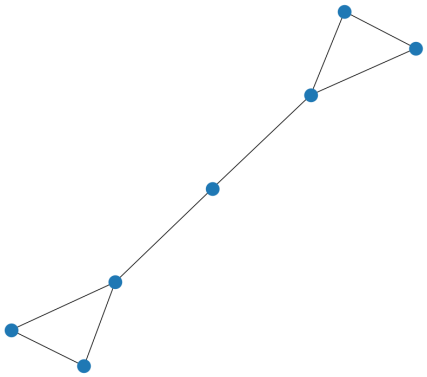


Figure 1.5: The network from Example 1.8.

This is also what we see. The middle node has a b_i of 0.6. The two that it is connected to both have a betweenness centrality of 0.533... . The other four nodes have a betweenness of 0.4.

Due to the particular shape of this network, the closeness centrality follows a similar trend, with values of 0.6, 0.5454... and 0.4, where we take the same order of nodes as for the betweenness.

These basic neighbourhood features are the basis to uncover uncommon behaviour. This is done via the principles of anomaly detection discussed in the next section.

1.3.2 Anomaly Detection in Social Networks

This part is based on the literature overview of [8]. In the paper, anomalies are defined as patters whose behaviour varies significantly from the majority of the data. Anomalous behaviour in a pattern can be, e.g., one node that has a high betweenness centrality while all others have one that is much smaller. Although not all anomalous patterns are ill-natured, these can be used to uncover fraudulent behaviour. Since less focus will be put on these techniques, we quickly go over them for the sake of completeness.

Using network based anomaly detection gives an edge over the classic anomaly detection methods. As mentioned in [8] and [6], fraudsters try to mimic normal behaviour to avoid detection. However, due to the robustness of the network, it is much harder to hide your tracks in the network, since these are now detected using the patterns hidden in the network.

Different types of anomalies can be detected; anomalous nodes, edges or sub-graphs. For this, different approaches can be used. The first is clustering/community-based. This is used for both anomalous node and edge detection. The important part is to apply an algorithm to cluster the nodes/edges in some way, by defining communities and neighbourhoods in the network. The anomalies are those who interconnect different clusters but do not seem to belong to any of them.

A second method is *network-structure based*, which is used for anomalous node and sub-graph detection. The idea here is to apply basic network embeddings, as discussed in Section 1.3.1, and feed the newly constructed data points into classic anomaly detection algorithms to see which nodes/sub-graphs are outliers.

The last method we are going to consider is *signal processing based* and is specifically used for anomalous sub-graph detection. Here, comparatively small anomalous sub-graphs are treated as signals. A more general approach is to study the graph's Laplacian matrix's spectral properties to uncover possible anomalous sub-graphs in the network.

The interested reader can find more details and references in [8].

1.3.3 Guilt-by-Association

Guilt-by-association algorithms, also called collective inference algorithms, try to obtain stronger signals by combining weak ones [22]. The idea is that we can make well-educated guesses about the type of every node by just knowing a few node types in the network. This leverages the social phenomenon that fraudster tend to cluster together, which is called "homophily". If you have filed a new claim, and it is known that your previous two claims were fraudulent, it is highly expectable that this newly reported claim is also fraudulent. Inside our claim network, these claims will all be close to each other, since they are all connected to the same person, and will therefore influence each other.

It is shown in [22] that the most popular methods, Personalized PageRank, Semi-Supervised Learning and Belief Propagation, are very similar. They can all be transformed into the problem of solving a linear system in matrix notation. These are then solved in an iterative way.

The idea of the PageRank [30] has been extended to bipartite networks to be used in social security [39] and credit car fraud detection [38]. More recent work [28] has used the BiRank algorithm [19] in the context of insurance fraud detection. This BiRank method is the object of study in Chapter 3.

All of the above papers use the *fraudscore* coming from the iterative methods as an additional source of network information, pooled together with information from the node's neighbourhood and intrinsic features to build a fraud detection algorithm.

1.3.4 A Combination using an Expert System

One of the first papers dealing with fraud detection in automotive insurance using social networks [36] combines what we have seen so far. The main idea is to go through different steps in order to uncover groups of fraudulent persons. As mentioned in the paper, the authors are convinced that a stand-alone fraud detection algorithm is not enough to uncover maleficent claims. That is why the input of an industry expert is needed in the different steps. In the end, it is also this person that needs to interpret the results in order to see which claims need to be further investigated. Hence, no decisions are done by the system alone. In this section, we go a bit deeper into the paper [36] to indicate where the different aspects of the previous sections come into play.

The first step is, of course, building the network from the dataset. After this is done, one needs to break up the connected components in the network into smaller communities. A community is a set of nodes that are closely connected to each other, with less connections to the entities outside of their community. This splitting is done by recursively deleting connections from the networks. The authors propose to do this using the betweenness of the edges. Before, we have talked about betweenness centrality of the nodes using Equation (1.4). The same can be defined for the connections by counting the number of geodesic paths that contain the connection in consideration. Similarly to what we have seen in Example 1.8, edges that connect different communities will have a greater betweenness than those inside a community. Therefore, this metric is used to remove the edges and decompose the network into the different communities as best as possible.

After the decomposition of the network, multiple structural characteristics are set in collaboration with the domain expert in order to detect suspicious components. These rules are set up via indicator variables to express whether a characteristic is suspicious or not. To help with setting these indicators, one can try to gain insights using anomaly detection methods, like we have seen in Section 1.3.2.

Each of these suspicious communities are looked at in more detail to see which of the nodes inside them are really malicious. This is done with what is called in the paper [36] an Iterative Assessment Algorithm (IAA). This starts by assigning a preliminary suspicion score to each node based on different rules, set up together with the domain expert. These scores are then iteratively recalculated using the scores of the neighbours of each node until convergence is reached. This IAA is nothing more than a guilt-by-association algorithm, which we have already discussed briefly in Section 1.3.3. This way, the nodes with the highest suspicion score are reported back to the domain expert who can then do the final assessment to see if the claims are fraudulent or not.

Hence, the expert system as described in [36] is a good case study that summarises the different aspects we have seen so far. The next sections will deal with more novel techniques to extract meaningful information from a social network.

1.3.5 Random Walks with Natural Language Processing

The final branch of literature studied here uses natural language processing algorithms as its workhorse. Each of the algorithms starts by sampling multiple short paths from the network. The sequence of nodes in these paths are seen as sentences and are fed into an NLP algorithm to find a community structure in the network [20]. A popular choice for homogeneous networks is the skip-gram method [25], while for heterogeneous networks metapath2vec [14] is an important method. This last algorithm will be studied in Chapter 4 in the context of motor insurance fraud.

The way in which these paths are selected is an active research question. The question is how to find a performant method that also scales easily to very large networks.

The first of these algorithms is DeepWalk [32]. The path sampling is done by taking short random walks through the network. The starting point of each walk is uniformly picked from the network's nodes. The walk proceeds by picking one of the neighbours uniformly as the next node.

After the walk is finished, the new *sentence* is processed and the representations updated using the NLP algorithm. Then one picks another node and repeats the process. In reality, it is advised to loop over this a couple of times, in order to have multiple paths starting at each node.

In the end, the nodes obtain, what is called in the literature, their final vertex representations, denoted by $\Phi \in \mathbb{R}^{|V| \times d}$. This is just the lower-dimensional representation of the network. These representations are then used to cluster the nodes using one-vs-rest logistic regression.

It is observed that the DeepWalk algorithm is used as an additional baseline model against which the following models are compared in the literature, some of which are extensions to the basic idea of the random walk used in DeepWalk.

One of these extensions is node2vec [17]. This method introduces two new hyperparameters in order to have biased random walks through the network. Depending on the values, the walk is more inclined to come back to the previous node or move away from it. This has as a consequence that the clusters can be vastly different.

A nice example given in the paper [17] is based on the novel Les Misérables. The nodes are the characters, and an edge exists between two nodes if characters co-appear. Depending on the choice of the bias parameters, one can uncover either homophily or structural equivalence in the network. The homophily presents itself as the characters that mostly appear together and form something resembling a community in the storyline. The structural equivalence manifests itself by clustering characters that act as bridges between storylines, and grouping all peripheral characters that do not interact much with other characters.

The previous two methods are very similar in what they do. This also means that they have more or less the same limits in what they can do. Although the different ways to sample the paths makes it possible to uncover different aspects of the data, they are still limited in their use due to the fact that they rely solely on the skip-gram algorithm.

In addition, sampling the paths and updating the representations takes time. This could mean that the algorithms are not scalable to very large graphs.

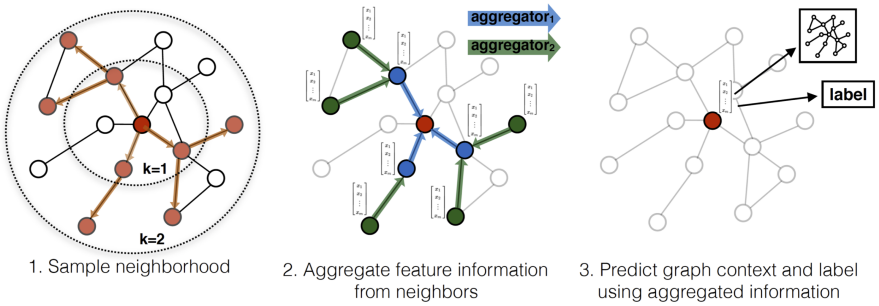


Figure 1.6: Illustration of the GraphSAGE algorithm. Source: [18].

The next sections explores some algorithms that each tackle one of these two problems.

1.3.6 Further Extensions

The first extension, namely the FastRP [10] algorithm, takes another route, where its main objective is to speed up the procedures compared to DeepWalk and node2vec. The idea here is to use sparse random projections to extract the network features. Using the same notation as in Definition 1.5, we take a sparse random matrix $R \in \mathbb{R}^{|V| \times d}$, where $d << |V|$, and use it to project the adjacency matrix onto the lower dimensional space:

$$N_1 = A \cdot R.$$

In practice, one first multiplies A with a matrix L to rescale the values.

An iterative procedure is used to capture higher order neighbourhood information of the network:

$$N_k = A \cdot N_{k-1}$$

The final projection is then a weighted sum of the different N_k.

It is claimed that this speeds up the process with a factor of 4000 without loosing to much performance. It comes, however, at a cost of having to calibrate much more hyperparameters, like the k and all the weights.

GraphSAGE [18] extends the idea of DeepWalk and node2vec. It samples a random neighbourhood of a node at depth K (one or multiple steps away from the original node). It takes the features of the nodes furthest away and aggregates this to the nodes one step closer. The aggregation step is done via non-linear functions. This is repeated until a final label for the source node is obtained. Here, one can choose which aggregator to use. This is visualised in Figure 1.6.

Using inputs and aggregating them via a non-linear function is the key idea of a neural network. When we do this in a context of network embeddings, it is therefore called a graph neural network (GNN) [1]. Here, the aggregation functions can each be individual neural networks. We illustrate this in Figure 1.7.

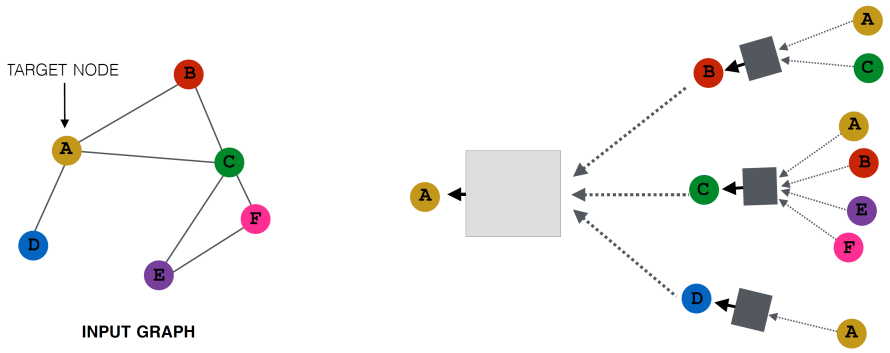


Figure 1.7: The illustration of the network and the corresponding GNN, where each aggregation is done by a neural network, represented via the box. Source: [1, 06-GNN1]

1.4 This Work

From the sections above, one sees that there is already extensive literature out there covering network embeddings. These network techniques have also found their way into fraud detection methods. However, the literature on fraud detection using social networks specifically for insurance claims is still scarce. This thesis adds to that literature in two ways.

First of, we apply different techniques to our dataset of motor insurance claims. To our best knowledge, some of these techniques have not been applied to this context thus far. This helps to extend the scientific research on the topic.

Secondly, when dealing with existing use cases, we extend these to gain further insights. On the one hand, we are interested in how these additional features obtained by the different embeddings influence the predicted fraud probability. On the other hand, we study the additional knowledge that is gained by incorporating these network features.

Chapter 2

The Data and its Transformations

In this smaller chapter, we explore the dataset provided, and how to transform it for later use.

The transformations we are going to use are based on projections that were already briefly illustrated on a toy example in Example 1.4. This example is extended in Section 2.2 to, what we will call, an n-partite network. These projections are essential in a couple of ways. As a start, most algorithms discussed in the literature are build for homogeneous graphs, i.e. graphs for which the nodes all have the same type. As we will see, these projections can reduce some graphs to their homogeneous form. The concept of projections will be used in Chapter 3 to project our network to a bipartite graph, not a homogeneous one, since the algorithm we are going to look at is specifically built for that.

Next to the technical need for projections, it can be that we want to project the network to one with fewer types of nodes. As we will see in Section 2.1, there are a lot of actors involved in the network that are not claims. A consequence of this will be that the claim nodes themselves will be far apart. For the purpose of this thesis, we are only really interested in these claim nodes. Plus, the only fraud data we have is for the claims. Although it might theoretically be possible to work with the network right out of the box, it may help the stability of the results if we first project it down to a graph with fewer node types without losing (too much) information. Therefore, Section 2.2 will be essential for our work.

The final section of this chapter, Section 2.3, analyses a simple but crucial aspect of the network, namely homophily. The term homophily means, in simple terms, that fraudsters have a tendency to cluster together in the network. This concept is important when dealing with guilt-by-association algorithms, as we will do in Chapter 3, since these algorithms start from the assumption that someone who is closely connected to a known fraudster is more likely to be one themselves.

2.1 The Dataset

Throughout this thesis, we are going to work with the same subsample of the graph dataset provided[1]. It consists of around 994 000 nodes of four different types. The nodes that are of most interest to us are the ones that represent claims. Next to the claims, we also have car nodes, policy nodes and nodes that represent the brokers. The number[2] of the different nodes in our network are summarised in Table 2.1.

Note type	Number
Claims	400 000
Cars	400 000
Policies	190 000
Brokers	4 000
Total	994 000

Table 2.1: The number of different nodes in our network.

With nodes alone, we do not have a network. Edges are also required. For the ease of use, we assume that all 1 030 000 edges are undirected. These are constructed in such a way as to make interpreting and working with the graph easier. Cars are connected to claims in which they were involved. Next to the connection with the claims, there are links connecting cars with the policies that cover them. The final kind of connections are between the policies and the brokers who sold them. Note that we can already see that the brokers will serve as hubs, since there are much more edges connected to brokers (190 000) than there are brokers in the network (4 000). Table 2.2 summarises the number of edges according to their start and end point.

Edge type	Number
Claim-Car	600 000
Car-Policy	240 000
Policy-Broker	190 000
Total	1 024 000

Table 2.2: The number of different nodes in our network.

Due to types of links in the network, we are dealing with a 4-partite network. This is a generalisation of the bipartite network defined in Definition 1.2. We have a clear direction between the nodes, i.e.
claims $\longleftrightarrow$ cars $\longleftrightarrow$ policy $\longleftrightarrow$ broker, and there are no other type of connection, e.g. there are no links between claims and policies directly. This is important for the theory in Section 2.2 to work.

We take a closer look at the structure of the network. We start by looking how (dis)connected our graph is. We have that the graph consists of around 4 000 connected

[1]This dataset is provided by Allianz. For confidentiality reasons, all numbers represented here are rounded.

[2]Since claim and fraud data are highly sensitive, we only give a rough approximation of the numbers, which can either be rounded up or down.

components. Between any two nodes of a connected component, one can find a path. In addition, we say that there are no paths between this connected component and any other node in the network that is not part of it. There is a significant difference between these sub-graphs. The largest one consists of around 962 000 nodes, while the second largest connected component is only 283 nodes big. Then the number of nodes goes down quickly, with more than half the sub-graphs containing only two observations.

In addition, we calculate different summary statistics for the degree of each node to get a grip on the connectedness of the graph. The information is summarised in Table 2.3. Note that we look at a more granular level to the tail of the degree distribution, since this captures some crucial insights.

Node	mean	std	min	25%	50%	75%	85%	95%	99%	max
Claims	1.49	0.58	0	1	1	2	2	2	3	103
Cars	2.01	5.93	1	1	2	2	3	5	7	2351
Policies	2.26	30.27	2	2	2	2	2	3	3	8797
Brokers	45.97	114.19	1	2	9	37	74	210	556	1915

Table 2.3: The summary statistics of the node degrees for the different types of nodes.

We see a few different things. First of all, claims have on average the lowest degree, while brokers have the highest average number of connections. For all but the broker nodes, the degree stays low for most, with only a few nodes with many connections. Only for the brokers, we see a much heavier tail with relatively more observations in this tail. This corroborates the observation made above that the brokers will serve as the glue of our network. These insights will be important later on when we introduce the embedding algorithms, since it may be crucial for the information flow that we have larger connected components in our network. Despite this, the observation with the highest degree is a policy node[3].

These relatively low degrees for the different nodes have as a consequence for our adjacency matrix is sparse[4]. To illustrate that this is also the case for our network, we plot the adjacency matrix in Figure 2.1 from the homogeneous network of claims (see Section 2.2 for the construction of this). The first 300 claims are shown, and each non-zero value in the matrix is indicated in blue. One can clearly see that most claim nodes are connected to just a few other claims.

2.2 Projections

One of the challenges with the network introduced in the previous section, is that the claim nodes can be far apart. Since these are the only ones carrying the fraud data, the data can be diluted too much when applying different algorithms on the full graph.

[3]It should be noted that the maximal numbers for the different nodes are extraordinary high. This could indicate some data quality issues. This is, however, not further investigated in this thesis. For practical implementations of the different algorithms, one should first look into this further before going to the modelling part.

[4]This sparseness of the adjacency matrix is widely observed for other social networks as well.

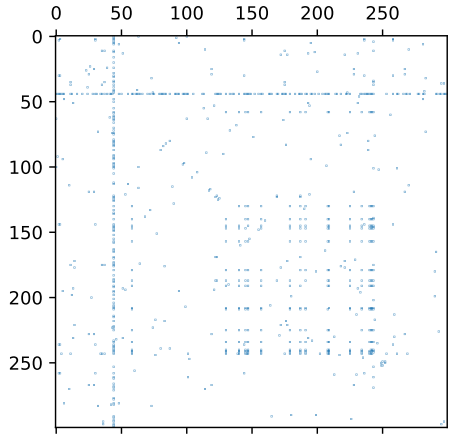


Figure 2.1: The adjacency matrix of the claims network is a sparse matrix.

Hence, embeddings coming from these algorithms may be harder to exploit for fraud detection purposes. That is why it is important that we find a way to transform the network such that, on the one hand, the claim nodes are closer together, and on the other, the distance and strength of the connections in the original graph is still there in some way. What we mean by the last objective is illustrated in Figure 2.2.

Figure 2.2 illustrates a network where there is one broker, B, who has sold two policies, P1 and P2. For policy P1, two claims were filed, namely C1 and C2. One additional claim C3 was filed falling under policy P2. In our dataset, the car nodes are also included, but we have not included them in this example to make the illustration of our problem easier.

All claims are connected to each other, but we see that four steps are required to go from C3 to any of the other claims. This would grow to as much as six steps if we also included car nodes. Hence, there is a need to move to a network representation where all claims are still connected, but much closer than in the current one. However, we need to be careful. It is clear that C1 and C2 are much closer to one another than to C3. It is important to also have this information after transforming the network when we want to use the newly constructed network for fraud detection. When we know C1 is a fraudulent claim, it will have a higher impact on C2 being fraudulent, since they are covered by the same policy, than on C3, since they are only connected via the broker. This is an important consideration that needs to be taken into account when moving forward.

In this thesis, we will use two transformations of our original network. To do this, we build further on the idea of making projections from a bipartite to a homogeneous network, as introduced in Section 1.1.2. For our full network, we construct three adjacency matrices. One representing the links between claims and cars, one for the links between cars and policies, and one for the policies and brokers. We denote these with

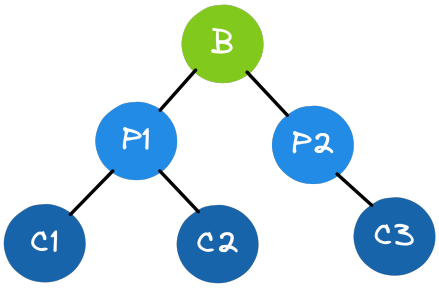


Figure 2.2: A network with one broker who sold two policies for which three claims were filed in total.

A_{cc}, A_{cp} and A_{pb}, respectively. In order to obtain the desired results for our homogeneous network, we first need to move to a bipartite representation. This is needed to preserve important information about the strength of the connections from the original graph. An illustration on how it is done and why it is important is given in Example 2.1 later on.

We divide the graph into two groups, namely the claims and all other nodes. This last group will be called the parties in the network. Then the bipartite adjacency matrix is constructed as follows. We begin by constructing the matrices that represent the links between the claims and all other parties individually. We already have the one for claims and cars for free, namely the matrix A_{cc}. Next, we construct the adjacency matrix for claims and policies, denoted by A_{cp} as follows:

$$A_{cp} = A_{cc} \cdot A_{cp}.$$

Finally, in the same spirit, we can construct the adjacency matrix for claims and brokers, A_{cb}:

$$A_{cb} = A_{cp} \cdot A_{pb}.$$

We now have the three adjacency matrices representing the links between the claims and each subgroup of the parties. The final step is to combine these matrices into one adjacency matrix for the bipartite network, by putting the columns of the matrices one after the other in one large matrix:

$$A_{\text{bi}} = \begin{pmatrix} A_{cc} & A_{cp} & A_{cb} \end{pmatrix}.$$

Moving to the homogeneous network with only claim nodes can easily be done in the same way as for the bipartite network in Example 1.4:

$$A_{\text{hom}} = A_{\text{bi}} \cdot A_{\text{bi}}^T.$$

The next example illustrates why it is necessary to first make the move to the bipartite matrix in order to have the desired outcome for the homogeneous network.

Example 2.1. We go back to the network as represented in Figure 2.2. Contrary to our full network, we only need to construct two adjacency matrices.

$$\begin{aligned} A_{cp} &= \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}, \\ A_{pb} &= \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \end{aligned}$$

This then leads to the following results for the bipartite and homogeneous network, which are represented in Figure 2.3.

$$\begin{aligned} A_{cb} &= A_{cp} \cdot A_{pb} \\ &= \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \\ A_{\text{bi}} &= \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}, \\ A_{\text{hom}} &= A_{\text{bi}} \cdot A_{\text{bi}}^T \\ &= \begin{pmatrix} 2 & 2 & 1 \\ 2 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}. \end{aligned}$$

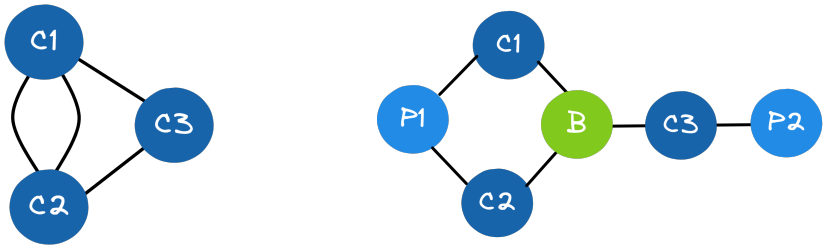
From this results, we can clearly see that the claims C1 and C2 are more closely connected to each other than to C3, since they have two edges between them, but only one to C3. We note that the representation of the projection in Figure 2.3a can also be adapted using weights as defined at the end of Section 1.1.1. Instead of using multiple connections, we could assign just one connection between each pair of the nodes, where the connections between C_1 and C_3, and C_2 and C_3 have weight 1, while the connection between C_1 and C_2 has weight 2.

Finally, we explain why we stress that it is important to construct the bipartite network like this, in order to have the desired result. This is illustrated via the following equation. One could also do the calculations using the adjacency matrices as

$$A_{\text{hom}} = A_{cp} \cdot A_{pb} \cdot (A_{cp} \cdot A_{pb})^T,$$

the result would be

$$A_{\text{hom}} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$



(a) The homogeneous network after projection.

(b) The bipartite network after projection.

Figure 2.3: The results of the different projection steps.

All claims are again connected, but we lose the information from the original network, since the strength connections between the three claims cannot be distinguished.

In the next chapter, we are going to apply the BiRank algorithm to our network after applying the projections as explained in this chapter. The procedure requires a bi-partite network, so the projection procedure described above will be a very important step. Before moving to the BiRank algorithm in the next chapter, which is a procedure falling under guilt-by-association, as described in Section 1.3.3, we need to make sure that it makes sense to do so. For this, we are going to investigate whether there is homogeneity in our network of claims in the next section.

2.3 Homophily in the Network

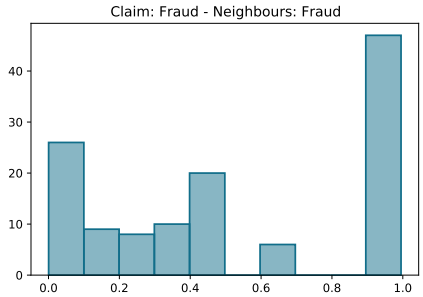
The premise of the next chapter in this thesis is that we can derive meaningful information for unlabelled nodes in the network via the fraud information from the other nodes in its vicinity. The concept that is important here is homophily [24]. Homophily is the social phenomenon where people who have similar characteristics, e.g., gender, age, occupation, tend to know each other better than people that are more different from each other.

In the context of fraud and social network analysis, it means that people who have bad intentions are more closely connected to one another, and less to people who file legitimate claims. It can be explained by the fact that people need to work together with multiple parties in order to stage the accident and file the wrongful claim. In addition, having more connections outside of your 'crime network' puts you at higher risk of being detected.

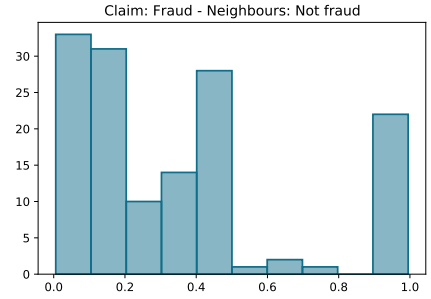
This homophily in the insurance fraud social networks was the onset of the work presented in [28]. Hence, if we want to apply a similar approach to our data, we need to check if there is indeed homophily in our network. We do this by considering the neighbourhoods of the labelled nodes in the dataset. For each neighbourhood, we calculate the ratio of number of frauds or non-fraud over all nodes in the neighbourhood. We then compare the histograms and average ratios over all fraudulent and legitimate claims to

uncover whether fraudulent claims tend to be more connected to other frauds than to non-frauds.

We start with the fraudulent claims. As can be seen on Figure 2.4, there are more fraudulent claims in the neighbourhood of a proven fraud case than non-fraudulent claims. The x-axis shows the ratios, and the y-axis represents the frequency (in absolute numbers). We note here that we did not include those claims that did not have any labelled claims in their neighbourhood. Including these would make the figures less clear since this one bar would dominate the picture, but would not change our conclusions. We see for example on Figure 2.4a that there are more than 40 fraudulent claims for which almost all neighbours are also labelled as fraudulent, i.e., having a ratio close to 1. On the other hand, Figure 2.4b tells us that there are only between 20 and 25 fraudulent claims whose neighbourhood is almost entirely made up of non-fraudulent claims, again having a ratio close to 1. When analysing the averages, we have that the average ratio of fraudulent neighbours is 7.89%, while the percentage of legitimate neighbours is 6.24%. This is already a confirmation that there is some homophily present in our network.



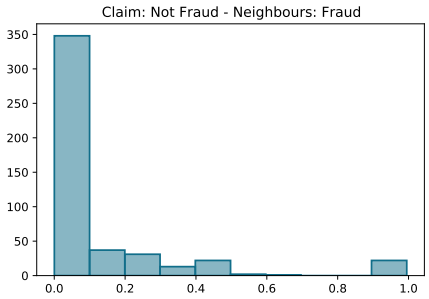
(a) The histogram of fraudulent claims in the neighbourhood of the fraudulent claims. The x-axis denotes the ratio of fraudulent neighbours to all neighbours of a claim.



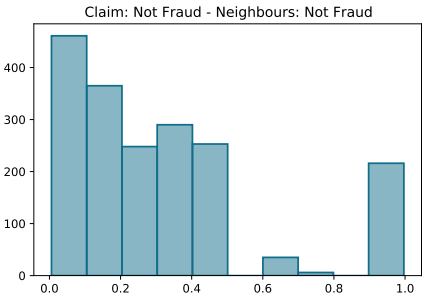
(b) The histogram of non-fraudulent claims in the neighbourhood of the fraudulent claims. The x-axis denotes the ratio of non-fraudulent neighbours to all neighbours of a claim.

Figure 2.4: The neighbourhoods of fraudulent claims.

Next, we do the same for the claims that have a non-fraud label. Their histograms are plotted in Figure 2.5. We did not include the neighbourhoods with no labelled claims, similarly as before. Here, the results are much more pronounced. Almost none of the neighbours of legitimate claims were labelled as being a fraud, as can be seen on Figure 2.5a. Figure 2.5b, on the other hand, shows that the ratios of non-fraudulent neighbours are more often away from zero. The average ratio tells a similar story. When looking at the neighbours of a legitimate claim, the average ratio of fraudulent claims is 0.42%, while this is 4.81% for non-frauds. This clearly shows that there is homophily in the network. We will now try to leverage these insights in the next chapter.



(a) The histogram of fraudulent claims in the neighbourhood of the non-fraudulent claims. The x-axis denotes the ratio of fraudulent neighbours to all neighbours of a claim.



(b) The frequency of non-fraudulent claims in the neighbourhood of the non-fraudulent claims. The x-axis denotes the ratio of non-fraudulent neighbours to all neighbours of a claim.

Figure 2.5: The neighbourhoods of non-fraudulent claims.

Chapter 3

Assigning Fraud Scores with the BiRank Algorithm

With this chapter, we start with the most important part of the thesis. We begin applying different algorithms to extract meaningful information from our network. In this chapter, we introduce the BiRank algorithm [19], and apply it to the network introduced in the previous chapter.

In Section 3.1, the BiRank algorithm is introduced and we explain how this can be used to uncover insurance fraud. This is inspired by the work done in [28]. Next, we apply the algorithm on our dataset in Section 3.2. Section 3.3 summarises the predictive power we obtain from this score. For this, we introduce two ways of doing the train-test split. Finally, further extensions to the procedure are introduced in Section 3.4 in order to try to enhance the results obtained in the Section 3.3.

3.1 The Algorithm

Before discussing the algorithm, we introduce some notation. As already mentioned, we deal with a bipartite network. The nodes in the network either represent a claim or a (counter-)party involved (car, policy or broker). This is why we will denote the set of nodes as $V = C \times P$. The goal is to obtain two vectors $\boldsymbol{c} = (c_i)$ and $\boldsymbol{p} = (p_j)$ that contain the fraud scores of the different nodes. Of course, only the vector $\boldsymbol{c}$ with scores for the claims will be of interest to us. To reach this goal, we apply the BiRank algorithm in this chapter.

The BiRank algorith [19] is a bipartite generalisation of the PageRank [30]. It is used to assign a score to each node, representing its importance in the network. In addition, the procedure is set up to incorporate a *smoothing effect*. The score of a node is updated using the scores of all its neighbours. This also means that a node is seen as "more important" if it is connected to other important nodes in the network. Notice that for our use case, being important means that you have a high fraud score. For the sake of explaining the algorithm, we will stick to the word "importance" for now.

Another desirable feature of the algorithm is that the scores are normalised using the square root of the degrees of the nodes (cf. infra). Firstly, we argue why it is desirable that the influence of a neighbour is inversely proportional to the node's own degree. Suppose the node under consideration is the only connection of an important node. Then, we can suppose that the node considered is also important, since the other node will have a high influence on it. On the other hand, if the considered node is one of many neighbours of an important node, we may find the information coming from the neighbouring node less relevant. In summary, if you are the only friend of a criminal, you are likely to be an accomplice, but if you are one of many acquaintances of that same criminal, it is less obvious.

A similar argument can be made for the node under consideration in the other way around. Hence, we also argue why it is desirable that the influence of a neighbour is inversely proportional to the neighbour's degree. If it only has a couple of neighbours, then one of them being important has a higher influence on the node. In the case it has one important neighbour and a lot more neighbours with a lower score, the influence of this important neighbour will be less. If your only friend is a criminal, it is not unreasonable to think that you are too, but if you have many friends, one of which is a criminal, then you being one is less straight forward.

Hence, the influence depends both on the degree of the node under consideration as well as the degree of the neighbouring nodes. Keeping all this in mind, the basic BiRank scores propagate through the network as follows:

$$\begin{aligned} p_j &= \sum_{i=1}^{|C_j|} \frac{w_{ij}}{\sqrt{d_i}\sqrt{d_j}} c_i, \\ c_i &= \sum_{j=1}^{|P_i|} \frac{w_{ij}}{\sqrt{d_i}\sqrt{d_j}} p_j; \end{aligned}$$

where $\mid C_j \mid$ and $\mid P_i \mid$ are the claim and counter-party neighbours of the nodes p_j and c_i, respectively.

In order to be relevant for our case about fraud detection, we want to introduce the available fraud data into the information flow of the network. This is done by using a prior belief vector $\boldsymbol{c}^0$ that encodes the original knowledge we have about the nodes. The final score is a weighted average between this query vector, and the score calculated earlier. Based on the procedure in [39] and [28], we first set c_i^0 equal to 1 if claim c_i is a proven fraud, and 0 otherwise. Next, we divide the scores by the total number of entries/claims to rescale $\boldsymbol{c}^0$. Here, we do not use any prior belief about the party nodes, since we are only interested in the claims being fraud or not. The formulas to calculate

the scores then become:

$$\begin{aligned} p_j &= \sum_{i=1}^{|C_j|} \frac{w_{ij}}{\sqrt{d_i}\sqrt{d_j}} c_i \\ c_i &= \alpha \sum_{j=1}^{|P_i|} \frac{w_{ij}}{\sqrt{d_i}\sqrt{d_j}} p_j + (1-\alpha) c_i^0. \end{aligned} \tag{3.1}$$

The weight α is a hyper-parameter that can be tuned, and should be taken between 0 and 1.

It is clear that each score depends on the scores of the other nodes in the network. Hence, the final score vectors need to be obtained iteratively. To do this, we translate Equation (3.1) into matrix notation:

$$\begin{aligned} \boldsymbol{p} &= \boldsymbol{S}^T \boldsymbol{c} \\ \boldsymbol{c} &= \alpha \boldsymbol{S} \boldsymbol{p} + (1-\alpha) \boldsymbol{c}^0. \end{aligned} \tag{3.2}$$

Here, the normalisation is done using the matrix $\boldsymbol{S} = \boldsymbol{D}_c^{-\frac{1}{2}} \cdot \boldsymbol{W} \cdot \boldsymbol{D}_p^{-\frac{1}{2}}$. Here, $\boldsymbol{W}$ represents the weight matrix, as defined in Definition 1.3. This has entries w_{ij}, which are either zero or non-zero. A non-zero entry indicates that there is a connection between nodes i and j, where the value represents the weight of the edge. If w_{ij} is 0, there is no edge between the two nodes. Note that in an unweighted setting, w_{ij} is either 1 or 0. The parameter α is fixed at 0.85, as this is the value used in the literature [6] and [28]. In Section 3.4.2, this value will be challenged.

Equation (3.2) gives us an iterative procedure to update the scores. The only thing that is left to do is to initialise the starting values of $\boldsymbol{p}$ and $\boldsymbol{c}$. This is done by sampling uniform random values between 0 and 1. The procedure is stopped when a convergence criterion is met, or the algorithm has executed a predefined number of updates of the scores. The full algorithm is given in Algorithm 1.

Algorithm 1 BiRank as given in [28]

Input: Weighted adjacency matrix $\boldsymbol{W}$, initial fraud labels $\boldsymbol{c}^0$
Output: Fraud score $\boldsymbol{c}$

1: Calculate $\boldsymbol{S} = \boldsymbol{D}_c^{-\frac{1}{2}} \cdot \boldsymbol{W} \cdot \boldsymbol{D}_p^{-\frac{1}{2}}$
2: Initialise $\boldsymbol{c}$ and $\boldsymbol{p}$ randomly
3: **while** stopping criteria not met **do**
4: $\quad \boldsymbol{p} = \boldsymbol{S}^T \boldsymbol{c}$
5: $\quad \boldsymbol{c} = \alpha \boldsymbol{S} \boldsymbol{p} + (1-\alpha) \boldsymbol{c}^0$
6: **end while**
7: **return**: $\boldsymbol{c}$

Example 3.1. Before going to the full network, we give a small example to illustrate the algorithm. The main principles will be shown using a small network. This network

is shown in Figure 3.1. It consists of four claim nodes and four counterparties. It is known that node C_1 is a fraud node. We are going to apply the BiRank algorithm to see which other nodes are most suspicious and need further investigation by the fraud expert.

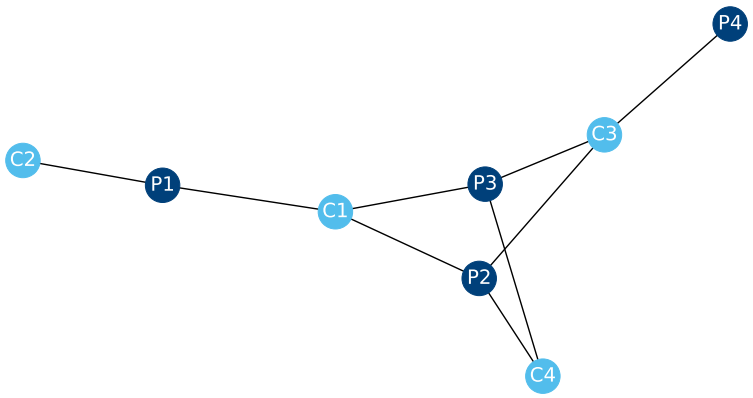


Figure 3.1: The bipartite network from Example 3.1. The counterparties are in dark blue and the claims in light blue.

From our understanding of the algorithm, both nodes C_3 and C_4 should have a higher score than C_2, since these are connected to the fraudulent one via two counterparties. On the other hand, since claim C_3 has an additional neighbour, its fraud score should be lower than that of C_4. Let's now see if our intuition of the algorithm is correct.

In order to let the algorithm run, we need to determine three things. We need to set up the matrix $\boldsymbol{S}$, determine the vector $\boldsymbol{c}^0$ and fix the parameter α. Beginning with the simplest of the three, we fix α at 0.85. This is the value we are going to use throughout the rest of the thesis, and it is based on the literature [28].

Next, we move to the vector, $\boldsymbol{c}^0$, with a priori knowledge of the illicit claims. As mentioned before, we have four claims with only the first one being flagged as fraud. This results in $\boldsymbol{c}^0 = (1, 0, 0, 0)^T$. Note that a "0" either means that the claim has been investigated and deemed legitimate or that it has not been investigated at this point.

Finally, the matrix $\boldsymbol{S}$ is constructed. In this example, the weight matrix is just the adjacency matrix and equal to

$$W = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

Next to that, we need the two diagonal matrices D_c and D_p containing as their diagonal elements the degrees of the different claim and counterparty nodes, respectively. This

then becomes

$$D_c = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

$$D_p = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

With all the different parts into place, we do the iterations of the algorithm. The resulting fraud scores for the different claims are 0.45, 0.19, 0.26 and 0.23. Hence, as expected, claim C_2 has the lowest fraud score. Contrary to our intuition, however, the score for claim C_3 is a bit larger than that of C_4.

3.2 Our Problem

The network we introduced in Chapter 2 is a 4-partite network, while the BiRank algorithm is specifically defined for a biparite one. Hence, we first need to apply the projections as defined in Section 2.2 in order to obtain the bipartite network representing the claims and parties.

Next, we also need to be able to test the performance of this procedure. If we would just take the network as is and set the entries of the query vector $\boldsymbol{c}^0$ to 1 for all fraudulent claims, we cannot say anything about the predictive power of the BiRank algorithm in the future. All fraud data is already included and we almost guarantee a high score of all the fraudulent claims in the network.

It is, hence, necessary to split our data into a training set, where we keep the fraud information as before, and a test set, for which we set all a priori information to 0. There are two ways of doing this split. We will first explain the two and then give a small illustration in Example 3.2.

The first way, which is called the *transductive* setting (see [1]), keeps the full structure of the dataset. No nodes or edges are removed in the training or validation steps. The only thing that is *removed* are the labels. We do as if we have access to the full network, but only know a few labels. In a fraud detection setting with BiRank scores, one can see it as follows. Suppose we have two sets of data. The first part contains the claims and parties that are part of our historical dataset which have already some fraud labels assigned to it. This will be called our training set. Then some new claims come in of which we only know their connections and place in the network, but not if they correspond to legitimate claims or not. This is our test set. What we do is initiate the BiRank algorithm with the $\boldsymbol{c}^0$ having 1 for the known fraud cases. Then, we run it once for the full network and try to determine which new claims have a high probability of being fraud.

The second way is called the *inductive* setting. Here, we break the network. The idea of this setting is to remove the nodes in the test set from the network. After training the model on the smaller graph, the test nodes are added, information is extracted and predictions are made. In the fraud setting, this can be seen as having the historical data, and already running the BiRank algorithm on that. Afterwards, we add the new claims, and use for $\boldsymbol{c}^0$ the fraud scores that were calculated earlier. We do not start with 1 as the fraud scores (of the fraudulent claims).

Afterwards, we can use these fraud scores coming out of the algorithm for the test set to classify the test nodes as fraudulent or legitimate. These are compared to the labels that were assigned in the dataset to see which were correctly picked up by the algorithm.

Example 3.2. Before moving on, we give a small illustration of the effect of the inductive and transductive way of working. We stay with the network from Example 3.1, given in Figure 3.1. We do as if claim C_2 is in our test set, and all others are in the training set.

Starting with the transductive setting, we do not need to do much. Here, we apply the full algorithm once on the full network. The results are hence exactly the same as in Example 3.1.

The inductive train-test split is a bit more involved. We remove claim C_2 from the network and let the algorithm run a first time. This gives only fraud scores for the remaining claims, as illustrated in Table 3.1 These new scores are then used a second time by the algorithm for the full network, i.e., including C_2. The final scores are also given in Table 3.1.

Claim	Transductive	Inductive Step 1	Inductive Step 2
C_1	0.445444	0.500356	0.340438
C_2	0.190087	-	0.145278
C_3	0.259112	0.291054	0.316148
C_4	0.227431	0.255468	0.263022

Table 3.1: Results from the BiRank Algorithm.

From the table, we can observe that the scores and order of the claims are very similar for both methods. The overall order of the four nodes has stayed the same. However, for the inductive step, the scores of the training set have moved closer to each other, while the score for the test set, i.e., node C_2, is smaller.

How this affects the performance of the model will be further investigated in the coming sections of this chapter.

For evaluating the BiRank in this thesis, we apply both ways of doing the train-test split. In a first try, we set the alpha equal to 0.85 and take the training set to be equal to 60% of our data (=240 000 claim nodes). We just take the first 60% since we are informed by the industry expert supplying the data that there is a time element present

in the order of the nodes. Since in reality, we use older data to find patterns and classify new claims, this way of working represents the real-life use of the algorithm.

3.3 The Results

3.3.1 A First Look

Before jumping into the algorithm with the train-test split, we apply it on the full network (without the split) to see if we can already extract some meaningful insights about how it works. This may give us some additional ideas of what to do to improve its performance.

The results of the BiRank algorithm are represented below in Table 3.2 for some claims. We see that different numbers are returned. We have the ID, which represents the place of the different nodes in our dataset. This is the row-index of the node in the adjacency matrix. The second column contains the score calculated by the procedure. This score is rescaled to the StdScore, where we subtract the mean from the score and divide it by the standard deviation. This way, the mean of the StdScore is zero and the standard deviation one. The final column is also a rescaling of the original score, in order to have it between 0 and 1. This is done as follows for node i:

$$c_{\text{scaled},i} = \frac{c_i - \min(\boldsymbol{c})}{\max(\boldsymbol{c}) - \min(\boldsymbol{c})},$$

with $\boldsymbol{c}$ the score vector coming out of the algorithm.

	ID	Score	StdScore	ScaledScore
0	0	5.670397e-07	-0.083258	0.000479
1	1	4.776474e-07	-0.088344	0.000403
2	2	5.170834e-07	-0.086100	0.000436
3	3	8.184590e-07	-0.068955	0.000691
4	4	4.646187e-07	-0.089085	0.000392

Table 3.2: Results from the BiRank Algorithm.

We will first look at the neighbourhood of the (fraudulent) claims that have the highest fraud score. These are known fraud cases whose connected components only contain the claim and car. This means that in case we do a train-test split in a future step, and one of these claims is in the test set (score set to 0), it will be impossible to detect it as a fraud, just by using the BiRank scores.

It is not surprising that the fraudulent claims get the highest fraud score, since we gave this information to our algorithm (remember that at this stage, we did not use the train-test split). What is surprising though, is that we have some claims in the network that received a high fraud score, but are not labelled as such. We represent these in Table 3.3. We see that the first two are on place 383 and 384 (place 1 is ID 0) which

means that they have a higher fraud score than 55% of the confirmed fraud cases. These two claims are plotted in Figure 3.2 together with their neighbourhood. We see that they are closely linked with a fraudulent claim, since they share the same involved car, policy and broker. Hence, we can conclude that it was the same person who filed the three claims, of which one was investigated and deemed fraudulent. We notice that this is also the entire connected component of these nodes. Therefore, the person is isolated from the rest of the network and the fraud claim has a high influence on the non-fraudulent claims.

It is therefore not unthinkable that these two other claims were fraudulent as well. Upon closer inspection, we see that the fraudulent claim was filed last, so this person first filed two claims that were not investigated. Later, he/she filed a third claim that was investigated and deemed fraudulent. This highlights an important problem that may arise when one is not careful, namely time-leakage. When taking a test set, one needs to make sure that only data available at the moment the filed claim came in is used for the predictions. Here, for example, we have that the scores of the green nodes in Figure 3.2 are calculated using the fraud score of a claim that was filed later. This time-leakage can have unwanted consequences and can skew the conclusion on how well an algorithm would work in practice.

Next, one also needs to be careful since the disconnectedness and lack of other fraudulent nodes in the neighbourhood means that we will not be able to pick it up using the BiRank algorithm as represented here, when it is part of our test set.

	ID	Score	StdScore	ScaledScore	Fraud	Labelled
382	116408	0.000336	18.982870	0.283333	0	0.0
383	122134	0.000336	18.982870	0.283333	0	0.0
566	38447	0.000287	16.224239	0.242408	0	0.0
567	319763	0.000287	16.224239	0.242408	0	0.0
591	384618	0.000277	15.667893	0.234154	0	0.0

Table 3.3: The unlabelled claims with the highest fraud score.

For the claim with the third highest fraud score amongst the non-labelled nodes, we also plotted the neighbourhood in Figure 3.3. We see that it is connected to fraudulent claims via its broker. Since it is only via the broker, it is less likely that fraud was committed by filing this claim. This shows us that brokers with few connections, may serve as hubs via which the fraud information flows throughout the neighbourhood. It is less imminent when we have brokers with a very high degree, since the BiRank procedure reduces the importance of information flowing through them, as was explained using Formula (3.1).

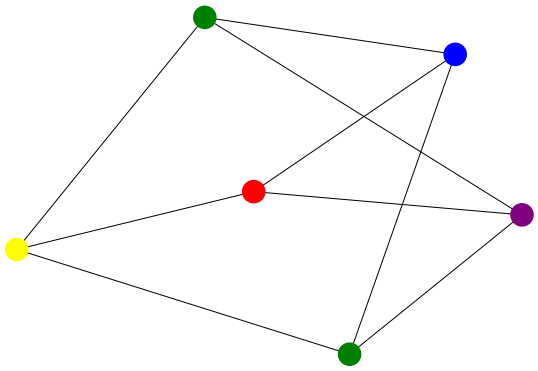


Figure 3.2: The two non-labelled claims (in green) with the highest fraud score, together with the fraudulent claim (in red). The node in purple represents the car, the blue node is the broker and the yellow node is the policy.

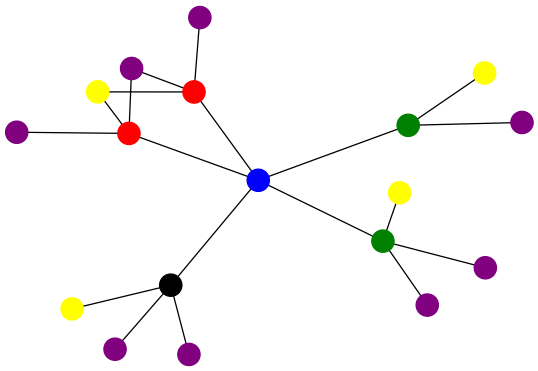


Figure 3.3: The non-labelled node under consideration (black) with a high fraud score, the non-labelled claims (in green) in its neighbourhood, together with the fraudulent claims (red). The nodes in purple represent the car, the blue node is the broker and the yellow nodes are the policies.

3.3.2 The Performance

Now that we have additional insights in how the procedure works, we introduce the train-test split as described above, in order to have a first meaningful conclusion on its performance. As an illustration, we show the scores of the 5 claims with the highest fraud score in Table 3.4. The table represented here only shows the results from the transductive method.

As mentioned above, we get multiple values from the procedure. From these, we need to choose one that we are going to use for the classification. In this case, the scaled score is of most interest. Since it is between 0 and 1, we can consider it as some sort of probability of fraud. If we introduce a cut-off value, we can say that any node with a higher scaled score is classified as fraud, and each node with a lower score as non-fraud.

	ID	Score	StdScore	ScaledScore
703	703	0.001553	77.103377	1.000000
45509	45509	0.001553	77.103377	1.000000
44427	44427	0.001553	77.103361	1.000000
209355	209355	0.001019	50.560786	0.656201
213329	213329	0.001019	50.560786	0.656201

Table 3.4: The five highest scores with the transductive train-test split.

The performance of the BiRank algorithm on our network is measured using the area under the ROC curve (AUC). For this, we need two values related to the classification, namely the true positive rate and the false positive rate. The true positive rate is defined as follows:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Here, the TP represents the number of claims classified as fraud that are actual fraud, called the true positive. The false negative elements, FN, are those fraudulent claims that were not detected by the classification algorithm. Hence, the TPR is the ratio of all fraud claims that are uncovered by our model.

In addition, we need the false positive rate, which is equal to:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}.$$

In the same spirit of the true negative rate, the false positive rate expresses the percentage of non-fraudulent claims that were mistakenly seen as unjustly reported.

We plot the false positive rate on the x-axis and the true positive on the y-axis. For each cut-off value, we get a specific FPR and TPR value pair. These are plotted for all values on the graph, to obtain what is called the ROC curve. For the lowest cut-off, i.e. 0, we obtain the point with coordinates (1,1). All claims are classified as fraud (positive) and none as negative. Therefore, none of the maleficent claims are falsely classified as fraud, so the number of false negatives is 0:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{TP} + 0} = 1.$$

With the same reasoning, the true negative rate will be 0, since none are classified as negative. Therefore, the false positive rate is also 1.

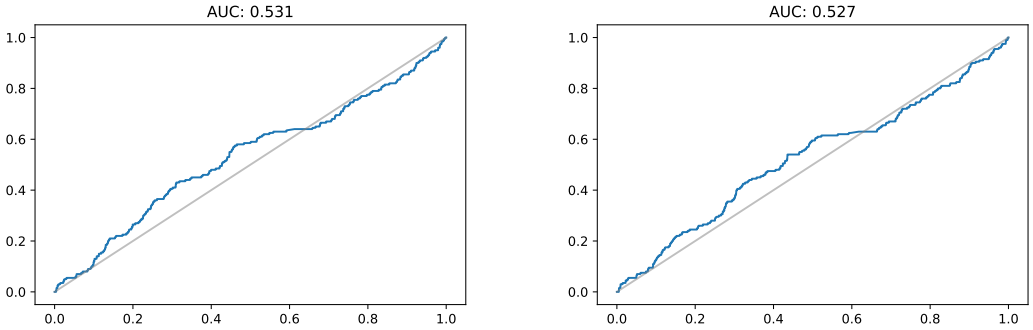
Similarly, if we take a cut-off of 1, where all claims are classified as legitimate, the TPR and FPR are both zero. So, for the highest cut-off, we end up in the bottom left corner of the plot.

When the ROC curve is close to the diagonal line between the origin and (1,1), our model is not that different from just randomly guessing the label of the nodes. This closeness is measured by the AUC, i.e., the area under this ROC curve. Note that the

diagonal line representing random guessing defines an area of 0.5. If the AUC of our model is well above this 0.5, we have a useable model. If, however, we would obtain an area smaller than 0.5, our model performs even worse than random guessing. In that case, it would obviously be adviced to just discard that model.

The AUC for our model is given in Figure 3.4 for both train-test splits. We see that we have a marginal improvement over random guessing. Nonetheless, we do not have a model that gives enough additional information to be able to implement it in practice to hunt for fraudulent claims. Especially in the upper-right part of the curve, we seem to have an underperforming model. Going back to what we have observed in Section 2.3 about the homophily in our network, we have noticed that the difference in the ratio of fraudulent and non-fraudulent claims in the neighbourhood was most pronounced for the non-fraudulent claims. This has an effect on the BiRank scores. For non-fraud cases, i.e. those with label 0, most of the neighbours have label 0. Hence, when we have such a node in our test set, the fraud scores in the neighbourhood will be low, and the BiRank algorithm will assign to this node a low score as well. On the other hand, for fraudulent claims, i.e. having label 1, there was less of a difference between the ratios of fraudulent and non-fraudulent neighbours. This implies that the fraud score calculated by BiRank based on its neighbourhood will be diluted by non-fraudulent (and not labelled) neighbours. As an effect, this fraud score will be less pronounced and these claims become harder to spot.

In the next section and chapters, we proceed with our analysis and adapt the above procedure to try and improve our results. Chapter 4 contains a brief discussion on the importance of this upper-right tail of the ROC curve.



(a) Inductive split.

(b) Transductive split.

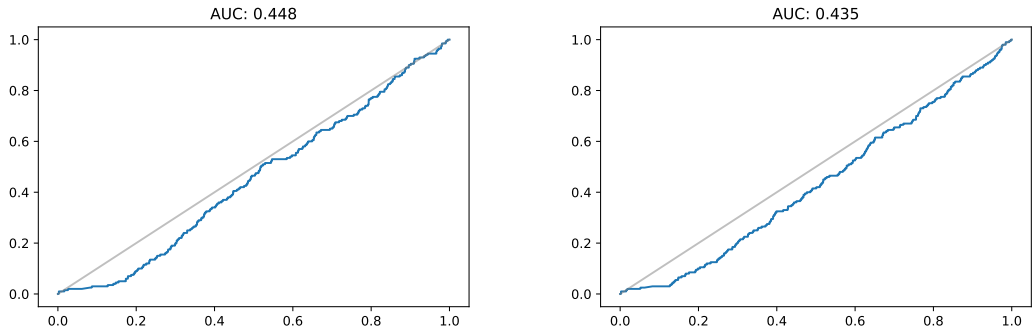
Figure 3.4: The ROC curves with the AUC of the BiRank algorithm for the two train-test splits.

3.4 Further Extensions of the Procedure

In this section, we will investigate five adaptations of the previous analysis to form a conclusion for the BiRank algorithm. Firstly, we do the same as before, but without the broker nodes in the network. We already saw above that brokers may artificially increase the fraud score of some nodes, if there is a maleficent case in its neighbourhood. Another extension will be to tune the hyper-parameter alpha. The third procedure will be to use a different way of doing the train-test split with leave-one-out cross validation. Furthermore, we are going to investigate how much the connectedness plays a role by only considering the largest connected component in our network. Finally, the dataset is enriched with some additional, but basic, network features.

3.4.1 Excluding the Brokers

As mentioned multiple times already, brokers influence the fraud score of some nodes, which in turn alters the results of the fraud detection algorithm. Hence to see how much they can skew our results, we do the exact same procedure as before, with the same train-test split, but now on a smaller and more disconnected network where we exclude the broker nodes. As can be seen in Figure 3.5, the BiRank gives an AUC below 0.5. This means that we are better of just guessing the label of the claim than relying on the algorithm.



(a) Inductive split.

(b) Transductive split.

Figure 3.5: The ROC curves with the AUC of the BiRank algorithm for the two train-test splits. Here, the broker nodes were excluded from the network.

One explanation could be that the network becomes too disconnected when not including the brokers. The connected components are then almost always very small islands with only a couple of fraud labels. If we then remove their lables for the train-test split, there is no way for fraud data to flow inside those components, and the scores stays (near) zero. Another explanation could be that having the same broker can carry some additional information. Going back to the example in Figure 3.3, we saw that brokers with few connections spread the fraud data strongly. It is possible that

a fraudster tries to evade suspicion by underwriting a new policy for another car after being caught once. Like this, it is possible that only the broker is the common factor for these claims. Brokers also carry some geographical data. If you have a broker with only a few claims reported, but where a couple of them are fraudulent, it may indicate that the broker operates in a region with higher risk.

This point also highlights an important issue that needs to be considered when using network information. It could be that there is a bias towards certain groups of people inherent in the network information coming from a bias of the fraud investigators, be it conscious or unconscious bias. When you have persons in a community who are more often investigated for fraud when they file a claim, more illicit claims will be picked up. This also means that the sub-part in the network representing this community will have a higher concentration of fraud labels, which will propagate more easily to the other nodes by, e.g., the broker that operates in that community. Hence, these people will have higher fraud scores, which results in a further build-up of bias. However, since the performance drops too much, we need to keep the brokers into the network. It is up to the modeller in subsequent step to quantify the bias inherent in the procedure before making final decisions based on its output.

We can, hence, conclude that it is beneficial to our algorithm to keep the connectivity of our network by including the brokers that sold the policies. Only when we would move to implementing it in real-life, one must consider the unwanted bias that may be part of the predictions.

3.4.2 Tune the Alpha Hyper-Parameter

Before, we have fixed the value of α in Equation (3.2). Since this is purely based on what we have seen in the literature, we are not sure that $\alpha = 0.85$ is the most optimal value for our problem. That is why we will tune this hyper-parameter to hopefully increase the performance of our algorithm.

For this, we need to make a couple of choices. The first is the values of α we want to test. We use values from 0.05 to 0.95 in steps of 0.05. Next, we choose the splits we want to make in our network. We opt to only use the inductive split for this procedure. The test set consists again of the 40% most recently filed claims. The other part is used to define the train and validate set. For the training set, we use 60% of the remaining network. This is taken as is, without modifying the labels. The validation set, which is the last bit of data we did not handle yet, takes on the role of the *test set* when tuning the parameter.

We fix a value of α in our selection, and use the train-validate set to calculate the AUC just in the same way as with our original train-test split. Using these different AUC's, we can select the most optimal value of α. Finally, the training and validation sets are combined into a larger training set, and we calculate a final AUC value using the test set where the α is fixed to the one we have chosen. The important thing here is that we did not use the test set during the tuning of our hyper-parameter.

Doing this, we obtain an optimal value of 0.4. The final result of our classification is given in Figure 3.6. We have an AUC of 0.518 which is suboptimal to the one we obtain

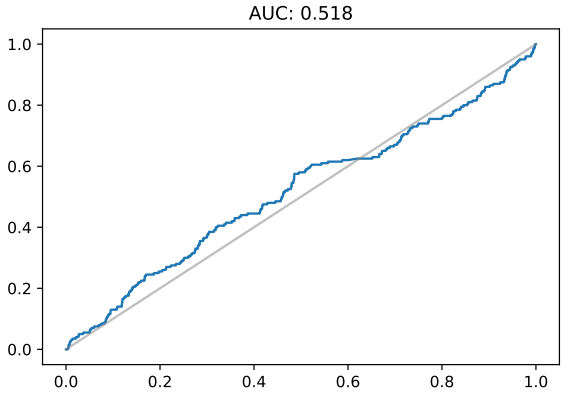


Figure 3.6: The resulting ROC curve and AUC for the tuned $\alpha = 0.4$.

using $\alpha = 0.85$. Therefore, we will keep using 0.85 as taken from the literature.

3.4.3 Leave-one-out cross validation

The main problem with the simple train-test split, is that it is a very invasive procedure. By removing 40% of the network in the inductive setting, its topology changes drastically. In the transductive setting, we do not change the topology, but it could be that the scores become too diluted by introducing more 0's in the labels, i.e. removing fraud labels.

In order to deal with these "shortcomings", we extend the splits by using leave-one-out cross validation. We randomly select 200 nodes[1], but with the constraint that we want 100 unlabelled and 100 fraudulent claims. One by one, we do the same procedure as before, but we only change the label of the node in question to 0 (if necessary). This gives 200 test predictions, without changing the topology too much. In this way, we can see whether the network is able to reproduce the fraud labels accurately, when we introduce only 1 claim to the network. This is also something that would be interesting in practice, since one usually receives the claims one by one.

With these results, we can calculate the AUC. The result is given in Figure 3.7. The AUC is a bit better than before, with a value of 0.582. But this is still not as high as one might have hoped.

3.4.4 Only the Largest Connected Component

Until now, we have worked with the complete network. As was pointed out in Section 2.1, our network consists of multiple different connected components. Section 3.3.1 hinted already at one of the problems. When working with smaller components, it could be that the train-test split removes the only fraud data available in that sub-graph. This

[1]We limit ourselves to 200 nodes out of time constraints. This took 2 hours and 49 minutes, and we believe that we already have an accurate enough picture of the performance of our model.

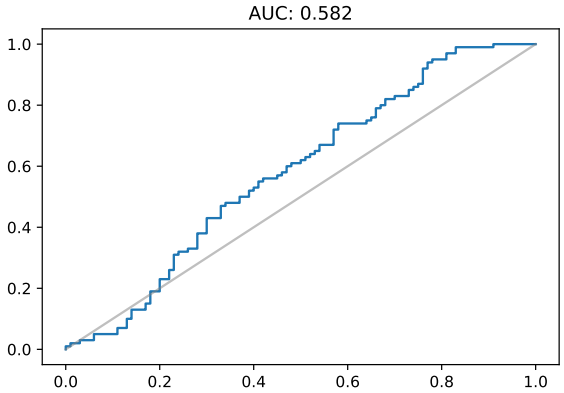
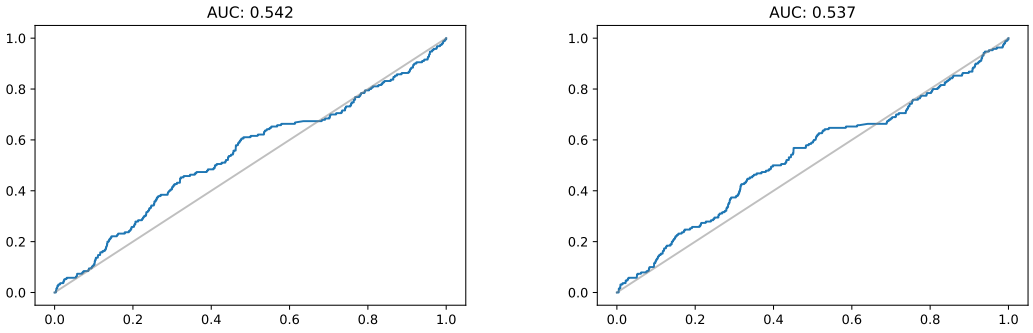


Figure 3.7: The AUC for the inductive method of the leave-one-out cross validation based on 200 nodes.

leads to an inherent inability to ever uncovering the fraudulent claim again. We should note here that we are again prone to the effects of time leakage.

In order to avoid such situations, we are only going to work with the largest connected component in this section. This enables us to make sure that there is always some possibility of fraud information reaching any node in the network. We will take the same plan of attack we used before as to preserve comparability amongst the results.

We plot the AUC of both the inductive and transductive train-test split in Figure 3.8. We see marginal increases for both splits with around 0.01. This shows that the BiRank algorithm indeed benefits ever so slightly from the network being fully connected.



(a) Inductive split.

(b) Transductive split.

Figure 3.8: The ROC curves with the AUC of the BiRank algorithm for the two train-test splits. Here, only the largest connected component is taken under consideration.

One explanation that there are no massive improvements can be due to the fact that we had most of our fraudulent data still present. We went from around 840 frauds to

around 800. This means that the special cases indicated above may be less prevalent than initially thought, i.e., only a small portion of known fraudsters are fully disconnected from the other claim nodes. Hence, they do not impact the performance much.

In what comes next, we will again work with the full network. There are two main reasons for this. The first is that we only see small improvements, which does not justify throwing away a large portion of our data. The second is that there may be valuable information in the fact that a node is not part of a large connected component. If you are undertaking in maleficent business practices, it could be that you try to hide it from other people. Then it is less likely that you will be connected to a larger connected component. You will most likely only be part of your own small one. This and extra network features will be the topic of discussion for the next section.

3.4.5 Enriching the BiRank data

As we could see in the previous section, the BiRank algorithm on its own did not give a convincing fraud detection model. In this section, we go further and extract more network information. This is combined with the fraud scores in order to build a larger fraud detection algorithm. As we obtain a larger set of features, it becomes necessary to look at other possible modelling techniques to obtain the classification. Hence, after discussing the additional data that we are going to extract, we make a little detour to discuss the model that we are going to consider in this section, namely gradient boosting machines.

In order to extract more features from the network other than the information flow with the fraud labels, we enrich the dataset with additional information. This way of working will come back when constructing the full model in Chapter 6. The three features that we will be adding for now are the geodesic distance, the number of cycles the claim is a part of, and the degree centrality. For the number of cycles and the geodesic distance, we will remove the brokers of the network. This way, these two features can be used to uncover fraud rings in our network.

We already mentioned some centrality measures in Section 1.3.1, but the degree centrality used in this section is not defined by taking paths in the network. It expresses the degree of the node, i.e., the number of connections to other nodes, divided by the total number of possible connections. We use a small simplification, and take the denominator as the number of all other nodes in the network. There is a bipartite equivalence of this, which takes the degree of the claim nodes and divides this by the number of all party nodes. This small layer of complexity does not add any additional information, so we just use the most simple one.

The geodesic distance on the other hand, is the shortest path from the node back to itself. Here, we will say that it is not allowed to use the same edge twice. Hence, a node can only have a geodesic distance (other than infinity) if it is part of a cycle. Due to the nature of our bipartite graph, the shortest distance any claim can have to itself is 4, and the other possible distances can only be 4 plus a multiple of 2. If the geodesic distance is small, this means that this claim is closely related to other claims, via multiple parties. This could point to a fraud ring in the network. This interest in fraud rings is the reason

why we do not include brokers in the network. If you need to have a broker node in order to form a cycle, it would dilute the fraud ring information that could be present. We generally do not assume that the broker is involved in covering up the fraud.

Including this information with the BiRank scores already gives us a couple of features that we can use to try to solve our classification problem.

The model we consider is a gradient boosting machine. For a discussion on tree based classifiers, one can consult [21]. For now, we take a gradient boosting classifier as implemented in sklearn [31], and set the number of trees equal to 50. We choose to not take this number too large, since this could lead to overfitting. For simplicity, we use a transductive train-test split. When looking at Figure 3.9, we have a slight improvement in AUC for our model. Next to the larger AUC, we note that the ROC curve does not go under the bisector anymore, which is another desirable trait.

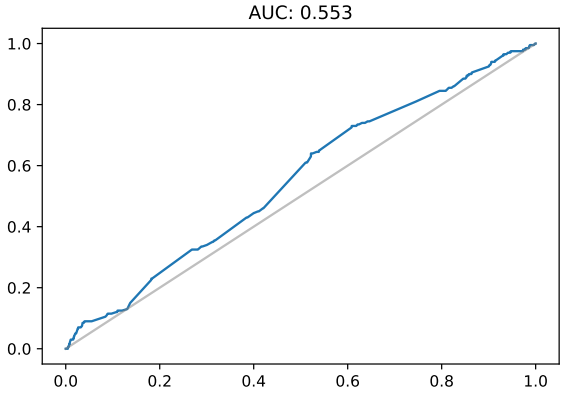


Figure 3.9: The AUC for the gradient boosting machine with the five network features.

The BiRank algorithm of this chapter has already demonstrated that some insights might be gained when using the network features. In the next chapter, we will introduce the metapath2vec algorithm. This will take a whole other approach at finding patterns and illicit claims in the network.

Chapter 4

The Metapath2vec Algorithm

In this chapter, we introduce the first real embedding algorithm of this thesis, namely metapath2vec [14]. This method generates random paths through the network, and afterwards utilises the Word2vec algorithm [26] from natural language processing to make a small dimensional projection of the full network. This results in new features that are leveraged by our gradient boosting classifier to build a fraud detection model.

The structure of this chapter is as follows. Section 4.1 describes the metapath2vec method in more detail. In Section 4.2.1, we translate the set-up of the algorithm to our network. Finally, a first result for our network is given in Section 4.2.2. These outcomes are then compared with the BiRank algorithm from Chapter 3.

4.1 The Algorithm

The full method is given in Algorithm 2. In what follows, the different steps will be handled in more details. Section 4.1 defines the metapaths and how these will be sampled. Next, the processing of the metapaths is handled in Section 4.1.2.

4.1.1 Sampling the Metapaths

The metapath2vec method was first introduced in [14]. This algorithm falls into the category of Section 1.3.5, where a random walk generator is combined with NLP to obtain an embedding of the network. Contrary to the methods introduced prior to [14], metapath2vec is developed for usage on heterogeneous network, where the others are meant to be used on homogeneous ones.

Despite what the name may suggest, metapath2vec is not a generalisation of node2vec. As will be discussed later, the next node in the path is chosen uniformly from all possible candidates. This means that it is more similar to a heterogeneous generalisation of DeepWalk.

The necessity to introduce additional embedding algorithms specifically for heterogeneous graphs comes from the following observation. Using the notation as in Definition 1.5, we have that the full network is mapped onto a representation $\boldsymbol{X} \in \mathbb{R}^{|V| \times d}$,

Algorithm 2 Metapath2vec as given in [14]

Input: Network $G(V, E)$, scheme for the metapaths $\mathcal{P}$, length of walks l, number of walks per node n, context size c, embedding dimension d
Output: The embedding $\boldsymbol{X} \in \mathbb{R}^{|V| \times d}$.

```
Initialise X
for i from 1 to w do
    for v ∈ V do
        MP[1] = v
        for j from 1 to l − 1 do
            draw u according to Equation (4.1)
            MP[i + 1] = u
        end for
        X = SkipGram(X, k, MP)
    end for
end for
return: X
```

where $d << |V|$. For a heterogeneous network, this means that all the node representations are mapped into the same latent space, even though the nodes are of different types. It is, hence, necessary to introduce new heterogeneous techniques. This makes sure that adequate care is taken to incorporate the heterogeneity of the nodes.

In order to incorporate the heterogeneity, the normal random walks are substituted by meta-path-based random walks. These meta-paths are represented using a meta-path scheme which we denote by $\mathcal{P}$. Using the notation from [14], we represent this scheme as

$$\mathcal{P} : V_1 \xrightarrow{R_1} V_2 \xrightarrow{R_2} \dots V_t \xrightarrow{R_t} V_{t+1} \dots \xrightarrow{R_{l-1}} V_l.$$

When constructing a walk through the network, the set of possible next steps is fully determined by the previous ones, according to the metapaths defined. One node is picked uniformly from the possible ones, i.e. for the network $G(V, E)$ we have that [14]

$$\mathbb{P}\left(v^{i+1} \mid v_t^i, \mathcal{P}\right) = \begin{cases} \frac{1}{|N_{t+1}(v_t^i)|} & (v_t^i, v^i) \in E, v^{i+1} \in V_{t+1} \\ 0 & (v_t^i, v^i) \in E, v^{i+1} \notin V_{t+1} \\ 0 & (v_t^i, v^i) \notin E \end{cases}; \tag{4.1}$$

where $N_{t+1}(v_t^i)$ represents the neighbours of node v_t^i of type V_{t+1}.

The following example shows the importance of a well thought out definition for the metapaths.

Example 4.1. For this example, we go back to the set-up of Example 2.1. We reshow that graph in Figure 4.1. Note that we assume in this example that we have the same node types, but that our network is much larger with multiple nodes from each type.

Here, we have three types of nodes, namely claims (C), policies (P) and brokers (B), and there are two particular meta-paths that are of interest. The "CPC" meta-path

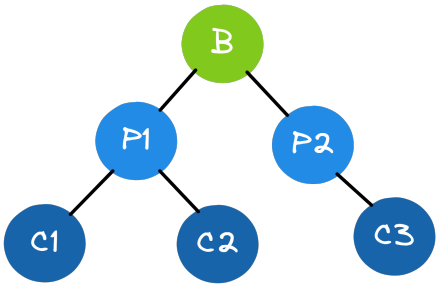


Figure 4.1: A network with one broker who sold two policies for which three claims were filed in total.

represents claims filed under the same policy, and "CPBPC" represents filed claims for which the policies were purchased from the same broker.

The algorithm samples uniform random walks using the pre-defined meta-paths over our network. Just like for the other embeddings using (biased) random walks, the skip-gram algorithm is used to construct the embeddings from the random walks. In the next section, we apply metapath2vec on our network.

In [14], it was mentioned that not using these meta-paths (i.e. just applying a random walk) would result in walks that are biased to nodes that have a high number of paths going through them or to nodes belonging to a group to which most paths lead. In our case, this would result in paths that favour broker nodes, see also [37]. Therefore, the definition of the meta-paths, like "CPBPC", is important to draw the random walker away from the broker, and towards the claim nodes, which we are more interested in.

This is the main reason why we will not make a projection to the bipartite network, as we did for the BiRank algorithm in Chapter 3. Using the bipartite equivalent could result in paths of the form "CBCBCBC", hence falsely representing that claims that are (only) connected via a string of brokers are closely related. This could result in a lot of unwanted noise on the embeddings, resulting in sub-optimal results.

4.1.2 Applying Natural Language Processing

The sampled metapaths can be seen as *sentences* where each node is another *word* in our vocabulary. The idea is that the skip-gram, which is used for natural language processing, then takes these sentences and makes representations of the words, i.e. the nodes of our network. These representations can express the similarity between the words. The algorithm produces a multi-dimensional embedding. Staying in the context of real languages, we have that multiple degrees of similarity are possible [25]. In the paper, the authors mention two intuitive degrees, namely syntactic and semantic similarities. Syntactic similarities express the fact that words can have multiple endings, but still have similar meaning, e.g. fraudster and fraudulent. The semantic similarities arise by the context of the different words around them. The vector representations of different words can then be combined using simple algebraic operations to obtain other

representations. As an example, one can have the following in ideal circumstances:

$$\text{vector}(\text{“Car”}) - \text{vector}(\text{“Engine”}) + \text{vector}(\text{“Pedals”}) = \text{vector}(\text{“Bike”})$$

The skip-gram algorithm takes a sentence, and for each word, it tries to predict the words that come before and after that word within a certain range, which is called the context. The representations of the words and the prediction of the other words in the context of a given word are calculated using a feed-forward neural network.

The quality of these predictions are measured using the log-probability, which the algorithm tries to maximise. This is expressed mathematically as follows. Suppose that we have a sentence of words, i.e. nodes from the network, $v_1, v_2, \ldots, v_N$, then the function that needs to be maximised, is

$$\frac{1}{N}\sum_{n=1}^{N}\left(\sum_{-c\leq m\leq c|m\neq 0}\log p(v_{n+m} \mid v_n)\right), \tag{4.2}$$

where c expresses the context size. In the case of the skip-gram algorithm, the softmax function is used for the probability function $p(v_{n+m} \mid v_n)$. For more details on this algorithm, we refer the interested reader to [26] and [25].

The final embedding representation of the sampled metapaths, after maximising the log-probabilities, gives us the output of the metapath2vec algorithm. This can then be used by any traditional machine learning algorithm to do the fraud classification.

4.2 Embedding our Network

4.2.1 Defining the embedding

The network that we are currently using, has four different node types, namely claims, cars, policies and brokers. This gives us a multitude of possibilities to define the meta-paths. As we are mainly interested in the claims, we want that the random walker starts and ends in a claim. Next, we keep the meta-paths symmetrical, like in the examples we gave above.

We decide to use the following three meta-paths:

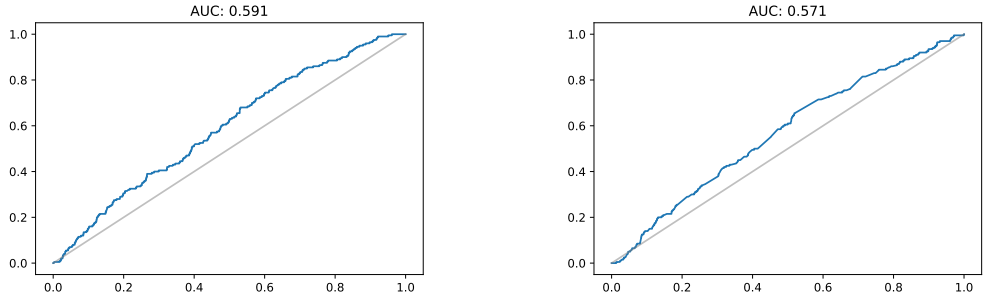
$$\begin{aligned}
\mathcal{P}_1 &: \text{Claim} \to \text{Car} \to \text{Claim}\\
\mathcal{P}_2 &: \text{Claim} \to \text{Car} \to \text{Policy} \to \text{Car} \to \text{Claim}\\
\mathcal{P}_3 &: \text{Claim} \to \text{Car} \to \text{Policy} \to \text{Broker} \to \text{Policy} \to \text{Car} \to \text{Claim}
\end{aligned}$$

In the same spirit as Section 3.4.1, we will compare the results using all previously defined metapaths with those obtained when we exclude $\mathcal{P}_3$, i.e., when we do not allow the paths to pass through a broker.

For this, we embed the network into a low-dimensional space of 20 dimensions, where the walk length is set to 13, which is the number of steps needed to go from one claim to another via a broker twice. Each claim is also taken twice as starting point.

4.2.2 The Results

The embedding is fed into a gradient boosting classifier built up of 100 estimators, using a 60-40 train-test split. The resulting AUC scores for both models, with and without brokers, is given in Figure 4.2.



(a) Using meta-paths where going through brokers is allowed.

(b) Using meta-paths where going through brokers is not allowed.

Figure 4.2: The ROC curve with the AUC of metapath2vec.

We see again that the model is a bit better than random. This shows that there is some structural information in the network that can point to fraudulent claims. What is important is that we see that, in terms of the AUC, we have similar but better results than for the BiRank algorithm. This is surprising, since BiRank incorporates the fraud data, while metapath2vec only relies on the network structure itself. The trade-off for this *better* prediction power is that the metapath2vec method takes much longer to execute.

As an additional comparison between the BiRank algorithm and metapath2vec, we overlay both ROC curves in Figure 4.3. We see that the curve from metapath2vec dominates the one from BiRank most of the time. Especially the dominance in the upper-right corner is of most interest. When implementing a fraud detection system in reality, there are additional costs related to misclassification. The cost of flagging a legitimate claim as being fraud is much lower than not detecting a fraudulent claim. Therefore, the true positive rate staying higher on the right hand side of the plot indicates that the metapath2vec embedding is preferable to the BiRank fraud score.

One explanation comes from the fact that we have very few labelled data. When doing the train-test split, we have fewer fraud data still. This can lead to a dilution of the data in parts of the network that is too severe, when applying the BiRank algorithm. As a consequence, we have multiple cases that remain undetected due to the fact that there are no or very few sources of fraud data left in their neighbourhoods after the split. The metapath2vec method does not suffer from this, since the embedding is intrinsic to the network, and the fraud data only comes into play when training the gradient boosting classifier, like in any other normal fraud detection model. Adding or removing some fraud labels does not affect the embedding itself, while it does have a significant

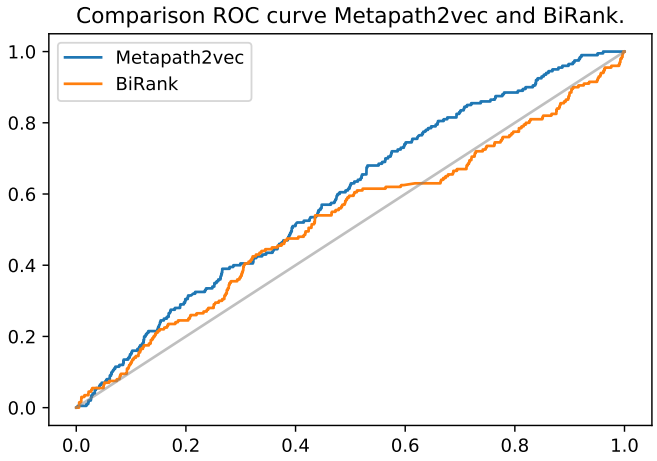


Figure 4.3: The ROC curves of the BiRank and metapath2vec method.

impact on the final result of the BiRank algorithm.

Next to the higher AUC value, we note in Figure 4.2b that the drop in AUC when not including the brokers is much less severe for metapath2vec than for BiRank. Metapath2vec stays slightly better than random, while BiRank without brokers did very poorly, with AUC scores around 0.44. This again has to do with the fact that the local network context for claims remains largely the same when the brokers are not considered for metapath2vec. A metapath containing a broker is also less likely. On the other hand, excluding the brokers can cut off the fraud information flow for a large portion of the network when applying the BiRank algorithm.

There are two drawbacks to using the metapath2vec embedding instead of the BiRank method. The first one is the time it takes to obtain results. In a first step, we need to do a lot of random walks in the network, which increases linearly with the number of nodes. Thereafter, these paths are embedded using the word2vec algorithm. Finally, a model is trained using this embedding to give a classification to the nodes. If we increase the embedding dimension, the training of our classifier will also take longer. When using BiRank as a stand-alone method, this procedure goes much faster, because all updates of the fraud scores happen simultaneously using matrix multiplication. As already noted before, these scores can, after rescaling, directly be seen as classification probabilities.

A second drawback is that there is some randomness involved. Each time the random walks are done, they are different than before. This difference alters the solution of the embedding slightly, which in turn has an effect on the predictive power of the classifier. We try to show and quantify this effect as follows. The algorithm is repeated 15 times, and each time, we calculate the AUC of the model as before. The resulting histogram of the 15 AUC's is plotted in Figure 4.4. We see that there is some variability in the resulting AUC. In our example, we have values between 0.5345 and 0.5978. Additionally, the average AUC is equal to 0.5685, with a standard deviation of 0.0199. This means

that, although there is some randomness involved, the results are more or less stable and we can extract meaningful insights from the AUC values.

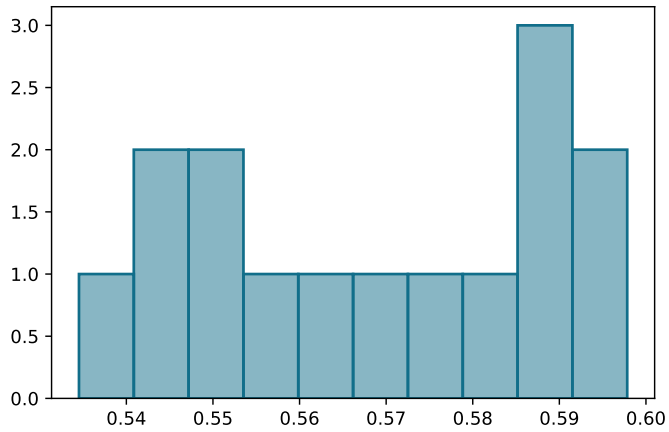


Figure 4.4: The distribution of the AUC of 15 metapath2vec iterations.

As can be seen in this chapter and in Chapter 3, the gain in predictive power over a random model, expressed using the AUC, is limited when only focussing on the social network. Hereafter, we will introduce a second dataset containing some intrinsic features. Network features will be added sequentially to these intrinsic features. We start with simple ones, coming directly from the dataset, but these will gradually become more complex with the introduction of a.o. the BiRank and metapath2vec features.

In addition, we are going to introduce a second metric to evaluate our model, namely the precision-recall curve. As with the ROC curve, our main interest will be the area under this curve. This is called the average precision. Why we do this, is explained in the next chapter.

Chapter 5

An Extension to the Dataset and Additional Metrics

In this chapter, we extend the dataset from Chapter 2 in Section 5.1 in order to extract meaningful insights. The starting point of this dataset is the same as the one introduced in Chapter 2, but it introduces the intrinsic, i.e. claim-specific data, to the network. It is also represented slightly differently, since it already comes in the form of a bipartite network.

In addition, we define two other concepts to analyse the performance of our model and its output. The average precision is introduced in Section 5.2.1. Section 5.2.2 deals with the Shapley value and the accompanying Python library used in this thesis. These concepts are used to try to get insights into and explain the decision-making process of our algorithm.

In Chapter 6, this information will be used to iteratively build and evaluate fraud detection models.

5.1 The Dataset

The new graph will be built out of two datasets, both being a subset of the full portfolio. As before, the data covers reported claims for car insurance. The first dataset contains intrinsic features about the claims, e.g. the date and cause of the accident/claim. Furthermore, it also contains some network information, including a reference number for the contract and the broker.

The second set is the backbone of our network. The data therein shows which counterparties were involved in the claims. For each of the counterparties, we have a reference number and the type of counterparty. In case this counterparty is a car, the table has a column for the type of car (brand and model), and the car's power. Other parties can be, e.g., witnesses, passengers, lawyers etc. We note that there is redundant information when pooling both datasets, since brokers can be present in both the first and second dataset. We deal with this when constructing the graph in order to de-duplicate the edges in our network.

If we make abstraction of the different types of non-claims, and say that we classify all as *counterparty*, the resulting network is bipartite by design. Hence, it is not necessary anymore to do additional projections. It is unavoidable that we are going to use the word counterparty interchangeably in the narrow sense, i.e. the ones coming from the second dataset, or in the broad sense, namely all nodes that are not a claim, i.e. those counterparties in the narrow sense together with the contracts and brokers. It should be clear from the context which definition we are referring to.

Similarly as before, there is an additional table indicating whether or not a claim was investigated, and if the investigated claim was deemed fraudulent by the investigator. We have that around 3% of the claims were investigated, and that in total, 0.3% of all claims are labelled as fraud.

The constructed network has around 1 750 000 nodes connected by roughly 2 670 00 edges. Of these 1 750 000 nodes, 660 000 represent claims. The rest are counterparty nodes, of which 800 000 are counterparties in the narrow sense, 300 000 represent contracts, with brokers being around 5 000 nodes in our network. Due to some very rough rounding of the figures, we are aware that the sum over the different types is not exactly the same as the total number of nodes. The sensitivity of the data makes this necessary.

Looking at the edges and using the fact that each connection in the network either begins or ends in a claim, we can summarise them as follows. Starting from a claim, there are 1 350 000 edges ending in a counterparty, 660 000 connected to a broker, and 650 000 going to a contract.

Additional insights are gained when looking at the different quantiles of the distribution of the edges per node type. This is done in Table 5.1. Some important findings are that brokers seem to be the glue that holds the full network together. Most of them connect many claims with each other. The counterparties, on the other hand, seem to be mostly unique for each entry.

Node	mean	std	min	25%	50%	75%	85%	95%	99%	max
Claims	4.07	1.11	3	3	4	5	5	6	8	107
Contracts	2.18	31.51	1	1	1	2	3	4	7	9653
Counterp.s	1.70	27.08	1	1	1	1	2	3	7	9653
Brokers	139.45	685.20	1	5	22	93	195	595	1780	35671

Table 5.1: The summary statistics of the node degrees for the different types of nodes.

All these nodes and connections result in a network of 825 connected components, the largest of which contains the bulk of the nodes, namely around 1 750 000 nodes. The second largest is only 109 nodes big.

This network will be studied in the remainder of this thesis. The next section introduces new metrics to better understand the models that we are going to build for the analysis of this network.

5.2 Additional Metrics

5.2.1 Precision-Recall Curve

In the previous chapters, we have worked with the AUC as the sole metric to measure the predictive power of our model. Here, we introduce another, namely the average precision and the precision-recall curve [33]. As the name suggests, we need two metrics to construct this curve. This is similar to the ROC curve for the AUC. Using the notation as in Section 3.3.2, we define the precision as follows:

$$\mathrm{Pr} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}.$$

Hence, the precision denotes the fraction of the observations that are predicted to be positive is actually positive.

The recall on the other hand is an old acquaintance. It is define as

$$\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}},$$

which is in fact the definition of the true positive rate from Section 3.3.2.

Similarly to the ROC curve, the precision-recall curve is plotted using different cut-off points for the predicted probability of an observation being positive or not. The recall is plotted on the x-axis and the precision on the y-axis. A similar analysis with the starting and ending point of the precision-recall curve, as for the ROC curve, is not fully possible. If we set the cut-off at 0, we already know that the recall (R) will be equal to 1. When looking at the precision, we have that all observations are labelled positive. This means that the denominator, TP+FP, will be equal to all observations, and the number of true positives is equal to the number of the positive observations in the set. Hence, the end point of the curve will be at 1 on the x-axis, while its y-value will be equal to the percentage of positive observations in the dataset. In our case, this is 0.003.

For the case where the cut-off is as high as possible, i.e. where the recall is 0, we cannot say anything about the precision. Ideally, it should be 1, but any other value between 0 and 1 is also possible.

Also for the precision-recall curve, we are interested in the area under the curve. As before, a higher area is better, since a higher precision gives us a higher confidence in the fact that an observation that is labelled positive is actually positive. As noted in [35], this area is equal to the average precision. Illustrations of this will be given in Chapter 6 when building the model.

The remaining question now is why we introduce another metric, if we have already the AUC to work with. As discussed in [12] and corroborated by [29], the accuracy results coming from the AUC may be misleading when dealing with a highly imbalance dataset, i.e., one with a large number of negative labels and only a few positive. The average precision is a better metric in this case.

It is not claimed, however, that using the AUC is completely false. The paper [12] shows that there is a one-to-one correspondence between precision-recall and ROC curve.

It proves an even stronger result that says that if one ROC curve dominates another, then the corresponding precision-recall curve of the first also dominates the second, and vice versa. Here, dominating means that all the points of the curve are above or equal to the points of the other curve. This implies that a higher AUC will lead to a higher average precision. Only when the curves cross, it could be that one has contradicting results depending on the metric that is used.

In [29], one can see that the AUC may give overly optimistic results when analysing the performance of a model. It is shown via simulations that the correlation between the AUC and average precision goes from 0.990 for a perfectly balanced dataset to 0.706 for the case when the positive cases only make up 0.99% of the dataset. In our case, we have that only 0.3% of the claims are labelled as fraud, so this imbalance is even more pronounced.

We now have a better understanding of the different metrics for evaluating our model. In the next section, we are going to investigate how we can get additional insights from the model. These insights are important to investigate whether the classification seems plausible.

5.2.2 The Shapley Value and the SHAP Package

Most of the machine learning models beyond the most simple one are seen as black boxes, where it is hard for a human observer to understand why the inputs lead to a particular output. In this thesis, we rely on the Shapley value [34] to make the decisions of the classification algorithm interpretable.

The original paper of Shapley [34] deals with games played with n players. The Shapley value tries to capture the contribution of one player i to the final outcome of the game. Translating this to a mathematical model, the Shapley value captures the effect of the different features on the outcome.

In order to come to the mathematical formula from [23], we need to introduce some new concepts. These notations will be a mix of those in the paper [23] and the book [27, Chapter 9]. We denote our model with f. This model gives an output $f(\boldsymbol{x})$ for a specific input vector $\boldsymbol{x}$ with the feature values. For this $\boldsymbol{x}$, we introduce the simplified input $\boldsymbol{x}'$, which is linked to $\boldsymbol{x}$ via the function $h_{\boldsymbol{x}}$, where $h_{\boldsymbol{x}}(\boldsymbol{x}') := \boldsymbol{x}$. Using this function, we can define $g_{\boldsymbol{x}}(\boldsymbol{x}') = g(h(\boldsymbol{x}'))$.

In what follows, we take $\boldsymbol{x}' \in \{0,1\}^N$, where N denotes the number of features. The function $h_{\boldsymbol{x}}(\boldsymbol{x}')$ can then be interpreted as selecting those feature values for which the entry is 1 in $\boldsymbol{x}'$. The way in which this particular function operates becomes clear when combining it with our model f. If we take $\boldsymbol{z}' \subseteq \{1, 2, \ldots, N\}$, we define:

$$f_{\boldsymbol{x}}(\boldsymbol{z}') := \int f(x_1, \ldots, x_N) d\mathbb{P}_{x_i' \notin \boldsymbol{z}'}, \tag{5.1}$$

hence, we integrate out the effect of those particular features values that are not selected by z'. Then, the Shapley value of feature j is given by

$$\phi_j(f) = \sum_{z' \subseteq \{1,\ldots,N\}} \frac{|\boldsymbol{z}'|!(N - |\boldsymbol{z}'| - 1)!}{N!} \left(f_{\boldsymbol{x}}(\boldsymbol{z}') - f_{\boldsymbol{x}}(\boldsymbol{z}' \backslash \{j\}) \right). \tag{5.2}$$

We make the convention that $\phi_0 = f(h_{\boldsymbol{x}}(0))$, which is the average prediction of our model per Equation (5.1).

The Shapley value can be interpreted as the average contribution of the feature to the outcome, i.e. the change in value, after introducing the feature value j. We start with a particular choice of features to include. We then look at the change in prediction before and after adding feature j to the set of selected features. This is repeated for all possible combinations of features. The average over all these contributions is used to define the overall contribution of feature j to the prediction.

The SHAP values (SHapley Additive exPlanation), introduced in [23], give the effect of the different features by going from the average prediction to the actual prediction using conditional expectations. Here, we have that $f_{\boldsymbol{x}}(\boldsymbol{z}') = \mathbb{E}[f(\boldsymbol{z}) \mid \boldsymbol{z}_{\boldsymbol{S}}]$, where $\boldsymbol{S}$ represents the features chosen by $\boldsymbol{z}'$. This is illustrated in Figure 5.1. For more detail, the interested reader can consult the paper [23].

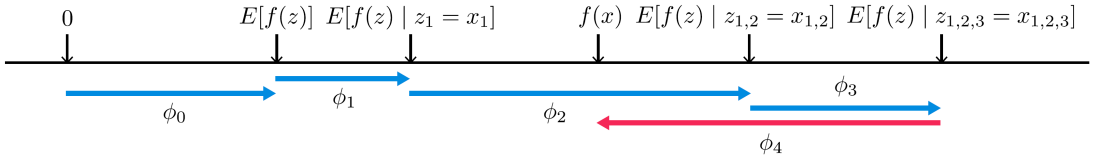


Figure 5.1: Illustration of the SHAP values. Source: [23].

Figure 5.1 is an illustration of what is done for 1 observation. The full effect of a feature j comes from looking at all the contributions ϕ_j over the different values of the feature j for all the observations in the dataset.

In the following chapters, plots generated via the SHAP package in Python will be used. These will make the interpretation of the SHAP values and how they can be used to explain a model more tangible. A detailed discussion on the resulting plots is given in Section 6.1 when the first plots with SHAP values are presented.

Chapter 6

Constructing a Full Model for Fraud Detection

This chapter will use the newly obtained dataset from Chapter 5 to build a fraud detection model with a host of different features. The model will be built iteratively to see the effect of the different features. The first model is based on the intrinsic features alone. This closely resembles what is done for classic fraud detection model, where one does not incorporate network features. Thereafter, we introduce the network data to our model. We start with the most simple data, which can quickly be determined from the dataset itself. Both these models are the item of study for Section 6.1.

Then, we incorporate what we have learned in Chapter 1 from the literature. We start by introducing the basic network features in Section 6.2, e.g., the degree centrality and geodesic distance of the nodes.

Finally, Section 6.3 applies more advanced embeddings. We start by applying the BiRank algorithm to calculate the fraud scores and see if further improvements for our detection model are gained. The last part of the section is about the metapath2vec embedding. This will introduce multiple new features to our data. It includes a discussion on the effects that this has on the interpretability of the model.

Whenever possible, we reflect on the effect that the different features have on our outcome, using the SHAP value from Section 5.2.2. This makes it possible to see if the resulting models correspond to our intuition on what may give away a fraudulent claim.

6.1 The Basic Features

Section 5.1 gave a first short introduction on the new dataset. In this section, we go into more depth in the available data. Using the data at hand, we construct new features that are deemed to be more predictive. Hence, the first part of this section deals with the feature engineering that is done, and the second part covers the resulting performance and interpretation of the model.

A first selection of features can be used right out of the box. These include the code for the specific cause of the claim/accident and the product family the policy belongs

to. To supplement these features, we do some feature engineering based on the data at hand and on expert insights from the literature.

In [36], it is mentioned that most staged accidents happen at late hours. Therefore, we construct a feature that expresses the hour of the accident. When no hour is available, we set it equal to 12 o'clock, i.e., noon. The judgement made to do this, is based on the information that the police is often called when the accident is staged [36]. Thus, we believe that fraudulent claims, happening at night, will most likely have the right time stamp. Putting missing hours to, e.g. 0, would skew the results for claims happening late at night.

Furthermore, the paper [5] applies discrete choice models to analyse the impact of different features on the probability of fraud for car insurance claims in Spain. It must be noted that the results in [36] and [5] are sometimes conflicting. As an example, when a police report is available, the chance of (organised) fraud is higher according to [36], but it lowers the chance in the models constructed in [5]. As such, we take the features as a source of inspiration, but we do not bias our expectations using these papers.

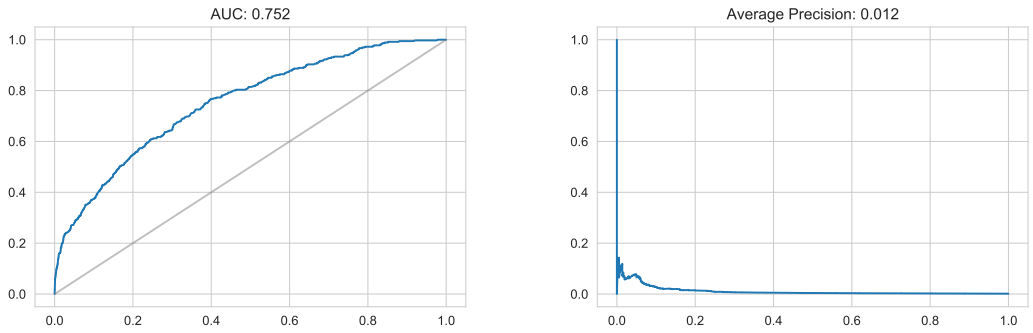
The resulting features are as follows. For each claim, the date of the accident and the date of reporting is available. From this we distil multiple features. The first is the reporting delay, which is the number of days between the occurrence and reporting date. We cap this value at 90 days to avoid extreme outliers that can skew our results. In addition, we extract both the day and month of the accident. This way, we can try to uncover seasonality effects. In winter months, it is dark longer, so it is possible that fraudsters have a longer time frame to set up illicit accident scenes.

With the above mentioned features, we train a full model. We use the gradient boosting classifier with 100 estimators and a maximal depth of 2. For this final model, we calculate the AUC, average precision (AP) and the SHAP values. The first two values are used to compare the different models that are built in this chapter, and the last is used to see the effects and importance of the features within the models.

This first simple model results in a classifier with an AUC of 0.752, but an AP of 0.012. Both graphs corresponding to these metrics are plotted in Figure 6.1. This small average precision is inherent to our problem, since the labelled data is highly imbalanced. The authors of [5] refer to a study that claims that only 1 in 3 illicit claims are actually detected [9]. So, for each fraud that is picked up by the insurance company, there are two that stay under the radar. If we accept this number, we have the following result. Assume that we have access to a perfect model that can classify all claims according to reality. If we were to compare its results with the labels in the dataset[1], one would find that the precision of this model would only be 1/3.

We now move to the SHAP values and the graphs that are produced by the package. We start with the summary plot. Here, the features are ranked with the most important feature at the top. The importance of a particular feature is measured by taking the mean of the absolute values of the SHAP values of all observations for said feature. In addition, a dot is placed for each observation per feature that represents its SHAP value, where a higher SHAP value means that this observation has a higher inclination to be

[1]Here, one needs to assume that a fraud label is always correct.



(a) The ROC curve.

(b) The precision-recall curve.

Figure 6.1: The curves for the simple model with only the intrinsic features.

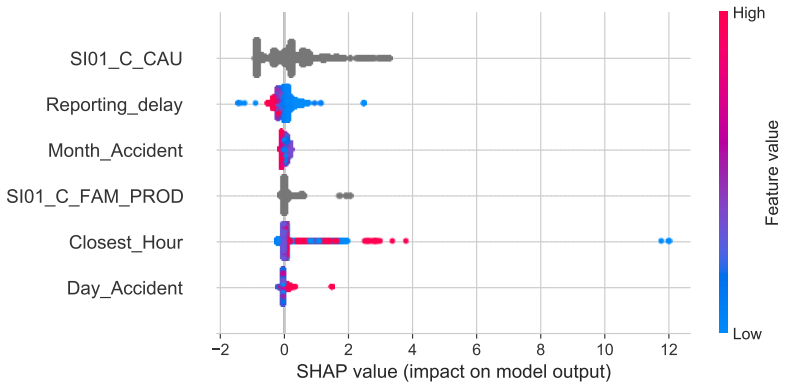


Figure 6.2: The summary plot for the first model with only intrinsic features.

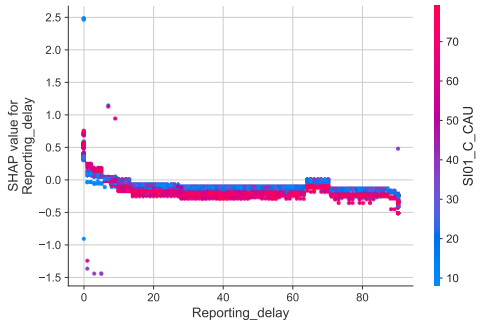
fraudulent according to the feature value. All these dots are coloured using the feature value, where red represents a high value and blue a low one. The combination of this information helps us to gain an understanding of how the different features affect the final classification probabilities.

Our very first summary plot is given in Figure 6.2. There are two things that we can notice right away. The first is that the cause of the accident is the most important predictor for our dataset. Notice how there are no colours for this feature and the one corresponding to the product family. This is not a problem for us, because being a "high" or "low" product family has no meaning at all. The second observation made is that the reporting delay seems to be correlated with the chance of being fraudulent. The blue dots, i.e. shorter delays, have a higher SHAP value than red dots, which correspond to higher reporting delays. This seems to be reasonable, since a fraudster will most likely not sit around for a couple of months before filing a claim to get paid, after staging an accident.

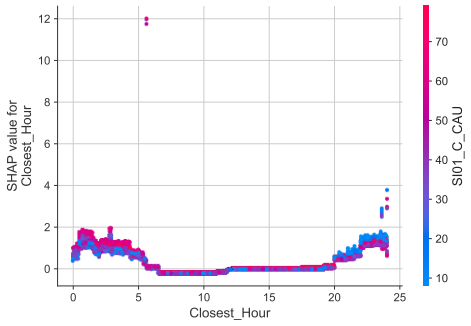
However, it is clear from the explanation of the summary plot that a lot is going

on and that much information is condensed into one figure. More detailed insights are obtained by studying the dependence plots. In these figures, one can plot the SHAP values for one particular feature. Here, additional colours are added representing the value of the feature that interacts most with the selected feature under consideration. This secondary information is of less importance to us and will mostly serve as an aesthetic characteristic of the plots.

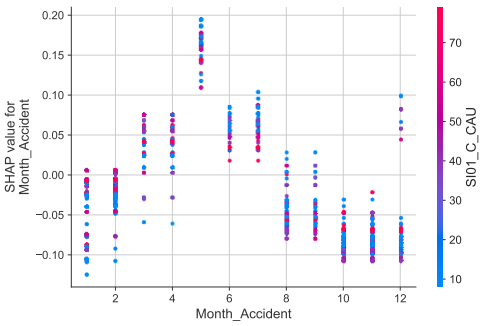
The four dependence plots corresponding to the reporting delay, hour of the accident, month and day of the occurrence are given in Figure 6.3. They give us the following, more detailed insights. The very short reporting delays seem to be associated with higher SHAP values, while longer ones do not seem to indicate more or less fraud. When looking at the time of the accident, we see that both the left and right tail have higher values. This corresponds to the nightly hours, and corroborates the above-mentioned hypothesis that illicit accidents are more likely to be staged at night. Next, we notice that the SHAP values indicate that accidents happening on a Sunday have a higher probability of being fraudulent than on other days, although this is not very pronounced.



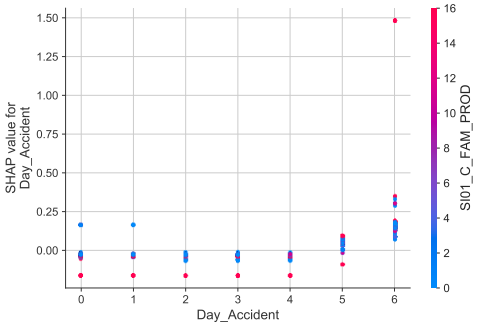
(a) The reporting delay.



(b) The hour of the accident.



(c) The month when the accident happened.



(d) The day on which the accident happened.

Figure 6.3: The dependence plots of the four intrinsic features.

Finally, we consider the dependence plot for the month of the accident. We notice that there seem to be some sort of seasonality, where claims reported in May have

much higher SHAP values than the rest of the months, with some outliers in December. Currently, there is no explanation as to why this is.

The model we have described in this section will serve as a base-line. In the following sections, we add different network features sequentially. We start with very simple and basic network information, and move to the more state-of-the art embeddings using the BiRank and Metapaht2vec algorithms.

6.2 Simple Network Features

The first network features that are used for the new fraud detection model, are extracted from the dataset directly. Each claim has information on different counterparties that are involved, e.g., passengers and third-party witnesses. This is summarised by assigning the number of each type of counterparty to each claim node. The data itself contains a multitude of different kinds of counterparties. To keep the results tangible and avoid having features where most entries are zeros, we only consider three different counterparty categories. These are the two mentioned above, namely other passengers in the car and other witnesses, and we add to that the number of lawyers involved[2].

The next set of network information, we have already used in the model constructed in Section 3.4.5, namely the geodesic distance. Closely related to this, is the number of cycles the node is a part of. A cycle is a path that begins and ends in the same node, and that does not use the same connection twice. When a node is not part of a cycle, we cannot calculate its geodesic distance, and we artificially set both values equal to 0.

The last set of simple network features are different centralities. These measure how information can flow in the network and which nodes serve as crucial links. In this thesis, we utilise three of them, namely the degree, betweenness and closeness centrality. The last two values require extensive calculations since they rely on the position of the nodes to all other nodes and possible paths (see Section 1.3.1). For this, we will use approximations.

These approximations are based on [11] and [16]. Both use a similar reasoning to estimate the metrics. They start with constructing a (small) subsample of the nodes in the network, call it C, for which the centralities are calculated exactly. For the other nodes, they make use of a *pivot*. This is where the two approaches differ. In the explanation of the procedures, we do not go into full detail since this is outside the scope of this thesis. The goal is to bring the main idea across. The full details can be found in the corresponding papers.

The closeness centrality of node j is based on the average distance to all other nodes from node j. Hence, we only need to estimate the total sum of all distances for j. For the closeness centrality [11], the pivot of a node j is defined as the node in the original subsample C that is closest to it, according to the distance measure, and this is denoted

[2]It must be noted that the usage of lawyer information may skew the results. If we know one or more are involved, it is likely that the claim is already being investigated as a fraud. This information is not available when a new claim comes in and the domain expert has to decide if it is illicit or not. Hence, it is possible that some data leakage may occur.

by $c(j)$. We are working with an undirected graph, i.e the distance metric is symmetric, so if for each node in C, we have the exact distance to all nodes in the network, we can pick any node at random, and say that we have the exact distance from it to each one in C. Therefore, we can already sum these distances between j and those of C. The particular idea in [11] is to focus more on nodes that are far away from j, which we measure as being further than a particular distance from the pivot $c(j)$[3]. An extra part is added to the sum. This term is constructed as the sum of all distances from the pivot $c(j)$ to every node in the network that is *far away* from j. The authors noted that far away nodes were not well represented in the classical approximations, while these outliers influence the exact calculation, which is based on the mean, in an important way. The interested reader can go into more detail in the paper [11].

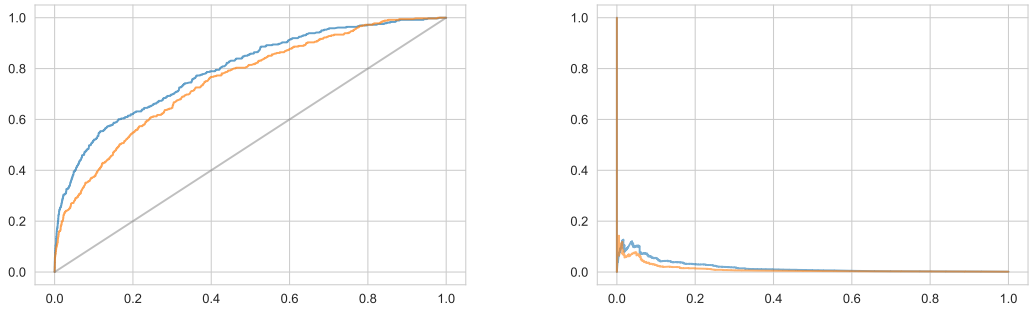
When looking at the betweenness centrality [16], a *pivot* is any node that is part of the subsample C. Since the betweenness centrality for a node j is based on the shortest paths passing through j, this approximation is done using the shortest paths between all pivots. For this problem, the authors noted that the results coming from the classical approximations are skewed when the node j is *too close* to the starting node in the path. Therefore, a *bisection scaling* is applied. This entails that we take a shortest path between two pivots s and t, and if j is in the first half of the path, we say it is too close to the pivot s and we do not use this path for the calculation of the betweenness centrality. The interested reader can go into more detail in the paper [16].

For our model, we made the approximation of the betweenness and closeness centrality using a subsample of 10 000 nodes. The resulting features can now be incorporated in our model. As we have already done many times in the past, we use a gradient boosting classifier with 100 simple learners. The results are as follows. The new model with the simple network features has an AUC of 0.793 and an average precision of 0.018. This is already a good improvement over the basic model, where the AUC was equal to 0.75 and the average precision was 0.011. We plot the ROC and precision-recall curves for both the base model and the new one in Figure 6.4. It can be seen that the ROC curve with network features dominates the one without, for the most part, indicating that we are dealing with a superior model. There is, however, some crossing of the precision-recall curve on the left.

The above results show that, in our case, the information contained inside the network adds to the prediction power of our fraud detection model. Next to the fact that we know that the network model seems to be superior, we are interested in why it is better, i.e., which network values affect the model in what way. We turn to the SHAP value to gain these insights.

Firstly, we turn our attention to the summary plot in Figure 6.5 It can be seen that some network features are very important, according to their SHAP value. The most important one is the betweenness centrality, with the degree centrality in third place. On the other hand, the Closeness Centrality and TE, which denoted the number of

[3]This distance varies with each node j in the network, and depends on the distance from j to $c(j)$. Otherwise, it could be that the node in question would be defined as far away from itself, which would be less than ideal.



(a) The ROC curve.

(b) The precision-recall curve.

Figure 6.4: The curves for the simple model (orange) with only the intrinsic features and the one with the simple network features added (blue).

witnesses, bring no added value to our model.

On the summary plot, we can already make some preliminary conclusions. For our network, a higher degree is associated with a larger chance of fraud, and the same is true for the number of cycles, although this is less pronounced. This can give away a fraud ring, where multiple parties are involved, and where they work closely together in staging the illicit accidents. This results in a high degree for the different claim nodes. Also, the number of (shortest) paths going through these nodes will be higher, and hence increase the betweenness centrality.

On the other hand, the number of third parties involved (expressed via TI) seems to impact the fraud chance negatively. The blue dots (few third parties) are on the right side (positive SHAP value), while red dots (multiple TI) are on the left side (negative SHAP value). Other SHAP values appear to be less interpretable.

To end this section, we analyse the dependence plot for the degree and betweenness centrality. On Figure 6.6a we have the betweenness centrality. The general trend indeed seems to indicate that a higher value is associated with a higher SHAP value. However, due to the presence of some outliers, the plot is less clear to interpret.

The plot for the degree centrality in Figure 6.6b is much clearer. We can easily see that a higher degree is associated with a higher chance of the claim being fraudulent.

In this section, we have seen that even simple network features can already give novel insights when hunting for illicit claims. In the next section, we proceed with more advanced methods, namely the BiRank and metapath2vec algorithms, to see if we can increase the model's performance even still.

6.3 Advanced Embeddings

In this section, we utilise multiple state-of-the-art algorithms to extract information from a social network in order to increase the performance of a fraud detection model. We

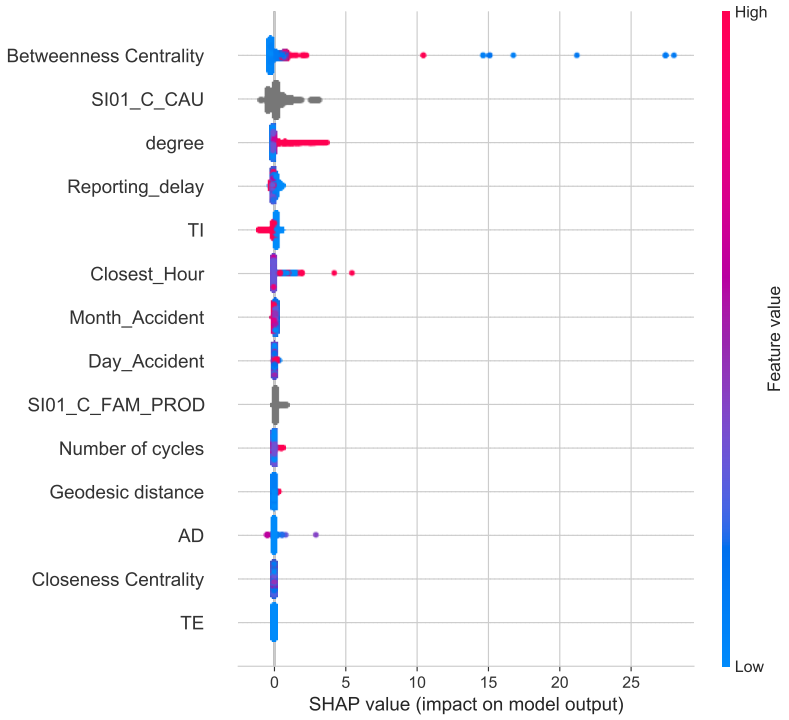
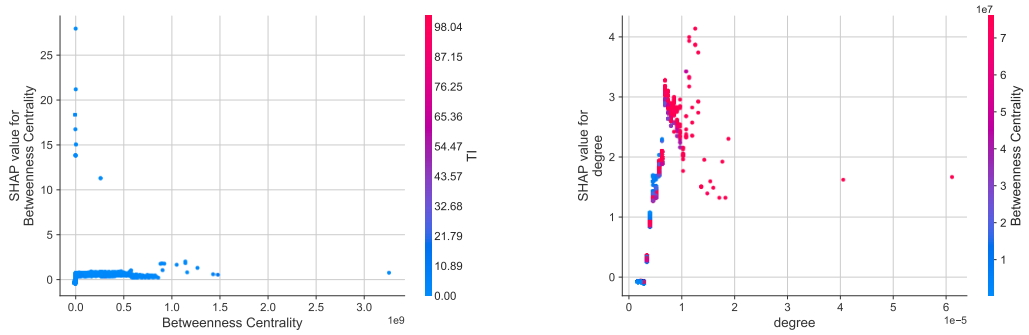


Figure 6.5: The summary plot for the model with the simple network features included.



(a) The betweenness centrality.

(b) The degree centrality.

Figure 6.6: The dependence plots for the centrality measures.

begin with the BiRank method. We explain how we adapt the train-test split and the additional information we extract from it, which were not used before. Afterwards, we move to metapath2vec and see how this one influences the model performance with and without the fraud scores coming from BiRank.

6.3.1 BiRank

The way the BiRank algorithm works is explained in Chapter 3. There, we focused on the scores as they came out of the algorithm. In this section, we extend the procedure based on what is done in [28], where the authors used summary statistics from the neighbourhood of the node as additional features in the model. More precisely, they looked at the first- and second-order neighbours and calculated the first quartile, median and maximal fraud scores. Since it is not possible for this thesis to do this in a time-efficient way, we will focus only on the first order neighbours. The scores we use will be the standardised ones.

As already mentioned before, the hard part is to have a meaningful train-test split, since training your model also incorporates the fraud labels. We will use a combination of the inductive and transductive procedures as discussed in Chapter 3. The training set is taken and split into two parts. In the first step, we keep the labels from the first part and discard the ones from the second. We let the BiRank algorithm run until convergence and keep the resulting scores for the second set for the future feature values. Hence, this follows the transductive approach. Afterwards, the scores for the test set are calculated using the obtained values from above, i.e., using the inductive approach.

One important remark to make, is that we will have to adapt the train and test data when calibrating the whole model. We can only use the second part of the full train set from before, since the first part already incorporates the fraud labels. If we would incorporate all nodes, this would result in massive data leakage, which would result into a useless model.

We enhance the BiRank scores in the same way as we did before. For the claim

nodes in the train set, we look at their claim neighbours, which corresponds to their strict second degree neighbours in the bipartite network. As before, we exclude the brokers to sample these neighbourhoods from the network. This will exclude claims from the neighbourhood that can only be reached via a broker. We deem these to be too far away. Since brokers are in many cases connected to various claims, excluding these make the neighbourhood construction faster, since significantly fewer claims can be reached from a given node. Hence, this speeds up the computing time of the algorithm. The nodes corresponding to the claims in the test set are also excluded for this first part. When no neighbours are found, we set the three summary statistics to zero. The four new features, namely the BiRank standardised fraud score, the first quartile, median and maximal value of the first order claim neighbourhood, are added to the feature set.

As always, we fit a gradient boosting classifier with 100 weak learners. This results in an AUC of 0.759 and an average precision of 0.009. It seems that adding the BiRank data does not add much to the performance of the model. When looking at the summary plot of the SHAP values on Figure 6.7, we see that the BiRank features seem to be the highly important ones. The results of the graph are not always in line with the intuition. What is clear, is that when a node at least one neighbour with a high fraud score (high n1_max), the model says that that node is more likely to be fraudulent as well.

On the other hand, for the standardised BiRank scores themselves, we see that some of the higher scores actually lead to a fraud prediction that is lower. This may be caused by interactions with other features, that the SHAP values were not able to separate. Looking at Figure 6.8, we clearly see the interaction with the n1_max feature. We already see that a higher n1_max value gives you a higher SHAP value. This is amplified if this is in combination with the node itself also having a high score. This can be seen by the red dots having a higher SHAP value. Hence, if both the node under consideration and at least one of its neighbours has a high fraud score, than the former is more likely to be an illicit claim. This pattern is reversed when the n1_max score is very small. This means that the model gives lower fraud probabilities to nodes with a higher standardised BiRank score if all of its neighbours have a very low score. This may explain a group of red dots on the left in Figure 6.7 for StdScore.

6.3.2 Metapath2vec

The final set of features that will be added, comes from the metapath2vec algorithm. As we are working with a bipartite network, we need to be careful when defining our metapaths. As mentioned in Chapter 5, the bipartite network consists of claim nodes and counterparty nodes (in the broad sense). These counterparty nodes actually consist of three sub-types, namely the contracts, brokers and counterparties (in the narrow sense of the word). Our network itself is only aware of these three types together with the

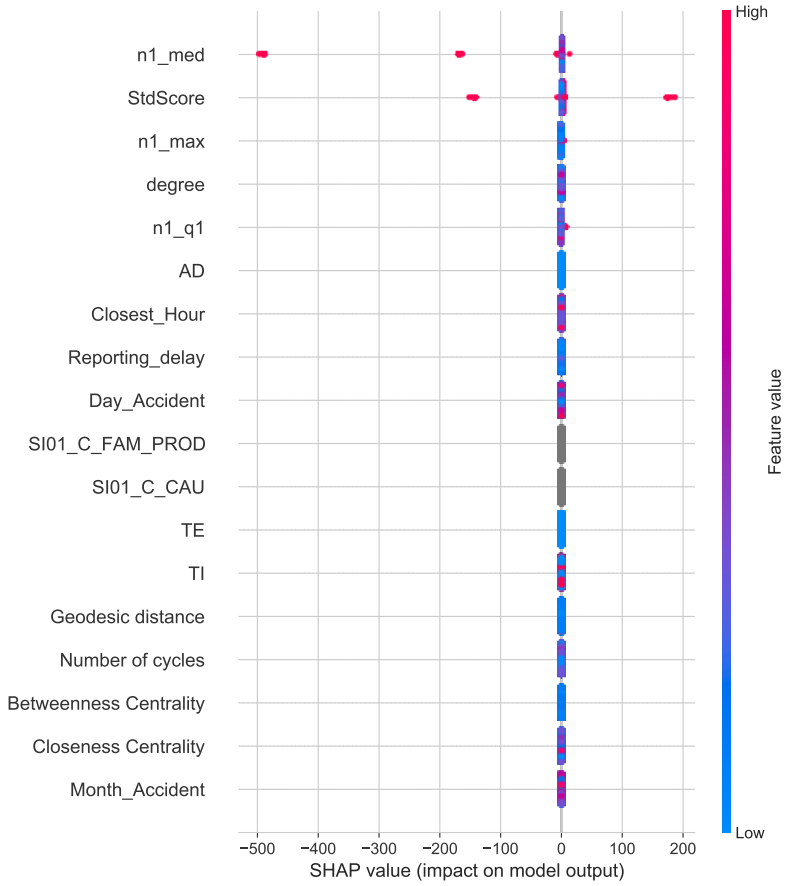


Figure 6.7: The summary plot for the model including the four features coming from the BiRank score.

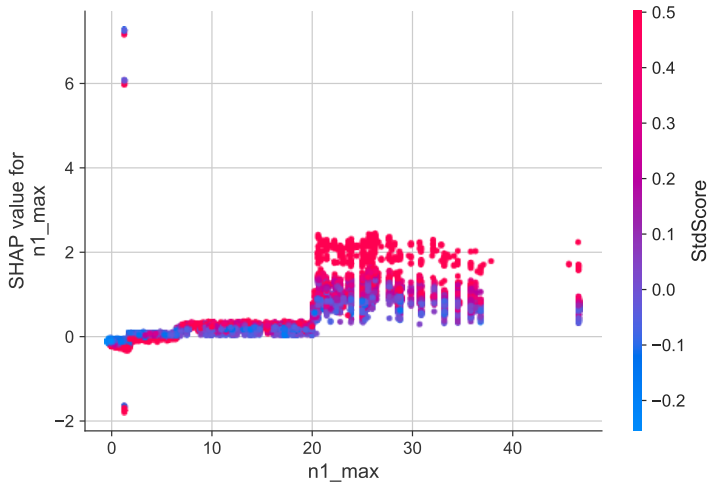


Figure 6.8: The dependence plot for the model for the n1_max scores.

claim type. Therefore, the metapaths are defined as follow[4]:

$$\begin{aligned} \mathcal{P}_1 &: \text{Claim} \to \text{Counterparty} \to \text{Claim} \\ \mathcal{P}_2 &: \text{Claim} \to \text{Contract} \to \text{Claim} \\ \mathcal{P}_3 &: \text{Claim} \to \text{Broker} \to \text{Claim} \end{aligned}$$

In order to capture the structure of the network well, we take our paths to have length 9. This way, each path will contain 5 claim nodes. Each of the nodes will be considered two times as a starting point. As before, the dimension of the embedding is taken to be 20.

These 20 new features are added to the rest. We make two models to also capture the marginal effect of adding the metapath2vec embedding. One with the BiRank features included and one without these fraud scores.

The resulting model without the BiRank scores, which is a gradient boosting classifier as before, obtains an AUC 0.792 with an average precision of 0.017. This is similar to the model without the metapath features.

There are three possible explanation why we do not see further improvements. The first is that the additional network features do not add any additional information when it comes to fraud detection. Secondly, it may well be that the 100 weak learners are not enough to deal with the complexity of all the different features that are now part of our embedding. Hence, the model would be underfitting. A third explanation would be

[4]In Chapter 4, we mentioned that the bipartite structure may have unwanted consequences for the embedding. Here, however, the dataset was given in bipartite form. A further point of study would be to construct a 4-partite network from this dataset. Due to time constraints, this is outside the scope of this thesis.

that we have reached the limit of what is possible to predict with this dataset, and an AUC around 0.79 and AP of 0.02 is the best we can hope for. This would be a result of the fact that only a handful of claims can be investigated by the domain expert, with multiple fraudulent claims going unnoticed as a consequence. The model analyses all claims and can flag some of these unlabelled claims. This results in lower AUC and AP performance. We will deal with the first possible problem below. The others are outside of the scope of this thesis.

The first problem can be investigated using the SHAP values. These numbers will only be indicative of the importance of the feature. Interpretability is not possible, since a high value of one or multiple features coming from metapath2vec will be abstract with no interpretable link to reality.

For our analysis, we have calculated the values for all 20 features, which are numbered from 0 to 19. The result is represented in Figure 6.9. From the twenty feature that are shown on the plot, half of them come from the metapath2vec algorithm. It seems that the individual features do contain important information that helps to identify illicit claims.

To end this section, we add the BiRank scores once again. We get similar results as before, namely an AUC of 0.792 and an average precision of 0.014. For completeness, we give the ROC curves and the precision-recall curves of the base model and the full model in Figure 6.10. Here, the curves behave similarly as before.

Above, we have two contradictory conclusions. Using the AUC and average precision, some gains were made over the initial model when incorporating the simple network features. However, when adding more information using the BiRank fraud scores and metapath2vec embedding, we saw no real improvements. On the other hand, the SHAP values indicated that these were important for the final fraud detection probabilities. In the next chapter, we dive deeper into these results.

We are going to look whether network features are able to detect fraud cases that would go unnoticed when only using intrinsic features. This is done by studying whether there is any complementarity in the predictions of the different models. If this were not the case, both the network and intrinsic features would contain the same information, which would explain why the AUC and AP do not change. Then, the incorporation of network features only results in unnecessary extra work.

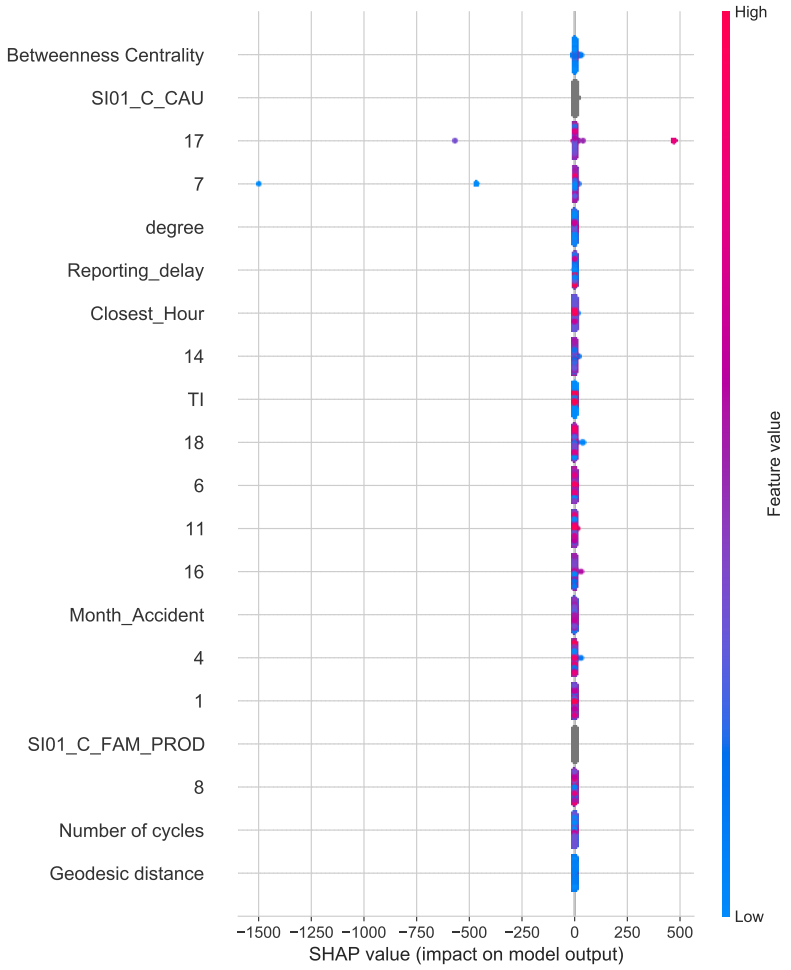
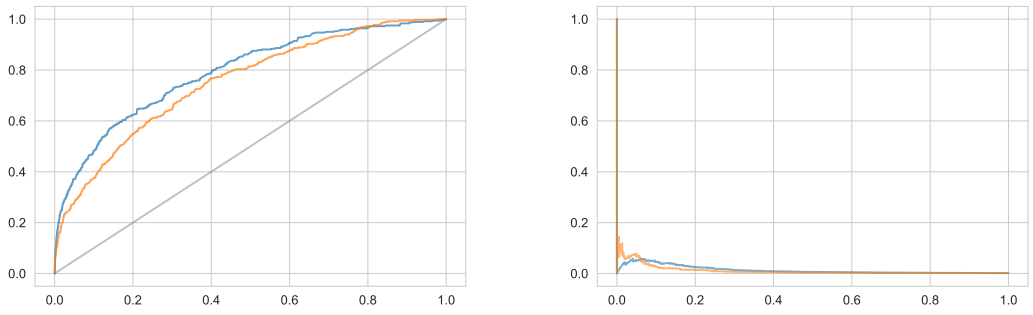


Figure 6.9: The summary plot including the 20 embedding features from metapath2vec. These are represented by the numerical values in the plot.



(a) The ROC curve.

(b) The precision-recall curve.

Figure 6.10: The curves for the simple model (orange) with only the intrinsic features and the one with all the network features added (blue).

Chapter 7

The Added Value of Using Social Network Analysis

The previous chapter combined the different kinds of network features into multiple fraud detection models. On the one hand, we saw that there is some additional predictive power present in the network models. On top of that, the SHAP values indicated that they make an important contribution to the predicted chance of fraud.

However, on the other hand, this additional predictive power was not reflected in the models using both the intrinsic and network features. In some situations, the AUC and AP metrics were worse than before. This encourages us to take a step back and study whether it is indeed sensible to include these network features.

In this chapter, we use another approach to prove or refute the information gain by using network features for our problem, i.e. the detection of claim fraud for this particular database. This will be done by studying the complementarity of the fraud predictions. More specifically, we are going to look at the predictions of illicit claims coming from both a model with intrinsic and one with network features. In case we see that the flagged cases have a large overlap, both the intrinsic claim data and the network data point in the same direction. Therefore, including the latter does not make much sense. If, on the other hand, there is complementarity, i.e., the network model points to other fraud cases, there really is a case to be made for the inclusion of these embeddings in fraud detection models. When we are in this situation, we need to give an answer to the question as to why this does not show up in the performance metrics like the average precision and AUC.

The build-up of this chapter is as follows. We start in Section 7.1 with the theory behind the lift curve and the complementarity measure used in this thesis. Afterwards, we apply these to the models constructed before in Section 7.2. Finally, Section 7.3 discusses the conclusions we can draw from this.

7.1 Comparing the Models

In this section, we introduce how we are going to study whether social network features bring anything new to the table. This is based on the ideas given in [40]. There, the authors analyse the performance of different models that use social network information to predict customer churn for telecom operators. The paper starts with the lift curve and extends that idea to see the complementarity of the models. This complementarity is exactly what is of interest for us in this chapter.

We start with the lift. Suppose we interpret the output of the classifier as the probability of a claim being fraudulent. We look at those claims with the highest $p\%$ of scores. For this subsample, we calculate the proportion of claims being illicit, call this r_p. This number r_p is then compared with the overall percentage of fraudsters in the whole dataset, denoted here by r. The lift at level p is now equal to $l = \frac{r_p}{r}$. This expresses how much higher the concentration of fraud score is in this sub-set, compared to the overall population. The lift curve plots the lift l_p for different levels of p. This will be illustrated in the next example.

Example 7.1. We come back to the network from Example 3.1, which is reshown in Figure 7.1. Contrary to Example 3.1, we assume that both claim C_1 and C_3 are fraudulent claims. To keep this illustration simple, we take only C_1 as our training set, and all other nodes as our test set. The BiRank algorithm is applied using the transductive method, i.e. we do all calculations just once immediately on the full network, but with only the fraud label of claim C_1 included. Hence, the resulting fraud scores are the same as in Example 3.1. C_3 has the highest score, then C_4, and finally there is claim C_2 with the lowest.

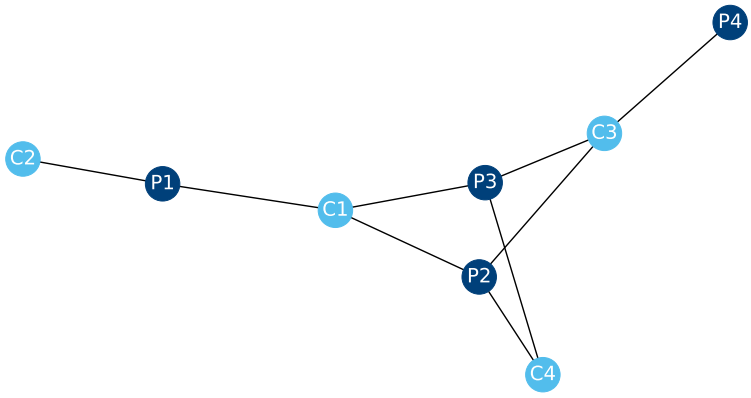


Figure 7.1: The bipartite network from Example 3.1. The counterparties are in dark blue and the claims in light blue.

The results from the test set are used to construct the lift curve. In this case, we have 1 fraud case over a total of 3 claims filed. This gives us an $r = \frac{1}{3} \approx 33.3\%$. Since we only have three points, we only need to consider two probabilities p to know the full lift curve, namely 33.3% and 66.6%.

For the first level, we only have claim c_4 in the set of claims with the highest scores. Since this one is a fraudulent claim, the fraction r_p is equal to 1. This gives a lift of $l_p = \frac{1}{\frac{1}{3}} = 3$. It is then easily checked that for $p \in [33\%, 66\%]$, the lift is equal $\frac{3}{2}$, and for all other values p higher than 66%, the lift is equal to 1.

As already mentioned above, the main object of interest for this chapter and the thesis is the complementarity, or the lack thereof, between a model using intrinsic features and one utilising network features. We follow the approach of [40] and build on top of the idea of the lift curve. As for the lift, we select the claims with a probability of p or higher of being fraud according to the two different models we want to compare. The complementarity of both classifiers is measured using the ratio of fraudulent cases detected by the first model that are not flagged by the second model, and vice versa.

If this ratio is close to 100%, the models are highly complementary and detect different segments of fraudsters. On the other hand, if the ratio is close to 0%, the classification results of one model can be substituted by those of the other model. Hence, it does not make much sense to use both models.

The hard part here is to choose a meaningful level of p to compare these models. If p is chosen too high, almost all claims are considered, which results in low fraction. Suppose on the other hand that we have two models that both have the fraudulent claims c_i and c_j as their top 2 highest scoring claims. However, due to some randomness in the training stage, we have that, according to model one, we have $\mathcal{P}_1(c_i = 1) = 0.987$ and $\mathcal{P}_1(c_j = 1) = 0.985$, but for the second model, it is the other way around, i.e. $\mathcal{P}_2(c_i = 1) = 0.985$ and $\mathcal{P}_2(c_j = 1) = 0.987$. If p is set so small that we only look at the claim with the highest probability, we only consider c_i for model 1 and c_j for model 2. Therefore, we get that the ratio of frauds detected by the first model, but not flagged by the second is equal to 1. The other way around, we again have a ratio of 1. Hence, this very low value of p would indicate that both models are completely complementary, while in reality they give very similar results[1].

In the next section, these concepts will be applied to the different models we have developed in the previous chapter.

7.2 Application and Results

In the same spirit as the previous chapters, we are going to gradually build to the ultimate model incorporating all network data. In this section, we will show the complementarity results for five different cut-off points. We look at the claims with the 1%, 5%, 10%, 20% and 50% highest predicted fraud probability.

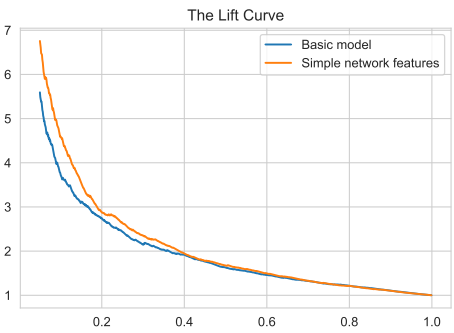
Unless stated otherwise, however, we will exclusively discuss the results for the 1% level. These results are also of most interest for practical implementations. At an insurance company, the fraud investigators will mostly concentrate on those claims with

[1]Here, we implicitly made the simplifying assumption that the fraud probabilities for both models are the same for all other claims in the test set.

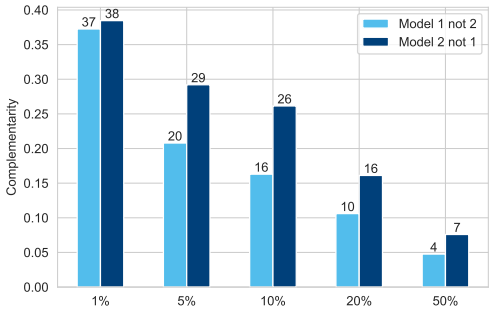
the highest predicted probability. This is because the time of these experts is limited, so they cannot consider all claims coming in.

7.2.1 The Simpler Network Features

As a start, the model with the simple network features together with the geodesic distance and the number of cycles is considered. Both the lift curve and a summary of the complementarity are given in Figure 7.2.



(a) The lift curves.



(b) The complementarity measure. Model 1 is the model with intrinsic features, and model 2 is the one with the network features.

Figure 7.2: The resulting lift curve and complementarity measure after adding the simple network features together with the geodesic distance and the number of cycles.

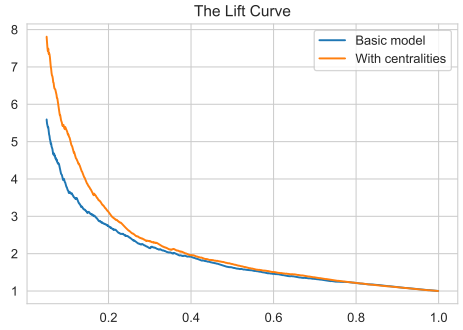
Looking at the plots, we can make out two things. First of all, the lift curve is higher when this first set of network features is added to the intrinsic features. This does not come as a surprise, since we also saw that the AUC was better, hence the fraud predictions are more reliable. This has as an effect that we capture more true positives with the top percentiles of the predicted probabilities. We see on Figure 7.3a, e.g. that amongst the claims with the 5% highest score, the ratio of illicit claims went from around 5.5 times that of the general dataset to almost 7.

Secondly, the complementarity gives additional insights. The addition of just these simple network features make both models already quite complementary. This is, e.g., visible for the claims with the 1% highest predicted probabilities. Of all the fraudulent claims that are picked up by the model with the network features added, 38% of them are not uncovered using just the intrinsic features. However, the reverse is also true. 37% of the uncovered illicit claims using only the intrinsic features were not picked up by the model with the additional network features. This shows that the new network model does not simply detect more cases[2], but that the models carve out different parts of the illicit claims in the dataset.

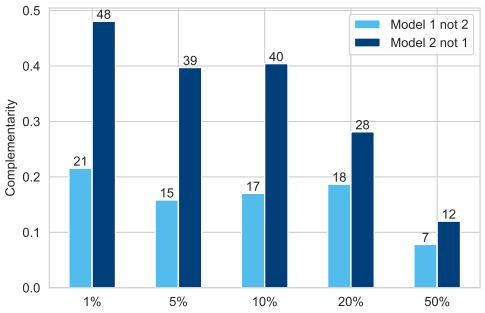
[2]In which case the height of the light blue bar would be close to 0.

Now, we move one step further. On top of the features of the previous model, we add the two centralities, namely the betweenness and closeness centrality. The graphs on Figure 7.3 already give a slightly different picture. Recall from Section 6.2, that the AUC for this model increased from 0.752 to 0.793 and the average precision went from 0.012 to 0.017. Hence, this model is already superior to the simple model, according to these two measures. The lift curve in Figure 7.3a gives a similar conclusion. The ratio of cases actually being fraud for the high prediction probabilities seems to be much higher than for the model with only intrinsic features. For the top predictions, we have that this concentration is almost 8 times higher than for the full dataset. For the former model having only the intrinsic features, this was around 5.5.

When we look at the complementarity, it can be seen that the fraud cases detected by the simple model but not by the network model has gone down. The other way around, this number has increased to 48%. This shows that this new model is more of an extension to the first model than when not including the centralities.



(a) The lift curves.



(b) The complementarity measure. Model 1 is the model with intrinsic features, and model 2 is the one with the network features.

Figure 7.3: The resulting lift curve and complementarity measure after adding the simple network features together with the geodesic distance and the number of cycles.

However, for the highest 1% predicted probabilities, we still see that 1 in 5 of the frauds from the simple model is not labelled as such by the model with the network features. Hence, this new model still picks up on different patterns and fraudulent cases when using the network information.

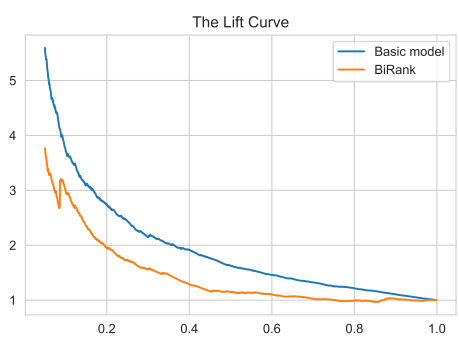
Next to these simpler network models, we are interested in how and if the network embedding algorithms add additional information.

7.2.2 The Network Embeddings

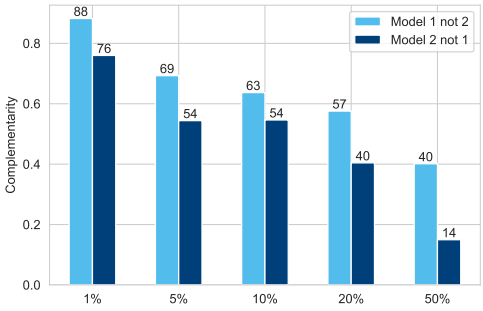
This final section deals with the network embedding algorithms as introduced in the Chapters 3 and 4. As always, we start with considering the BiRank fraud scores and then afterwards, we study the metapath2vec embeddings.

Compared to the previous section, we are not going to jump directly to a full model with intrinsic features. We are first going to analyse the information and complementarity coming from the embeddings as is. Only in a second stage, we add these features to the intrinsic and simple network features to form a final conclusion.

We start with the BiRank scores. As a reminder, we not only use the standardised fraud scores, but the minimal, median and maximal scores of its first order claim neighbours. Contrary to prior expectations, the lift curve in Figure 7.4a stays above 2 for quite some time. This again shows that it has some additional prediction power, even though the the AUC would say that is only slightly better than random guessing.



(a) The lift curves.



(b) The complementarity measure. Model 1 is the model with intrinsic features, and model 2 is the one with the BiRank scores.

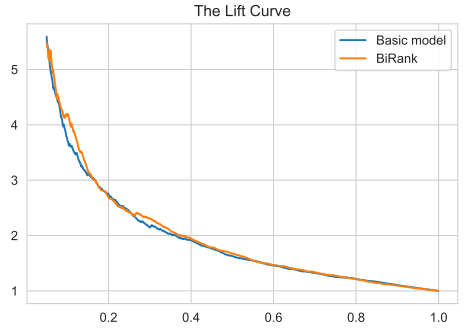
Figure 7.4: The resulting lift curve and complementarity measure for the BiRank algorithm.

The complementarity number in Figure 7.4b show that the illicit cases that are found by the two models almost do not overlap with each other. This is again a strong case for the use of network information as a way of finding other fraud patterns in the data. There is one note of caution when jumping to conclusions. Section 6.3.1 showed that the BiRank score, as they are implemented now, are not a strong indicator for detecting new fraud cases. The model with only the intrinsic features (model 1 in the figure) is superior according to all metrics we have used so far.

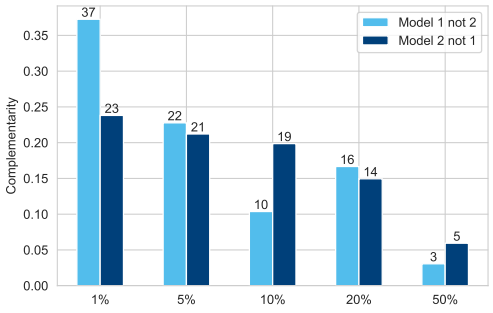
So, hypothetically speaking, we could have the following. Say that we look at the ten claims with the highest scores. Furthermore, we suppose that over the two sets for the models of ten claims each, there are also 10 fraudulent claims present in total. For convenience, we call these frauds 1 to 10. Assume now that model 1 has fraud 1 to 9 in its set of ten claims with highest fraud probability, and that the set of model 2 only contains fraud 9 and 10. Then the complementarity measure for the light blue bar would be 88% ($\approx 8/9$), while the dark blue bar would have a height of 50%. This would indicate that both models are highly complementary, but it is harder to justify that we do have qualitative additional information coming from this second model.

This qualitative issue seem to disappear when adding the BiRank information to

the intrinsic features. Figure 7.5 shows us very similar results for both models when looking at the lift-curve. However, there is quite some complementarity in between the uncovered frauds.



(a) The lift curves.



(b) The complementarity measure. Model 1 is the model with intrinsic features, and model 2 is the one with the BiRank scores added.

Figure 7.5: The resulting lift curve and complementarity measure for the BiRank scores added to the intrinsic and simple network features and the centralities.

Finally, we turn to the 20-dimensional embedding from the metapath2vec method. The features on their own already have some predictive power, as we also saw in Section 6.3.2. This is corroborated by what we see in Figure 7.6a, where the lift curve is again above 3 for the very high prediction probabilities. Looking at the complementarity results in Figure 7.6b, we see similar numbers as for the BiRank model.

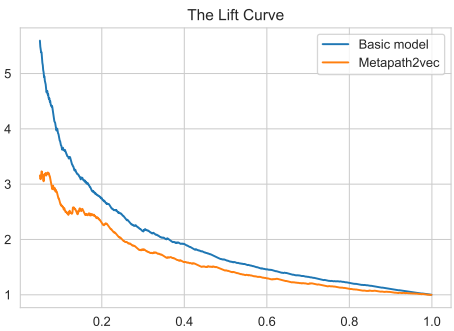
We add this embedding to the intrinsic, simple network and centrality features. The resulting numbers, as presented in Figure 7.7, closely resemble of Figure 7.3. We do the same for the model where all network features are included, so adding both metapath2vec and BiRank. Not surprisingly, Figure 7.8 results in a similar conclusion as before.

This again shows that adding these network features to the intrinsic ones does not just uncover additional illicit claims on top of those that were already known. The complementarity goes both ways. These network features shine a new light on the data present and point to other patterns hidden in the network.

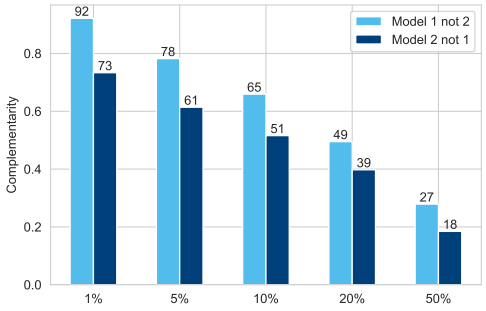
7.3 Conclusion on Complementarity

To conclude this chapter, we summarise the results. We have seen that the different techniques of extracting network information made it possible to uncover new patterns in the data. This gave complementary results when building a fraud detection algorithm.

By studying both the lift curve and complementarity numbers for the top percentage of the results, we got an glimpse of the real-life performance of these algorithms. Since a company only has limited resources to study incoming claims, the fraud expert must



(a) The lift curves.



(b) The complementarity measure. Model 1 is the model with intrinsic features, and model 2 is the one with the metapath2vec embedding.

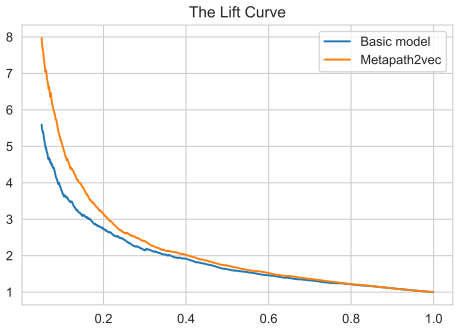
Figure 7.6: The resulting lift curve and complementarity measure for the metapath2vec embedding.

direct their attention to the claims with the highest probability of being illicit. Hence, these practical considerations make it important to study this local performance as well, when building a model, and not base the conclusions on the global metrics, like the AUC and AP, alone.

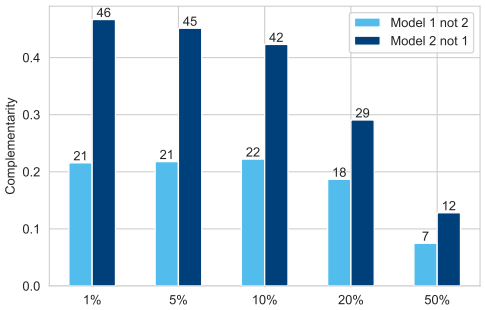
The network features are used to pick up the *social fraud* committed. This involves multiple actors in order to stage an accident and report a claim. One needs to work together with multiple people to make sure that a claim looks as legitimate as possible. These social interactions make it possible to apply the different network techniques to uncover these fraud rings, something that was not possible before. However, since this is a new technique, the data on social fraudsters is limited, hence even fewer labelled data for this sub-category is available. This is a reason why the AUC and AP metrics did not improve much.

This chapter has shown that, although we did not see much improvements in the average precision and AUC, it is worthwhile to study these network embeddings to obtain novel insights and build stronger models for assigning fraud scores to the reported claims. It will guide the fraud experts more into the direction of social fraudster, which will ultimately improve the availability of data on the problem.

Although this works shows that there is important information hidden in the network structure that can help fraud detection techniques, the hardest part is still finding a way to incorporate this into an effective algorithm. Further study should be directed at how the two complementary algorithms, namely those with only claim-specific information and those with network features, can be combined in a meaningful way. Unfortunately, this is outside the scope of this thesis.

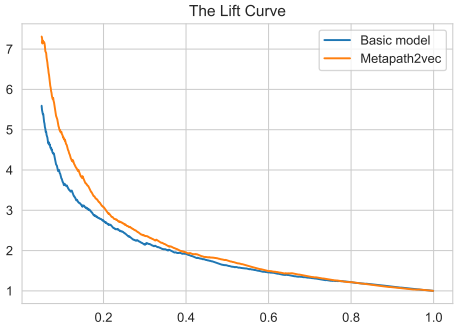


(a) The lift curves.

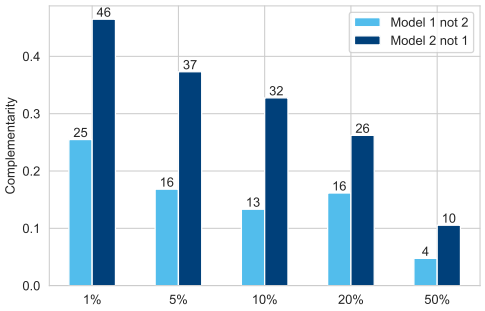


(b) The complementarity measure. Model 1 is the model with intrinsic features, and model 2 is the one with the metapath2vec embedding added.

Figure 7.7: The resulting lift curve and complementarity measure when adding the metapath2vec embedding to the intrinsic features.



(a) The lift curves.



(b) The complementarity measure. Model 1 is the model with intrinsic features, and model 2 is the one with all the network features added.

Figure 7.8: The resulting lift curve and complementarity measure when adding all the network features to the intrinsic features.

Chapter 8

General Conclusion

In this thesis, we have studied how network data can be leveraged to enhance the classical claim specific features in the context of fraud detection algorithms for motor insurance claims. This was done in multiple steps throughout this work.

In Chapter 1 we started with a basic introduction of what social networks are. To do this, we introduced multiple definitions of different aspects of these networks that were needed to build up the different algorithms later down the line. In addition, we discussed the low-dimensional representations of network information, i.e. network embeddings. These are crucial to feed network information into machine learning algorithms. It was uncovered in the literature that there are different kinds of embeddings possible. We defined the four main branches, namely anomaly detection, guilt-by-association, expert systems and random walks with NLP.

The dataset used in this thesis was introduced in Chapter 2. The network that we could extract from this, was a special kind of network, namely a 4-partite network. The four different node types were the claims (which can be illicit or not), the cars, the policies and the brokers. This special 4-partite structure made it possible to define projections, with which we could reduce the distance between claim nodes, and ultimately strengthen the stability of the algorithms. These projections were constructed in such a way as to preserve as much network information as possible. We then projected the network down to its bipartite counterpart. With this, we investigated the possible homophily present in the data. We saw a tendency for fraudulent claims to cluster together. On the other hand, these illicit claims were also less connected to the legitimate ones. This was a strong indication that there was indeed homophily present in the network, which was crucial for what came next.

Our first fraud detection method, namely the BiRank algorithm, was introduced in Chapter 3. This guilt-by-association algorithm is built on the assumption that there is homophily present amongst the filed claims. This chapter was inspired by the work done in [28]. It was used to detect fraud in our own dataset. The performance was measured via the AUC using two types of train-test split, namely the inductive and transductive split. We could observe that the stand-alone BiRank scores gave results that were slightly better than random guessing, which means that we can extract some

additional information from the network using this algorithm. Another conclusion we reached was that the brokers form crucial links in the network. We already saw in Chapter 2 that the brokers are the ones that hold the other nodes together and give us a workable network structure. Chapter 3 enforced that they are essential for fraud information to flow through the network.

Chapter 4 introduced the metapaht2vec algorithm, which can be considered as the only real embedding method in this thesis. This method combines guided random walks through the network with natural language processing to obtain low-dimensional representations of the network. This algorithm does not incorporate known fraud data to construct the embedding. It solely relies on the structure of the network and the place of the nodes in it. We could show that for our network, metapath2vec gave superior results to the BiRank scores. This was evident for two reasons. Firstly, the AUC value was higher for the metapath2vec algorithm. Secondly, the shape of the ROC curves indicated that the results of metapath2vec were more useful when implementing these in practice. This conclusion was reached using the fact that not detecting in fraud is more costly for the undertaking than falsely detecting a legitimate claim. Using this cost-based analysis, the metapath2vec algorithm was clearly superior to BiRank for our dataset.

We extended the tools at our disposal in Chapter 5. We added new claim-specific data. Next, we introduced the average precision metric to supplement the AUC. Using evidence from the literature, we argued that this average precision is more suited to compare fraud detection models, since its performance was better when the data is highly skewed. This is the case for this work, since there are far fewer fraudulent claims than non-fraudulent ones. In addition, we introduced the SHAP values. Since we would be dealing with much more features than before, these SHAP values helped us to see the effect of the different features on the predicted fraud probabilities. They also have a built-in procedure to assign feature importance using the absolute values of the SHAP values of the different data points.

In Chapter 6, we started with building a simple fraud detection algorithm only using the intrinsic features. We added the network data iteratively to study the increase of performance when leveraging this additional data, based on the AUC and AP measures. The most gains were already booked using the simpler network features, i.e. the degree, geodesic distance and the centralities. Only minor improvements were observed when adding the embedding from the BiRank and metapath2vec algorithm. On the other hand, the SHAP values indicated that these were actually amongst the most important features in the model.

To analyse the added value of these features, we studied the complementarity and lift curve in Chapter 7. There, we could clearly see that the algorithms could pick up on different types of fraud. This new type of fraud, namely the “social fraud”, is less prevalent in the network, or at least in the part of the network that has been labelled. Therefore, our methods for embedding the network data did not give major improvements over the classical algorithms only using claim-specific data. However, the complementarity showed that this social fraud was more easily uncovered using the

network features. Hence, it is worthwhile to include these features when hunting for illicit claims. When time goes on, this will increase the labelled cases of social fraud and ultimately lead to more high quality data that can show the real power of these network embeddings.

Although this thesis is extensive and a lot of work has already been done, there is still much room for further research. We saw that the fraud detection literature incorporating social networks is still scarce for insurance-related cases. Hence, further research can be done that compares the performance of more of these network embeddings to each other. Next to that, it would be interesting to further study how to combine the algorithms with intrinsic features to those using network data. This thesis showed that feeding all features together into one simple algorithm may hamper the increase in performance. Other ways to leverage the complementarity should be investigated. Lastly, a more elementary study into these methods is needed. In this work, we could only draw conclusions for our particular network. Further analysis could be done in the structure present in these kinds of network, and how these can influence the performance of the different embedding techniques. This would make it possible to extend conclusions reached for the network of one insurance undertaking to what would be possible at another.

List of Figures

List of Tables

Bibliography

[1] Cs224w: Machine learning with graphs. http://web.stanford.edu/class/cs224w/. Lecture slides Stanford. Accessed: 2021-08-21.

[2] Fraudebestrijdingsbeleid. https://www.assuralia.be/nl/31-sectorinfo/zo-werkt-de-verzekering/376-fraudebestrijdingsbeleid. Accessed: 2022-02-12.

[3] Insurance fraud. https://www.fbi.gov/stats-services/publications/insurance-fraud#:~:text=Approximately%201.6%20million%20insurance%20claims,as%20much%20as%20%246%20billion. Accessed: 2022-02-12.

[4] Arsov, N., and Mirceva, G. Network embedding: An overview. *arXiv preprint arXiv:1911.11726* (2019).

[5] Artís, M., Ayuso, M., and Guillén, M. Detection of automobile insurance fraud with discrete choice models and misclassified claims. *Journal of Risk and Insurance 69*, 3 (2002), 325–340.

[6] Baesens, B., Van Vlasselaer, V., and Verbeke, W. *Fraud analytics using descriptive, predictive, and social network techniques: a guide to data science for fraud detection*. John Wiley & Sons, 2015.

[7] Barabáse, A.-L. *Network Science*, 5 ed. Cambridge University Press, 2020.

[8] Bindu, P., and Thilagam, P. S. Mining social networks for anomalies: Methods and challenges. *Journal of Network and Computer Applications 68* (2016), 213–229.

[9] Caron, L., and Dionne, G. Insurance fraud estimation: more evidence from the quebec automobile insurance industry. In *Automobile Insurance: Road Safety, New Drivers, Risks, Insurance Fraud and Regulation*. Springer, 1999, pp. 175–182.

[10] Chen, H., Sultan, S. F., Tian, Y., Chen, M., and Skiena, S. Fast and accurate network embeddings via very sparse random projection. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (2019), pp. 399–408.

[11] Cohen, E., Delling, D., Pajor, T., and Werneck, R. F. Computing classic closeness centrality, at scale. In *Proceedings of the second ACM conference on Online social networks* (2014), pp. 37–50.

[12] Davis, J., and Goadrich, M. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning* (2006), pp. 233–240.

[13] Dekimpe, K. Fundamenten voor de informatica. University course at KU Leuven KULAK.

[14] Dong, Y., Chawla, N. V., and Swami, A. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining* (2017), pp. 135–144.

[15] EIOPA. Big data analytics in motor and health insurance: A thematic review.

[16] Geisberger, R., Sanders, P., and Schultes, D. Better approximation of betweenness centrality. In *2008 Proceedings of the Tenth Workshop on Algorithm Engineering and Experiments (ALENEX)* (2008), SIAM, pp. 90–100.

[17] Grover, A., and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining* (2016), pp. 855–864.

[18] Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (2017), pp. 1025–1035.

[19] He, X., Gao, M., Kan, M.-Y., and Wang, D. Birank: Towards ranking on bipartite graphs. *IEEE Transactions on Knowledge and Data Engineering 29*, 1 (2016), 57–71.

[20] Hou, M., Ren, J., Zhang, D., Kong, X., Zhang, D., and Xia, F. Network embedding: Taxonomies, frameworks and applications. *Computer Science Review 38* (2020), 100296.

[21] James, G., Witten, D., Hastie, T., and Tibshirani, R. *An introduction to statistical learning*, vol. 112. Springer, 2013.

[22] Koutra, D., Ke, T.-Y., Kang, U., Chau, D. H. P., Pao, H.-K. K., and Faloutsos, C. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2011), Springer, pp. 245–260.

[23] Lundberg, S. M., and Lee, S.-I. A unified approach to interpreting model predictions. In *Proceedings of the 31st international conference on neural information processing systems* (2017), pp. 4768–4777.

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

[25] Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[26] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems 26* (2013).

[27] Molnar, C. *Interpretable Machine Learning*. 2019.

[28] Óskarsdóttir, M., Ahmed, W., Antonio, K., Baesens, B., Dendievel, R., Donas, T., and Reynkens, T. Social network analytics for supervised fraud detection in insurance. *Risk Analysis* (2021).

[29] Ozenne, B., Subtil, F., and Maucort-Boulch, D. The precision–recall curve overcame the optimism of the receiver operating characteristic curve in rare diseases. *Journal of clinical epidemiology 68*, 8 (2015), 855–859.

[30] Page, L., Brin, S., Motwani, R., and Winograd, T. The pagerank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab, 1999.

[31] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[32] Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (2014), pp. 701–710.

[33] Raghavan, V., Bollmann, P., and Jung, G. S. A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transactions on Information Systems (TOIS) 7*, 3 (1989), 205–229.

[34] Shapley, L. S. *17. A value for n-person games*. Princeton University Press, 2016.

[35] Su, W., Yuan, Y., and Zhu, M. A relationship between the average precision and the area under the roc curve. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval* (2015), pp. 349–352.

[36] Šubelj, L., Furlan, Š., and Bajec, M. An expert system for detecting automobile insurance fraud using social network analysis. *Expert Systems with Applications 38*, 1 (2011), 1039–1052.

[37] Sun, Y., Han, J., Yan, X., Yu, P. S., and Wu, T. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment 4*, 11 (2011), 992–1003.

[38] Van Vlasselaer, V., Bravo, C., Caelen, O., Eliassi-Rad, T., Akoglu, L., Snoeck, M., and Baesens, B. Apate: A novel approach for automated credit card transaction fraud detection using network-based extensions. *Decision Support Systems 75* (2015), 38–48.

[39] Van Vlasselaer, V., Eliassi-Rad, T., Akoglu, L., Snoeck, M., and Baesens, B. Gotcha! network-based fraud detection for social security fraud. *Management Science 63*, 9 (2017), 3090–3110.

[40] Verbeke, W., Martens, D., and Baesens, B. Social network analysis for customer churn prediction. *Applied Soft Computing 14* (2014), 431–446.

FACULTY OF BUSINESS AND ECONOMICS
Naamsestraat 69 bus 3500
3000 LEUVEN, België
tel. + 32 16 32 66 12
fax + 32 16 32 67 91
feb.leuven@kuleuven.be
www.feb.kuleuven.be