Link Prediction Applied to Tract-tracing Data

Jules Kruijswijk[1]
s4140230
Artificial Intelligence
Radboud University Nijmegen

Morten Mørup[2]
Department of Informatics and Mathematical Modelling, Kgs. Lyngby
Technical University of Denmark

Rembrandt Bakker[3]
Donders Institute for Brain, Cognition and Behaviour, Nijmegen
Radboud University Nijmegen

Marcel van Gerven[4]
Donders Institute for Brain, Cognition and Behaviour, Nijmegen
Radboud University Nijmegen

Bachelor's Thesis in Artificial Intelligence

August 26, 2014

[1] jma.kruijswijk@student.ru.nl
[2] External supervisor
[3] Affiliated supervisor
[4] Internal supervisor

Contents

Abstract

Tract-tracing studies are invasive and costly, but are still applied since they are more accurate than other techniques that expose the brain's structural connectivity. To reduce the costs of future tract-tracing studies, the present study investigates whether link prediction algorithms, that are normally used for exposing new information in social networks, can be used to maximize the information gained by future tract-tracing studies. Before using a link prediction algorithm on tract-tracing data, the performance is tested using simulated networks that mimic the topological features of human brain networks. The results show that the algorithm performs well on the simulated data and also when applied to tract-tracing data. Various ways to improve the empirical results are discussed.

1 Introduction

Neuroscience has come a long way from a paradigm where specific brain areas were thought to contribute to characteristics such as arrogance, affection and sense of witchcraft [1], to modern neuroscience where new paradigms such as computational models for neuronal activity are dominant [2]. In modern day neuroscience, there are two main theories that describe the cognitive functions of the brain. In the first place, there is the theory of modularity, which supports specialization of brain areas to specific functions [3]. The other theory is called distributive processing, which proposes that the functions of the brain are distributed over all regions [4]. A collaboration between the two theories could be a more probable explanation of the brain's structure [3]. This view is also adhered to in more recent studies [5]. Functional segregation and integration are mediated by the structural connectivity which links together different brain regions.

1.1 Tract-tracing

One path to expose the complete structural connectivity of the brain is to use tract-tracing techniques. The technique traces the axoplasmic transport of a neuron using tracers that can be labelled with a fluorescence microscope [6]. Several tracers, either anterograde or retrograde, are injected in the animal that is being observed to highlight the downstream or upstream neurons. The anterograde tracers highlight the transport that goes from the neuron's soma (the source) to its axon terminals, whereas retrograde tracers work the other way around [7]. This means that when retrograde tracers are used the injected area is the target area and the area that contains the labelled neurons is the source area. The tracing results in detailed data, exposing structural connectivity at a neuronal level. It tells us how regions are interconnected and which have stronger connections [6].

Not only is the technique invasive on the level of the injection that is needed, it leads to the eventual death of the animal that is used as study object. In each study, the animal will first be anesthetized, after which the tracers will be injected. Before the neurons can be counted under the microscope it is required that the brain is removed from the animal after several days of survival [6, 8]. Despite these downsides, the main reason that tract-tracing is still being used is that it has better sensitivity and specificity compared to other techniques [9]. To minimize the costs and to spare the maximum amount of lives, but maintaining the goal of exposing the connectivity in the brain on such a detailed level, it would be useful to know which study should be applied next.

1.2 Link prediction

A lot of time has been devoted to the research of complex, social networks, where the properties of these networks are analyzed [10, 11]. In social networks, people or other agents

are represented as nodes and their edges represent the interaction or influence between these agents. In reality, these networks transform a lot, because the relations between agents change. For example, when you have a group of scientists where the edges represent their collaboration, it can happen that two scientists decide to stop their collaboration for various reasons, in which case the social network structure will shift. The link prediction problem asks to what extent we can predict these changes based on the features and information that the network contains [10]. One feature could be transitivity, which means that if agent A knows agent B and agent B knows agent C, then there is a high possibility that agent A also knows agent C. Another would be clustering, in which the agents tend to cluster into groups so that ties within a group are more dense than between groups. Such predictions based on network features could be useful for various applications. One example would be to expose new or "missing" collaborations in a terrorist network, so that future attacks can be prevented [12].

1.3 Present study

The brain to a certain degree resembles a social network, where the neurons would be the agents and the connections between neurons the interaction between those agents. It is also divided in regions similar to groups in a social network and such as interactions between agents change new interactions between neurons are constructed through learning. Since link prediction has been proven to work in the context of social network analysis and reveal new information [10, 11], perhaps the network of the brain hides the same type of information. If new links could be discovered using existing data, then that would mean tract-tracing researchers could make studies more specific and still gain the same amount of information. This would result in more lives could be spared and money could be saved. I therefore would like to propose the following research question:

> Can link prediction algorithms be used to maximize the information gained by future tract-tracing studies?

Based on studies of link prediction, the expectation is that a link prediction algorithm can be utilized to reveal new information in a brain's network.

2 Methods

2.1 Procedure

Hoff et al. proposed a link prediction algorithm where the probability of a relation between two nodes depends on the Euclidean distance of those nodes in an unobserved "social space" [13]. The distance between the nodes is determined by the characteristics that they share in

relation to the network. Several improvements have been made by Handcock et al. [14] and Krivitsky et al. [15] The latest edition of the algorithm will be used, which includes several improvements to the older versions [15, 16]. The input of the algorithm can be either binary or weighted count data. Weighted data can lead to better performance because the data contains more information. This thesis will focus on binary data, since the Markov data does not readily allow the use of weighted counts and can be easily transformed into binary data using a cut-off, as shown in Figure 4. Before the algorithm is applied, it is useful to see how the algorithm performs on artificial data which can be controlled. Therefore, a simulation will be done first.

2.1.1 Algorithm

The model assumes that each node i has an unobserved position in a two-dimensional Euclidean latent space Z. The algorithm also allows extensions into other dimensions, but in this case we restrict ourselves to the two-dimensional case. The probability of a link between a pair of nodes i and j in the actual network is determined by the positions of the nodes z_i and z_j in the latent space and an offset term β:

$$p(y_{ij}|\beta, z_i, z_j) = \frac{1}{1 + \exp(-(\beta - ||z_i - z_j||_2))}\,. \tag{1}$$

The latent space is estimated with a Bayesian approach and inference is done using a Markov chain Monte Carlo (MCMC) algorithm. There are a few hyper-parameters that have to be specified by the user and in this case it is chosen to use the default values [15]. At each MCMC iteration, the algorithm makes two updates using Metropolis-Hastings algorithms. Firstly, the actor-specific latent space positions are updated for each actor in a random order using a multivariate normal distribution - in this case a bivariate normal distribution because we restrict ourselves to a two-dimensional space. In the second update the new β and Z are proposed using a correlated multivariate normal distribution and accepted as a block. Eventually all samples are returned. The use of the Metropolis-Hastings algorithm requires a burn-in period, where a number of initial samples are thrown away. This burn-in period is required because the initial samples will be arbitrary and do not reflect samples from the posterior.

2.2 Simulation

Several networks are constructed to check the performance. As research suggest, brain connectivity can be described in terms of two different kinds of networks [5]. The performance will be checked on both types. In the first type the ties within a region are stronger than between regions (network type one). The second type is more distributed, based on similar connectivity profiles (network type two). Each created network will have a dense structure

based on its type.

2.2.1 Construction of a network

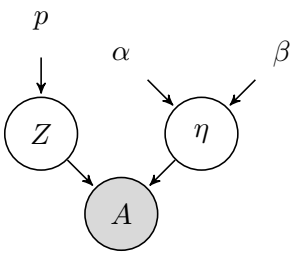

Figure 1: Creation of a binary network. p, α and β are the hyperparameters that eventually determine the structure of network A. Z is a categorical distribution and η is a Beta distribution. A is created by combining Z and η.

The network starts with two components, Z and η, which will form a binary network A as shown in Figure 11. The assignment matrix Z is N_k by K, where N_k is the number of nodes per cluster and K the total number of clusters. Z is defined as a Categorical distribution:

$$z_{ij} \sim \text{Cat}\left(\frac{1}{K}\right). \tag{2}$$

This means that every row in the matrix, and thus every node, gets a cluster assigned between 1 to K with a chance $p = \frac{1}{K}$. When the number of clusters K is 5, every node has the probability of $\frac{1}{5}$ to get into a cluster between 1 and 5.

The second component is a $K \times K$ link probability matrix η. Each value η_{ij} represents the chance that a link between clusters i and j exists. It is defined as a Beta distribution, having α and β as parameters:

$$\eta_{ij} \sim \text{Beta}(\alpha, \beta). \tag{3}$$

Using α and β, the probability density can be manipulated, such that a desired density is obtained. The probability density function of the Beta distribution is defined as:

$$f(x; \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha) \cdot \Gamma(\beta)} \cdot x^{\alpha - 1} \cdot (1 - x)^{\beta - 1}. \tag{4}$$

Where $\Gamma(n) = (n - 1)!$ for integers n.

Figure 2 shows what happens when α and β are varied. When both parameters are equal, for example $\alpha = \beta = 1$, you can see that the density is equally distributed over x. A parabolic distribution will be found when $\alpha = \beta = 2$, with the distribution more centered around 0.5. The higher the values of α and β with $\alpha = \beta$, the more the probabilities are distributed around 0.5. For the simulation, a variable c where $c = \alpha = \beta < 1$ is picked. With $c < 1$, the probability density will be will be more distributed around 0 and 1, as you can see with

$\alpha = \beta = 0.5$ in Figure 2. Because of this, the link probability between a pair of clusters will either be very low or very high, such that most of the connections between these clusters will be the same, thus resulting in a more consistent structure. This will create a network of type two. To create a network of type one, the within-cluster link chances are drawn from a Beta distribution with parameters $\alpha = 5$ and $\beta = 0.5$ and the between-cluster link chances are drawn with parameters $\alpha = 0.5$ and $\beta = 5$. For examples see Figure 3.

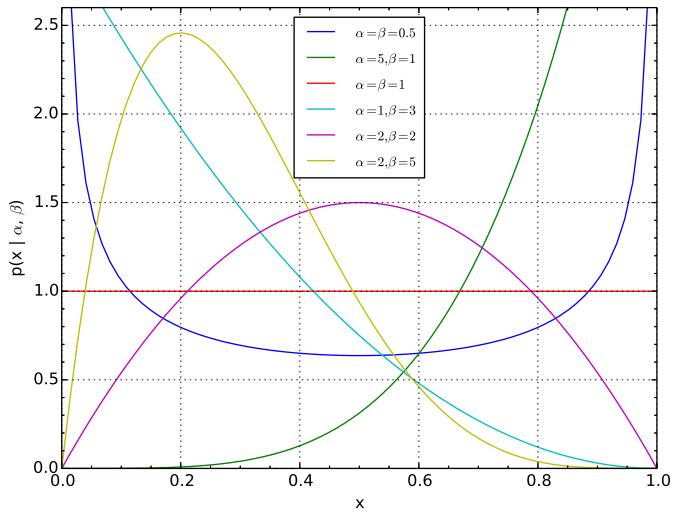

Figure 2: Different probability densities of the Beta distribution where the parameters α and β are varied.

For each pair of nodes (i, j) in Z we draw a link with probability θ_{ij} where

$$\theta_{ij} \sim \text{Bernoulli}(z_i^T \eta z_j) \tag{5}$$

where z_i is the i-th column of Z. This will result in a network A as seen in Figure 11.

2.2.2 Analysis of simulated data

When the networks are generated, they will be used as input for the algorithm to test its accuracy. From each network, a set of random edges is assumed unobserved. This is done multiple times, where the number of unobserved edges increases each time, to test the performance when having different amounts of information. The number of edges that is assumed unobserved for each network is 1, 10, 100, 1000, 2500, 5000, 7500, 8500, 9500 and 10000, so that the results will show what happens when the amount of information declines in relatively small steps.

The burn-in for the MCMC algorithm will be set to 20000, to ensure proper sampling from the posterior. When the algorithm is used on the network with unobserved edges, 4000

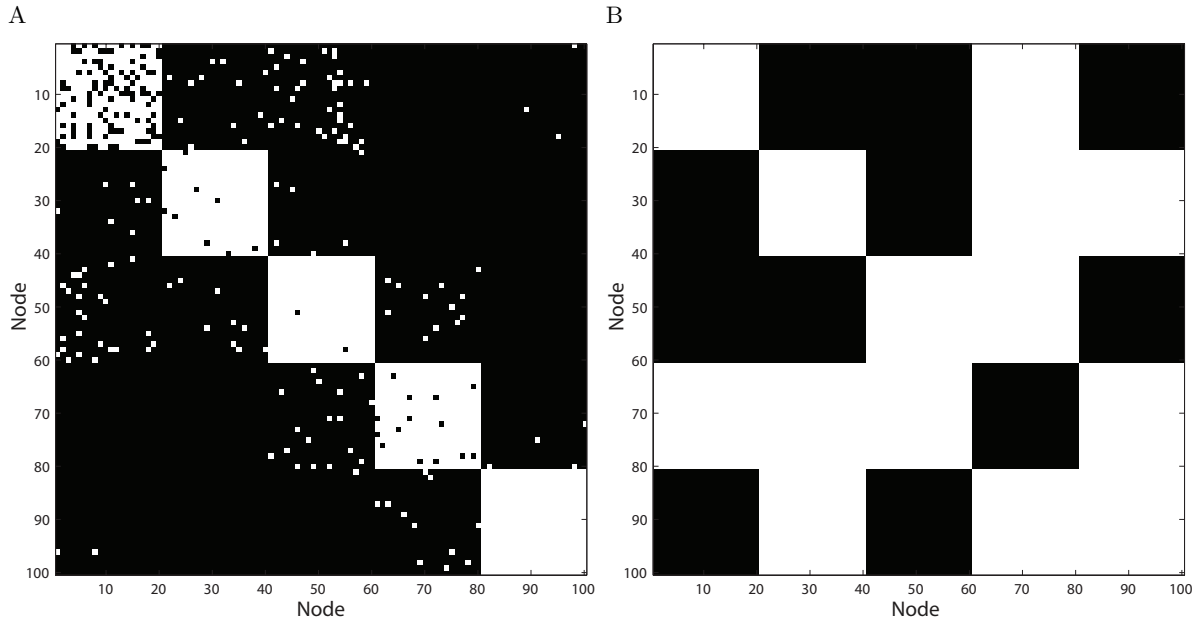

Figure 3: Visualization of simulated networks. Present connections are represented as white and the absent as black. Panel A shows mostly connections on the diagonal, which represents the structure of a network that has strong within and weak between links (network type one). Panel B shows a randomly structured network that has a strong density within the clusters (network type two).

samples of latent space matrices Z and the corresponding slopes β are used as a representation of the posterior. The probability that an edge in the original network is existent, is given by Equation (1). When the probability is calculated for each pair of nodes that has been unobserved, the probabilities are compared to their true value to determine the accuracy. When the true value in the network is 1, the accuracy is equal to (1) and when the true value is 0, the accuracy is equal to flip of (1), namely $(1 - p)$. The accuracy uses this flip and is defined as:

$$\text{accuracy}(y_{ij}) = p^q \cdot (1 - p)^{1-q} \tag{6}$$

where q is the true probability. If the accuracy is calculated over more than one link, the average accuracy is:

$$\text{average accuracy} = \frac{1}{N} \sum_{i=1}^{N} p_i^{q_i} \cdot (1 - p_i)^{1-q_i}. \tag{7}$$

The accuracy of the simulations is then compared using a boxplot for each network and each number of unobserved edges. The algorithm should perform better than chance. In the case of these networks, chance level performance is defined in terms of the majority class of the density of the network. The majority class is defined as the maximal density relative to the number of present or absent edges. If for example, the density of zeros in a network is 0.7, zero would be the majority class, i.e. we set the algorithm to guess everything as zero, the

algorithm would get an accuracy of 0.7. Since the algorithm should not perform on chance level, it should perform better than that and thus the density of the majority class is set as the baseline.

2.3 Markov data

To find out whether or not new links can be predicted in tract-tracing data, the research data from Markov et al. will be used [6]. The data consists of multiple tract-tracing experiment results, where retrograde tracing was applied to several macaques. Since retrograde tracing was used, the injection area is called the target area, because it is the point of termination, and the labeled area is called the source area. Before the tracers were injected, the monkeys were anesthetized. After injection, the monkeys were held alive and monitored for a certain survival time, so that the tracers had enough time to travel through the brain. Later the brains were removed and processed such that the tracers could be labeled in the soma of each neuron for each different region using a fluorescence microscope [6].

To answer the question for this thesis, only a specific part of the available data will be used. Markov et al. have put all the counted labels in a weighted connectivity matrix, as seen in Figure 4. This matrix represents the extrinsic fraction of labeled neurons (FLNe) for every region, which is estimated from the number of labeled neurons relative to the total number of labeled neurons, excluding the labeled neurons in the injected area itself. This connectivity matrix will be used in the link prediction algorithm to predict new links between two regions.

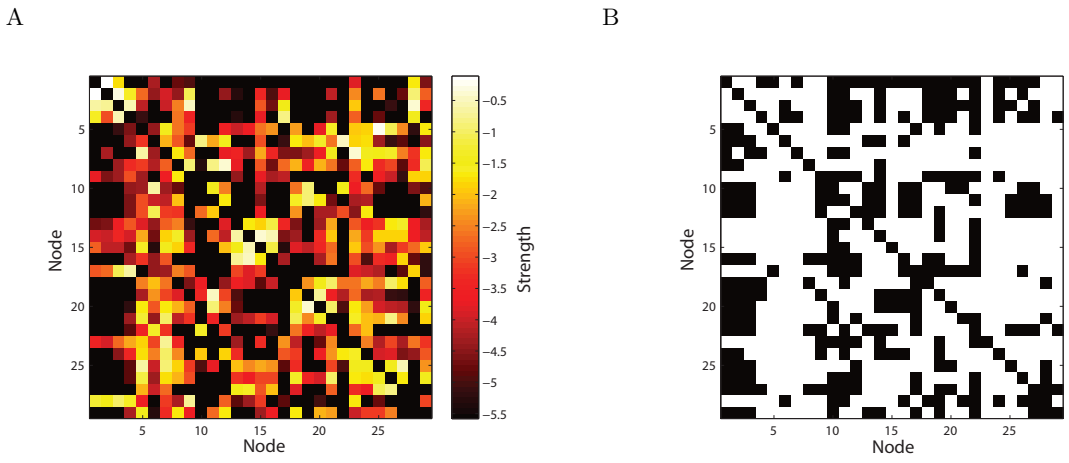

Figure 4: The FLNe values of 29 areas. Panel A shows the $\log_{10}$(FLNe). In Panel B the data is binarized. Because during the process of labeling of neurons mistakes can be made, Markov et al. suggest three different levels of reliability, where the lowest values can be discarded to reduce the number of false positives at the expense of false negatives [6].

2.3.1 Analysis of Markov data

The analysis on the Markov data will be nearly the same as that of the simulated data. The original FLNe values will be binarized. Markov et al. suggest a cut-off value for the data, which will increase the reliability of the data in terms of false positive connections. Connections are strong when $\log_{10}(\text{FLNe}) > -2$, moderate when $-4 < \log_{10}(\text{FLNe}) \leq -2$ and sparse when $\log_{10}(\text{FLNe}) < -4$. For this analysis, all sparse connections will be discarded. This results in the input as seen in Figure 5. Another difference with the simulated data, is that the Markov data is smaller in size. The network consists of 29 areas, which results in a total of 841 connections. The number of edges that is assumed unobserved is 1, 5, 10, 25, 50, 75, 100, 250, 500, 750 and 841.

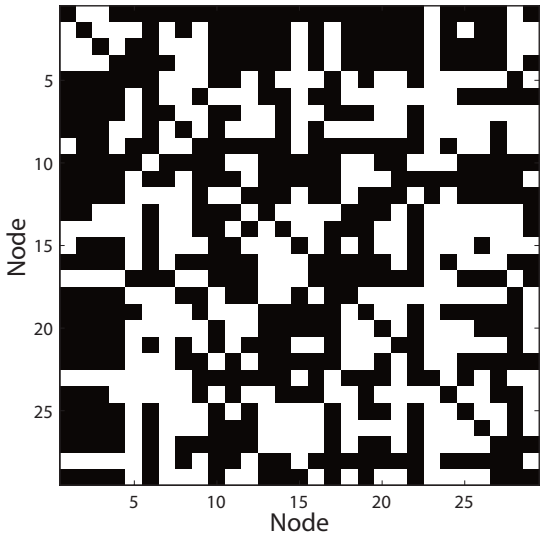

Figure 5: The used Markov data, where all the sparse connections $\log_{10}(\text{FLNe}) < -4$ are discarded and then binarized. Discarding the sparse connections results in more reliable data in terms of false positive connections. Present connections are represented as white and the absent connections as black.

3 Results

3.1 Simulation

Figure 6 shows the performance on both types of networks. The algorithm performs best, in absolute numbers, on network type one. However, relative to the density of the networks, the algorithm performs best on the second type. For both network types, the algorithm improves greatly when compared with random guessing, which would be around the baseline.

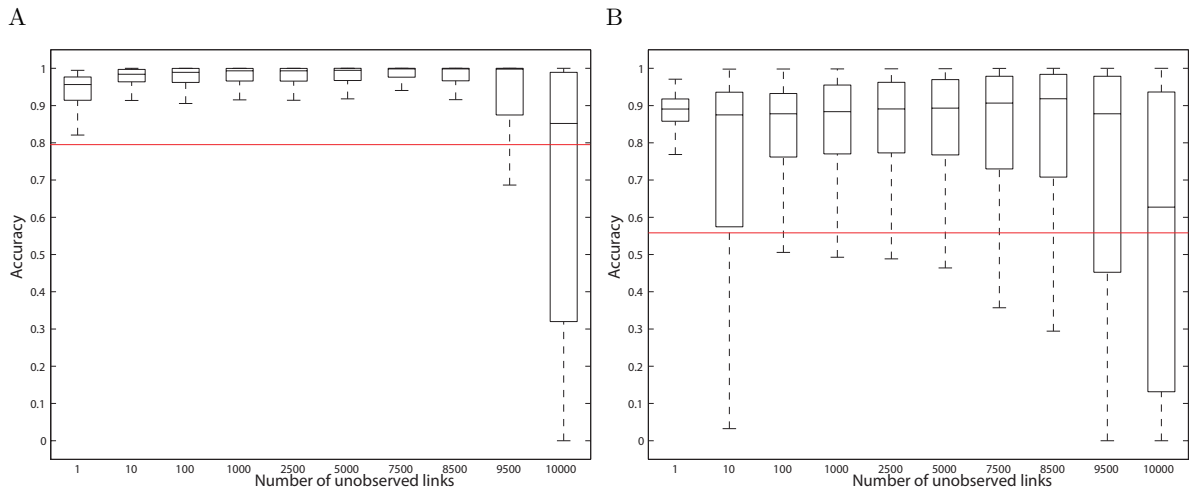

Figure 6: Accuracy of simulated networks. The red line is the density of the majority class of the network, which is a baseline for the performance. Panel A shows the results for the network type one and Panel B shows the results for network type two. The larger distribution in the boxplots in Panel B is caused by only a few edges. This means that the performance is only caused to deteriorate by noise on a small amount of edges and that the algorithm still performs good apart from those edges.

The results on the second network type show a broad distribution before the second quartile, especially at 10 deleted items. This is caused by the performance on just a few of the selected edges which happen to be hard to predict by the model. Figure 7 shows an example of this phenomenon, where only a few edges cause the model's prediction to deteriorate.

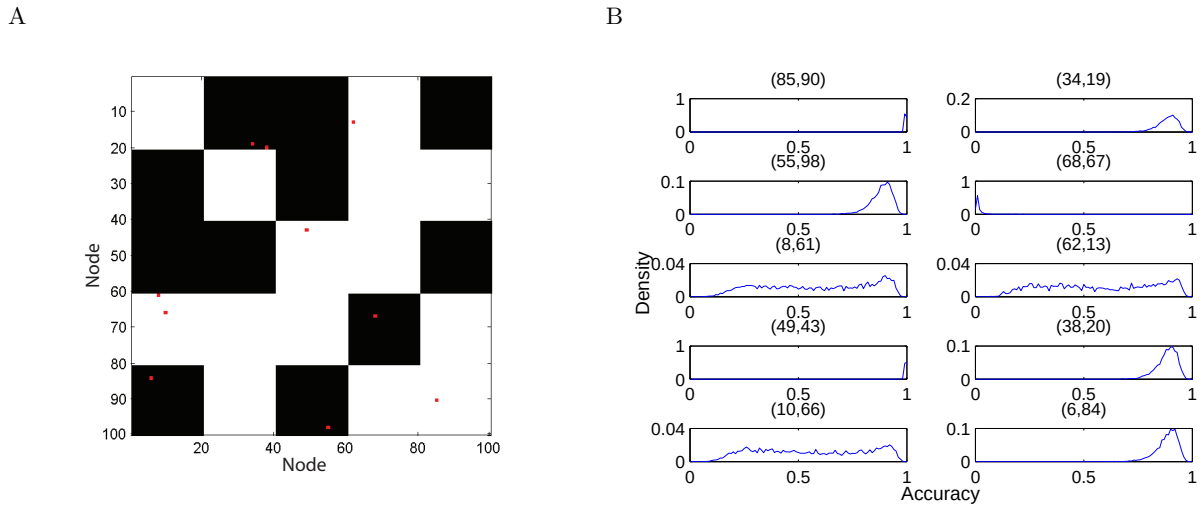

Figure 7: Positions of unobserved links and their probability density. The red dots in Panel A are the unobserved links. Panel B shows that only a few edges cause the noise in the accuracy. Especially matrix elements $(8, 61)$, $(10, 66)$ and $(62, 13)$ are very inconsistent. The performance on edge $(68, 67)$ is consistent, but wrong.

Figure 8 shows the latent space of network type one when there are 1000 unobserved links. The plot is based on a minimization of a Kullback-Leibler (MKL) divergence in the posterior of the output [15]. The figure shows that there are 5 clusters, just as in the original data (see Figure 3 Panel A), which means that the algorithm has made a good estimation of what the original data looks like.

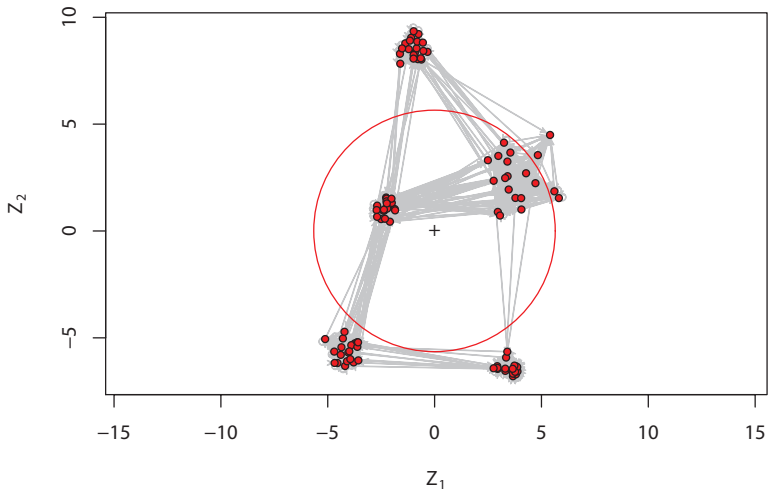

Figure 8: Latent space of network type one with 1000 unobserved links. The plot is based on a MKL using the posterior and is the most representative distribution.

3.2 Markov data

Figure 9 shows the performance of the algorithm on the Markov data. The algorithm seems to perform significantly better than chance as Panel A suggests, which is what the simulated data also shows. However, when only one item was set unobserved, the algorithm performed much worse than chance level. Note that the result of this may depend on the identity of the chosen edge, whereas an other chosen edge may perform better. The performance of the separate edges, as shown in Figure 9 Panel B, is much less specific when it is compared with the performance of the separate edges of the simulated data (see Figure 7). Even with this less specific separate edge performance, the algorithm still performs better than chance level.

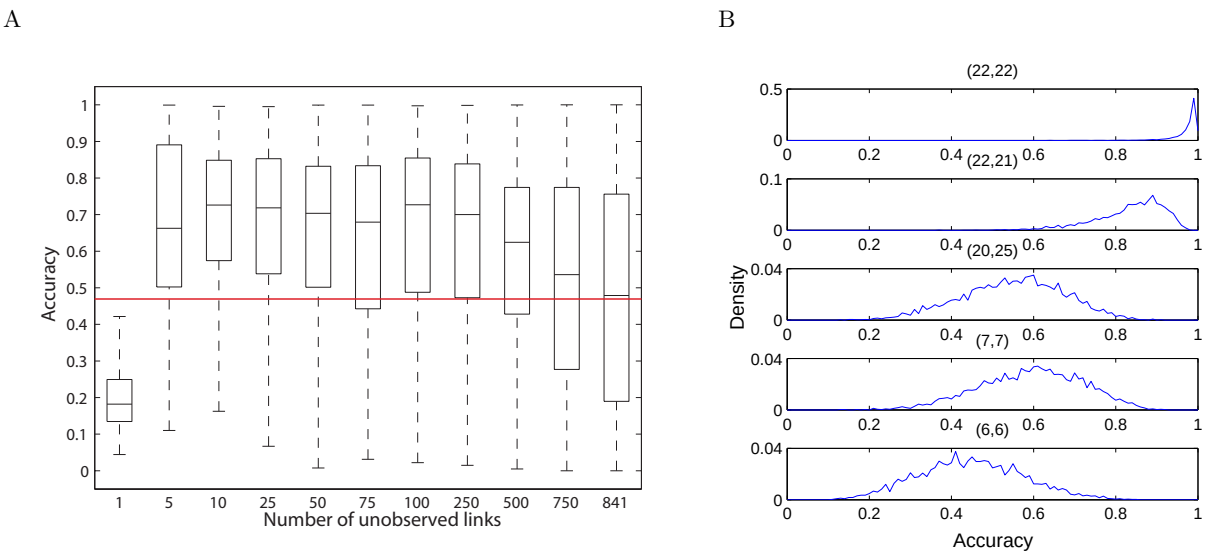

Figure 9: Panel A shows the accuracy of Markov data. The red line is the density of the majority class of the network, which is a baseline for the performance. Panel B shows the probability distribution over different edges. It shows that the algorithm is not specific within the samples of one edge. The title of each distribution refers to the position of each edge in the data of Figure 5.

Figure 10 shows the latent space of the Markov data when there are 100 unobserved links. The plot shows a slightly more random distribution when compared with the latent space of the simulation (Figure 8). Albeit being a bit more random, the plot still suggests a division of clusters in the Markov data.

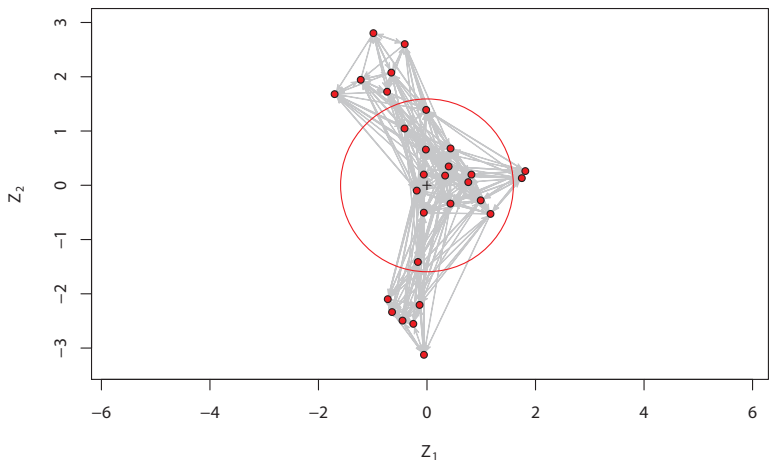

Figure 10: Latent space of Markov data with 100 unobserved links. The plot is based on a MKL using the posterior and is the most representative distribution.

4 Discussion

After taking a closer look at the results an answer can be formulated to the question whether link prediction can be used to maximize the information gained by future tract-tracing studies. The results show that the latent space approach allows for link predictions above chance level on the simulated data whose structure is based on the structure of the brain. Also there are predictions that are significantly above chance level for the tract-tracing data, where the results showed that the algorithm performs better than randomly guessing the probability of the links.

Although the results are promising, there still are several options to improve the current results, that are not tested due to time constraints. Firstly, the method that is used here mainly focuses on one link, whereas the MCMC could make better predictions on other links. An extension of the model into weighted counts could help the algorithm to make better estimates, which is also supported by the implemented algorithm. Next to that, the model can also be extended using sender and receiver node connection probabilities, other latent space dimensions and the assignment of nodes in particular groups [15]. These extensions would give the model more information about the network to work with and could result in better predictions.

Neuroscientists can use this research or future improved research to optimize their current experimental design. The algorithm should not be run on random links, but instead used on the links in the data that show no or nearly no connection. The missing information should be viewed as unobserved, in the same way we treated the unobserved links in this research, and then used as input into the algorithm. Eventually, the output can be used in an optimal experimental design study to plan the future tract-tracing experiments [17].

Acknowledgments

I would like to offer my special thanks to Morten Mørup for his advice given in the field of link prediction and Rembrandt Bakker for his advice given in the field of tract-tracing. Their willingness to give their time has been very much appreciated. Last but not least I am particularly grateful for the professional supervision given by Marcel van Gerven.

References

[1] F. Gall and W. Lewis, *On the Functions of the Brain and of Each of Its Parts: On the Organ of the Moral Qualities and Intellectual Faculties, and the Plurality of the Cerebral Organs*. Marsh, Capen & Lyon, 1835.

[2] P. S. Churchland, C. Koch, and T. J. Sejnowski, *Computational Neuroscience*. MIT Press, 1993.

[3] J. Fodor, *The Modularity of Mind: An Essay on Faculty Psychology*. A Bradford book, The MIT Press, 1983.

[4] A. R. Mcintosh, "Mapping cognition to the brain through neural interactions," *Memory*, vol. 7, no. 5-6, pp. 523–548, 1999.

[5] M. Hinne. Personal communication, May 2014.

[6] N. T. Markov, M. M. Ercsey-Ravasz, A. R. Ribeiro Gomes, C. Lamy, L. Magrou, J. Vezoli, P. Misery, A. Falchier, R. Quilodran, M. A. Gariel, J. Sallet, R. Gamanut, C. Huissoud, S. Clavagnier, P. Giroud, D. Sappey-Marinier, P. Barone, C. Dehay, Z. Toroczkai, K. Knoblauch, D. C. Van Essen, and H. Kennedy, "A weighted and directed interareal connectivity matrix for macaque cerebral cortex," *Cerebral Cortex*, vol. 24, pp. 17–36, Jan. 2014.

[7] D. Purves, *Neuroscience*. Sinauer Associates, Incorporated, 2012.

[8] N. T. Markov, P. Misery, A. Falchier, C. Lamy, J. Vezoli, R. Quilodran, M. A. Gariel, P. Giroud, M. M. Ercsey-Ravasz, L. J. Pilaz, C. Huissoud, P. Barone, C. Dehay, Z. Toroczkai, D. C. Van Essen, H. Kennedy, and K. Knoblauch, "Weight consistency specifies regularities of macaque cortical networks," *Cerebral Cortex*, vol. 21, pp. 1254–72, June 2011.

[9] R. Bakker, T. Wachtler, and M. Diesmann, "Cocomac 2.0 and the future of tract-tracing databases," *Frontiers in Neuroinformatics*, vol. 6, p. 30, Jan. 2012.

[10] D. Liben Nowell and J. Kleinberg, "The link prediction problem for social networks," *Journal of the American Society for Information Science and Technology*, vol. 54, pp. 556–559, 2003.

[11] L. Linyuan, "Link prediction in complex networks: A survey," *Physica A*, vol. 390, no. 6, 2011.

[12] V. E. Krebs, "Mapping networks of terrorist cells," *Connections*, vol. 24, no. 3, pp. 43–52, 2002.

[13] P. D. Hoff, A. E. Raftery, and M. S. Handcock, "Latent space approaches to social network analysis," *Journal of the American Statistical Association*, vol. 97, pp. 1090–1098, Dec. 2002.

[14] M. S. Handcock, A. E. Raftery, and J. M. Tantrum, "Model-based clustering for social networks," *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, vol. 170, pp. 301–354, Mar. 2007.

[15] P. N. Krivitsky, M. S. Handcock, A. E. Raftery, and P. D. Hoff, "Representing degree distributions, clustering, and homophily in social networks with latent cluster random effects models," *Social Networks*, vol. 31, pp. 204–213, July 2009.

[16] P. N. Krivitsky and M. S. Handcock, "Fitting latent cluster models for networks with latentnet," *Journal of Statistical Software*, vol. 24, pp. 1–23, May 2008.

[17] F. Pukelsheim, *Optimal Design of Experiments*. Classics in Applied Mathematics, Society for Industrial and Applied Mathematics, 2006.

Appendices

A Construction of Weighted Simulation Data

Although the final version of my thesis only considered binary Markov data, also time has been invested in working with weighted simulated data. In the beginning it was not certain if any weighted data or only binary data was available and thus there has been time invested on both types so that the thesis could be shaped using either of the two. Below you will find a piece of text and which would be added if any weighted Markov data had been available. Together with this text also code for weighted data has been created. After the text, you will firstly find Matlab code which creates simulation data and secondly R code which evaluates the weighted simulated data using the model.

> After a binary network has been created, the network is extended to a weighted network M (see Figure 11) using a Poisson distribution. The probability mass function of the Poisson distribution is defined as:
>
> $$f(k;\lambda) = P(X = k) = \frac{\lambda^k \cdot \exp(-\lambda)}{k!}. \tag{8}$$
>
> In the generation of a network, the values of A are used as input together with a parameter λ. λ is defined differently for the two different types of values in A, which are 0 and 1. Since λ is the expected value of the Poisson distribution, the structure of M is controlled using that parameter. For the entries that are 0, a low value of λ is picked, such as 10^{-100}, and the opposite is done for the entries that are 1 (e.g. 10). This results in a bigger difference between absent and present links in the generated network.

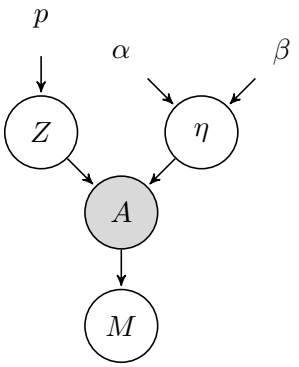

Figure 11: Creation of a binary network. p, α and β are the hyperparameters that eventually determine the structure of network M. Z is a categorical distribution and η is a Beta distribution. A is created by combining Z and η and M is created from A by using a Poisson distribution.

A.1 Create a weighted network

```
clear all;
close all;

%% hyperparameters

K = 5; % Number of clusters
Nk = 20; % Nodes per cluster

%Alpha and beta parameter for the beta distribution
%If you use this, you need 'eta = betarnd(a,b,K,K); in the for-loop
%a = b = 0.01 is for network type two
%a = 0.01; b = 0.01;

%If you use this setup for alpha and beta, use eta=betarnd(a,b); in the for-
    loop
%This is for network type one
a = [5 0.5 0.5 0.5 0.5;
     0.5 5 0.5 0.5 0.5;
     0.5 0.5 5 0.5 0.5;
     0.5 0.5 0.5 5 0.5;
     0.5 0.5 0.5 0.5 5];
b = [0.5 5 5 5 5;
     5 0.5 5 5 5;
     5 5 0.5 5 5;
     5 5 5 0.5 5;
     5 5 5 5 0.5];

%% Start generative process
N = Nk*K;

%Make cluster assignment matrix
Z = zeros(N,K);
for k=1:K
Z((k-1)*Nk + (1:Nk),k) = 1;
end

%Make cluster link probabilities

%Use this for the network type two configuration
%eta = betarnd(a,b,K,K);

%Use this for the network type one configuration
eta = betarnd(a,b);
eta = triu(eta);
eta = eta + eta';
eta(1:(K+1):end) = eta(1:(K+1):end)/2;
```

```
%Create the final binary network
A = zeros(N,N);
for i=1:N
    for j=1:N
        ci = find(Z(i,:));
        cj = find(Z(j,:));
        A(i,j) = (eta(ci,cj) > rand);
    end
end

%Save the network
save(['A_1_5a_0.5b_intra_extra' '.mat'],'A')

%Create weighted network based on the binary network
W = zeros(N,N);
for i=1:N
  for j=1:N
    W(i,j) = poissrnd(lambda(1+A(i,j)));
  end
end

%Save the network
save(['W_' num2str(l) '_5a_0.5b_intra_extra_10E-100lambda_10lambda.mat'], 'W')
```

A.2 Use of MCMC algorithm on weighted data

```
#The purpose of this script is to read different networks that were generated
    with Matlab
#With these networks I want to perform an ergmm calculation and export the Z
    and beta.
#The script also deletes a given number of random nodes in the network A

library(statnet)
library(R.matlab)

#Determine how many items you want to have deleted.
#We're going for: 1 10 100 1000 2500 5000 7500 10000
del <- c(1,10,100,1000,2500,5000,7500,10000)

#Set the number of nodes of the network
#100 is the number of nodes for the simulated data
ncol <- 100

#Loop over the different delete-values
for(d in del){
    #Save the old values to check accuracy later in matlab
```

```
oldValue <- matrix(0,d,1)
#Save the old row numbers
sampleRow <- matrix(0,d,1)
#Save the old col numbers
sampleCol <- matrix(0,d,1)
    #Read in file, it saves as variable W
    #The strings will be used later on
    str <- paste0('simulated_weighted')
    filetype <- '.mat'
    #Read the matrix, gives a list
    W<- readMat(paste0(str,filetype))
    #Convert the list to a matrix
    A <- matrix(unlist(W, use.names=FALSE), ncol = 100)

#Calculate the density to save it later on
dens <- network.density(A)

    #Using the random generated data, make those respective nodes Not
        Available (NA)
#A while-loop is used here instead of a for-loop because if we skip a for-
    loop with the 'next' function, it will eventually delete less edges than
     you would like
#The loop might seem redundant or inefficient, but R does not easily and
    readily support a sample of for example 10000 different coordinates in
    random order
    while(i <= del){
                    sampleColumn <- sample(100,1)
                    sampleRoww <- sample(100,1)
        #If this edge is already set to NA, try another one
                    if(is.na(A[sampleRoww,sampleColumn])){
                            next
                    }
        #Save the old values so we can use that later to calculate the
            accuracy
                    sampleRow[i] <- sampleRoww
                    sampleCol[i] <- sampleColumn
                    if(A[sampleRoww,sampleColumn] > 1){
                            oldValue[i,1] <- 1
                    }else{
                            oldValue[i,1] <- 0
                    }
        #Set to NA
                    A[sampleRoww,sampleColumn] <- NA
        #Increase i for the while-loop
                    i <- i + 1
    }
```

```
        #Set the weights when constructing new network
        W <- as.network.matrix(A, loops=FALSE, ignore.eval=FALSE, names.eval='
            weight', control=control.ergmm(burnin=40000))

        #Calculate the density for the accuracy
        dens <- network.density(W)

        #Now calculate the fit of this network
        W.fit <- ergmm(W ~ euclidean(d=2), response='weight', family="Poisson.
            log")

        #Save these two to a .mat file
        #The .mat file should be named rather equal to the opened file
        writeMat(paste0(str,'_R_deltest',d,filetype), Z=W.fit$sample$Z, b=W.fit
            $sample$beta, sampleRow=sampleRow, sampleCol=sampleCol, oldValue=
            oldValue, dens=dens)
}
```

B Source Code

This is the source code that was used for the thesis and the results.

B.1 Create a network

```
clear all;
close all;

%% hyperparameters

K = 5; % Number of clusters
Nk = 20; % Nodes per cluster

%Alpha and beta parameter for the beta distribution
%If you use this, you need 'eta = betarnd(a,b,K,K); in the for-loop
%a = b = 0.01 is for network type two
%a = 0.01; b = 0.01;

%If you use this setup for alpha and beta, use eta=betarnd(a,b); in the for-
    loop
%This is for network type one
a = [5 0.5 0.5 0.5 0.5;
     0.5 5 0.5 0.5 0.5;
     0.5 0.5 5 0.5 0.5;
     0.5 0.5 0.5 5 0.5;
     0.5 0.5 0.5 0.5 5];
b = [0.5 5 5 5 5;
```

```
      5  0.5  5  5  5;
      5  5  0.5  5  5;
      5  5  5  0.5  5;
      5  5  5  5  0.5];

%% Start generative process
N = Nk*K;

%Make cluster assignment matrix
Z = zeros(N,K);
for k=1:K
Z((k-1)*Nk + (1:Nk),k) = 1;
end

%Make cluster link probabilities

%Use this for the network type two configuration
%eta = betarnd(a,b,K,K);

%Use this for the network type one configuration
eta = betarnd(a,b);
eta = triu(eta);
eta = eta + eta';
eta(1:(K+1):end) = eta(1:(K+1):end)/2;

%Create the final binary network
A = zeros(N,N);
for i=1:N
    for j=1:N
        ci = find(Z(i,:));
        cj = find(Z(j,:));
        A(i,j) = (eta(ci,cj) > rand);
    end
end

%Save the network
save(['A_1_5a_0.5b_intra_extra' '.mat'],'A')
```

B.2 Use of MCMC algorithm

```
#The purpose of this script is to read a network, delete a specified number (or
     numbers) of links and perform an ergmm calculation.
#This ergmm calculation is a MCMC algorithm that will output 4000 samples.

#Load the used libraries (statnet is used because it loads both network and
    latentnet packages)
#R.matlab is for opening and saving matlab files
```

```
library(statnet)
library(R.matlab)

#Determine how many items you want to have unobserved.
del <- c(1,5,10,25,50,75,100,250,500,750,841)

#Set the number of nodes of the network
#29 is the number of nodes for the markov data
ncol <- 29

#Loop over the different delete-values
for(d in del){
    #Save the old values to check accuracy later in matlab
    oldValue <- matrix(0,d,1)
    #Save the old row numbers
    sampleRow <- matrix(0,d,1)
    #Save the old col numbers
    sampleCol <- matrix(0,d,1)

    #Read in file, it saves as variable A
    #The seperate strings will be used to save the output later on
    str <- paste0('markov_binary')
    filetype <- '.mat'
    #Read the matrix, gives a list
    A <- readMat(paste0(str,filetype))
    #Convert the list to a matrix
    A <- matrix(unlist(A, use.names=FALSE), ncol)
    #Convert the matrix to a network
    if(d < (ncol*ncol)){
        A <- as.network.matrix(A, loops=TRUE)
    }else{
        #For some reason, if we delete all nodes, the ergmm function doesn't
            accept loops.
        #Since loops are not available when the network is completely cleared,
            when can leave those out and still run the function
        A <- as.network.matrix(A, loops=FALSE)
    }

    #Calculate the density to save it later on
    dens <- network.density(A)

    #Initiate i for the while loop over d number
    i <- 1

    #Using the random generated data, make those respective nodes Not Available
        (NA)
```

```
    #A while-loop is used here instead of a for-loop because if we skip a for-
        loop with the 'next' function, it will eventually delete less edges than
         you would like
    #The loop might seem redundant or inefficient, but R does not easily and
        readily support a sample of for example 841 different coordinates in
        random order
    while(i <= d){
        sampleColumn <- sample(29,1)
        sampleRoww <- sample(29,1)
        #If this edge is already set to NA, try another one
        if(is.na(A[sampleRoww,sampleColumn])){
            next
        }
        #Save the old values so we can use that later to calculate the accuracy
        sampleRow[i] <- sampleRoww
        sampleCol[i] <- sampleColumn
        oldValue[i,1] <- A[sampleRoww,sampleColumn]
        #Set to NA
        A[sampleRoww,sampleColumn] <- NA
        #Increase i for the while-loop
        i <- i + 1
    }

    #Now calculate the fit of this network
    A.fit <- ergmm(A ~ euclidean(d=2), control=control.ergmm(burnin=20000))

    #Save these two to a .mat file
    #The .mat file should be named rather equal to the opened file
    writeMat(paste0(str,'_R_deltest',d,filetype), Z=A.fit$sample$Z, b=A.fit$
        sample$beta, sampleRow=sampleRow, sampleCol=sampleCol, oldValue=oldValue
        , dens=dens)
}
```

B.3 Calculate accuracy

```
clear all;
close all;

%% Calculating accuracy

%The different numbers of deletions
types = [1 5 10 25 50 75 100 250 500 750 841];

%A counter for the probability cells
l = 1;
%Make a cell for all the probabilities
allProbs = cell(1,size(types,2));
```

```
%Loop over all the types of files
for elm = types
    %Load the file
    load(['markov_binary_R_deltest' num2str(elm) '.mat']);
    %All the probabilities will be saved in this
    prob = zeros(elm,4000);
    %Loop over the samples
    for k=1:size(Z,1);
        z_sample = Z(k,:,:);
        z_sample = squeeze(z_sample);
        for j=1:size(sampleRow,1)
            %eta = beta - ||z_i - z_j||_2
            eta = b(k,:) - norm(z_sample(sampleRow(j),:)-z_sample(sampleCol(j)
                ,:));
            %calculate the probability
            p = 1/(1+exp(-eta));
            %save the probability
            prob(j,k) = (p^(oldValue(j,1)) * (1-p)^(1-oldValue(j,1)));
        end
    end
    %Put the list of probs in a cell and increase the counter
    allProbs{l} = prob;
    l = l + 1;
end

save('markov_binary_R_deltest_output.mat', 'allProbs', 'dens')
```

B.4 Make a boxplot

```
clear all;
close all;

%Specify the file you want to load the output from
load('output_markov_original2.mat');

%Reshape the cells from allProbs
M = allProbs{1};
vector1 = reshape(M.',[],1);
M = allProbs{2};
vector2 = reshape(M.',[],1);
M = allProbs{3};
vector3 = reshape(M.',[],1);
M = allProbs{4};
vector4 = reshape(M.',[],1);
M = allProbs{5};
vector5 = reshape(M.',[],1);
```

```
M = allProbs{6};
vector6 = reshape(M.',[],1);
M = allProbs{7};
vector7 = reshape(M.',[],1);
M = allProbs{8};
vector8 = reshape(M.',[],1);
M = allProbs{9};
vector9 = reshape(M.',[],1);
M = allProbs{10};
vector10 = reshape(M.',[],1);
M = allProbs{11};
vector11 = reshape(M.',[],1);

%Put vectors together and make groups for the boxplot
vectors = [vector1;vector2;vector3;vector4;vector5;vector6;vector7;vector8;
    vector9;vector10;vector11];
%Clear in between in the case the vectors are very large and might take up
%too much RAM
clear vector1 vector2 vector3 vector4 vector5 vector6 vector7 vector8 vector9
    vector10 vector11 M allProbs;
group = [repmat({'1'}, 4000, 1); repmat({'5'}, 20000,1); repmat({'10'},
    40000,1); repmat({'25'}, 100000, 1); repmat({'50'}, 200000,1 ); repmat({'75'
    }, 300000, 1); repmat({'100'}, 400000, 1); repmat({'250'}, 1000000, 1);
    repmat({'500'}, 2000000, 1);
    repmat({'750'}, 3000000, 1); repmat({'841'}, 3364000, 1)];

h = boxplot(vectors,group);
%If you want the outliers to be deleted, set this on
%set(h(7,:),'Visible','Off');
xlabel('Number of unobserved links');
ylabel('Accuracy');

hold on;
%Plot the density line
f = plot(repmat(dens,11,1));
```