Generalized linear models for network analysis

Harmen Dijkstra

Masterthesis Mathematics
April 2017
First Supervisor: Prof. dr. E.C. Wit
Second Supervisor: MSc. M. Signorelli
Abstract

Network modelling appear in a variety of disciplines, we can find them among other disciplines in computer science, sociology, economics and biology. Exponential random graph models are widely used to model these networks. We will focus on generalized linear models as a new approach to analyse the data. The derived approach by generalized linear models is implemented in R and is used to analyse a gene regulatory network. The results with generalized linear models is compared with an already existing model: Network Enrichment Analysis Test. The data analysis with generalized linear models gives results that are partly in line with the results achieved by Network Enrichment Analysis Test.
Contents

1 Introduction 1

2 Biological Networks 3
   2.1 Gene regulatory networks .................................................. 3
   2.2 Biological pathways ......................................................... 4
      2.2.1 Kyoto Encyclopedia of Genes and Genomes ..................... 6
   2.3 Gene set enrichment analysis .............................................. 6
      2.3.1 Gene ontology .......................................................... 7

3 Exponential Random Graph Models 8
   3.1 Network structures .......................................................... 9
      3.1.1 Self-organizing network processes .............................. 9
      3.1.2 Attribute-based processes .............................. 11
      3.1.3 Exogenous dyadic covariates .......................... 11
   3.2 Network models ............................................................... 12
      3.2.1 Bernoulli model ....................................................... 12
      3.2.2 Dyadic model .......................................................... 13
      3.2.3 Markov model .......................................................... 13
      3.2.4 Circuit model .......................................................... 14
   3.3 Disadvantages of ERGMs ................................................... 14

4 Generalized Linear Model for a binomial distribution 16
   4.0.1 The exponential family distribution .......................... 17
   4.0.2 A linear predictor and a link function ..................... 17
   4.1 Binary data ................................................................. 18
   4.2 The binomial distribution ................................................ 19
Chapter 1

Introduction

In this thesis we will model a biological network with Generalized Linear Models (GLMs). The computationally more expensive approach is to use Exponential Random Graph Models (ERGMs). The latter is especially in use in social network analysis.

The applications of network studies are in many disciplines; we can find them among other disciplines in computer science, sociology, economics and biology. In this thesis we will look in particular to biological networks. The study of biological networks are an important part of life science today. The outline of this thesis is as follows. We start in Chapter 2 by considering biological networks. In particular, we will describe the interaction of biological pathways which will be the foundation of the network.

In Chapter 3 we introduce ERGMs as the standard model to model a networks. The network structures are then formulated with their network statistics. We will go along some dependence assumptions and some models which use these assumptions. However, there are some disadvantages with ERGMs as can be found in Section 3.3.

To broaden the field of network modelling, we focus on a new approach to model networks. We will do this with GLMs which are commonly used in statistics. The background of GLMs will be discussed in Chapter 4. In Section 4.5 will we go into the issue of simplifying the model with model selection. The idea of simplifying the model comes due to William of Occam a philosopher of the fourteenth century. The principle of Occam’s razor is also called the law of parsimony. In science, this law can be stated as: ”What can be done with fewer [assumptions] is done in vain with more.” Meaning that simpler theories or models are preferable over the more complex ones.

The derived approach by generalized linear models is implemented in R and is used to analyse a gene regulatory network in Chapter 5. The results with the model with generalized linear model is com-
pared with an already existing model: Network Enrichment Analysis Test (NEAT). The data analysis with generalized linear models gives results that are partly in line with the results achieved by NEAT. Chapter 6 contains some short notions of more extensive models which overcomes some difficulties of GLMs. Suesse [32] introduced marginalized exponential random graph models, combining GLMs and ERGMs. And Hoff [16] adds random effects to generalized linear models to still capture network dependence.

In Appendix A are the derivations which are not in the thesis and in Appendix B are the R codes which were used to get the results.
Chapter 2

Biological Networks

With biological networks we can represent biological processes such as metabolic pathways, food webs, protein-protein interaction or gene regulatory networks [5]. We will focus on the last one, gene regulatory networks.

2.1 Gene regulatory networks

DNA is divided into functional units called genes. Genes are expressed by means of proteins. This conversion from genes to proteins consist of two steps: transcription and translation. During the transcription process, the gene sequence is copied into messenger RNA (mRNA). Translation is the second step in expressing a gene into a protein. The copy is translated into the protein sequence.

Proteins have different roles inside a cell. The best-known role of proteins in the cell is as enzymes. Enzymes influence the rate of chemical reactions. A second role of proteins is their involvement in the process of cell signalling and signal transduction. For example, there are proteins that transmit signals and others act as receptors that can bind to these signals which causes a reaction in the cell.

A third role of proteins are the so called structural proteins. Which refers to the structure these are able to make in biological components.

Some proteins have a certain interaction with genes. Namely to activate, repress or translate other genes. These are the proteins we are looking for in a gene regulatory network. In such a network are the proteins the nodes and the edges are the corresponding interaction between these proteins. Genes are mostly regulated by other genes through some proteins. That is why there are different levels of which we can model the network [7]. In Figure 2.1 we see a hypothetical gene network with the different functional levels.

In this thesis we have a data set which looks at the interactions between the genes. Another possibility
is to look into a gene regulatory network where the interactions are between proteins.

2.2 Biological pathways

Human bodies are constantly receiving chemical cues stimulated or generated by different things. From outside the body such as food, odour, light and injury. From inside the body such as hormones, cytokines and neurotransmitters. Cells send and receive signals to react on these cues. A biological pathway consist of all the molecules that interact with each other for a certain task. An example of such a task is to repair a little scratch on a hand, some cells will signal other cells nearby which can fix the injury.

These pathways are organised in different databases. One of the databases is the KEGG pathway
database, which is a popular pathway database, this database is used for the data in this thesis. In Figure 2.2 we see an example of a biological pathway from the KEGG database.

Researchers found out that biological pathways are much more complicated than once thought. As it seems, biological pathways work together and interact with each other. So, there is not only interaction between the molecules inside the biological pathways, there is also interaction between the biological pathways. Pathways have no boundaries such that they can work together to accomplish tasks. The interaction of these biological pathways make up a biological network and this is the network we will be studying. From now on will we call these different pathways ‘sets’.

1See Section 2.2.1
2.2.1 Kyoto Encyclopedia of Genes and Genomes

As described above is Kyoto Encyclopedia of Genes and Genomes (KEGG) one of the databases which contain biological pathways. The data derived by Vinciotti et al. has used this database [33].

“Given the estimated networks, the test developed by Signorelli, and implemented in the R package neat, is used to detect enrichment of the networks among KEGG pathways. In particular, the test detects whether the number of edges between two pathways in the inferred network is larger than what is expected by chance. For this, we download all human KEGG pathways using the R package KEGGREST Tenenbaum. Out of the total 299 pathways, we filter 62 pathways as those that contain at least 20 of the selected genes and test for enrichment amongst any pair of pathways. Finally, we rank the p-values and build a network with 62 nodes (the pathways) and with edges corresponding to the top enrichments.” [33]

2.3 Gene set enrichment analysis

A gene is considered to be expressed (or ‘turned on’) if a gene is producing its protein or RNA product. The amount of mRNA which is expressed can be measured. In a typical gene expression profiling experiment, the relative amount of expressed mRNA between two different experimental conditions are measured. The relative amount of expressed mRNA could than be the inputs of a list $L$. The inputs are than ordered by their differential expression of the two classes. The top of the list $L$ are the genes which are relative over-expressed and the bottom of the list are the genes which are relative under-expressed. The two extremes of the list $L$ are in particular interesting because those are the genes with the biggest difference in expression between the two experiments. An example could be to test whenever several tumours are resistant to a drug or not. If some genes are expressed higher with tumours which are resistant to a drug than tumours without this resistant than it might be that these genes plays a role in the resistant a drug. Therefore, in developing this drug it could be wise to perform gene expression profiling experiment [25].

Gene Set Enrichment Analysis (GSEA) is a method to determine whether a set of genes based on prior knowledge are significantly over-represented in the top or bottom of the gene list. Gene set enrichment are performed on a set of genes instead of focusing on single genes. This gene sets are group of genes that share some common biological characteristics. GSEA bears in mind the difference intensity expression between genes. In doing so, GSEA looks whenever the expression of the gene set of interest has significant difference between a certain condition. To test this significant difference
it depends on pre-defined gene sets from databases, such as GO or pathway databases. In [27] is stated that a limitation to both single and gene set enrichment analysis is that they do not consider interactions between genes. To represent these interactions one can make gene networks. In such a network are the interactions represented by edges. The method that integrate this information between genes into GSEA is called Network Enrichment Analysis (NEA), developed by Shojaie and Michailidis [26].

They suggested a linear model to represent the gene network. This method can reveal network patterns which would not be there by chance. Whether there is enrichment between two sets of genes can be determined by comparing two distributions, one with and one without enrichment between these two sets of genes. By this comparison is looked at the number of edges connecting the two sets of genes. A different method is proposed by Alexeyenko et al. [2] and McCormack et al. [22]. Their idea is that under the null hypothesis of no enrichment, the number of links between two sets will follow a reference distribution which is approximately normal. So, this reference distribution models the number of links between the two sets when there is no enrichment. This number is compared with the number of links between the two sets to estimate whenever there is enrichment between the two sets.

Signorelli et al. (2016) build upon the approach of [2] and [22], Signorelli et al. proposed a test based on the hypergeometric distribution, the Network Enrichment Analysis Test (NEAT) [27]. NEAT is also developed for directed networks. In this thesis we will compare the results of NEAT and GLM by performing a data analysis on a biological network.

2.3.1 Gene ontology

For biologist it is sometimes helpful to get all the information of a certain area of research. For example, if someone wants to create new antibiotics, he might want to know all the gene products which are involved in bacterial protein synthesis. The problem arises if one database describes these molecules as being involved in ‘translation’ and another database puts this molecule in ‘protein synthesis’. The Gene Ontology (GO) is a project which unify the descriptions of gene products [4]. This is helpful because biological terminology could for instance differ considerable due to a different in research area or species. The GO project describe gene products in three different structured ontologies: biological processes, cellular components and molecular functions.

\[2\text{See Section 2.3.1}\]
Chapter 3

Exponential Random Graph Models

Exponential random graph models (ERGMs) are a class of statistical models which can be used to analyse data about social and other networks. With ERGMs it is possible to understand small local tie-based structures. Such as edges, triangles 2-stars and 3-stars. Better understanding of these network structures give more insight into the underlying biological or social processes. Let $Y$ be a random $r \times r$ adjacency matrix, where $Y_{ij} = 1$ if there is an edge between node $i$ and $j$ and $Y_{ij} = 0$ if there is no edge between node $i$ and $j$. We call the variable for the existence of a certain tie $Y_{ij}$ a tie-variable. Note that a commonly used word in mathematics is the word ‘graph’ instead of the word ‘network’. In this thesis, we will use the word ‘network’ if there is a (biological) structure which represents the reality. We will use the word ‘graph’ if the link to reality is less clear and to clarify an underlying mathematical structure. For this thesis we use data which represents a biological network, that is why we would use in most cases the word ‘network’ in this thesis. We could however simplify this network (and underlying reality) to a graph.

$Y$ corresponds with a network of $r$ nodes, $\mathcal{R} = \{1, \ldots, r\}$. This network is the input for an ERGM. Furthermore, let $\mathcal{Y}$ be the set of all possible networks with $r$ nodes. We assume that the diagonal entries of $\mathcal{Y}$ are all one, meaning that two pathways with a certain gene are always connected by this gene. This makes sense, for example if one pathway changes the amount of a certain gene then this could affect another pathway where this gene occurs. The probability model for an ERGM is than

$$P_{\theta,Y}(Y = y) = \frac{\exp[\theta^T s(y)]}{z(\theta, \mathcal{Y})}, \quad y \in \mathcal{Y},$$

(3.1)
where $\theta \in \mathbb{R}^q$ are the model coefficients and $s : \mathcal{V} \to \mathbb{R}^q$ are statistics based on the adjacency matrix $y$ [12], [6]. The denominator,

$$
z(\theta, \mathcal{V}) = \sum_{y \in \mathcal{V}} \exp[\theta^T s(y)]
$$

is the normalizing factor such that Equation 3.1 is a valid probability distribution.

### 3.1 Network structures

In a ERGM we study local network patterns, called network configurations. Such network configurations are small subnetworks which contain some underlying process.

![Network Configurations](image)

**Figure 3.1:** Examples of network configurations for a directed network.

Observing a certain network configurations more than we would expect by chance gives information about how the individual nodes interact with each other. Local network structures can become quite complex by involving more ties in the local environment. These patterns of ties could also cause the presence of a new tie if we look at a network which evolves over time. Lusher [21] introduced three categories of network processes: self-organizing network processes, attribute-based processes and exogenous dyadic covariates.

#### 3.1.1 Self-organizing network processes

Self-organizing network processes are also called ‘purely structural effects’ because they do not have any result on the other two categories. These network processes are defined by only considering at the network ties obtained from the adjacency matrix. Important self-organizing network processes with there corresponding network statistics for a directed network are [20]:

![Network Configurations](image)
Arc: $A(y) = \sum_{i,j} y_{ij}$

Reciprocity:

$M(y) = \sum_{i<j} y_{ij} y_{ji}$

2-in-star: $S_I(y) = \sum_{i,j,k} y_{ji} y_{ki}$

2-out-star: $S_O(y) = \sum_{i,j,k} y_{ij} y_{ik}$

Two-mixed star: $S_M(y) = \sum_{i,j,k} y_{ij} y_{ik}$

Transitivity:

$T_T(y) = \sum_{i,j,k} y_{ij} y_{jk} y_{ik}$
And for a undirected network are the network statistics quite similar. The difference is that $y_{ij}$ is then similar to $y_{ji}$. We would not count these double by changing indices of the summation. For example we would formulate a 2-star as $S_2(y) = \sum_{i>j>k} y_{ij}y_{jk}$.

### 3.1.2 Attribute-based processes

Sometimes there is some similarity between a couple of nodes within the network. For a social network this could be: age, gender or profession. These terms are called ‘actor attributes’ [21]. Within a network, nodes of a particular attribute could contribute differently to the network. For example, people in the age group of 15 – 20 could be related to more ’friends’ then people in the age group of 70 – 75. In order to take into account the effect of node attributes, functions $f(X) = f(X_i, X_j)$ are frequently considered, where $X^T = (X_1, \ldots, X_p)$ is a vector of inputs with the node attributes. The corresponding statistic $\sum_{i \leq j} y_{ij}f(X_i, X_j)$ is then added to $s(y)$. An example of a node attribute function is the similarity effect of age group $f(X_i, X_j) = I(\text{agegroup}_i = \text{agegroup}_j)$, where $I$ is the indicator function. If there are $K$ different node attributes then we call the corresponding functions: $f_1(X) \ldots f_K(X)$ and the corresponding parameters $\theta_{f_1} \ldots \theta_{f_K}$.

### 3.1.3 Exogenous dyadic covariates

Exogenous dyadic covariates are covariates which could predict a part of the network. This could be another network, exogenous to the model. Mostly are networks not that simple that they do only depend on the variables of the model. Of course, the networks are only a simplification of the reality. For this reason could there be more covariates which influences the network, it is however often not possible to include all influences in the model.
3.2 Network models

Dependency is an important concept in the ERGM theory. It would not make any sense to search for local patterns if all the ties would be completely independence of each other. Hence, to give raise to any network configurations, we must propose some dependency between the ties. In other words, if we would formulate an ERGM, than we must introduce a theory of dependency. By introducing a theory about dependency we will see that we get automatically a definition of the term ‘local’: namely how far this dependency is reached. With a definition of dependency we can construct a dependence graph \( D \) as described in Wasserman and Faust [6]. \( D \) is a graph what can be used to determine which elements of \( Y \) are independent.

**Definition 3.1 (Dependence graph \( D \)).** $
D = (\mathcal{V}_D, \mathcal{E}_D)$ where,
\[
\mathcal{V}_D = \{(i, j); i, j \in \mathcal{R}, i \neq j\}
\]
and
\[
\mathcal{E}_D = \{((i, j), (k, l))\} \text{ where } Y_{i,j} \text{ and } Y_{k,l} \text{ are conditionally dependent given the rest of } Y \},
\]
The vertices of \( D \) are \( \mathcal{V}_D \) and the edges of \( D \) are \( \mathcal{E}_D \).

Different network dependence assumptions led to different types of configurations in the model which led to different combinations of statistics. This is why there are various models based on different dependence assumptions.

3.2.1 Bernoulli model

The most uncomplicated model to formulate dependency is done by the Bernoulli graph [21]. Here are all ties assumed to be independent distributed Bernoulli variables. Such a Bernoulli assumption is not realistic to most networks. Let \( Y \) be the adjacency matrix which creates the network, than we define the rest of the network by \( Y_{-(ij)} := Y \setminus Y_{ij} \). Every tie can than be modelled with a certain fixed probability

\[
P(Y_{ij} = 1|Y_{-(ij)} = y_{-(ij)}, \theta) = P(Y_{ij} = 1|\theta).
\]

Furthermore, we have that
\[
P(Y_{ij} = 1|\theta) = \sum_{y_{-(ij)}} P(Y_{ij} = 1, Y_{-(ij)} = y_{-(ij)}|\theta)
\]
\[
= \sum_{y_{-(ij)}} e^{\theta_{ij}} P(Y_{ij} = 0, Y_{-(ij)} = y_{-(ij)})
\]
\[
= e^{\theta_{ij}} P(Y_{ij} = 0),
\]
combining with \( P(Y_{ij} = 1|\theta) + P(Y_{ij} = 0|\theta) = 1 \)
and
\[
\frac{P(Y_{ij} = 1, Y_{-ij} = y_{-ij}|\theta)}{P(Y_{ij} = 0, Y_{-ij} = y_{-ij}|\theta)} = e^{\theta_{ij}}
\]
gives
\[
P(Y_{ij} = 1|\theta) = \frac{e^{\theta_{ij}}}{1 + e^{\theta_{ij}}}.
\]

The probability mass function according to this model is
\[
P_{\theta,Y}(Y = y) = \frac{\exp[\theta_A A(y)]}{z(\theta, Y)}, \quad y \in \mathcal{Y},
\]
where \( \theta_{NS} \) is the model coefficient for a certain network statistic \( NS \). See for the definitions of the network statistics on page 10. From now on will the subscript of \( \theta \) show to what network configuration it will belong.

### 3.2.2 Dyadic model

Only for directed graphs, we can construct the dyad-independent assumption [21]. Namely, that the tie from node \( i \) to node \( j \) is dependent of the tie from node \( j \) to node \( i \). If there is a tie to both directions we call this a ‘dyad’. The edge set of the dependence graph is in this case: \( \mathcal{E}_D = \{((i, j), (j, i)), \forall \ i \neq j\} \).

The probability mass function according to this model is
\[
P_{\theta,Y}(Y = y) = \frac{\exp[\theta_A A(y) + \theta_M M(y)]}{z(\theta, Y)}, \quad y \in \mathcal{Y},
\]
where the definitions of the network statistics could be found on page 10. The probabilities of the possible situations are then easily calculated
\[
P(Y_{ij} = y_{ij}, Y_{ji} = y_{ji}|\theta) = \begin{cases} 
\frac{1}{z(\theta)} & y_{ij} = y_{ji} = 0 \\
\frac{1}{z(\theta)} \exp[\theta_A] & y_{ij} = 0, y_{ji} = 1 \\
\frac{1}{z(\theta)} \exp[\theta_A] & y_{ij} = 1, y_{ji} = 0 \\
\frac{1}{z(\theta)} \exp[2\theta_A + \theta_M] & y_{ij} = y_{ji} = 1,
\end{cases}
\]
where \( z(\theta) = 1 + 2 \exp[\theta_A] + \exp[2\theta_A + \theta_M] \).

### 3.2.3 Markov model

The Markov dependence assumption is introduced by Frank and Strauss [12]. Two ties are independent of each other, conditional to the rest of the network, unless they share a common node. Directed networks have a lot of possible dependences structures, for only three nodes are there already fifteen
different configurations. If we include more nodes to the configurations then we get more configurations than we want to include in the model. That is why it is favourable to make a selection of configurations. By choosing $k$ different configurations the probability mass function will be

$$P_{\theta,Y}(Y = y) = \frac{\exp[\theta_1 s_1(y) + \cdots + \theta_k s_k(y)]}{z(\theta, Y)}, \quad y \in Y.$$ 

Nondirected networks have less configurations. Then there are four different statistics: the number of edges $L(y)$, the number of $k$-stars $S_k(y)$, the number of 2-paths $S_P$ and the number of triangles $T(y)$.

The probability mass function for a nondirected network with $n$ nodes is

$$P_{\theta,Y}(Y = y) = \frac{\exp[\theta_L L(y) + \theta_{S_2} S_2(y) + \cdots + \theta_{S_{n-1}} S_{n-1}(y) + \theta_{S_P} S_P(y) + \theta_T T(y)]}{z(\theta, Y)}, \quad y \in Y.$$ 

The higher order configurations contain lower order configurations. For example a 4-star contains 6 2-stars. In general a $k_1$-star contains $\binom{k_1}{k_2}$ $k_2$-stars, where $k_1 > k_2$. So if there are more then $\binom{k_1}{k_2}$ $k_2$-stars then there are $k_2$-stars which do not form a $k_1$-star. This tells us how much the number of $k_2$-stars influences the number of $k_1$-stars.

### 3.2.4 Circuit model

Pattison and Robins [24] expand this particular Markov dependence assumption by proposing that two ties who not share a common node could still be conditionally dependent given the presence of another tie-variable. Then these ties do not satisfy the classical assumption of Markov dependence but they satisfy the assumption of partial conditional dependence. To put it differently, two tie-variables are independent if and only if another tie-variables would not have any impact. A frequently given example is the circuit dependence. Give two disjoint pairs of actors $Y_{ij}$ and $Y_{kl}$ (thus if there are also ties between the pairs there would be a four-cycle). Then the tie-variable $Y_{ik}$ have no common node with tie-variable $Y_{jl}$ but the existence of a tie between $i$ and $k$ is likely to affect a tie between $j$ and $l$.

### 3.3 Disadvantages of ERGMs

ERGMs are one of the standard approaches in analysing networks. However there are also some disadvantages of ERGMs or issues that arrive with standard ERGMs. Maximum likelihood estimation is often complicated and therefore is this usually overcome by stochastic approximation of the log-likelihood through Markov chain Monte Carlo algorithms. But the papers that use Markov chain

---

1 More explanation of these configurations will be in Chapter 5

14
Monte Carlo algorithms describe some difficulties in convergence to realistic distributions and sometimes fails because of model degeneracy\cite{8,29}. Snijders\cite{29} also describes some strange properties, the sequences of realizations could change enormous with only little variation of the parameters. Lusher et al.\cite{21} describes five other issues. First, that the outcome depends on the choices one makes: such as model specification, the values driving the estimation algorithm and how someone deals with degenerate regions. The second he describes is about the robustness of the results with an incomplete specification of the model. The third is what he describes as the need of more rigorous approaches of model comparison. As fourth he pointed out the strong assumptions that are made in the processes. At last he issued that most work are now focused at binary variables and that there could be more development in categorical variables.

\footnote{That is, if the model places to much probability mass on only a small network configurations or variables. The extreme would be if the distribution places almost all its mass on the empty graph.}
Chapter 4

Generalized Linear Model for a binomial distribution

A Generalized Linear Model (GLM) is a generalization of a classical linear model. We start with an explanation of the classical linear models. Suppose that we observe a $r$-dimensional vector $y$. This vector of observations is a realization of a random response variable $Y$. And there are $p$ observations $x_1, x_2, \ldots, x_p$. These observations are realizations of the explanatory variables $X_1, X_2, \ldots, X_p$ which are the columns of the design matrix $X$. In this thesis, the elements of $Y$ are independently Bernoulli distributed with mean $\pi$, comparable with the previous chapter. For later purpose we consider $N$ response variables $Z_1, \ldots, Z_N$ corresponding to the number of links between the sets. Hence, $N$ is defined as the number of different ways to connect two sets, where the two sets are a subset of all the sets. We are also interested in the number of links within one set, that is why we include self loops. Therefore, we have $N = 0.5 \text{Gr} (\text{Gr} + 1)$, where $\text{Gr}$ is the number of groups (or sets). In the case of standard linear models, we have

$$E(Z_i) = \mu_i = \sum_{k=0}^{p} x_{ik}\beta_k, \quad i = 1, \ldots, N, \quad \text{(4.1)}$$

where the $\beta$’s are parameters which values are generally unknown and which we want to estimate from the data. Here is $x_0 = 1$ such that $\beta_0$ corresponds to the intercept. We write the model

$$Z = f(X) + \epsilon,$$

where $\epsilon$ is normally distributed $\epsilon \sim N(0,1)$.

With ERGM we looked at $Y_{ij}$ the possible link between node $i$ and $j$. Now are we looking at $Z_i$ the number of links between two sets. Note that we have now only one index to represent all the possible interactions between the sets.
The linear regression model has then the form

\[ f(X) = \sum_{k=0}^{p} x_k \beta_k. \]

A GLM is a generalization of above, a GLM should satisfy two elements:

1. The response variable should be from the exponential family distribution.
2. A linear predictor and a link function.

### 4.0.1 The exponential family distribution

Let \( Z \) be a distribution from the exponential family then we can write this in the form described in Faraway [11].

\[ f(z|\theta, \psi) = \exp \left[ z\theta - b(\theta) + c(z, \phi) \right] \quad (4.2) \]

The functions \( a, b \) and \( c \) specify the diverse distributions which are from the exponential family. The \( \phi \) is called the dispersion parameter which describes the scale and \( \theta \) is the canonical parameter which describes the location.

### 4.0.2 A linear predictor and a link function

The generalisation of Equation 4.1 is

\[ \eta = \sum_{k=0}^{p} x_k \beta_k, \]

where \( \eta \) is the linear predictor. If we write

\[ \eta = g(\mu), \]

then \( g \) is called the link function. The canonical link has \( g \) such that \( \eta = g(\mu) = \theta \). So the model is

\[ g(E[z_i|x_i, \beta]) = \eta_i = \sum_{k=0}^{p} x_{ik} \beta_k. \quad (4.3) \]
4.1 Binary data

In a network a tie-variable is zero or one, therefore we will focus on logistic regression such that the dependent variable is binary. We may write

\[ P(Y_i = 1) = \pi_i, \quad P(Y_i = 0) = 1 - \pi_i, \]

for the probabilities of an edge and no edge respectively. To insure that \( \pi \) is restricted to the interval \([0, 1]\), it is modelled by a cumulative probability distribution

\[ \pi = \int_{-\infty}^{t} h(s) ds, \]

where \( \int_{-\infty}^{\infty} h(s) ds = 1 \) and \( h(s) \geq 0 \). The distribution \( h(s) \) is called the tolerance distribution [10].

Three link functions which are frequently used for binary data are the logit function, the probit function and the complementary log-log function. We will shortly go along those link functions with a simple linear model \( \hat{g}_i(\pi) = \beta_0 + x\beta_1 \), for \( i = 1, 2, 3 \). The logit function is the canonical link for a binomial distribution

\[ \hat{g}_1(\pi) = \log \left( \frac{\pi}{1 - \pi} \right). \]

The tolerance distribution is

\[ h(s) = \frac{\beta_1 \exp(\beta_0 + \beta_1 s)}{(1 + \exp(\beta_0 + \beta_1 s))^2}, \]

which gives \( \pi = \frac{\exp(\beta_0 + x\beta_1)}{1 + \exp(\beta_0 + x\beta_1)} \) or equally \( \text{logit}(\pi) = \beta_0 + x\beta_1 \). This is the mostly used function for binomial data. The probit function has link function

\[ \hat{g}_2(\pi) = \Phi^{-1}(\pi), \]

where \( \Phi \) is the cumulative probability function of the standard normal distribution \( N(0, 1) \). The tolerance function is

\[ \pi = \frac{1}{\sigma \sqrt{2\pi}} \int_{-\infty}^{x} \exp \left[ -\frac{1}{2} \left( \frac{s - \mu}{\sigma} \right)^2 \right] ds = \Phi \left( \frac{x - \mu}{\sigma} \right), \]

The last link function is the complementary log-log function which has link function

\[ \hat{g}_3(\pi) = -\log(-\log(\pi)). \]
We get this function when we use the extreme value distribution as the tolerance distribution

\[ h(s) = \beta_1 \exp[(\beta_0 + \beta_1 s) - \exp(\beta_0 + \beta_1 s)]. \]

Then we have \( \pi = 1 - \exp[-\exp(\beta_0 + x\beta_1)] \) such that \( \log[-\log(1 - \pi)] = \beta_0 + x\beta_1 \). The simple linear models above are a special case of the general regression model in which \( g(\pi) = \sum_{k=0}^{p} x_k \beta_k \), with \( x_0 = 1 \).

4.2 The binomial distribution

The binomial probability distribution is

\[ f(z_i) = \binom{n_i}{z_i} \pi_i^{z_i} (1 - \pi_i)^{n_i - z_i}, \]

where \( n_i \) is the number of possible edges between two sets. The probability of this edge is stated by \( \pi_i \) and \( z_i \) is the number of edges we observed between the two sets.

After taking the logs we can write

\[ \log[f(z_i)] = z_i \log\left(\frac{\pi_i}{1 - \pi_i}\right) + n_i \log[1 - \pi_i] + \log\left(\frac{n_i}{z_i}\right). \]

So the canonical parameter for a binomial distribution is

\[ \theta_i = \log\left(\frac{\pi_i}{1 - \pi_i}\right). \]

Which gives us

\[ 1 - \pi_i = \frac{1}{1 + \exp[\theta_i]}. \]

We can now rewrite the binomial distribution as member of the exponential family with \( \theta = \log\left(\frac{\pi}{1 - \pi}\right) \), \( b(\theta) = n \log[1 + \exp(\theta)] \) and \( c(z, \phi) = \binom{n}{z} \).
4.3 Maximum likelihood estimation

Logistic regression model uses the logit link function. We will be using the logit link function because it is the most common link function for a binomial distribution. The purpose of logistic regression is to find the unknown parameters $\beta$ which can be found using maximum likelihood estimation. We follow the concepts of [9]. The joint probability density function of $Z$ is

$$f(z|\beta) = \prod_{i=1}^{N} \frac{n_i!}{z_i!(n_i-z_i)!} \pi_i^{z_i}(1-\pi_i)^{n_i-z_i}. \quad (4.4)$$

We want to maximize the maximum likelihood function

$$L(\beta|z) \sim \prod_{i=1}^{N} \left( \frac{\pi_i}{1-\pi_i} \right)^{z_i} (1-\pi_i)^{n_i}. \quad (4.5)$$

We get after some rewriting

$$L(\beta|z) \sim \prod_{i=1}^{N} \left( \exp \left[ z_i \sum_{k=0}^{p} x_{ik}\beta_k \right] \right) \left( 1 + \exp \left[ \sum_{k=0}^{p} x_{ik}\beta_k \right] \right)^{-n_i}. \quad (4.6)$$

Which gives us the log likelihood function

$$l(\beta) = \sum_{i=1}^{N} z_i \left( \sum_{k=0}^{p} x_{ik}\beta_k \right) - n_i \cdot \log \left[ 1 + \exp \left( \sum_{k=0}^{p} x_{ik}\beta_k \right) \right]. \quad (4.6)$$

\[2\text{See Appendix } A.\]
To find the maximum we need to differentiate the log likelihood

\[
\frac{\partial l(\beta)}{\partial \beta_k} = \sum_{i=1}^{N} z_i x_{ik} - n_i \cdot \frac{1}{1 + \exp \left( \sum_{k=0}^{p} x_{ik} \beta_k \right)} \cdot \frac{\partial}{\partial \beta_i} (1 + \exp \left( \sum_{k=0}^{p} x_{ik} \beta_k \right))
\]

\[
= \sum_{i=1}^{N} z_i x_{ik} - n_i \cdot \frac{1}{1 + \exp \left( \sum_{k=0}^{p} x_{ik} \beta_k \right)} \cdot \exp \left( \sum_{k=0}^{p} x_{ik} \beta_k \right) \cdot x_i
\]

(4.7)

\[
= \sum_{i=1}^{N} z_i x_{ik} - n_i \pi_i x_i.
\]

All the \((p + 1)\) \(\beta_i\)'s can now be found by setting every equation in Equation 4.7 to zero. Every solution of these equations are a minimum or a maximum depending on the sign after differentiating each equation a second time with respect to each element of \(\beta\), denoted by \(\beta_i\). These equations have the form

\[
\frac{\partial^2 l(\beta)}{\partial \beta_k \partial \beta_{k'}} = \frac{\partial}{\partial \beta_k} \sum_{i=1}^{N} z_i x_{ik} - n_i \pi_i x_i
\]

\[
= - \sum_{i=1}^{N} n_i x_{ik} \frac{\partial}{\partial \beta_k} \left( \exp \left( \sum_{k=0}^{p} x_{ik} \beta_k \right) \right) \frac{1}{1 + \exp \left( \sum_{k=0}^{p} x_{ik} \beta_k \right)}
\]

\[
= - \sum_{i=1}^{N} n_i x_{ik} \pi_i (1 - \pi_i) x_i.
\]
4.4 Fitting a GLM

In most cases, it is not possible to find analytically the exact solution after setting Equation 4.7 to zero. Therefore we use numerical optimization. McCullagh and Nelder [23] show that it is enough to converge iteratively reweighted least squares (IRWLS) instead of optimize Newton-Raphson method with Fisher scoring.

The IRWLS procedure from Faraway [11] is separated by five steps.

1 Starting with an initial estimate for the parameters \( \hat{\beta} \) we calculate
   \[ \hat{\eta}_0 = \sum_{j=0}^{p} x_{ij} \hat{\beta}_j \] and \( \hat{\mu}_0 = g^{-1}(\hat{\eta}_0) \).

2 Form the adjusted dependent variable
   \[ q_0 = \eta_0 + (z - \hat{\mu}_0) \frac{d\eta}{d\mu} |_{\hat{\eta}_0}. \]

3 Form the weights
   \[ W_0^{-1} = \left( \frac{d\eta}{d\mu} \right)^2 |_{\hat{\eta}_0} V(\hat{\mu}_0). \]  \( \text{(4.8)} \)

4 Give a new estimate of \( \beta \) and calculate corresponding \( \eta \) and \( \mu \).

5 Go back to step 2 until convergence.

We obtained the improved estimate of \( \beta \) when we calculate the weighted least-squares estimate

\[ \hat{\beta} = (X^T WX)^{-1} X^T W q, \]

where \( W \) is defined by Equation 4.8.

In case of a binomial response, we have

\[ \eta = \log \left[ \frac{\pi}{1-\pi} \right] = \log \left[ \frac{\mu}{n-\mu} \right], \]

\[ \frac{d\eta}{d\mu} = \frac{1}{n \pi (1-\pi)}. \]

So the adjusted dependent variable is

\[ q = \eta + \frac{z - n \pi}{n \pi (1-\pi)}, \]

and the iterative weight is in then

\[ w = n \pi (1-\pi). \]
4.5 Model selection and regularization

For a given data, one can choose between different models. Model selection is the process of choosing a model between all those possible models \[^{17}\]. For a set of observations \([x_1, \ldots, x_p]\) there is an underlying true distribution \(G(x)\). This probability distribution function generates the data. We want to estimate this distribution by a distribution \(F(x)\) from a training set \(Z_T = \{(x_1, z_1), \ldots, (x_M, z_M)\}\). Where \(F(X)\) is the distribution function for a specific model. The functions \(G(x)\) and \(F(x)\) have corresponding density functions \(g(x)\) and \(f(x)\). We estimate \(f(X)\) by the prediction model \(\hat{f}(X)\).

### 4.5.1 Error function

The goodness of the model \(f(x)\) is dependent on how close it is to the true model. In regression analysis is least squares a common approach to find an approximate solution which minimizes the Residual Sum of Squares (RSS). The vector with all the residuals is

\[
\epsilon = \begin{bmatrix} e_1 & \ldots & e_i & \ldots & e_N \end{bmatrix}^T,
\]

where the \(i\)th residual is defined as \(e_i = z_i - \hat{z}_i\), where \(\hat{z}_i\) is the prediction of \(Z\) based on the \(i\)th value of \(X\). Then we can define the RSS as

\[
\text{RSS} = e_1^2 + \cdots + e_N^2.
\]

For the general regression model \(z = X\beta + \epsilon\) we have as solution the least squares estimator \[^{3}\]

\[
\hat{\beta} = (X^TX)^{-1}X^Tz.
\]

We can now rewrite

\[
\text{RSS} = z^T(I - H)z,
\]

where \(H\) is the hat matrix defined as \(H = X(X^TX)^{-1}X^T\).

The loss function is the function which measures the error between \(Z\) and \(\hat{f}(X)\), denoted by \(L(Z, \hat{f}(X))\). Two examples are

\[
L(Z, \hat{f}(x)) = \begin{cases} (Z - \hat{f}(X))^2 & \text{quared error} \\ |Z - \hat{f}(X)| & \text{absolute error}. \end{cases}
\]

The test error or true error is

\[
\text{Err}_{Z_T} = E_{X_0, Z_0}[L(Z_0, \hat{f}(X_0)) | Z_T],
\]

\[^{3}\text{See for a derivation Appendix A.}\]
where $X$ and $Z$ is chosen randomly and the test error refers to this specific training set $Z_T$. Where $X^0$ and $Z^0$ notation means that we observe $M$ new values, one for every training point from $Z_T$. The expected prediction error is then $Err = E[Err_{Z_T}]$, we will normally estimate the expected error instead of the test error.

We can now rewrite the expected error for a certain input $X = x^0$ and using the squared-error loss

$$Err(x^0) = E[(Z - \hat{f}(x^0))^2|X = x^0] = \sigma^2_e + \text{Bias}^2(\hat{f}(x^0)) + \text{Var}(\hat{f}(x^0)),$$

where $\sigma^2_e$ is the irreducible error. This error is some constant noise of the system and can therefore not be decreased. If we want to reduce the expected error we have to make a trade-off between minimizing the bias and minimizing the variance. Decreasing one will in general increase the other.

A danger of measuring the error is that one can measure the error using the same data which is used to train the data. An example of such a problematic error is the training error

$$err = \frac{1}{M} \sum_{i=1}^{M} L(z_i, \hat{f}(x_i)).$$

In general is the training error not a good estimator of the test error. Let’s define the in-sample error

$$Err_{in} = \frac{1}{M} \sum_{i=1}^{M} E_{Z^0}[L(Z^0_i, \hat{f}(x_i))|Z_T].$$

The difference between the in-sample error and the training error is called the optimism, $op = Err_{in} - \overline{err}$. This is called this way because the training error will be usually to optimistic. It is to optimistic because the error is calculated using the same data.

The expectation of the optimism is then defined as

$$\omega = E[z][op] = \frac{2}{M} \sum_{i=1}^{M} \text{Cov}(\hat{z}_i, z_i).$$

---

$^4$The derivation of the last part is derived at Appendix A.
In the next paragraph, we will discuss fitting procedures for creating different models. These fitting procedures have as goal to choose a model which describes the data as good as possible. This can be by minimizing the RSS or the by minimizing the test error. There are two approaches to estimate the test error.

1 An indirect estimate of the test error by making an adjustment to the training error which takes the bias into account. We will discuss this in Section 4.5.5.

2 A direct estimate of the test error which we will discuss in Section 4.5.6.

4.5.2 Introduction to regularization

A linear regression model may result in a large number of predictors relative to the number of observations. This gives a lot of variability in the least squares fit which results in overfitting. To reduce the model one can use different methods, we will go along two approaches: subset selection and shrinkage methods. With subset selection we preserve a subset of the variables and remove the other variables in the model. Examples of strategies for selecting subsets are: best-subset selection, forward selection and backward selection. Best-subset selection takes a subset of the \( p \) predictors which predicts the model as good as possible. The model is fit using least squares on only the chosen subset of variables. Best-subset selection cost quite some time because it needs to calculate for every number of predictors which predictors must be chosen. Therefore it could be computationally more efficient to use a different subset selection method. With forward selection we start with an empty model and then stepwise select a predictor. This predictor should be chosen in such a way that the fit is mostly improved. Go further with adding with predictors until there is reached a certain maximum, the fit becomes worse by adding more terms in the model. Backward selection is the other way around. Starting with the full model there is a stepwise selection of predictors which can be removed. For each step there is one predictor removed such that the model is still as good as possible. Both selection procedures does not guarantee that the best model is found for a certain amount of predictors. Subset selection is a discrete process. Predictors are add to the model or are removed. Shrinkage methods are less discrete because it shrinks the regression coefficients by applying a penalty on the seize of the coefficients. Depending on the choice of shrinkage, some of the coefficients will not only shrinks to zero but become zero.

4.5.3 Ridge regression

The first shrinkage method we will discuss is ridge regression. The regression coefficients are shrunken by imposing a penalty on their size. Ridge regression is quite similar to least squares, the estimates
of the ridge coefficients are

$$\hat{\beta}_{\text{ridge}} = \arg \min_{\beta} \left( \sum_{i=1}^{N} \left[ z_i - \sum_{k=0}^{p} x_{ik} \beta_k \right]^2 + \lambda \sum_{k=1}^{p} \|\beta_j\|^2 \right) = \arg \min_{\beta} \left( \text{RSS} + \lambda \sum_{k=1}^{p} \|\beta_j\|^2 \right), \quad (4.10)$$

where $\lambda \geq 0$ is a complexity parameter which is to determined separately. And where $\|\beta_j\|^2$ is the $l_2$ norm of $\beta_j$. The second term is small if the $\beta_i$'s are close to zero, hence it shrinks the coefficients to zero. The estimates of the ridge coefficients depend on selecting a good parameter $\lambda$, we will discuss this in Section 4.5.6. Note the lower limits of $k$ in Equation (4.10) differs. In the RSS is $k = 0$ included but intercept $\beta_0$ is not applied to the the shrinkage penalty. We want to shrink each variable but not the intercept which is a measure of the responses. We can also write ridge regression as

$$\arg \min_{\beta} (\text{RSS}) \quad \text{subject to} \quad \sum_{k=1}^{p} \|\beta_j\|^2 \leq t. \quad (4.11)$$

Least squares coefficients corresponds to ridge regression with $\lambda = 0$. In that case is the variance high but is there no bias. If we increase $\lambda$ then this lead to an increase in bias and a decrease in variance. We want to minimize the expected error which we will in general not find at $\lambda = 0$, hence ridge regression improves least squares estimates. A high degree of multicollinearity can give several problems for estimating the coefficients by least square method. Furthermore, if $p > N$, then the least squares estimates do not even have a unique solution. Or if $p$ is close to $N$ then has least squares estimates a high variance. In these cases can ridge regression be a good alternative.

Example 4.1 shows this advantage of ridge regression over normal regression.
Example 4.1. In this example there is a high degree of multicollinearity. The two observations of this example are almost the same, however with the least square method we see that the coefficients of the two are completely different. This problem is solved with ridge regression.

```r
> x1 <- rnorm(50)
> x2 <- rnorm(50, mean=x1, sd=.0005)
> y <- rnorm(50, mean=5+x1+x2)
> lm(y~x1+x2)$coef
 (Intercept)       x1       x2
4.946235 -86.265461  88.310524
> lm.ridge(y~x1+x2, lambda=1)
 x1    x2
4.947048  1.006197  1.007758
```

4.5.4 Lasso regression

An advantage of subset selection over ridge regression is that these are models that involve only a subset of the variables. Ridge regression however will include all predictors. The penalty will only shrink the coefficients towards zeros but it will never be zero. Lasso regression is comparable to ridge regression, but lasso overcomes this disadvantage of ridge regression. The estimates of the lasso coefficients are

$$

\hat{\beta}_{lasso} = \arg \min_{\beta} \left( \sum_{i=1}^{N} \left[ z_i - \sum_{k=0}^{p} x_{ik}\beta_k \right]^2 + \lambda \sum_{k=1}^{p} ||\beta_j||_1 \right) = \arg \min_{\beta} \left( RSS + \lambda \sum_{k=1}^{p} ||\beta_j||_1 \right), \quad (4.12)

$$

Where $||\beta_j||_1$ is the $l_1$ norm of $\beta_j$. If the complexity parameter $\lambda$ is sufficiently large than are some of the coefficients exactly zero. Hence, like subset selection, the lasso will accomplish variable selection. The choice of the parameter $\lambda$ is important for the amount of shrinkage and for the number of variable selection. We will discuss more about this choice in the Section 4.5.6. We can also write lasso regression as

$$

\arg \min_{\beta} (RSS) \quad \text{subject to} \sum_{k=1}^{p} ||\beta_j||_1 \leq t. \quad (4.13)

$$

---

5The predictors will only be zero if we take $\lambda = \infty$. 

27
**Example 4.2.** So, the advantage of lasso over ridge is that some coefficients are estimated as zero, we can illustrate this in Figure 4.1.

![Figure 4.1](image_url)

Figure 4.1: An example of lasso (left) and ridge (right) regression with two coefficients. The areas at the origins are constructed by the constraint functions in Equation 4.11 and 4.13. The blue ellipses stand for the contours of the RSS. The left picture is for lasso and the right for ridge regression.

We see that only for lasso some of the coefficients becomes zero. If the least square estimate $\hat{\beta}$ lie inside the constraints than are the ridge and lasso estimates of this coefficient the same as the least squares estimates. This corresponds to a large $s$ and a small $\lambda$. If $\hat{\beta}$ lie outside the constraints then are the lasso and ridge regression coefficients estimates given by the point at which the ellipse contacts the constraint region. The constraint region of ridge regression is a circle, this is why such a intersection point will in general not be on a axis and therefore will the ridge regression coefficients not become zero. In Figure 4.1 we see that lasso has a intersection at $\beta_1 = 0$. In higher dimensions, there may be a lot of coefficients equal to zero especially if $\lambda$ is big.

The combination of the lasso and ridge method is called the elastic net penalty [15]

$$\hat{\beta}_{\text{elasticnet}} = \arg\min_\beta \left( \text{RSS} + \sum_{k=1}^p \alpha \| \beta_j \|_1 + (1 - \alpha) \| \beta_j \|_2^2 \right),$$

where $\alpha = 0$ gives ridge regression and $\alpha = 1$ gives the lasso regression. It is of course possible to something between that, such ass $\alpha = 0.5$. 

28
4.5.5 Information criterion

Another measurement to determine the closeness of the model \( f(x) \) is the Kullback-Leibner information

\[
I\{g; f\} = EG \left( \log \left( \frac{g(X)}{f(X)} \right) \right) = EG(\log[g(X)]) - EG(\log[f(X)]),
\]

where \( EG \) is the expectation with regard to the distribution \( G(x) \). The term \( EG(\log[g(X)]) \) is constant with respect to different models. So to compare different models, it is sufficient to check the expected log-likelihood \( EG(\log[f(X)]) \). If the expected log-likelihood becomes larger than is the Kullback-Leibner information smaller and therefore will the model be better. We need a good estimator of the expected log-likelihood to find a good information criterion.

The Kullback-Leibner information is zero when \( g(x) = f(x) \) and otherwise will it be more then zero. To proof this one need first make a distinguish between continuous models and discrete models:

\[
I(g; f) = \begin{cases} 
\int \log \left( \frac{g(x)}{f(x)} \right) dG(x) 
& \text{for continuous models}, \\
\sum_{i=1}^{\infty} g(x_i) \log \left( \frac{g(x_i)}{f(x_i)} \right) 
& \text{for discrete models}.
\end{cases}
\]

A familiar estimator of the expected log-likelihood is \( p^{-1}l(\hat{\beta}) \),

\[
E_G(\log[f(X|\beta)]) = \int \log[f(x|\beta)d\hat{G}(x)] = \frac{1}{p} \sum_{i=1}^{p} \log[f(x_i|\beta)],
\]

where \( \hat{G} \) is an empirical distribution function which replaces the unknown distribution \( G \). To compare the goodness of the model one can determine the magnitude of the maximum log-likelihood. Unfortunately, the maximum log-likelihood contains a bias as an estimator of the expected log-likelihood. This bias appears because we use the data two times. Namely, for estimating the parameters and for estimating the expected log-likelihood. This bias is the optimism as we discussed before. We do not know the exact value of the optimism, the difference estimates of the optimism describes the difference in methods as AIC, BIC and others. The general information criterion has a correction estimator of the optimism

\[
IC(X, \hat{G}) = -2(\text{maximum log-likelihood} - \text{optimism estimator}).
\]

\footnote{For a prove see [19]}
The Akaike information criterion (AIC) is one of the mostly used information criterion

$$\text{AIC} = -2(l(\hat{\beta}_{mle}, x) - d),$$

where $d$ is the number of parameters in the model and $l(\beta_{mle}, x)$ is the maximized log-likelihood which is defined as

$$l(\hat{\beta}_{mle}, x) = \max_{\beta} l(\beta, x).$$

### 4.5.6 Cross-validation

In Section 4.5.1 did we introduced the difference between the test error and the training error. We saw that the easily obtained training error is not a good estimate of the test error. So we want to calculate the test error, but in order to do that we need a designated validation set [17]. A way to do this, is to divide the data set into a training set and a validation set. To fit the model we use the training set and to estimate the prediction error or the deviance we use the validation set. Unfortunately, statisticians have not always access to a large amount of data. If we therefore reserve enough data to validate the data then we possibly lacking ourselves in training data. And of course vice versa.
K-fold cross-validation tries to overcome this difficulty. First we divide the data $Z$ into $K$ random and approximately similar sized parts $(D_1, \ldots, D_K)$. Now we model the data $K$ times, such that one part is the validation part and the remaining parts are used for training. In doing so, every part has exactly be one time the validation part. We define $D^*_i = D_{-i} = (D_1, \ldots, D_{i-1}, D_{i+1}, \ldots, D_K)$ with corresponding covariates $X^*_i$. Then we follow the steps [14]:

1. We model the $i$th part with $D^*_i$ and $X^*_i$ with a lasso penalty. This is done for a grid of $\lambda$’s. From every $\lambda$ we get an estimate of $\beta$.

2. Calculate the log-likelihood with the validation part $D_i$ for all these $\beta$’s.

3. Take $\lambda$ and corresponding $\hat{\beta}$ which maximizes the penalized log-likelihood.

4. Calculate for this $\hat{\beta}$ the deviance $\text{Dev}_{\hat{\lambda}}(i) = -2l(\hat{\beta}^i)$.

5. Go back to step 1 for the next part until all parts are used for validation.

6. The $K$-fold CV estimate is then computed by averaging these deviances

$$\text{CV}(\hat{\lambda}) = \frac{1}{K} \sum_{i=1}^{K} \text{Dev}_{\hat{\lambda}}(i).$$

With cross-validation we have now find $\hat{\lambda}$ which can be used to find the regression coefficients $\hat{\beta}_{\hat{\lambda}}$

$$\hat{\beta}_{\hat{\lambda}} = \arg\min_{\beta} [-l(\beta) + \lambda ||\beta||_1].$$

This penalized likelihood is controlled by the value $\lambda$, the tradeoff between the penalty and the fit. If we choose $\lambda$ to small then we tend to overfit the data. We would have a lot of parameters with a high variance. But if we choose $\lambda$ to large then we tend to underfit the data. We would have less parameters which would make the model to simplistic and biased.

\footnote{With Equation 4.6}
4.6 Glmnet

To model the data, I used the program R with the package glmnet [13]. This package fits a GLM via penalized maximum likelihood. The package glmnet minimizes

\[-\frac{1}{N} \sum_{i=1}^{N} z_i \left( \sum_{k=0}^{p} x_{ik} \beta_k \right) - \log \left( 1 + \exp \left( \sum_{k=0}^{p} x_{ik} \beta_k \right) \right) + \lambda \left( (1 - \alpha) ||\beta||_2^2 + \alpha ||\beta||_1 \right) ,\]

where it minimizes over \( \beta \) for a grid of \( \lambda \) values.

To explain the package we will use the following data

```r
> set.seed(1)
> x = matrix(rnorm(100*20),100,20)
> y = sample(1:2,100,replace=TRUE)
> fit1 = glmnet(x,y,family="binomial", alpha = 0)
> fit2 = glmnet(x,y,family="binomial", alpha = 1)
```

4.6.1 Ridge and Lasso

We modelled the data two times, with a ridge penalty and with a lasso penalty. We can now also see the difference between ridge and lasso in Figure 4.2.

```r
> plot(fit1,label=TRUE)
> plot(fit2,label=TRUE)
```

Every line in the plots corresponds to a coefficient. The \( y \)-as shows the value of the coefficient and the \( x \)-as shows \( \log(\lambda) \). The numbers above the plot indicates how much coefficients are nonzero. This is the main difference between ridge and lasso. As discussed before, by ridge are all coefficients shrunken to zero but never become zero. If we make the model more simplistic than are more and more lasso coefficients shrunken to zero. Furthermore, we see that the ridge coefficients are different shrunken to zero then the lasso coefficients. Note also the different scale of the \( x \)-as. The two models become more similar when the model becomes more complicated. We labelled the coefficients. If we compare the coefficients for ridge and lasso for the most complicated model than we see that the coefficients are almost the same.
Figure 4.2: Visualisation of the coefficients with ridge regression and lasso regression.
4.6.2 Penalty factors

It could be beneficial to exclude some variables from the model. We can do this by changing the penalty factors. The default is 1 for every parameter, such that the penalty term is normally applied to the variable. If we set for a certain variable the penalty factor equal to zero than is this variable not penalized at all. Let $v_k$ be the penalty factor for the $k$th variable. Let’s change the penalty term by

$$\lambda \sum_{k=0}^{p} v_k \left[ (1 - \alpha) \| \beta_k \|^2_2 / 2 + \alpha \| \beta_k \|^1_1 \right].$$

(4.14)

**Example 4.3.** In this example will we set some penalty factors to zero. We will perform a lasso regression on the same data as used earlier in this section such that we get a similar plot as in Figure 4.2. Let’s set the penalty factor for the variables 3 and 15 to zero.

![Figure 4.3: Performing a lasso regression where the penalty factor for the variables 3 and 15 are set to zero. These two variables will never be equal to zero due to this adjustment.](image)

34
This R code gives us a plot where the variables 3 and 15 are always in the model as can be seen in Figure 4.3.

### 4.6.3 Cross-validation

For cross-validation we need the coefficients for a certain $\lambda$. If we want the coefficients for $\lambda = 0.05$, we simply type \( \text{coef}(\text{fit2, s = 0.05}) \). We can perform cross-validation stepwise as in Section 4.5.6 where we use glmnet. We can also use the function \( \text{cv.glmnet} \) which can do all the cross-validation steps for us.

```r
> cvfit = \text{cv.glmnet}(x, y, family = "binomial", type.measure = "deviance")
> plot(cvfit)
> lambda.min<-cvfit$lambda.min
> coef(cvfit, s = lambda.min)
```

![Figure 4.4: Performing cross-validation on a small data set.](image)
Chapter 5

Generalized Linear Models Network Analysis

5.1 Combining GLM and ERGM and introducing pseudolikelihood

ERGMs have some similarities to GLMs. In a previous chapter we defined the probability model for an ERGM\(^1\), this implies

\[
\log \left[ \frac{P(Y_i = 1|Y_{-i} = y_{-i})}{P(Y_i = 0|Y_{-i} = y_{-i})} \right] = \theta^T \Delta (s(y))_i, \tag{5.1}
\]

where \(\Delta (s(y))_i\) is the difference in \(s(y)\) between \(Y_i = 0\) and \(Y_i = 1\) with the rest of the graph remains fixed. Let’s make a division in \(s(y)\) by defining \(s_1\) the statistics where \(\Delta (s(y))_i\) remains constant and \(s_2\) the statistics where \(\Delta (s(y))_i\) is not constant. For example, the total number of edges \(L(y)\) is in \(s_1\) because \(\Delta (s(y))_i = 1\) independent of the choice of \(i\). Most configurations depends on more then one edge such as \(k\)-stars and triangles, those are in \(s_2\).

If we have an ERGM with \(K = 3\) node attributes then we can express Equation 5.1 as

\[
\log \left[ \frac{P(Y_i = 1|Y_{-i} = y_{-i})}{P(Y_i = 0|Y_{-i} = y_{-i})} \right] = \eta_L + \sum_{k=1}^{K} \eta_{f_k} f_k(X_i) + \eta^T \Delta (s_2(y))_i, \tag{5.2}
\]

where \(f_k(X_i)\) are symmetric functions. If the ERGM has only statistics in \(s_1\) then is the ERGM the same as a logistic regression model, as we see if we look at a GLM in the form described by McCullagh

\(^1\) See Equation 3.1
and Nelder [23]

\[ g(P(Y_{ij} = 1)) = \beta_0 + \sum_{k=1}^{K} \beta_k \hat{f}_k(X_i). \] (5.3)

The normalizing factor in Equation 3.1 generally makes maximum likelihood estimation impossible [31]. We define the pseudolikelihood function to be

\[ P(Y_{ij} = y_{ij}) = \prod_{ij} P(Y_{ij} = y_{ij} | Y_{-ij} = y_{-ij}). \] (5.4)

This pseudolikelihood is easier to maximize than the true likelihood of Equation 3.1 since the normalizing factor is no longer in the equation. We will be maximizing the pseudolikelihood instead of maximizing the true likelihood. Strauss presented that there is an equivalence between maximise Equation 5.4 and maximum likelihood fit of logistic regression to the model of Equation 3.1 [30,31]. In this Chapter, we will analyze a data set using GLM. GLM is used to maximize the pseudolikelihood.

### 5.2 Network structures

To model a GLM on a network, we used the data derived by Vinciotti et al. [33]. We used GSEA on databases as KEGG to get a biological network. This is an undirected network of 1435 genes which are contained in 62 biological pathways. Those genes are not only connected within the pathways but also to genes who are in different pathways. Furthermore, the pathways are not disjoint. An simple network with three pathways A, B and C could look be as in Figure 5.1.

![Figure 5.1: Three pathways connected by genes.](image)
Example 5.1. To illustrate how we count the edges between two sets, we will construct a bipartite as can been seen in Figure 5.2. In this case we look at edges between set \( A \) and \( B \). For example, node 1 which is in set \( A \), is connected by node 4 which is in set \( B \). We see therefore a corresponding edge in the bipartite graph from node 1 to node 4. The edge between node 3 and node 4 is a special edge because both nodes are contained in both sets. We see therefore this edge two times in the bipartite graph. However, we do not want to count this edge twice. We have namely no different information about an edge from one set to another or precisely the other way around. So if two nodes are members of two sets which are the same, than will we count the corresponding edge between those nodes only one time. Hence, the number of edges between \( A \) and \( B \) is 6 if we include self loops and 4 if we do not include self loops.

For this thesis, we included self loops into the model. If a certain gene is included in two different pathways than it is likely that those two pathways are connected with each other due to this gene. For example, the concentration of this gene in pathway \( A \) can influence the concentration in pathway \( B \). This is why we included self loops into the model. However, if we excluded the self loops from the model than we got approximately the same end results.

![Figure 5.2: Bipartite of two sets](image)

We find out that there are 6 edges between set \( A \) and set \( B \). For the maximum number of edges between \( A \) and \( B \) we could not simply multiply the number of members of set \( A \) with the number of members of set \( B \). The same reasoning as above, there could only be one edge between two nodes which are in both sets. So \( M_n = A_n * B_n - AB_n(AB_n - 1)/2 \), where \( M_n \) is the maximum number of edges between \( A \) and \( B \). \( A_n \) and \( B_n \) are the number of nodes inside set \( A \) and in set \( B \) and \( AB_n \) is
the number of nodes which are in both sets. So in this example $M_n = 11$. Which gives $\hat{p} = \frac{6}{11}$.

In Section 3.1 we defined several network configurations. Most network configurations will not be applicable for our model because our network is undirected. Furthermore, if we model a GLM, than will we focus on the links between the nodes and not on the nodes themselves. This is why we will use the following network configurations: edges, 2-paths, added 2-stars, $\ldots$, added 10-stars and added triangles. In Figure 5.3 is a simplification given of the different network configurations for an undirected network. We will explain those network configurations in an example.

![Figure 5.3: Simplification of the different network configurations for an undirected network.](image-url)
Example 5.2. Suppose that we have a small network as in Figure 5.4.

![Small network diagram](image)

Figure 5.4: Small network

We will focus in this example on the 'link' between A and B. There is no edge between A and B therefore is the number of edges between A and B zero. If we focus on such a network, than could the number of edges between A and B only be one or zero. However, if we focus on the edges between sets this could of course be higher, as we have seen earlier. The number of 2-paths between A and B is the number of possible ways to go from A to B in two edges. In this example, there are 3 2-paths.

The number of added 2-stars from node A and B is the number of 2-stars from node A together with the number of 2-stars from node B. This is analogue for higher order stars. The number of added 2-stars is \( \binom{5}{2} + \binom{3}{2} = 13 \). The number of added 3-stars is \( \binom{5}{3} + \binom{3}{2} = 11 \), the number of added 4-stars is \( \binom{5}{4} = 5 \), there is one 5-stars and there are no higher order in this network. The number of added triangles from A and B is the number of triangles where A is an angular point together with the number of triangles where B is an angular point. In this case there are 3 added triangles between A and B. Notice that if there would be a edge between A and B than there would be two triangles where A and B are an angular point of the same triangle. In that case we would count those triangles twice, once for A and once for B. This would make the total added triangles now 7.

We have now only focused at the 'link' between A and B, so in our model the interaction between two genes. The extension to sets, or in our model biological pathways, is quite simple. We would simply focus on all possible links between two sets. So if we have a pathway of M genes and a pathway of N genes then we must investigate \( M \times N - U(U - 1)/2 \) possible links where \( U \) is the number of genes which are in both pathways.
5.3 The model

Remember the probability model 3.1

\[ P_{\theta,Y}(Y = y) = \frac{\exp[\theta^T s(y)]}{z(\theta, Y)}, \quad y \in \mathcal{Y}. \]

We are interested which model coefficients are chosen unequal to zero. We are especially interested in those who are more then zero, which means that there are more links between two sets then we would expect by chance.

Furthermore, we have the generalized linear model described in Section 4. Where the elements of \( Y \) are Bernoulli distributed, so the elements of \( Z \) are independently Binomial distributed. Lets choose the canonical link for a binomial distribution

\[ \text{logit}(\pi) = X\beta. \] (5.5)

Then we can rewrite the above for directed networks, for a certain link as

\[ \text{logit}(\pi_{ij}) = \beta_0 + \alpha_{c(i)} + \gamma_{c(j)} + \delta_{c(i),c(j)} + \varphi^1_{c(i),c(j)} + \cdots + \varphi^r_{c(i),c(j)}. \]

Where \( c(i) \) is the set in which node \( i \) is located and \( \beta_0 \) is the intercept corresponding to the density of the network. Furthermore, \( \alpha_{c(i)} \) are the model coefficients directed from the set of node \( i \) to some other set and \( \gamma_{c(j)} \) are the model coefficients directed to the set of node \( j \) from some other set. The \( \delta_{c(i),c(j)} \)'s are the model coefficients describing the interaction of the two sets. And \( \varphi^1_{c(i),c(j)} \cdots \varphi^r_{c(i),c(j)} \) are all the \( r \) different model coefficients describing certain network configurations for directed networks such as reciprocity and out-degree.

Our network is undirected, that is why we adjust the above in

\[ \text{logit}(\pi_{ij}) = \beta_0 + \gamma_{c(i)} + \gamma_{c(j)} + \delta_{c(i),c(j)} + \varphi^1_{c(i),c(j)} + \cdots + \varphi^s_{c(i),c(j)}. \]

Again, \( c(i) \) is the set in which node \( i \) is located and \( \beta_0 \) is the intercept corresponding to the density of the network. Since the network is undirected we have now \( \gamma_{c(i)} \) instead of \( \alpha_{c(i)} \). For undirected network are \( \gamma_{c(i)} \) and \( \gamma_{c(j)} \) homologous factors. Such that there is no difference between an edge from set \( A \) to set \( B \) or an edge from set \( B \) to set \( A \). The model coefficient \( \delta \) still describes the interaction between the two sets, but the dimension is changed. For undirected networks is the interaction term \( \delta_{c(i),c(j)} \) the same as \( \delta_{c(j),c(i)} \). So we delete one of the two columns in the design matrix \( X \) in Equation 5.5. The model coefficients describing the network configurations are now \( s \) different coefficients describing certain network configurations for undirected networks such as 2-paths and added 2-stars.
5.4 Cross-validation

We can perform a cross-validation on the data to select the complexity parameter. We can also use AIC or BIC, but cross-validation has the advantage over these in that it provides a direct estimate of the test error. See [17] for more advantages of cross-validation over AIC and BIC.

Using R we can evaluate the prediction performance for every $\lambda$ and pick the model with the best performance. Figure 5.5 illustrates cross-validation on the data set of Vinciotti.

![Figure 5.5: Five-fold cross-validation on the data, using the R package cv.glmnet. It illustrates that the model with the best performance is with complexity parameter $e^{-9.398} = 8.29 \times 10^{-5}$. The binomial deviance is the deviance as described in Section 4.5.6. See the appendix for the R code.]

5.4.1 Step by step

Instead of a standard defined package in R, we can also perform a stepwise cross-validation using the steps of Section 4.5.6. Before we can follow these steps we have to randomly divide the data into 5 approximately similar sized parts. In Figure 5.6 we illustrated step 2 for all the five folds. The five-fold CV estimate is then

$$CV(\hat{\lambda}) = 1.66 \times 10^{-4},$$

where $\hat{\lambda} = 8.65 \times 10^{-5}$. 

42
Figure 5.6: Performing step by step five-fold cross-validation on the data. The complexity parameter is now approximated by the average of the five parameters: $8.65 \times 10^{-5}$. See page 84 for the corresponding R code in the appendix.

5.5 Enrichment test

We use GLM to model a network instead of ERGM. We can compare our results with network enrichment analysis by using the package NEAT in R [28]. This package gives as output whenever there is enrichment between two sets. The method of the test is described by Signorelli et al. [27].

We will only explain enrichment test for undirected networks.

If there is enrichment between two sets of genes $A$ and $B$, we expect the number of links between the two sets to be larger (or smaller) than we would expect by chance. The number of edges between set $A$ and $B$ is denoted by $n_{AB}$. The observed $n_{AB}$ is a realization from the random variable $N_{AB}$. Furthermore, $\mu_{AB}$ is the expected value of $N_{AB}$ and $\mu_0$ is the number of edges that we would expect to observe from $A$ to $B$ by chance. Given that there is enrichment, we say that there is over-enrichment between $A$ and $B$ if $\mu_{AB}$ is higher than $\mu_0$ and that there is under-enrichment if $\mu_{AB}$ is lower than $\mu_0$.

The degree of a node is the number of edges which are linked by that node, the total degree of set $A$ is then defined as $d_A$ and similar for other sets. At last, we define $V$ as the set of all nodes.

Signorelli et al. proposed a hypergeometric distribution where $n_{AB}$ can be viewed as the number of successes in a random sample of size $d_A$ or $d_B$, drawn from a population of size $d_V$. If there is no
relation between $A$ and $B$ than is the distribution of $N_{AB}$ (i.e. the null distribution of $N_{AB}$)

$$N_{AB} \sim \text{hypergeom}(d_A, d_B, d_V).$$

The null hypothesis of no enrichment can then be formulated as

$$H_0 : \mu_{AB} = \mu_0,$$

against the alternative hypothesis of enrichment

$$H_1 : \mu_{AB} \neq \mu_0.$$

## 5.6 Results

To compare enrichment analysis with GLM, we investigate a simpler model

$$\logit(\pi_{ij}) = \beta_0 + \gamma c(i) + \gamma c(j) + \delta c(i) c(j),$$

so without the network configurations. We perform only a penalty factor on the interaction term $\delta$ and set the penalty factor for $\beta_0$ and $\gamma$ to zero. By doing so, we set $\beta_0$ and $\gamma$ standard in the model and are we only looking for the interaction terms.

There are 61 sets such that there are 1830 possible interactions between the sets. At $\alpha = 1\%$ level, NEAT detects enrichment between 452 sets of which 313 are over-enrichments.

For GLM we used $\hat{\lambda} = 8.65 \times 10^{-5}$, which we derived through cross-validation. The number of significant coefficients for GLM are in that case 391.

There are 164 over-enrichments detected by NEAT that are also significant using GLM. This number is approximately 2.5 times higher than random, nevertheless is there a big difference between the two methods.

We can focus more on the p-values of NEAT by making a histogram as can be seen in Figure 5.7. The left plot shows p-values of all the coefficients corresponding to the interactions which have more links than by chance. The right plot shows only the 313 over-enrichments at $\alpha = 1\%$ level. The height of the dark blue bars stands for the frequency of p-values using NEAT. Furthermore, the height of the light blue bars stands for the frequency of p-values using NEAT that are also significant using GLM. For example, by looking at the right plot, one can see that there are slightly less than 200 over-enrichments using NEAT with a p-value less than 0.001 and that there are less than 100 over-enrichments using...
NEAT with a p-value less than 0.001 that are also significant using GLM. If we compare the proportion light blue bars versus dark blue bars than are the proportion light blue bars higher by small p-values relative to bigger p-values. So the interactions with smaller p-values are relative more often significant for both models. Hence, the models have some consensus about which interactions are essential with this data set.

Figure 5.7: The left plot shows p-values of all the coefficients corresponding to the interactions which have more links than by chance. The right plot shows only the 313 over-enrichments at $\alpha = 1\%$ level. The height of the dark blue bars stands for the frequency of p-values using NEAT. Furthermore, the height of the light blue bars stands for the frequency of p-values using NEAT that are also significant using GLM.
In Section 5.3 is the model introduced for a binomial distribution

$$\text{logit}(\pi) = X\beta,$$

where

$$\text{logit}(\pi_{ij}) = \beta_0 + \gamma_c(i) + \gamma_c(j) + \delta_{c(i),c(j)} + \varphi_{c(i),c(j)}^1 + \cdots + \varphi_{c(i),c(j)}^s.$$ \hspace{1cm} (5.6)

Equation 5.6 can be adjusted by removing or adding $s$ different network configurations, this give rise to 13 different models as is presented in Table 5.1. Every model has a different penalty factor $\lambda$ which is achieved by cross-validation. After that is the quality of each model compared the others using Akiaike information criteria.

<table>
<thead>
<tr>
<th>Model</th>
<th>Network configurations</th>
<th>AIC value</th>
<th>Dif</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 1</td>
<td>$L(y)$</td>
<td>567892.5</td>
<td>589.4</td>
</tr>
<tr>
<td>Model 2</td>
<td>$L(y), S_P(y)$</td>
<td>567645.6</td>
<td>342.5</td>
</tr>
<tr>
<td>Model 3</td>
<td>$L(y), T(y)$</td>
<td>567889.8</td>
<td>586.7</td>
</tr>
<tr>
<td>Model 4</td>
<td>$L(y), S_2(y)$</td>
<td>567892.4</td>
<td>589.3</td>
</tr>
<tr>
<td>Model 5</td>
<td>$L(y), S_P(y), S_2(y)$</td>
<td>567312</td>
<td>8.9</td>
</tr>
<tr>
<td>Model 6</td>
<td>$L(y), S_P(y), T(y)$</td>
<td>567320.7</td>
<td>17.6</td>
</tr>
<tr>
<td>Model 7</td>
<td>$L(y), S_2(y), T(y)$</td>
<td>567892.4</td>
<td>589.3</td>
</tr>
<tr>
<td>Model 8</td>
<td>$L(y), S_P(y), S_3(y)$</td>
<td>567358.3</td>
<td>55.2</td>
</tr>
<tr>
<td>Model 9</td>
<td>$L(y), S_P(y), S_2(y), S_3(y)$</td>
<td>567312</td>
<td>8.9</td>
</tr>
<tr>
<td>Model 10</td>
<td>$L(y), S_P(y), S_2(y), S_3(y), S_4(y)$</td>
<td>567312</td>
<td>8.9</td>
</tr>
<tr>
<td>Model 11</td>
<td>$L(y), S_P(y), S_2(y), T(y)$</td>
<td>567303.1</td>
<td>0</td>
</tr>
<tr>
<td>Model 12</td>
<td>$L(y), S_P(y), S_2(y), \ldots, S_{10}(y)$</td>
<td>567315.5</td>
<td>12.4</td>
</tr>
<tr>
<td>Model 13</td>
<td>$L(y), S_P(y), S_2(y), \ldots, S_{10}(y), T(y)$</td>
<td>567357.2</td>
<td>54.1</td>
</tr>
</tbody>
</table>

Table 5.1: The 13 different models produced by adding and removing different network configurations. The last column ‘Dif’ stands for the difference in AIC value compared with the best model, model 11.

The intercept of model 1 is $-3.28$ which is approximately the same for the other models. This number matches roughly with the theory, the probability at a link is $\pi \approx 0.037$ which gives an estimation for the intercept: $\log \frac{1}{\pi} = -3.25$. In Table 5.1 are the different models with their corresponding network configurations. The model which performs the best is the model with network configurations: number of edges, number of two-paths, number of added 2-stars and number of added triangles. The number of edges is grouped into the different sets and interactions of those sets. The number of two-paths is positive correlated, the number of added 2-stars and the number of added triangles are negative...
correlated. Meaning that the sets in the data are linked with each other not only in one step but also with a detour of two steps. Furthermore, the genes have less complex relations with other genes than expected by chance.

5.7 Discussion

The data analysis with GLM obtained by using glmnet gives results that are in line with the results giving by NEAT. However, the two models do not completely correspond with each other. A possible reason is the difference approach of the two methods. If one coefficient are added in glmnet than will this influence all the other coefficients. While, in NEAT are all the coefficients mainly apart checked for enrichments.

Furthermore, the data has a remarkable distribution of the p-values. This is already visible in Figure 5.7 this positive skewness of the p-values remained when we changed the range of the p-values. Figure 5.8 shows a P-P plot of the p-values clarifying that the p-values are not uniform distributed. A possible consequence of this skewness could be that it is more difficult to find consensus which interactions to choose as being the most significant. Apparently there are quite a lot coefficients being interpreted as being significant by NEAT.

![P-P plot of p-values](image)

Figure 5.8: A P-P plot showing that the p-values are not uniform distributed.
Chapter 6

Extended Generalized Linear Models

There are more complex models which extends the generalized linear model. Two of these approaches will shortly be discussed in this chapter.

6.1 Generalized linear mixed-effects model

A generalized linear mixed model (GLMM) extends the GLM. GLMMs can account for the dependence by adding random effects to the model. The resulting model is a mixed model with the common fixed effects together with the random effects. Remember the model with Equation 4.3, where we assume that the observations are conditionally independent:

$$g(E[z_i|x_i, \beta]) = \eta_i = \sum_{k=0}^{p} x_{ik} \beta_k.$$  

The GLMM then becomes

$$g(E[z_i|x_i, \beta]) = \eta_i = \sum_{k=0}^{p} x_{ik} \beta_k + \gamma_i,$$

where $\gamma_i$ is a random effect. The network observations are now modelled as conditionally independent, given the random effects terms. An approach to model the random effect is to use random intercept[1]

$$\gamma_i = a_s + b_r + \epsilon_i,$$

where $s$ stands for sender and $r$ stands for receiver. So, $a_s$ and $b_r$ represents the sender and receiver effects for the two nodes who are connected with each other in row $i$. The distribution of these effects

[1]This is for a directed network. For an undirected network we will delete the sender or receiver intercept.
are normally distributed with mean zero and variance could be estimated with the data.

Note the analogy of Equation 5.2 to GLMM for binary data. In particular in Agresti [11] where they choose $\eta^T \Delta (s_2(y))_i$ as the random effects.

An advantage of GLMM is that there are quite some different types of random effects models [10]. This could therefore provide a rich class of new models for dependent network data.

### 6.2 Marginalized exponential random graph models

In Section 5.1 we have already showed the similarity between ERGMs and GLMs. In this Section, we will extends this even more, following the result of Suesse [32]. He introduced marginalized ERGMs (MERGMs). This is a combination of GLMs and ERGMs.

Suesse uses the method exponential tilting [32]. The variable $Y$ are tilted with density $f(y)$ with tilting parameter $\theta$, which gives the tilted distribution

$$f_{\varphi}(y) = \frac{\exp[\varphi y]}{z(\varphi, Y)} f(y), \quad y \in Y,$$

where

$$z(\varphi, Y) = \sum_{y \in Y} \exp[\varphi y].$$

We apply now above transformation of the ERGM from Equation 3.1. After this transformation will the ERGM marginally follow the GLM given by Equation 5.3. The density of this ERGM has the form

$$P(Y = y|\varphi, \theta) = \exp[\varphi Ty + \theta^T \log(s(\theta, Y)) - \log(z(\theta, \varphi, Y))]$$

with the normalizing factor

$$z(\theta, \varphi, Y) = \sum_{y \in Y} \exp[\varphi^T y + \theta^T s(y)].$$

The MERGM is specified by Equation 5.3 and 6.1. The parameters in the vector $\varphi$ are chosen such that the ERGM match marginally with the GLM defined by 5.1.
Conclusion

The purpose of this thesis was to investigate a new way to model a biological network. The more common approach is to use Exponential Random Graph Models (ERGMs). The pseudolikelihood is used as an approximation of the number of possible links within the network. The pseudolikelihood can be modeled with Generalized Linear Models (GLMs). ERGMs and GLMs have extensively been clarified in this thesis. To model a GLM on a network we used the data of a gene regulatory network derived by Vinciotti et al. The derived approach by GLM is implemented in R and is used to analyse the gene regulatory network. The results with the model with GLMs is compared with an already existing model: Network Enrichment Analysis Test (NEAT). The data analysis with GLMs gives results that are partly in line with the results achieved by NEAT.
Bibliography


Appendix A

Derivations

Likelihood

After taking the exponential by the link function we get
\[
\frac{\pi_i}{1 - \pi_i} = \exp \left[ \sum_{k=0}^{p} x_{ik} \hat{\beta}_k \right],
\]
(A.2)
or solving for \( \pi_i \)
\[
\pi_i = \frac{\exp \left[ \sum_{k=0}^{p} x_{ik} \hat{\beta}_k \right]}{1 + \exp \left[ \sum_{k=0}^{p} x_{ik} \hat{\beta}_k \right]}.
\]
(A.3)

Now we substitute Equation A.2 and Equation A.3 in Equation 4.5 to get
\[
L(\beta | z) \sim \prod_{i=1}^{N} \left( \exp \left[ \sum_{k=0}^{p} x_{ik} \hat{\beta}_k \right] \right)^{z_i} \left( 1 - \frac{\exp \left[ \sum_{k=0}^{p} x_{ik} \hat{\beta}_k \right]}{1 + \exp \left[ \sum_{k=0}^{p} x_{ik} \hat{\beta}_k \right]} \right)^{-n_i}.
\]
(A.4)

After replacing the 1 with
\[
\frac{1 + \exp \left[ \sum_{k=0}^{p} x_{ik} \hat{\beta}_k \right]}{1 + \exp \left[ \sum_{k=0}^{p} x_{ik} \hat{\beta}_k \right]},
\]
we can rewrite Equation A.4 now as
\[
L(\beta | z) \sim \prod_{i=1}^{N} \left( \exp \left[ z_i \sum_{k=0}^{p} x_{ik} \hat{\beta}_k \right] \right) \left( 1 + \exp \left[ \sum_{k=0}^{p} x_{ik} \hat{\beta}_k \right] \right)^{-n_i}.
\]

Least squares estimator

Differentiating RSS with respect to \( \beta \) gives
\[
\frac{\partial \text{RSS}}{\partial \beta} = -2X^T(z - X\beta).
\]

Setting this first derivative to zero gives
\[
X^T(z - X\beta) = 0,
\]
hence the solution is \( \hat{\beta} = (X^TX)^{-1}X^Tz \).
Expectation of the optimism

\[
w = E_z[\text{op}]
\]

\[
= E_z[\text{Err}_n] - E_z[\overline{\text{Err}}]
\]

\[
= E_z \left[ \frac{1}{M} \sum_{i=1}^{M} E_{Z^0}[L(Z_i^0, \hat{f}(x_i))] \right] - E_z \left[ \frac{1}{M} \sum_{i=1}^{M} L(z_i, \hat{f}(x_i)) \right]
\]

\[
= \frac{1}{M} \sum_{i=1}^{M} E_z E_{Z^0}((Z_i^0 - \hat{z}_i)^2) - E_z[(z_i - \hat{z}_i)^2]
\]

\[
= \frac{1}{M} \sum_{i=1}^{M} E_z E_{Z^0}[(Z_i^0)^2] + E_z E_{Z^0}[\hat{z}_i^2] - 2E_z E_{Z^0}[Z_i^0 \hat{z}_i] - E_z[z_i^2] - E_z[\hat{z}_i^2] + 2E_z[z_i \hat{z}_i]
\]

\[
= \frac{1}{M} \sum_{i=1}^{M} E_z[z_i^2] + E_z[\hat{z}_i^2] - 2E_z[z_i]E_z[\hat{z}_i] - E_z[z_i^2] - E_z[\hat{z}_i^2] + 2E_z[z_i \hat{z}_i]
\]

\[
= \frac{2}{M} \sum_{i=1}^{M} E_z[z_i \hat{z}_i] - E_z[z_i]E_z[\hat{z}_i]
\]

\[
= \frac{2}{M} \sum_{i=1}^{M} E_z[z_i \hat{z}_i] - z_i E_z[\hat{z}_i] - E_z[z_i] \hat{z}_i + E_z[z_i]E_z[\hat{z}_i]
\]

\[
= \frac{2}{M} \sum_{i=1}^{M} E_z[(\hat{z}_i - E_z[\hat{z}_i])(z_i - E_z[z_i])]
\]

\[
= \frac{2}{M} \sum_{i=1}^{M} \text{Cov}(\hat{z}_i, z_i)
\]
Appendix B

Main R code

```r
set.seed(1)
install.packages("curl")
install.packages("glmnet")
install.packages("network")
install.packages("ergm")
install.packages("neat")
library(curl)
library(neat)
library(ergm)
library(network)
library(glmnet)

mydata = read.csv(file.choose(),header=FALSE) #Loading the data
matrixnetwork=as.matrix(mydata)
matrixnetworknonames1=matrixnetwork[,-1]
matrixnetworknonames=matrixnetworknonames1[,,-1] #the network without the names
matrixnetwork1=matrixnetwork #a copy of the network

#Some calculations are quite time intensive. Therefore did I saved those calculations.

load("C:\Users\harme\Desktop\masterscriptie2edata1.28b\Laatsteversie\network2edata.Rdata")
load("C:\Users\harme\Desktop\masterscriptie2edata1.28b\Laatsteversie\networkingroepen2e.Rdata")
load("C:\Users\harme\Desktop\masterscriptie2edata1.28b\Laatsteversie\testpnea.Rdata")
load("C:\Users\harme\Desktop\masterscriptie2edata1.28b\Laatsteversie\FGSgroepen.Rdata")
load("C:\Users\harme\Desktop\masterscriptie2edata1.28b\Laatsteversie\networkingroepenmatrix2edata.Rdata")

Laatsteversie \networkingroepenmatrix2edata.Rdata)"
load("C:\Users\harme\Desktop\masterscriptie2edata1.28b\Laatsteversie\fsgnamen.Rdata")
load("C:\Users\harme\Desktop\masterscriptie2edata1.28b\Laatsteversie\matrixO.Rdata")
load("C:\Users\harme\Desktop\masterscriptie2edata1.28b\Laatsteversie\Designmatrix1small.Rdata")
load("C:\Users\harme\Desktop\masterscriptie2edata1.28b\Laatsteversie\Designmatrix2small.Rdata")
```

56
namen=names(FGS.list)  # names of the pathways

Twopaths  # (so between group A and B there are no links between node s to node s which are in both groups)

Nafs

Naf  # (collumnames of the adjacency matrix)

g.names<−as.vector(as.character(mydata[2:1436,1]))  # names of the genes

# (collumnnames of the adjacency matrix)

Nafs<−o2  # number of links between the 61 groups

Nafs<−o2s  # number of links between the 61 groups without selfloops

# (so between group A and B there are no links between node s to node s which are in both groups)

Twopaths<−o3  # number of twopaths between the 61 groups

namen=names(FGS.list)  # names of the pathways
k = matrix(data = NA, nrow = length(FGS.list) * length(FGS.list), 1)
for (i in 1:length(FGS.list)) {
    k[((length(FGS.list) * (i - 1)) + 1):(length(FGS.list) * (i)),] <- c(rep(namen[i], length(FGS.list)))
}
l = c(rep(namen, length(FGS.list)))
m = cbind(k, l)  # A vector with the names of all the possible links between the pathways

k2 = matrix(data = NA, nrow = length(FGS.list) * length(FGS.list), 1)
for (i in 1:length(FGS.list)) {
    k2[((length(FGS.list) * (i - 1)) + 1):(length(FGS.list) * (i)),] <- c(rep(i, length(FGS.list)))
}
l2 = c(rep(1:length(FGS.list), length(FGS.list)))
m2 = cbind(k2, l2)  # A vector with all the possible links between the pathways (now with numbers)

q = cbind(m, Naf)
qs = cbind(m, Nafs)

# af is the maximum number of links between the pathways
af = matrix(0, nrow = length(FGS.list) * length(FGS.list), ncol = 1)
for (k in 1:length(o2s)) {
    together <- c(FGS.list[[m[k, 1]]], FGS.list[[m[k, 2]]])
    samegenes <- together[duplicated(together) | duplicated(together, fromLast = TRUE)]
    nrsamegenes <- length(samegenes) / 2
    af[k] <- (length(FGS.list[[m[k, 1]]]) * length(FGS.list[[m[k, 2]]])) - ((nrsamegenes * (nrsamegenes - 1)) / 2)
}

# afs is the maximum number of links between the pathways when there are no selfloops
afs = matrix(data = NA, nrow = length(FGS.list) * length(FGS.list), 1)
for (k in 1:length(o2s)) {
    ki <- m2[k, 2]
    kj <- m2[k, 1]
    ni <- length(FGS.list[[ki]])
    nj <- length(FGS.list[[kj]])
    if (ki != kj) {
        58
double <- c(FGS.list[[kj]], FGS.list[[ki]])
whichdouble <- double[duplicated(double) | duplicated(double, fromLast = TRUE)]
length1 <- (length(whichdouble) / 2)
afs[k] <- af[k] - length1
}
else {
  afs[k] <- af[k] - ni
}
}
completematrix <- cbind(q, af)
completematrixs <- cbind(qs, afs)

# Changing completematrix into a matrix with only n(n-1)/2 rows, by leaving out possibility
# to go to the same group and divided by two because the data is undirected.
# So if we have a row from group 1 to group 2 than we excluded the row from 2 to 1.
completematrix2 <- completematrix
for (i in 3721:1) {
  if (m2[i, 1] >= m2[i, 2]) {completematrix2 <- completematrix2[-i,]}
}
af3 <- as.numeric(completematrix2[, 4])
Naf3 <- as.numeric(completematrix2[, 3])
l3 <- completematrix2[, 1]
k3 <- completematrix2[, 2]

# Now for the case without selfloops
completematrix2s <- completematrixs
for (i in 3721:1) {
  if (m2[i, 1] >= m2[i, 2]) {completematrix2s <- completematrix2s[-i,]}
}
af3s <- as.numeric(completematrix2s[, 4])
Naf3s <- as.numeric(completematrix2s[, 3])
l3s <- completematrix2s[, 1]
k3s <- completematrix2s[, 2]
#now \( (n\times n)/2 \) rows, so with going to the same group
completematrix2 <- completematrix
for (i in 3721:1) {
  if ( m2[i,1] > m2[i,2] ) {completematrix2 <- completematrix2[-i,]}
}

af4 = as.numeric(completematrix2[,4])
Naf4 = as.numeric(completematrix2[,3])
l4 = completematrix2[,1]
k4 = completematrix2[,2]

#Designmatrix1small is a designmatrix with only from 1 group
#So if we have a row from group 1 to group 2 than we excluded the row from 2 to 1.
pen.fac1 <- rep(1,(dim(Designmatrix1small)[2]))
pen.fac1[1:62] <- 0
pen.fac2 <- rep(1,(dim(Designmatrix2small)[2]))
pen.fac2[1:62] <- 0
pen.fac3 <- rep(1,(dim(Designmatrix3small)[2]))
pen.fac3[1:62] <- 0
pen.fac4 <- rep(1,(dim(Designmatrix4small)[2]))
pen.fac4[1:62] <- 0
pen.fac5 <- rep(1,(dim(Designmatrix5small)[2]))
pen.fac5[1:62] <- 0
pen.fac5a <- rep(1,(dim(Designmatrix5asmall)[2]))
pen.fac5a[1:62] <- 0
pen.fac5b <- rep(1,(dim(Designmatrix5bsmall)[2]))
pen.fac5b[1:62] <- 0
pen.fac5c <- rep(1,(dim(Designmatrix5csmall)[2]))
pen.fac5c[1:62] <- 0
pen.fac5d <- rep(1,(dim(Designmatrix5dsmall)[2]))
pen.fac5d[1:62] <- 0
pen.fac5e <- rep(1,(dim(Designmatrix5esmall)[2]))
pen.fac5e[1:62] <- 0
pen.fac6 <- rep(1,(dim(Designmatrix6small)[2]))
pen.fac6[1:62] <- 0
pen.fac7 <- rep(1,(dim(Designmatrix7small)[2]))

60
# Lambda here under are achieved by performing:

```r
set.seed(1)
test <- cv.glmnet(x=Designmatrix6small, y=cbind(af3-Naf3,Naf3), family="binomial",
  nfolds=5, alpha=1)
test$lambda.min
```

# Similar for other x's and y's.

# To see the results, I put it temporary in a text document;

```r
sink("vergelijken123g4.txt")
```

```r
lambda<-8.086613e-05
desresultaat1=glmnet(Designmatrix1small,cbind(af3-Naf3,Naf3),family="binomial",penalty.factor=pen.fac1)
u<-coef(desresultaat1,s=deslambda)
y=which( u!=0, arr.ind=TRUE)
P=matrix(0, nrow = dim(y)[1], ncol = 2)
for(i in 1:dim(y)[1]){
  number<-which( u!=0, arr.ind=TRUE)[i,1]
P[i,1] <- rownames(u)[number]
P[i,2] <- summary(u)[i,3]
}
P1<-P[which(P[,2]>0),] # overenrichments
```

```r
lambda<-3.044558e-05
desresultaat1=glmnet(Designmatrix2small,cbind(af3-Naf3,Naf3),family="binomial",penalty.factor=pen.fac2)
u<-coef(desresultaat1,s=deslambda)
y=which( u!=0, arr.ind=TRUE)
P=matrix(0, nrow = dim(y)[1], ncol = 2)
for(i in 1:dim(y)[1]){
  number<-which( u!=0, arr.ind=TRUE)[i,1]
P[i,1] <- rownames(u)[number]
```

61
\( P[i,2] <- \text{summary}(u)[i,3] \)
}
\( P2 <- P[\text{which}(P[,2] > 0),] \) #overenrichments
\( P2 \)

\( \lambda <- 8.086613e-05 \)
\( \text{resultaat1} = \text{glmnet}(\text{Designmatrix3small, cbind(af3-Naf3,Naf3)}, \text{family="binomial"}, \text{penalty.factor=pen.fac3}) \)
\( u <- \text{coef}(\text{resultaat1}, s=\lambda) \)
\( y = \text{which}(u != 0, \text{arr.ind=TRUE}) \)
\( P = \text{matrix}(0, \text{nrow = dim}(y)[1], \text{ncol = 2}) \)
\text{for (i in 1:dim}(y)[1]) {
\text{number} <- \text{which}(u != 0, \text{arr.ind=TRUE})[i,1]
\text{P}[i,1] <- \text{rownames}(u)[\text{number}]
\text{P}[i,2] <- \text{summary}(u)[i,3]
}
\( P3 <- P[\text{which}(P[,2] > 0),] \) #overenrichments
\( P3 \)

\( \lambda <- 8.086613e-05 \)
\( \text{resultaat1} = \text{glmnet}(\text{Designmatrix4small, cbind(af3-Naf3,Naf3)}, \text{family="binomial"}, \text{penalty.factor=pen.fac4}) \)
\( u <- \text{coef}(\text{resultaat1}, s=\lambda) \)
\( y = \text{which}(u != 0, \text{arr.ind=TRUE}) \)
\( P = \text{matrix}(0, \text{nrow = dim}(y)[1], \text{ncol = 2}) \)
\text{for (i in 1:dim}(y)[1]) {
\text{number} <- \text{which}(u != 0, \text{arr.ind=TRUE})[i,1]
\text{P}[i,1] <- \text{rownames}(u)[\text{number}]
\text{P}[i,2] <- \text{summary}(u)[i,3]
}
\( P3 <- P[\text{which}(P[,2] > 0),] \) #overenrichments
\( P3 \)

\( \lambda <- 3.500502e-05 \)
\( \text{resultaat1} = \text{glmnet}(\text{Designmatrix5small, cbind(af3-Naf3,Naf3)}, \text{family="binomial"}, \text{penalty.factor=pen.fac5}) \)
\( u <- \text{coef}(\text{resultaat1}, s=\lambda) \)
\( y = \text{which}(u != 0, \text{arr.ind=TRUE}) \)

62
```r
P <- matrix(0, nrow = dim(y)[1], ncol = 2)
for (i in 1:dim(y)[1]) {
    number <- which(u != 0, arr.ind=TRUE)[i,1]
    P[i,1] <- rownames(u)[number]
    P[i,2] <- summary(u)[i,3]
}
P5 <- P[which(P[,2] > 0),] # overenrichments
P5

lambda <- 4.216367e-05
resultaat1 <- glmnet(Designmatrix5asmall, cbind(af3 - Naf3, Naf3), family = "binomial", penalty.factor = pen.fac5a)
u <- coef(resultaat1, s = lambda)
y <- which(u != 0, arr.ind=TRUE)
P <- matrix(0, nrow = dim(y)[1], ncol = 2)
for (i in 1:dim(y)[1]) {
    number <- which(u != 0, arr.ind=TRUE)[i,1]
    P[i,1] <- rownames(u)[number]
    P[i,2] <- summary(u)[i,3]
}
P5a <- P[which(P[,2] > 0),] # overenrichments
P5a

lambda <- 8.086613e-05
resultaat1 <- glmnet(Designmatrix5bsmall, cbind(af3 - Naf3, Naf3), family = "binomial", penalty.factor = pen.fac5b)
u <- coef(resultaat1, s = lambda)
y <- which(u != 0, arr.ind=TRUE)
P <- matrix(0, nrow = dim(y)[1], ncol = 2)
for (i in 1:dim(y)[1]) {
    number <- which(u != 0, arr.ind=TRUE)[i,1]
    P[i,1] <- rownames(u)[number]
    P[i,2] <- summary(u)[i,3]
}
P5b <- P[which(P[,2] > 0),] # overenrichments
P5b
```

63
\[ \lambda < -2.648001 \times 10^{-5} \]
resultaat1 = glmnet(Designmatrix5csmall, cbind(af3 - Naf3, Naf3), family = "binomial", penalty.factor = pen.fac5c)
\[ u = -coef(resultaat1, s = \lambda) \]
y = which( u != 0, arr.ind = TRUE)
P = matrix(0, nrow = dim(y)[1], ncol = 2)
for (i in 1:dim(y)[1]) {
    number = which( u != 0, arr.ind = TRUE)[i, 1]
    P[i, 1] = rownames(u)[number]
    P[i, 2] = summary(u)[i, 3]
}
P5c = P[which(P[,2] > 0),]
# overenrichments
P5c

\[ \lambda < -3.500502 \times 10^{-5} \]
resultaat1 = glmnet(Designmatrix5dsmall, cbind(af3 - Naf3, Naf3), family = "binomial", penalty.factor = pen.fac5d)
u = -coef(resultaat1, s = \lambda)
y = which( u != 0, arr.ind = TRUE)
P = matrix(0, nrow = dim(y)[1], ncol = 2)
for (i in 1:dim(y)[1]) {
    number = which( u != 0, arr.ind = TRUE)[i, 1]
    P[i, 1] = rownames(u)[number]
    P[i, 2] = summary(u)[i, 3]
}
P5d = P[which(P[,2] > 0),]
# overenrichments
P5d

\[ \lambda < -3.500502 \times 10^{-5} \]
resultaat1 = glmnet(Designmatrix5esmall, cbind(af3 - Naf3, Naf3), family = "binomial", penalty.factor = pen.fac5e)
u = -coef(resultaat1, s = \lambda)
y = which( u != 0, arr.ind = TRUE)
P = matrix(0, nrow = dim(y)[1], ncol = 2)
for (i in 1:dim(y)[1]) {
    number = which( u != 0, arr.ind = TRUE)[i, 1]
    P[i, 1] = rownames(u)[number]
    P[i, 2] = summary(u)[i, 3]
}

\begin{verbatim}
resultaat1=glmnet(Designmatrix6small,cbind(af3−Naf3,Naf3),family="binomial",penalty.factor=pen.fac6)

\end{verbatim}
for (i in 1:dim(y)[1]) {
    number <- which(u != 0, arr.ind = TRUE)[i, 1]
    P[i, 1] <- rownames(u)[number]
    P[i, 2] <- summary(u)[i, 3]
}
P8 <- P[which(P[, 2] > 0), ]  # overenrichments
P8

sink()

# checking intercept: log((sum(Naf3)/sum(af3))/(1−(sum(Naf3)/sum(af3)))) gives −3.248667
boxplot(testen$pvalue ~ I(1*(u[1, 1] != 0)))
boxplot(testen$pvalue ~ I(1*(u[1, 1] > 0)))

# Calculating the AIC values
# Filling in choose.designmatrix= ... For example designmatrix1small
# After calculating 'resultaat1' for corresponding lambda fill in:
Choose.designmatrix=Designmatrix6small
P = P6
le <- (dim(Choose.designmatrix)[2]) + 1
cof <- coef(resultaat1, s = lambda)[1:le]
eta1 = ((Choose.designmatrix[, -1] %*% cof[3:le]) + cof[1])
loglik = 0
for (j in 1:length(Naf3)) {
    loglik <- -Naf3[j] * eta1[j] - af3[j] * (log(1 + exp(eta1[j]))) + loglik
}
AIC = -2*(loglik - dim(P)[1])
AIC

NonNAindex <- which(is.na(s))  # s is computed in 'network2edata'
firstNonNA <- min(NonNAindex)
s2 <- s[1:firstNonNA,]
s3 <- s2 - 1
a <- g.names[s3[, 1]]
b <- g.names[s3[,2]]
networknames <- cbind(a, b)
networknames <- networknames[1:58892,]

# Performing NEAT
testen = neat(alist = FGS.list, blist = NULL, network = networknames,
nettype = 'undirected', nodes = g.names, alpha = 0.01)
save(testen, file = 'testpnea.Rdata')
# the first 10 results:
print(testen)

over = testen[testen$conclusion == 'Overenrichment',]
mirkovp <- print(over, nrow = 'ALL') # display overenrichments
dim(print(over, nrow = 'ALL'))
under = testen[testen$conclusion == 'Underenrichment',]
dim(print(under, nrow = 'ALL')) # display underenrichments
summary(testen)

# So at 1% level there are 452 enrichments where 139 are underenrichment and 313 are overenrichment.
plot(testen)

# Now checking similarity with mirkov
hlist <- matrix(data = 0, nrow = dim(P1)[1], 1)
mirlength <- dim(mirkovp)[1]
harlength <- dim(P1)[1]
hlist <- matrix(data = 0, nrow = dim(mirkovp)[1], 1)
for (j in 1:harlength) {
  for (i in 1:mirlength) {
    if (paste("l", mirkovp[i, 2], "k", mirkovp[i, 1], sep = "") == P1[j, 1] |
      paste("l", mirkovp[i, 1], "k", mirkovp[i, 2], sep = "") == P1[j, 1]) {
      hlist[j] <- i
    }
  }
}
pvaluesmirkov <- mirkovp[, 5]
hlistnonzero <- hlist[which(hlist == 0)]
hist(pvaluesmirkov, col = rgb(0, 0.1, 3/4), main = paste("Histogram of", paste("p-values")),

67
xlab = paste("P-values of the coefficients using NEAT"))

hist(pvaluesmirkov[hlistnonzero], col=rgb(0,1,1,3/4), add=T)

# Performing cross-validation with cv.glmnet
install.packages("C:\zoom_2.0.4.tar.gz", repos = NULL, type="source")
library(zoom) # Invoke the Library
# Call plot

test = cv.glmnet(x=Designmatrix1smallmirkov, y=cbind(af3-Naf3,Naf3), family="binomial",
nfolds=5, alpha=1)
plot(test, xlim=c(-9.504, -3.992), ylim=c(0.022, 0.026))
plot(test)
zm()
test$lambda.min
coef(test, s = "lambda.min")
Time consuming R codes

#These R codes did I run only once.
#From Veronica, R script that she uses to get the data.
#only first time on a computer:
source("https://bioconductor.org/biocLite.R")
biocLite("KEGGREST")

#Library KEGGREST
library(KEGGREST)
a <- keggLink("pathway", "hsa")
a <- unique(a)

#remove pathways which contain none of the 1435 genes
#only report the genes in the pathways that are in the networks.
g.names <- as.vector(as.character(mydata[2:1436,1])) # names of the genes
FGS.list <- list()
fgs.names <- NULL
h <- 0
for (i in 1:length(a)){
  b <- keggGet(a[i])[1]$GENE
  if (length(b)>0){
    b <- unlist(strsplit(as.character(b[seq(2,length(b),2)]),","))
    b.names <- g.names[g.names %in% b[seq(1,length(b),2)]]
    if (length(b.names)>0){
      h <- h+1
      FGS.list[[h]] <- b.names
      fgs.names <- c(fgs.names,strsplit(keggGet(a[i])[[1]]$NAME," - Homo")[[1]][1])
    }
  }
}
names(FGS.list) <- fgs.names
length(fgs.names)
#269 pathways
length(unique(FGS.list))
#262
# Only considers unique pathways and pathways larger than 20.

idx <- NULL
for (i in 1:(length(FGS.list) - 1))
  for (j in (i + 1):length(FGS.list))
    {
      if (length(FGS.list[[i]]) == length(FGS.list[[j]]) )
        {
          if ( all (FGS.list[[i]]==FGS.list[[j]]) )
            idx <- c(idx, i)
        }
      if (length(FGS.list[[i]]) < 20)
        idx <- c(idx, i)
    }
  if (length(FGS.list[length(FGS.list)]) < 20)
    idx <- c(idx, length(FGS.list))

FGS.list <- FGS.list[−unique(idx)]
fgs.names <- fgs.names[−unique(idx)]
length(fgs.names)
# 61 pathways left

save(FGS.list, file = 'FGSgroepen.Rdata')
save(fgs.names, file = 'fgsnamen.Rdata')

# Counter for the number of links between the groups
o2=matrix(data=NA, nrow = length(FGS.list) * length(FGS.list), 1)
for (k in 1:length(o2)) {
  counter <- 0
  ki <- m2[k, 2]
  kj <- m2[k, 1]
  ni <- length(FGS.list[[ki]])
}
nj <- length(FGS.list[[kj]])

for (j in 1:nj) {
  groupsinnumbers <- NULL
  for (i in 1:ni) {
    groupsinnumbers <- c(groupsinnumbers, which(FGS.list[[ki]][[i]] == g.names))
  }
  together <- c(which(matrixnetworknonames[which(FGS.list[[kj]][[j]] == g.names),] == "1"), groupsinnumbers)
  samegenes <- together[duplicated(together) | duplicated(together, fromLast = TRUE)]
  counter <- (length(samegenes)/2) + counter
}

if (ki != kj) {
  counter1 <- 0
  double <- c(FGS.list[[kj]], FGS.list[[ki]])
  whichdouble <- double[duplicated(double) | duplicated(double, fromLast = TRUE)]
  length1 <- (length(whichdouble)/2)
  checkforlinks <- whichdouble[1:(length1)]

  for (m in 1:length1) {
    groupsinnumbers <- NULL
    for (l in 1:length1) {
      groupsinnumbers <- c(groupsinnumbers, which(checkforlinks[l] == g.names))
    }
    together <- c(which(matrixnetworknonames[which(checkforlinks[m] == g.names),] == "1"), groupsinnumbers)
    samegenes <- together[duplicated(together) | duplicated(together, fromLast = TRUE)]
    counter1 <- (length(samegenes)/2) + counter1
  }
  o2[k] <- counter - (counter1 - length1)
} else {
  o2[k] <- (counter - length(FGS.list[[kj]]))/2 + length(FGS.list[[kj]])
  # for two the same groups we must subtract the links which we counted double.
}

print(k)
save(o2, file = 'o2.Rdata')

# now counting the links without selfloops,
# if you decide to use this one then also use corresponing afs
o2s=matrix(data=NA, nrow= length(FGS.list)* length(FGS.list),1)
for (k in 1:length(o2s)) {
    ki <- m2[k,2]
    kj <- m2[k,1]
    ni <- length(FGS.list[[ki]])
    nj <- length(FGS.list[[kj]])
    if (ki != kj) {
        double <- c(FGS.list[[kj]], FGS.list[[ki]])
        whichdouble <- double[duplicated(double) | duplicated(double, fromLast=TRUE)]
        length1 <- (length(whichdouble)/2)
        o2s[k] <- o2[k] - length1
    } else {
        o2s[k] <- o2[k] - ni
    }
}
save(o2s, file = 'o2s.Rdata')

# Counter for the number of twopaths between every group.
matrixdiag0 <- matrix(networknonames)
    diag(matrixdiag0) <- 0
o3 <- matrix(data=NA, nrow= length(FGS.list)* length(FGS.list),1)
for (k in 1:length(o3)) {
    counter <- 0
    ki <- m2[k,2]
    kj <- m2[k,1]
    ni <- length(FGS.list[[ki]])
    nj <- length(FGS.list[[kj]])
    for (j in 1:nj) {

for (i in 1:ni) {
    comparing1 <- which(FGS.list[[kj]][[j]] == g.names)
    comparing2 <- which(FGS.list[[ki]][[i]] == g.names)
    if (comparing1 != comparing2) {
        comparing1links <- which(matrixdiag0[which(FGS.list[[kj]][[j]] == g.names),] == "1")
        comparing2links <- which(matrixdiag0[which(FGS.list[[ki]][[i]] == g.names),] == "1")
        together <- c(comparing1links, comparing2links)
        samegenes <-一起[duplicated(together) | duplicated(together, fromLast=TRUE)]
        counter <- (length(samegenes)/2) + counter
    } 
} 

o3[k] <- counter
print(k)

save(o3, file = 'o3.Rdata')

# Counter for the number of added k-stars. (2-stars till 10-stars)
matrixdiag0 <- matrixnetworknonames
diag(matrixdiag0) <- 0
star2 <- matrix(data=NA, nrow = length(FGS.list) * length(FGS.list), 1)
star3 <- matrix(data=NA, nrow = length(FGS.list) * length(FGS.list), 1)
star4 <- matrix(data=NA, nrow = length(FGS.list) * length(FGS.list), 1)
star5 <- matrix(data=NA, nrow = length(FGS.list) * length(FGS.list), 1)
star6 <- matrix(data=NA, nrow = length(FGS.list) * length(FGS.list), 1)
star7 <- matrix(data=NA, nrow = length(FGS.list) * length(FGS.list), 1)
star8 <- matrix(data=NA, nrow = length(FGS.list) * length(FGS.list), 1)
star9 <- matrix(data=NA, nrow = length(FGS.list) * length(FGS.list), 1)
star10 <- matrix(data=NA, nrow = length(FGS.list) * length(FGS.list), 1)
for (k in 1:length(star2)) {
    twostarscounter <- 0
    threestarscounter <- 0
    fourstarscounter <- 0
    fivestarscounter <- 0
    }
sixstarscounter <- 0
sevenstarscounter <- 0
eightstarscounter <- 0
ninestarscounter <- 0
tenstarscounter <- 0
ki <- m2[k,2]
kJ <- m2[k,1]
ni <- length(FGS.list[[ki]])
nJ <- length(FGS.list[[kJ]])

for (j in 1:nj){
    for (i in 1:ni){
        protein1 <- which(FGS.list[[kf]]][[j]]=g.names)
        protein2 <- which(FGS.list[[ki]][[i]]=g.names)
        if (protein1!=protein2) {
            linksprotein1 <- sum((matrixdiag0[which(FGS.list[[kj]][[j]]=g.names),]=="1")*1)
            linksprotein2 <- sum((matrixdiag0[which(FGS.list[[ki]][[i]]=g.names),]=="1")*1)
            if (matrixdiag0[protein1,protein2]==1) {
                linksprotein1 <- linksprotein1 - 1
                linksprotein2 <- linksprotein2 - 1
            }
            twostarscounter <- twostarscounter + choose(linksprotein1,2) + choose(linksprotein2,2)
            threestarscounter <- threestarscounter + choose(linksprotein1,3) + choose(linksprotein2,3)
            fourstarscounter <- fourstarscounter + choose(linksprotein1,4) + choose(linksprotein2,4)
            fivestarscounter <- fivestarscounter + choose(linksprotein1,5) + choose(linksprotein2,5)
            sixstarscounter <- sixstarscounter + choose(linksprotein1,6) + choose(linksprotein2,6)
            sevenstarscounter <- sevenstarscounter + choose(linksprotein1,7) + choose(linksprotein2,7)
            eightstarscounter <- eightstarscounter + choose(linksprotein1,8) + choose(linksprotein2,8)
            ninestarscounter <- ninestarscounter + choose(linksprotein1,9) + choose(linksprotein2,9)
            tenstarscounter <- tenstarscounter + choose(linksprotein1,10) + choose(linksprotein2,10)
        }
    }
}

star2[k]<-twostarscounter
star3[k]<-threestarscounter
star4[k]<-fourstarscounter
star5[k]<-fivestarscounter
star6[k] <- sixstarscounter
star7[k] <- sevenstarscounter
star8[k] <- eightstarscounter
star9[k] <- ninestarscounter
star10[k] <- tenstarscounter
print(k)
}
save(star2, file = 'star2.Rdata')
save(star3, file = 'star3.Rdata')
save(star4, file = 'star4.Rdata')
save(star5, file = 'star5.Rdata')
save(star6, file = 'star6.Rdata')
save(star7, file = 'star7.Rdata')
save(star8, file = 'star8.Rdata')
save(star9, file = 'star9.Rdata')
save(star10, file = 'star10.Rdata')

#tempor is a counter. How much triangles a certain node an angular point from is.
matrixdiag0 <- matrixnetworknonames
diag(matrixdiag0) <- 0
tempor <- matrix(data=NA, nrow= dim(matrixdiag0)[1], 1)
for (k in 1:dim(matrixdiag0)[1]) {
    countertri1 <- 0
    if (length(which(matrixdiag0[k,] == "1")) > 0) {
        length2 <- length(which(matrixdiag0[k,] == "1"))
        for (s in 1:length2) {
            for (t in 1:length2) {
                onecorzero1 <- 0
                if (t != s) {
                    onecorzero1 <- if (matrixdiag0[which(matrixdiag0[k,] == "1")][t],
                                    which(matrixdiag0[k,] == "1")[s] == 1) {1} else {0}
                }
            }
        }
    }
}
if (length(which(matrixdiag0[k,] == "1")) > 0) {tempor[k] <- countertri1 / 2}
}

# Counter for the number of added triangles
matrixdiag0 <- matrixnetworknonames
diag(matrixdiag0) <- 0
triang <- matrix(data = NA, nrow = length(FGS.list) * length(FGS.list), 1)
for (k in 1:length(triang)) {
  triangcounter3 <- 0
  ki <- m2[k, 2]
  kj <- m2[k, 1]
  ni <- length(FGS.list[[ki]])
  nj <- length(FGS.list[[kj]])
  for (j in 1:nj) {
    triangcounter1 <- 0
    triangcounter2 <- 0
    for (i in 1:ni) {
      protein1 <- which(FGS.list[[kj]][[j]] == g.names)
      protein2 <- which(FGS.list[[ki]][[i]] == g.names)
      if (protein1 != protein2) {
        length1 <- 0
        if (matrixdiag0[protein1, protein2] == "1") {
          double <- c(which(matrixdiag0[protein1, ] == "1"), which(matrixdiag0[protein2, ] == "1"))
          whichdouble <- double[duplicated(double) | duplicated(double, fromLast = TRUE)]
          length1 <- (length(whichdouble) / 2)
          triangcounter1 <- tempor[protein1] + tempor[protein2] - 2 * length1
          # length1 are the number of triangles with a consist of a line between protein1 and protein2.
        } else {triangcounter1 <- tempor[protein1] + tempor[protein2]}
      }
      triangcounter2 <- triangcounter2 + triangcounter1
    }
  }

76
triangcounter3 <- triangcounter3 + triangcounter2

for (i in 1435:1) {
  if (counter[i,] == 0) {
    matrixnetwork1 <- matrixnetwork1[-(i+1),]
    matrixnetwork1 <- matrixnetwork1[,-(i+1)]
  }
}

save(matrixnetwork1, file = 'matrixnetwork1.Rdata')

# The start of the calculation of the triangle effect. Making of the matrix O.
# This matrix has as indices the number of common neighbours of the two genes.
matrixnetwork2 <- matrixnetwork1
diag(matrixnetwork2) <- 0
n = dim(matrixnetwork2)[1]
O = matrix(0, nrow = (n-1), ncol = (n-1))
for (j in 2:n) {
  for (i in 2:n) {
    O[(i-1),(j-1)] = sum(!is.na(match(which(matrixnetwork2[i,] == "1", arr.ind = FALSE, useNames = TRUE), which(matrixnetwork2[j,] == "1", arr.ind = FALSE, useNames = TRUE))))
  }
  if (j%%50 == 0) print(j)
}
diag(O) <- 0
```r
save(O, file = 'matrixO.Rdata')
matrixnetwork3 <- matrixnetwork2 # A copy of matrixnetwork2

# making the network in a vector and with numbers instead of names.
# (my computer needed 40 hours for this calculation)
s = matrix(NA, nrow = 1436 * 1436, ncol = 2)
for (j in 1:1436) {
  for (i in 1:1436) {
    NonNAindex <- which(is.na(s))
    firstNonNA <- min(NonNAindex)
    if (j%%100==0) print(j)
    if (mydata[i,j]==1) {
      s[firstNonNA,1] <- i
      s[firstNonNA,2] <- j
    }
  }
}
save(s, file = 'network2edata.Rdata')

# Making the network but now not on node scale but on group scale.
NonNAindex <- which(is.na(s))
firstNonNA <- min(NonNAindex)
s2 <- s[1:firstNonNA,]
s3 <- s2 - 1
a <- g.names[s3[,1]]
b <- g.names[s3[,2]]
networknames <- cbind(a, b)
networknames <- networknames[1:58892,]
network = ynetgenesgroepen
genestypes = names(uniquelist)
k = length(uniquelist)
a = 0
B = matrix(0, nrow = nk, ncol = nk)
```

78
for (i in 1:dim(networknames)[1]) {
  gfrom = network[i,1]
  gto = network[i,2]
  from = which(ll<-(network[i,1]==genesgroups))
  to = which(ll<-(network[i,2]==genesgroups))
  if ((length(from)>0) & (length(to)>0)) {B[from,to] = B[from,to]+1} else {a=1}
}

save(B, file = 'networkinggroepenmatrix2edata.Rdata')

# A smaller designmatrix with less columns. The columns ‘to’ are deleted because ‘from’ and ‘to’
# are the same in undirected network. And after that are double columns deleted.
# Namely, if we have already from group A to B then we delete from B to A,
# because these are the same in undirected network. Also form A to A etc. are deleted.
# With”Alzheimer’s disease” through, constrain arg

namen=names(FGS.list)
k=matrix(data=NA, nrow=length(FGS.list)*length(FGS.list),1)
for (i in 1:length(FGS.list)) {
  k[((length(FGS.list)∗(i−1))+1):(length(FGS.list)*i),] <- c(rep(namen[i],length(FGS.list)))
} l=c(rep(namen,length(FGS.list)))
m=cbind(k,l)
df1 <- data.frame(l,k)
Designmatrixbig1=model.matrix(~ l∗k, data=df1, contrasts.arg=list(l=contrasts(df1$l, contrasts=F), k=contrasts(df1$k, contrasts=F)))
Designmatrixbig1 <- Designmatrixbig1[.−(63:123)]
Designmatrix1small <- Designmatrixbig1
for (i in 3721:1) { if ( m2[i,1]>=m2[i,2]) {Designmatrix1small <- Designmatrix1small[.−(i+62)]}}
for (i in 3721:1) { if ( m2[i,1]>=m2[i,2]) {Designmatrix1small <- Designmatrix1small[.−i,]}}

save(Designmatrix1small, file = 'Designmatrix1small.Rdata')

# Different models:
namen=names(FGS.list)
k=matrix(data=NA, nrow=length(FGS.list)*length(FGS.list),1)
for (i in 1:length(FGS.list) )
{k[((length(FGS.list)∗(i−1))+1):(length(FGS.list)∗i)),]<−c(rep(namen[i],length(FGS.list)))}

l=c(rep(namen,length(FGS.list)))
m=cbind(k,l)

k2=matrix(data=NA, nrow= length(FGS.list)∗ length(FGS.list),1)
for (i in 1: length(FGS.list) ) {k2[(( length(FGS.list)∗(i−1))+1):(length(FGS.list)∗i)),]<−c(rep(i,length(FGS.list))}

l2=c(rep(1: length(FGS.list), length(FGS.list)))
m2=cbind(k2,l2)

Designmatrixbig2=model.matrix(~ l∗k+Twopaths, data=df1,
contrasts.arg=list(l=contrasts(df1$l, contrasts=F), k=contrasts(df1$k, contrasts=F)))
Designmatrixbig2<−Designmatrixbig2[,−(63:123)]
Designmatrix2small<−Designmatrixbig2

Designmatrixbig3=model.matrix(~ l∗k+triang, data=df1,
contrasts.arg=list(l=contrasts(df1$l, contrasts=F), k=contrasts(df1$k, contrasts=F)))
Designmatrixbig3<−Designmatrixbig3[,−(63:123)]
Designmatrix3small<−Designmatrixbig3

Designmatrixbig4=model.matrix(~ l∗k+star2, data=df1,
contrasts.arg=list(l=contrasts(df1$l, contrasts=F), k=contrasts(df1$k, contrasts=F)))
Designmatrixbig4<−Designmatrixbig4[,−(63:123)]
Designmatrix4small<−Designmatrixbig4

Designmatrixbig5=model.matrix(~ l∗k+Twopaths+star2, data=df1,
contrasts.arg=list(l=contrasts(df1$l, contrasts=F), k=contrasts(df1$k, contrasts=F)))
Designmatrixbig5<−Designmatrixbig5[,−(63:123)]
Designmatrix5small<−Designmatrixbig5

80
Designmatrixbig5a = model.matrix(~ l*k+Twopaths+triang, data=df1, contrasts.arg=list(l=contrasts(df1$l, contrasts=F), k=contrasts(df1$k, contrasts=F)))
Designmatrixbig5a <- Designmatrixbig5a[-(63:123)]
Designmatrix5asmall <- Designmatrixbig5a

Designmatrixbig5b = model.matrix(~ l*k+triang+star2, data=df1, contrasts.arg=list(l=contrasts(df1$l, contrasts=F), k=contrasts(df1$k, contrasts=F)))
Designmatrixbig5b <- Designmatrixbig5b[-(63:123)]
Designmatrix5bsmall <- Designmatrixbig5b

Designmatrixbig5c = model.matrix(~ l*k+Twopaths+star3, data=df1, contrasts.arg=list(l=contrasts(df1$l, contrasts=F), k=contrasts(df1$k, contrasts=F)))
Designmatrixbig5c <- Designmatrixbig5c[-(63:123)]
Designmatrix5csmall <- Designmatrixbig5c

Designmatrixbig5d = model.matrix(~ l*k+Twopaths+star2+star3, data=df1, contrasts.arg=list(l=contrasts(df1$l, contrasts=F), k=contrasts(df1$k, contrasts=F)))
Designmatrixbig5d <- Designmatrixbig5d[-(63:123)]
Designmatrix5dsmall <- Designmatrixbig5d

Designmatrixbig5e = model.matrix(~ l*k+Twopaths+star2+star3+star4, data=df1, contrasts.arg=list(l=contrasts(df1$l, contrasts=F), k=contrasts(df1$k, contrasts=F)))
Designmatrixbig5e <- Designmatrixbig5e[-(63:123)]
Designmatrix5esmall <- Designmatrixbig5e

Designmatrixbig6 = model.matrix(~ l*k+Twopaths+star2+triang, data=df1, contrasts.arg=list(l=contrasts(df1$l, contrasts=F), k=contrasts(df1$k, contrasts=F)))
Designmatrixbig6 <- Designmatrixbig6[, -(63:123)]
Designmatrix6small <- Designmatrixbig6

Designmatrixbig7 <- model.matrix(~ l*k+Twophaths+star2+star3+star4+star5+star6+star7+star8+star9+star10, 
data=df1, contrasts.arg=list(l=contrasts(df1$l, contrasts=F), k=contrasts(df1$k, contrasts=F)))
Designmatrixbig7 <- Designmatrixbig7[, -(63:123)]
Designmatrix7small <- Designmatrixbig7

Designmatrixbig8 <- model.matrix(~ l*k+Twophaths+star2+star3+star4+star5+star6+star7+star8+star9+star10+triang, 
data=df1, contrasts.arg=list(l=contrasts(df1$l, contrasts=F), k=contrasts(df1$k, contrasts=F)))
Designmatrixbig8 <- Designmatrixbig8[, -(63:123)]
Designmatrix8small <- Designmatrixbig8

for (i in 3721:1) { if ( m2[i,1] >= m2[i,2]) {
  Designmatrix2small <- Designmatrix2small[, -(i+63)]
  Designmatrix3small <- Designmatrix3small[, -(i+63)]
  Designmatrix4small <- Designmatrix4small[, -(i+63)]
  Designmatrix5small <- Designmatrix5small[, -(i+64)]
  Designmatrix5asmall <- Designmatrix5asmall[, -(i+64)]
  Designmatrix5bsmall <- Designmatrix5bsmall[, -(i+64)]
  Designmatrix5csmall <- Designmatrix5csmall[, -(i+64)]
  Designmatrix5esmall <- Designmatrix5esmall[, -(i+66)]
  Designmatrix6small <- Designmatrix6small[, -(i+65)]
  Designmatrix7small <- Designmatrix7small[, -(i+72)]
  Designmatrix8small <- Designmatrix8small[, -(i+73)]
}
for (i in 3721:1) { if ( m2[i,1] >= m2[i,2]) {
  Designmatrix2small <- Designmatrix2small[-i,]
  Designmatrix3small <- Designmatrix3small[-i,]
  Designmatrix4small <- Designmatrix4small[-i,]

82
Designmatrix5small <- Designmatrix5small[-i,]
Designmatrix5asmall <- Designmatrix5asmall[-i,]
Designmatrix5bsmall <- Designmatrix5bsmall[-i,]
Designmatrix5csmall <- Designmatrix5csmall[-i,]
Designmatrix5dsmall <- Designmatrix5dsmall[-i,]
Designmatrix5esmall <- Designmatrix5esmall[-i,]
Designmatrix6small <- Designmatrix6small[-i,]
Designmatrix7small <- Designmatrix7small[-i,]
Designmatrix8small <- Designmatrix8small[-i,]}

save(Designmatrix2small, file='Designmatrix2small.Rdata')
save(Designmatrix3small, file='Designmatrix3small.Rdata')
save(Designmatrix4small, file='Designmatrix4small.Rdata')
save(Designmatrix5small, file='Designmatrix5small.Rdata')
save(Designmatrix5asmall, file='Designmatrix5asmall.Rdata')
save(Designmatrix5bsmall, file='Designmatrix5bsmall.Rdata')
save(Designmatrix5csmall, file='Designmatrix5csmall.Rdata')
save(Designmatrix5dsmall, file='Designmatrix5dsmall.Rdata')
save(Designmatrix5esmall, file='Designmatrix5esmall.Rdata')
save(Designmatrix6small, file='Designmatrix6small.Rdata')
save(Designmatrix7small, file='Designmatrix7small.Rdata')
save(Designmatrix8small, file='Designmatrix8small.Rdata')
R: Cross-validation

kf = 5  # number of folds. In this case we do a 5-fold cross-validation
Choose.designmatrix = Designmatrix1small
le <- (dim(Choose.designmatrix)[2]) + 1  # for later, for cof [] function

colm = dim(Choose.designmatrix)[2]
rows = dim(Choose.designmatrix)[1]
Designmatrixtemp1 = matrix(NA, nrow = rows, ncol = colm)
Naf3temp = matrix(0, nrow = rows, ncol = 1)
af3temp = matrix(0, nrow = rows, ncol = 1)

leaveoutnumbers <- c(1:rows)  # making this vector already for later
randomnumbers <- sample(1:rows, floor((rows/kf)), replace = F)
randomnumbers <- randomnumbers[(order(randomnumbers))]  # order them

for (j in 1: kf) {
  if (j < 5) {
    for (i in floor((rows/kf)):1) {
      Designmatrixtemp1[(i+(j - 1) * floor((rows/kf))],) <-
      as.vector(Choose.designmatrix[(randomnumbers[i]),])
      Naf3temp[i+(j - 1) * floor((rows/kf))]) <- Naf3[(randomnumbers[i])]
      af3temp[i+(j - 1) * floor((rows/kf))]) <- af3[(randomnumbers[i])]
    }
  }
  leaveoutnumbers <- setdiff(leaveoutnumbers, randomnumbers)
  # a vector of 1 to 1891 (if rows = 1891) {((3721-61)/2)+61}
  # where the numbers used before are leaved out.
  randomnumbers <- sample(leaveoutnumbers, floor(rows/kf), replace = F)
  randomnumbers <- randomnumbers[(order(randomnumbers))]  # order them
}

if (j == 5) { randomnumbers <- leaveoutnumbers
# if the rows/kf are not an integer then will the last fold be larger.
  for (i in ceiling((rows/kf)):1) {
    Designmatrixtemp1[(i+(j - 1) * floor((rows/kf))],) <-
  }
}
as.vector(Choose.designmatrix[(randomnumbers[i]),])
Naf3temp[i+((j-1)*floor(rows/kf))] <- Naf3[(randomnumbers[i])]
af3temp[i+((j-1)*floor(rows/kf))] <- af3[(randomnumbers[i])]
}
}

Dster1 <- Designmatrixtemp1[ -c(1:366),]
Naf3ster1 <- as.numeric(Naf3temp[ -c(1:366)])
af3ster1 <- as.numeric(af3temp[ -c(1:366)])
D1 <- Designmatrixtemp1[ c(1:366),]
Naf3.1 <- as.numeric(Naf3temp[ c(1:366)])
af3.1 <- as.numeric(af3temp[ c(1:366)])

Dster2 <- Designmatrixtemp1[ -c(367:732),]
Naf3ster2 <- as.numeric(Naf3temp[ -c(367:732)])
af3ster2 <- as.numeric(af3temp[ -c(367:732)])
D2 <- Designmatrixtemp1[ c(367:732),]
Naf3.2 <- as.numeric(Naf3temp[ c(367:732)])
af3.2 <- as.numeric(af3temp[ c(367:732)])

Dster3 <- Designmatrixtemp1[ -c(733:1098),]
Naf3ster3 <- as.numeric(Naf3temp[ -c(733:1098)])
af3ster3 <- as.numeric(af3temp[ -c(733:1098)])
D3 <- Designmatrixtemp1[ c(733:1098),]
Naf3.3 <- as.numeric(Naf3temp[ c(733:1098)])
af3.3 <- as.numeric(af3temp[ c(733:1098)])

Dster4 <- Designmatrixtemp1[ -c(1099:1464),]
Naf3ster4 <- as.numeric(Naf3temp[ -c(1099:1464)])
af3ster4 <- as.numeric(af3temp[ -c(1099:1464)])
D4 <- Designmatrixtemp1[ c(1099:1464),]
Naf3.4 <- as.numeric(Naf3temp[ c(1099:1464)])
af3.4 <- as.numeric(af3temp[ c(1099:1464)])

85
Dster5 <- Designmatixtemp1[-c(1465:1830),]
Naf3ster5 <- as.numeric(Naf3temp[-c(1465:1830)])
a3ster5 <- as.numeric(a3temp[-c(1465:1830)])
D5 <- Designmatixtemp1[c(1465:1830),]
Naf3.5 <- as.numeric(Naf3temp[c(1465:1830)])
a3.5 <- as.numeric(a3temp[c(1465:1830)])

# We have now only evenly and random divided the Designmatrix etc.

pen.fac1s <- rep(1,(dim(Dster1)[2]))
pen.fac1s[1] <- 0

# Now perform cross-validation for every part:
# Starting with Dster1
morelambda1 <- NULL
moreobjective1 <- NULL
moreloglik1 <- NULL
eta1 <- NULL
resultaat <- glmnet(Dster1,cbind(a3ster1-Naf3ster1,Naf3ster1),family ="binomial",penalty.factor=pen.fac1s)
lambda=exp(-6)
cof <- coef(resultaat,s=lambda)[1:le]
eta1=((D1[-1]*%*%cof[3:le])+cof[1])
loglik =0
for(j in 1:length(Naf3.1)){
  loglik <- Naf3.1[j]*eta1[j]-a3.1[j]*(log(1+exp(eta1[j]))) + loglik
}
lasso=0
for(i in 3:le){lasso <- lasso+abs(cof[i])}
objective=lambda*lasso-(loglik)
morelambda1 <- lambda
moreloglik1 <- loglik
moreobjective1 <- objective
for(i in 1:80){lambda <- exp(-(0.1*i+6))
  lasso=0
  cof <- coef(resultaat,s=lambda)[1:le]
eta1=(((D1[,-1]%%cof[3:le])+cof[1])
loglik=0
for (j in 1:length(Naf3.1)) {
    loglik <-Naf3.1[j]*eta1[j]-af3.1[j]*(log(1+exp(eta1[j])))+loglik
}
for (j in 3:le) {lasso<-lasso+cof[j]^2}
objective<-lambda*lasso-(loglik)
morelambda1<-c(morelambda1,lambda)
moreloglik1<-c(moreloglik1,loglik)
moreobjective1<-c(moreobjective1,objective)
}

#plot of the cross-validation. Note that the exact lambda
#and plot range depends on the randomly divided Designmatrix.
#So this will be different next time.
plot (morelambda1,moreloglik1,xlim=range(morelambda1),ylim=range(moreloglik1),pch=20,
cex=0.8,
main="Fold 1", xlab="lambda",ylab="loglikelihood",
text(0.0015, -56250, expression(paste(lambda[max],\phantom(x),'=8.27 \times 10^{-5}')))

morelambda1[which.max(moreloglik1)] #maximum

#Now for Dster2
morelambda2<-NULL
moreobjective2<-NULL
moreloglik2<-NULL
eta1<-NULL
resultaat <- glmnet(Dster2,cbind(af3ster2-Naf3ster2,Naf3ster2),family="binomial",penalty.factor=pen.fac)
lambda=exp(-6)
cof<-coef(resultaat,s=lambda)[1:le]
eta1=((D2[,-1]%%cof[3:le])+cof[1])
loglik=0
for (j in 1:length(Naf3.2)) {
    loglik <-Naf3.2[j]*eta1[j]-af3.2[j]*(log(1+exp(eta1[j]))) + loglik

bridge2=0
for (i in 3:le) {bridge2 <- bridge2 + cof[i]^2}
objective <- lambda * bridge2 - (loglik)
morelambda2 <- lambda
moreloglik2 <- loglik
moreobjective2 <- objective
for (i in 1:80) {
  lambda <- exp(- (0.1 * i + 6))
  bridge2 = 0
  cof <- coef(resultaat, s = lambda)[1:le]
  eta1 = ((D2[,-1] %*% cof[3:le]) + cof[1])
  loglik = 0
  for (j in 1:length(Naf3.2)) {
    loglik <- Naf3.2[j] * eta1[j] - af3.2[j] * (log(1 + exp(eta1[j]))) + loglik
  }
  for (j in 3:le) {bridge2 <- bridge2 + cof[j]^2}
  objective <- lambda * bridge2 - (loglik)
  morelambda2 <- c(morelambda2, lambda)
  moreloglik2 <- c(moreloglik2, loglik)
  moreobjective2 <- c(moreobjective2, objective)
}

plot (morelambda2, moreloglik2, xlim=range(morelambda2), ylim=range(moreloglik2), pch=20,
  cex=0.8,
  main="Fold 2", xlab="lambda", ylab="loglikelihood",
  text (0.0015, -53745, expression(paste(lambda[max], phantom(x), '=' 1.01 * 10^-4))))
morelambda2[which.max(moreloglik2)] #maximum

# Now for Dster3
morelambda3 <- NULL
moreobjective3 <- NULL
moreloglik3 <- NULL
eta1 <- NULL
resultaat <- glmnet(Dster3, cbind(af3ster3 - Na3ster3, Naf3ster3), family = "binomial", penalty.factor = pen.fac)
lambda = exp(-4)
cof <- coef(resultaat, s = lambda)[1:le]
eta1 = ((D3[,-1]*%*%cof[3:le])+cof[1])
loglik = 0
for (j in 1:length(Naf3.3)) {
  loglik <- Naf3.3[j]*eta1[j] - af3.3[j] * (log(1+exp(eta1[j]))) + loglik
}
bridge2 = 0
for (i in 3:le) {bridge2 <- bridge2 + cof[i]^2}
objective = lambda * bridge2 - (loglik)
morelambda3 <- lambda
moreloglik3 <- loglik
moreobjective3 <- -objective
for (i in 1:80) {
  lambda <- exp(-0.1*i+6)
  bridge2 = 0
  cof <- coef(resultaat, s = lambda)[1:le]
  eta1 = ((D3[,-1]*%*%cof[3:le])+cof[1])
  loglik = 0
  for (j in 1:length(Naf3.3)) {
    loglik <- Naf3.3[j]*eta1[j] - af3.3[j] * (log(1+exp(eta1[j]))) + loglik
  }
  for (j in 3:le) {bridge2 <- bridge2 + cof[j]^2}
  objective <- -lambda * bridge2 - (loglik)
  morelambda3 <- c(morelambda3, lambda)
  moreloglik3 <- c(moreloglik3, loglik)
  moreobjective3 <- c(moreobjective3, objective)
}
plot(morelambda3, moreloglik3, xlim = range(morelambda3), ylim = range(moreloglik3), pch = 20, cex = 0.8, main = "Fold 3", xlab = "lambda", ylab = "loglikelihood", text(0.0015, -59975, expression(paste(lambda[max], phantom(x) , '=' , 1.12* 10^(-4))))
morelambda3[which.max(moreloglik3)] # maximum

# Now for Dster4

89
morelambda4 <- NULL
moreobjective4 <- NULL
moreloglik4 <- NULL
eta1 <- NULL
resultaat <- glmnet(Dster4,cbind(af3ster4–Na3ster4,Naf3ster4),family="binomial",penalty.factor=pen.fac) lambda=exp(−6) cof<-coef(resultaat,s=lambda)[1:le]
eta1=((D4[-1]%*%cof[3:le])+cof[1])
loglik =0
for (j in 1:length(Naf3.4)){
  loglik <-−Naf3.4[j]*η1[j]−af3.4[j] *(log(1+exp(η1[j]))) +loglik
}
bridge2=0
for (i in 3:le){bridge2<-bridge2+cof[i]^2}
objective=lambda*bridge2−(loglik)
morelambda4 <-−lambda
moreloglik4 <-−loglik
moreobjective4 <-−objective
for (i in 1:80){lambda <-−exp(−(0.1*i+6))
  bridge2=0
  cof<-coef(resultaat,s=lambda)[1:le]
  eta1=((D4[-1]%*%cof[3:le])+cof[1])
  loglik =0
for (j in 1:length(Naf3.4)){
  loglik <-−Naf3.4[j]*η1[j]−af3.4[j] *(log(1+exp(η1[j]))) +loglik
}
for (j in 3:le) {bridge2<-bridge2+cof[j]^2}
objective<-−lambda*bridge2−(loglik)
morelambda4 <-−c(morelambda4,lambda)
moreloglik4 <-−c(moreloglik4,loglik)
moreobjective4 <-−c(moreobjective4,objective)
}
plot(morelambda4,moreloglik4,xlim=range(morelambda4),ylim=range(moreloglik4),pch=20,
cex=0.8,
main="Fold 4", xlab="\lambda", ylab="loglikelihood",
text(0.0015, -57450, expression(paste(lambda[max],\text{phantom(x)},'=9.14 \times 10^{-5}')))
morelambda4[which.max(moreloglik4)] #maximum

# Now for Dster5
morelambda5<-NULL
moreobjective5<-NULL
moreloglik5<-NULL
eta1<-NULL
resultaat <- glmnet(Dster5,cbind(af3ster5-Naf3ster5,Naf3ster5),family="binomial",penalty.factor=pen.fac)
lambda=exp(-6)
cof <- coef(resultaat,s=lambda)[1:le]
eta1=((D5[,-1]%*%cof[3:le])+cof[1])
loglik =0
for (j in 1:length(Naf3.5)){
  loglik <- Naf3.5[j]*eta1[j]-af3.5[j] *(log(1+exp(eta1[j]))) +loglik
}
bridge2=0
for (i in 3:le){bridge2<-bridge2+cof[i]^2}
objective=lambda*bridge2-(loglik)
morelambda5<-lambda
moreloglik5<-loglik
moreobjective5<-objective
for (i in 1:80){
  lambda<-exp(-(0.1*i+6))
  bridge2=0
  cof <- coef(resultaat,s=lambda)[1:le]
  eta1=((D5[,-1]%*%cof[3:le])+cof[1])
  loglik =0
  for (j in 1:length(Naf3.5)){
    loglik <- Naf3.5[j]*eta1[j]-af3.5[j] *(log(1+exp(eta1[j]))) +loglik
  }
  for (j in 3:le){bridge2<-bridge2+cof[j]^2}
  objective <- lambda*bridge2-(loglik)
  morelambda5<-c(morelambda5,lambda)
moreloglik5 <- c(moreloglik5, loglik)
moreobjective5 <- c(moreobjective5, objective)
}

plot(morelambda5, moreloglik5, xlim=range(morelambda5), ylim=range(moreloglik5), pch=20,
     cex=0.8,
     main="Fold 5", xlab="lambda", ylab="loglikelihood",
     text(0.0015, -57480, expression(paste(lambda[max], phantom(x), '=4.54*10^-5'))))
morelambda5[which.max(moreloglik5)]

# Calculating the average of the 5:
(morelambda5[which.max(moreloglik5)] + morelambda1[which.max(moreloglik1)] +
 morelambda3[which.max(moreloglik3)] + morelambda2[which.max(moreloglik2)] +
 morelambda4[which.max(moreloglik4)]) / 5