Karlsruhe Institute of Technology

Discovery of latent features and clusters based on similarities in brain function

Diplomarbeit
von

Moritz v. Looz

An der Fakultät für Informatik
Institut für Theoretische Informatik

Gutachterin:	Prof. Dr. Dorothea Wagner
Betreuende Mitarbeiterin:	Dipl-Inf. Andrea Kappes
Externer Betreuer:	Prof. Dr. Dr. med Klaas Enno Stephan
Bearbeitungszeit:	01. April 2013 – 01. October 2013

Acknowledgements. First of all, I want to thank Andrea Kappes, Jakob Heinzle and Sudhir Shankar for great and precise supervision, patience and guidance when I came up with enthusiastic half-baked ideas and for more instances of proofreading than I can count.

I want to thank Prof. Klaas Enno Stephan for presenting the problem, initial supervision, generous hospitality and final grading.

Kay H. Brodersen supplied the schizophrenia dataset used for evaluation and participated in the initial supervision, for which I am grateful. He was able to clear up confusion by joining a conversation, just listening for thirty seconds and immediately spotting where we had been talking past each other for the last 15 minutes.

I also want to thank Prof. Wagner for allowing me to write my thesis at her institute and accepting and grading the final work.

The members of `tex.stackexchange.com` deserve thanks for immediately answering even the most complicated LaTeX questions.

Finally, I want to thank my parents for supporting me and for supplying bits of academic wisdom when needed.

Declaration. I declare I have written this thesis by myself and have not used any sources or assistance other than those listed.

Karlsruhe, September 30, 2013

Abstract

In the field of psychiatric diseases, symptoms may not map consistently to underlying brain dysfunctions, leading to the idea to reclassify disease labels based on physiological data [32]. These dysfunctions are not understood and identified completely, making exploratory analysis on brain data necessary. Unsupervised learning and clustering is an important step in this approach. Because more than one dysfunction can be present in one patient, an unbounded feature model is a suitable prior to model the overlaps. We use Bayesian inference on an unbounded latent feature model to obtain multiple clusterings based on similarity data. For that purpose, we extend a model Palla [27] created for binary link prediction. The Bernoulli distribution originally needed to create binary links is replaced with Gaussian noise, making several priors conjugate and significantly improving the sampling speed. A Markov Chain Monte Carlo method is used for the model inversion algorithm. We also present an aggregation function to obtain a point estimate of the posterior distribution. Desorno [6] used magnetic resonance imaging (MRI) to discover the functional connectivity changes in brain function of schizophrenia patients in contrast to a control group. We evaluated the clustering algorithm on this dataset, obtaining partitions which roughly correspond with the diagnosis (NMI 0.20) and medication (NMI 0.21).

Zusammenfassung

Bei vielen psychischen Krankheiten lassen sich sichtbare Symptome nicht klar neurologischen Ursachen zuordnen. Eine Diagnose nach Symptomen ist daher oft langwierig und die Medikamentenwahl braucht mehrere Versuche. In den letzten Jahren kam daher der Wunsch auf, Störungen stärker nach neurologischen Ursachen anstatt nach Symptomen zu definieren [32]. Dabei sind Werkzeuge zur explorativen Datenanalyse notwendig, um in Messdaten von bildgebenden Verfahren Ähnlichkeiten zu erkennen. Diese Arbeit stellt einen Bayesschen Ansatz vor, um Objekte unüberwacht und parallel in verschiedenen Clusterungen zu partitionieren. Dabei werden verschiedene Clusterungen parallel erzeugt, die in ihrer Summe Struktur in den Daten modellieren. Die Anzahl der Clusterungen und der Cluster in ihnen ist nicht vorgegeben, sondern wird während der Inferenz mitgeschätzt. Das generative Modell beruht auf einem von Palla [27] vorgestellten. Wir erweitern es von binären auf reellwertige Matrizen und fügen normalverteiltes Rauschen hinzu. Außer dem erweiterten Anwendungsbereich ist die Samplinggeschwindigkeit des eingesetzten MCMC-Verfahrens dadurch auch schneller, da sich mehrere Verteilungen leichter berechnen lassen. Der Algorithmus wurde auf einem mit funktioneller Magnetresonanztomographie gewonnenen Datensatz getestet, der von Desorno [6] zur Verfügung gestellt wurde und mit Schizophrenie diagnostizierte Patienten sowie eine gesunde Kontrollgruppe enthält. Die von unserem Algorithmus auf diesen Daten gefundenen Clusterungen entsprachen ungefähr dem Krankheitsstatus (Transinformation: 0.2) und der Medikamentenwahl (0.22).

Contents

Glossary

$E^m[i][j]$ proportion of samples in which objects i and j share the same cluster in feature m. 34

H_n harmonic number, $H_n = \sum_k^N \frac{1}{k}$. 14, 39

$M(\text{algorithm}, s)$ memory complexity of algorithm with input s. 25

$T(\text{algorithm}, s)$ time complexity of algorithm with input s. 25

$X^m[i][j]$ average weights between objects i and j in feature m. 34

$\text{TLC}(s)$ Total Link Count, a complexity measure for model instances. $\text{TLC}(s) = \sum_{m=1}^{n_f(s)} n_{cf}(s,m)^2$. 25

n number of objects. 15

$R(i,j)$ similarity of i and j. 16

$T(i,j)$ expected similarity of objects i and j. 16

$c_c^m(s)$ cluster c of feature m of s. 15

$n_{cf}(s,m)$ number of clusters in feature m. 15

$n_{oc}(s,m,c)$ number of objects in cluster c of feature m. 15

$f^m(s)$ feature m of s. 15

$n_f(s)$ number of features in s. 15

$n_{of}(s,m)$ number of objects in feature m. 15

$\mathcal{L}(A|B)$ likelihood of parameter A given observation B. $\mathcal{L}(A|B) = \mathcal{P}(B|A)$. 5

$n(s)$ number of objects in s. 15

$\mathcal{P}(X)$ discrete probability of event X happening. 5

$p(X)$ continuous probability density at point X. 6

$s\backslash(i,m)$ s without object i in feature m. 26

ANOVA analysis of variance, a statistical tool to determine how diverse samples from different populations are. 63

average NMI reduction of NMI matrices obtained when comparing a sample with a ground truth. 57

BP Balanced Purity, a clustering quality measure. 54

Chlorpromazine well-known antipsychotic drug, used as benchmark to describe doses of more recent drugs. 63

conjugate prior prior distribution resulting in a posterior with the same distribution family. 8, 46

CRP Chinese restaurant process, a stochastic process resulting in a partition of objects. 12

dynamic causal modelling a statistical method used for the analysis of functional neuroimaging data. 61

fMRI functional magnetic resonance imaging. Using MRI to measure aspects of brain function. 3

IBP Indian buffet process, a stochastic process resulting in a feature assignment on objects. 13

inverse transform sampling a method to sample from probability distributions with an invertible cumulative distribution function. 12

MCMC Markov chain Monte Carlo sampling, a family of methods to draw samples from probability distributions without access to the density function. 9

MiB Mebibyte, $2^{2 \cdot 10}$ byte. 46

model instance a tuple of (Z, C, W, b), resulting in an expected similarity matrix $T(i, j)$. 15

NMI Normalized Mutual Information, a clustering quality measure. 56

1. Introduction

Unsupervised learning is an important field in machine learning and seeks to recover latent structures in input data. It is often used for exploratory analysis.

One of the challenges in unsupervised learning is choosing the correct amount of structure for a given dataset. For example, how many clusters are assumed when clustering? In a Bayesian setting, this takes the form of finding a good prior distribution [8]. In [1], the culinary themed *Chinese restaurant process* (CRP) is presented. It is a stochastic process resulting in a distribution over partitions. The number of clusters in these partitions is only bounded by the number of objects. This process was generalized by [12] to the *Indian Buffet Process* (IBP). In contrast to the disjoint clusters of the CRP, the IBP results in a distribution over feature participation. Each object can have zero, one or several of infinitely many features.

Especially with high-dimensional data, a single disjoint clustering may not capture the underlying structure effectively. Figure 1.1 shows an example where the data is best fit with overlapping clusters. We use a latent feature model, with each feature representing one partition. As the number of features and clusters are unbounded, they are inferred along with the actual clusterings.

1.1 Motivation

This work was in part motivated by an idea within psychiatry to reclassify disorders based on physiological data [32]. As of now, psychiatric diseases are classified by their symptoms. These classifications may not accurately map to the underlying disease mechanisms and there is indeed some evidence that they do not. For example, genes influencing calcium channels have been found to be associated with a range of psychiatric disorders [29].

Functional magnetic resonance imaging (fMRI) is a method to gather information about changes in brain connectivity. We describe a model-based inference algorithm to discover latent features and clusters among human subjects based on similarities of their fMRI data. Some of these hidden features may correlate with natural clusterings like age and gender, others with psychiatric illnesses or similar dysfunctions. These mechanisms may overlap, making a latent feature model suitable to model underlying dysfunctions. We test this generative model on a schizophrenia dataset provided by Desorno [6].

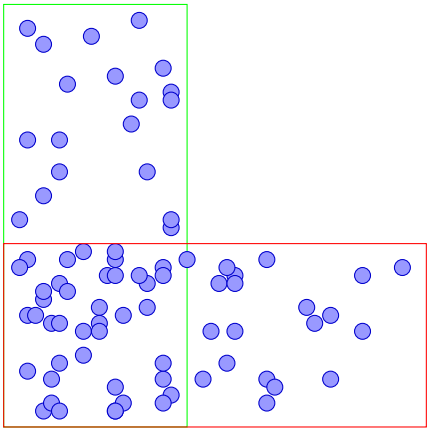


Figure 1.1: These data points in R^2 are better modeled with two overlapping clusters instead of two or three disjoint ones.

1.2 Related Work

The generative model presented in Section 3 is an extension of a latent feature model introduced by Palla [27]. We use a Markov chain Monte Carlo (MCMC) method for inference in Chapter 4. Markov chain Monte Carlo methods are commonly used to sample from probability distributions when a closed form solution is not available. A good introduction is given by Neal [24]. The generative model for feature presence is a so called Indian buffet process (IBP), which was presented by Griffiths et al. [12]. Wood [35] used an unbounded latent feature model with the Indian buffet process to model and infer stroke locations from fMRI data. Niu [26] also use a combination of the Indian buffet process and the Chinese restaurant process to cluster objects, but focusing on different clustering views.

1.3 Outline

We use a Bayesian approach to clustering, defining a generative model which gives us a prior distribution and likelihood. In Section 2, the underlying method of Gibbs samplers and our use of them is discussed, the generative model is presented in Section 3. The main algorithm and implementation are described in Section 4, followed by an evaluation with synthetic and fMRI data in Section 5. Finally, Section 6 concludes this work with a discussion and an outline of future work. A glossary of the notation can be found before this introduction.

2. Preliminaries

In this work, we use Bayesian inference to discover unbounded latent features and clusters in fMRI data of human subjects. Since the posterior probability distribution is difficult to derive analytically, we use a Markov chain Monte Carlo (MCMC) method to generate samples from it. Readers already familiar with Markov chains, Monte Carlo methods and Gibbs samplers may skip to Chapter 3 where the actual model is described. For the rest of us, we will give a short introduction to Bayesian updating in Section 2.1, describe MCMC methods in Section 2.3 and mention Gibbs Samplers in 2.3.3. While these introductions will be sufficient to understand the use of these concepts in our work, use of appropriate textbooks is recommended for a thorough and solid comprehension. "Information theory, inference and learning algorithms" from David MacKay [18] is an excellent introduction. We use prior distributions given by the Chinese restaurant process and the Indian buffet process. These stochastic processes are presented in Sections 2.5.1 and 2.5.2.

2.1 Bayesian Inference

Bayes' theorem is central to the field of probability theory. It describes how to get from an initial *prior* probability distribution to an updated *posterior* distribution when encountering new evidence. We will give some examples and describe the relation to our algorithm. Many introductions to Bayes theorem can be found online, for example the "Intuitive Explanation of Bayes' Theorem" by Yudkowsky [36]. For further study, the textbook "Probability Theory - the logic of Science" by E.T. Jaynes [14] is recommended.

In its simplest form, Bayes' theorem can be given as:

$$\mathcal{P}(B|A) = \frac{\mathcal{P}(A|B)\mathcal{P}(B)}{\mathcal{P}(A)} \tag{2.1}$$

When observing outcome A, the probability of cause B is proportional to the probability of A conditioned on B multiplied by the prior probability of B. The conditional probability of an outcome A given a cause B is also called the *likelihood function*:

$$\mathcal{L}(B|A) = \mathcal{P}(A|B)$$

2.1.1 Burglar Example

A popular example is a burglar alarm which can be set off by an actual burglar or by something else, perhaps an earthquake[28]. When observing a burglar alarm going off, what is the probability of an actual burglary being in progress? The answer depends on the probability of burglaries happening at a given point in time, the probability of an alarm given a burglary and the overall probability of an alarm.

$$\mathcal{P}(\text{burglary}|\text{alarm}) = \frac{\mathcal{P}(\text{alarm}|\text{burglary})\mathcal{P}(\text{burglary})}{\mathcal{P}(\text{alarm})} \tag{2.2}$$

An alarm can be set off by a burglar or by other things. We use $\sim$ for the absence of an event:

$$\mathcal{P}(\text{alarm}) = \mathcal{P}(\text{alarm}|\text{burglary})\mathcal{P}(\text{burglary}) + \mathcal{P}(\text{alarm}|\sim\text{burglary})\mathcal{P}(\sim\text{burglary}) \tag{2.3}$$

Let $\mathcal{P}(\text{burglary}) = 0.001 = 10^{-3}$, $\mathcal{P}(\text{alarm}|\text{burglary}) = 0.9$ and $\mathcal{P}(\text{alarm}|\sim\text{burglary}) = 0.004 = 4 \cdot 10^{-3}$. It follows:

$$\mathcal{P}(\text{burglary}|\text{alarm}) = \frac{0.9 \cdot 0.001}{0.9 \cdot 0.001 + 0.004 \cdot 0.999} = \frac{0.0009}{0.004896} \approx 0.183 \tag{2.4}$$

Even though the alarm notices 90% of burglars, the probability of a burglary given an alarm is only about 18%.

What happens if the rate of burglaries rises? Let $\mathcal{P}(\text{burglary}) = 0.005$.

$$\mathcal{P}(\text{burglary}|\text{alarm}) = \frac{0.9 \cdot 0.005}{0.9 \cdot 0.005 + 0.004 \cdot 0.995} \approx 0.53 \tag{2.5}$$

The properties of the alarm itself were unchanged, the rise in the posterior probability of a burglary reflects the rise of the prior probability.

For the second example, let us assume a baseline rate of burglaries and an overzealous alarm. Let $\mathcal{P}(\text{burglary}) = 0.001$ and $\mathcal{P}(\text{alarm}|\sim\text{burglary}) = 0.01$.

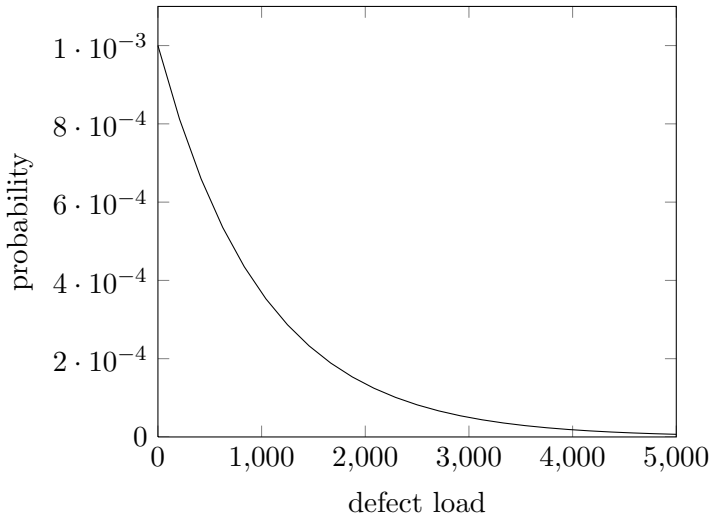
$$\mathcal{P}(\text{burglary}|\text{alarm}) = \frac{0.9 \cdot 0.001}{0.9 \cdot 0.001 + 0.01 \cdot 0.999} = \frac{0.0009}{0,01089} \approx 0.08 \tag{2.6}$$

With many more fake alarms, the probability of a burglary given an alarm shrinks.

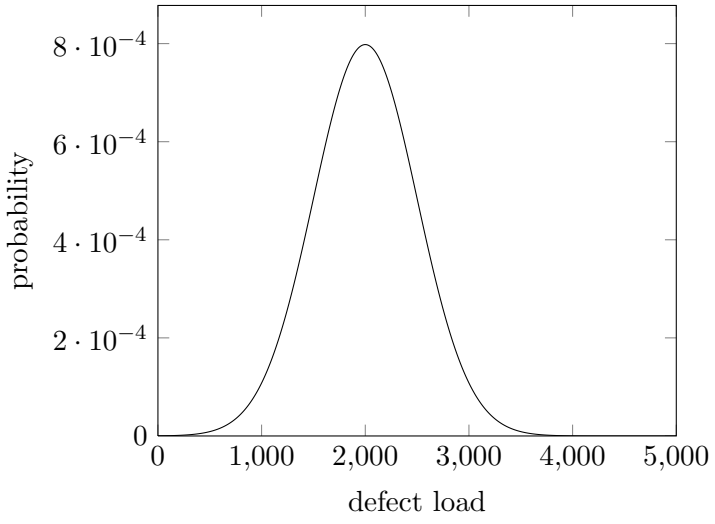
Something we used implicitly before defining it is the *probability mass function* of a random variable. When considering a discrete random variable, its probability mass function $f(x)$ gives the probability of the variable taking on value x. The sum of the probability mass function is always 1. If the random variable is continuous, the assorted function is called the *probability density function*, whose integral will be 1. The probability density at x is denoted by $p(x)$. *Likelihood mass functions* and *likelihood density functions* are defined analogously.

2.1.2 Mechanical Defect Example

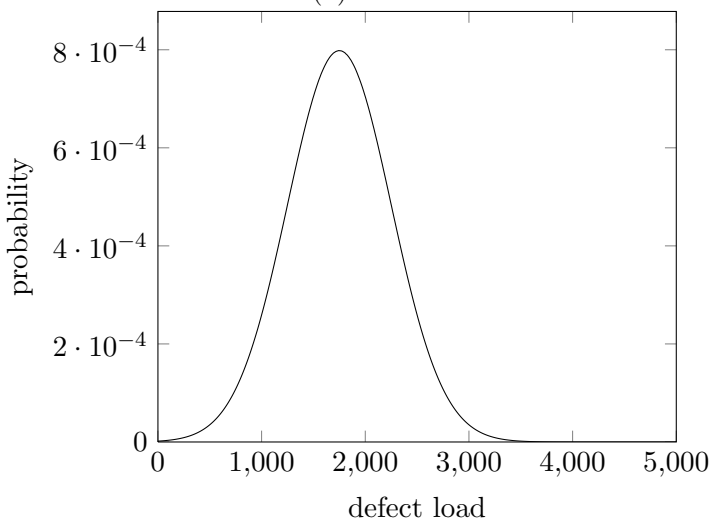
Forsooth Heavy Industries produces widgets. Each widget can have small structural defects. These defects are not severe and a widget only needs to be scrapped if it has a large number of defects. From prior experience, Forsooth Heavy Industries knows the defect load to be exponentially distributed, see Figure 2.1(a).



(a) Prior



(b) Likelihood



(c) Posterior

Figure 2.1: Estimating amount of material defects. The prior is exponentially distributed, the likelihood and posterior are Gaussian distributions. The posterior is the normalized product of prior and likelihood.

A test has been developed to measure the defect load in a given widget. Unfortunately, the defect load is difficult to measure and the true value may be different from the measurement result. Measurement noise is usually assumed to be normally distributed.[1] In this example a defect load of 2000 is measured, the likelihood function of the true defect load is given in Figure 2.1(b).

By multiplying the prior distribution with the likelihood, we obtain the *posterior distribution*, shown in Figure 2.1(c). A probability measure needs to integrate to 1, which is why we included the scaling factor of 6522.3 in Figure 2.1(c). For clearer and prettier equations, one often omits the scaling factor and writes "Probability density function $p_A(x)$ is proportional to probability distribution $p_B(x)$" or even shorter: "$p_A(x) \propto p_B(x)$".

When using a normal distribution as prior and likelihood, the posterior probability is also Gaussian but with a lower variance. This property has given rise to the quip "A Bayesian is one who, vaguely expecting a horse, and catching a glimpse of a donkey, strongly believes he has seen a mule."[16]. This joke illustrates why the choice of priors is important: If one vaguely expects a horse, but definitely an existing animal, the prior probability density falls off sharply when encountering beings not known as animals. Even when expecting a horse, an elephant is vastly more probable than an elephant-horse-chimera. In this case, a Gaussian prior is not advisable, as values closer to the mean are always more probable in a Gaussian distribution than more distant values.

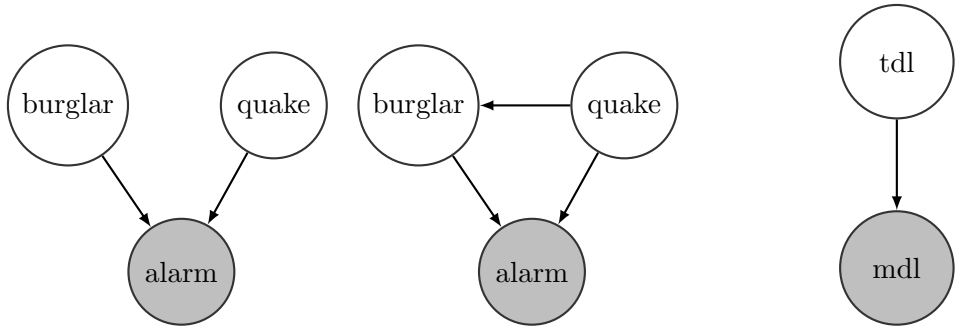
When the prior and likelihood result in a posterior distribution of the same family, the prior is called a *conjugate prior* [30] for the likelihood. This simplifies updates and sampling, as the posterior distribution can be parametrized. A Gaussian prior is conjugate to a Gaussian likelihood.

2.2 Bayesian Networks

In both examples in the previous section, observable events such as an alarm or a measured defect load were caused by hidden events, the earthquake/burglar or the true defect load. Many applications have even more layers of events influencing each other. One could introduce a major tectonic fault influencing the probabilities of earthquakes, natural disasters attracting opportunistic burglars or other hidden causes for the already hidden causes. To see at a glance which elements in a model influence what other elements, a *graphical model* is helpful. A Bayesian network is shown as a directed acyclic graph, with the nodes representing random variables and the edges representing conditional probabilities. Figure 2.2(a) shows the model of our burglar example: An alarm can be caused by a burglary or an earthquake, which are a priori independent. If we want to model opportunistic burglars striking during earthquakes, we add another link from earthquakes to burglaries as shown in Figure 2.2(b). Note that the edge leads from quakes to burglaries: Earthquakes cause burglaries, they are not triggered by burglaries. Figure 2.2(c) shows the defect load example.

Inversion. When given a model of conditional and prior probabilities, deducing the hidden causes from observations is called *inverting* the model. In the burglar example, one would start with an alarm going off or not and estimating probabilities for a burglary or an earthquake happening. The result of a Bayesian inversion is a posterior probability distribution for each of the hidden causes. Often one is interested in *point estimates*, values giving a "best guess" of the hidden variables. The mode and the mean of a distribution are common ways to get point estimates.

[1] A normal distribution is a popular choice because of the *central limit theorem*, which states that the sum of many independent random variables is normally distributed.



(a) Burglar or Earthquake (b) Opportunistic Burglar (c) **t**rue and **m**easured **d**efect **l**oad

Figure 2.2: Graphical models of the burglar and mechanical defect example. 2.2(a) shows independent burglars and earthquakes, both influencing the alarm probability. 2.2(b) shows an opportunistic burglar influenced by earthquakes. In 2.2(c), the true defect load influences the measured defect load. In each example, the observed outcome is marked in light gray.

2.3 Markov Chain Monte Carlo

For more complex models, the posterior distribution may have no analytical form or be too complicated to derive. If one can generate samples from the posterior distribution, it can be approximated. With an increasing number of samples, the distribution of samples approaches the true posterior. Several methods exist to generate samples from a distribution. The family of *Metropolis-Hastings* methods is popular in Bayesian inference and uses Markov chains. In this section, we give a brief introduction to Markov chains, Metropolis-Hastings algorithms, slice sampling and *Gibbs sampling*, a special case of a Metropolis-Hastings algorithm.

2.3.1 Markov Chain

A Markov process is a stochastic process where the probabilities of the next state only depend on the current state, but not on the previous history of states. This property of memorylessness is called the *Markov property*.

A *Markov chain* is a Markov process with discrete time steps. The states may be either discrete or continuous. One common example for discrete state Markov chains are board games played with dice: The next state only depends on the current state and the dice rolls, but not the history of states. Particularly well-behaved Markov chains have a single *limiting* probability distribution they converge to when the number of iterations is high enough, irrespective of the starting distribution. Such a distribution is called a *stationary* distribution.

If we can construct a Markov chain whose limiting distribution is the desired posterior, we can iterate through it sufficiently often until the chain distribution is sufficiently close to the stationary distribution and afterwards use the chain states as samples.

Numerous methods exist to judge whether a given chain has converged sufficiently[4]. We use multiple parallel chains and compare their cumulative means and the inter-chain variance.

Since consecutive chain states are autocorrelated, often a process called *thinning* is used: every kth sample is used and the rest discarded.

2.3.2 Metropolis-Hastings

A Metropolis algorithm gives samples from a probability distribution A if a function f proportional to the probability density function f_A is known.

The algorithm starts with an initial sample x_0. Each subsequent sample x_t is then generated using two steps:

1. A proposal x^* for the next sample is created from the current sample x_t using a *proposal distribution* $Q(x^*|x_t)$. In the Metropolis algorithm, Q must be symmetric, i.e. $Q(x|y) = Q(y|x)$, $\forall x, y$.
2. The *acceptance ratio* r is calculated: $r = f(x^*)/f(x_t) = f_A(x^*)/f_A(x_t)$.
3. If x^* has a higher probability than x_t, it is automatically accepted. If not, it is accepted with probability r.
4. If x^* was accepted, set x_{t+1} to x^*. Else, set x_{t+1} to x_t.

The first few samples still reflect the choice of the initial sample, which is why they are usually discarded. The time needed until the samples reflect the stationary distribution is called the *burn-in period*. A chain which approaches the stationary distribution quickly is called *fast-mixing*.

The variance of the proposal distribution determines the mixing behavior. If the proposal x^* is very similar to x_t, the acceptance ratio is high, but the chain takes long to move away from the starting point x_0. A proposal which is very different will lead to larger jumps when accepted, but the acceptance ratio will be smaller. The jumping width is often calibrated to target an acceptance ratio of 0.25 to 0.5. If the distribution A is multivariate with a high number of variables, calibrating the jumping width can be challenging.

The Metropolis algorithm was generalized by Hastings [13] who replaced the acceptance ratio with an *acceptance distribution*, thus creating the Metropolis-Hastings algorithm. In this generalization, the proposal distribution Q no longer needs to be symmetric as long as the acceptance distribution is chosen correctly.

2.3.3 Gibbs Sampling

Gibbs sampling is a method to sample from a multivariate distribution when only conditional probabilities are known and can be sampled from. This can be useful for Bayesian networks, since they are formulated as a set of conditional probabilities.

The Metropolis algorithm requires a density function f proportional to the probability density function of the distribution to sample from. In hierarchical Bayesian models, calculating f requires several integrals difficult to evaluate.

Gibbs sampling does not require f, as it moves in the sample space one variable at a time. Let $x_t = (x_t^{(1)}, x_t^{(2)}, ..., x_t^{(k)})$ be the current sample. Sample x_{t+1} is created in k steps, each variable $x^{(j)}$ is drawn from the distribution $p(x^{(j)}|x_{t+1}^{(1)}, x_{t+1}^{(2)}, ..., x_{t+1}^{(j-1)}, x_t^{(j+1)}, ..., x_t^{(k)})$. These individual sampling steps may in turn include Metropolis-Hastings or other sampling methods.

This algorithm generates a Markov chain whose limiting distribution is equal to the joint distribution of the variables. A more rigorous introduction to Gibbs samplers is given by Suess and Trumbo [33].

Example. We revisit the graphical model of the burglar example introduced in Figure 2.2(a). Let $\mathcal{P}(\text{burglary}) = 0.001$, $\mathcal{P}(\text{quake}) = 0.008$, $\mathcal{P}(\text{alarm}|\text{quake, burglary}) = 1$, $\mathcal{P}(\text{alarm}|\text{quake, } \sim \text{burglary}) = 0.5$, $\mathcal{P}(\text{alarm}| \sim \text{quake,burglary}) = 0.8$ and $\mathcal{P}(\text{alarm}| \sim \text{quake, } \sim \text{burglary}) = 0$.[2] Assume we just observed an alarm and want to infer the probability of a burglary by using Gibbs sampling. Burglaries and earthquakes are the hidden random variables. We pick an initial sample $x_0 = (\sim\text{quake, } \sim\text{burglary})$.

1.
   - Sample $x_1^{(1)}$ from $p(\text{quake}|\text{alarm, } \sim\text{burglary})$. $\mathcal{P}(\text{quake}|\text{alarm, } \sim\text{burglary}) = 1$. $x_0^{(1)} = \text{quake}$.
   - Sample $x_1^{(2)}$ from $p(\text{burglary}|\text{alarm, quake})$. $\mathcal{P}(\text{burglary}|\text{alarm, quake}) \approx 0.002$. When given an alarm and an earthquake, a burglary is more likely than the prior probability, but still very unlikely. This Markov chain would stay at this state for many samples, which is called *slow mixing*. For the sake of this example, we pick the improbable choice of $x_1^{(2)} = \text{burglary}$.
   - x_1 is thus (quake, burglary).
2.
   - Sample $x_2^{(1)}$: $\mathcal{P}(\text{quake}|\text{alarm, burglary}) \approx 0.02$ Set $x_2^{(1)}$ to $\sim$quake.
   - Sample $x_2^{(2)}$: $\mathcal{P}(\text{burglary}|\text{alarm, } \sim\text{quake}) = 1$. Set $x_2^{(2)}$ to burglary.
   - x_2 is ($\sim$quake, burglary).

Because of the very small prior probabilities, this chain mixes slowly and has a high degree of autocorrelation. In terms of the Metropolis algorithm, the acceptance ratio is very small. Even though, the distribution of samples eventually approaches the posterior distribution.

Auxiliary Variables. In some cases, even the conditional probability distributions $p(x^{(j)}|x_{t+1}^{(1)}, x_{t+1}^{(2)}, ..., x_{t+1}^{(j-1)}, x_t^{(j+1)}, ..., x_t^{(k)})$ are difficult to sample from. One workaround consists of sampling auxiliary variables y from the current state and afterwards sampling $x^{(j)}$ from the other variables and y. The conditional distributions are $p(y|x_{t+1}^{(1)}, x_{t+1}^{(2)}, ..., x_{t+1}^{(j-1)}, x_t^{(j)}, x_t^{(j+1)}, ..., x_t^{(k)})$ and $p(x^{(j)}|x_{t+1}^{(1)}, x_{t+1}^{(2)}, ..., x_{t+1}^{(j-1)}, x_t^{(j+1)}, ..., x_t^{(k)}, y)$. The auxiliary variables y are discarded after sampling $x^{(j)}$. This approach is more thoroughly introduced in Algorithm 8 of [23].

2.3.4 Slice Sampling

Slice sampling is another MCMC algorithm introduced by Neal [25]. It requires a function f proportional to the probability density function of the distribution to sample from. The name is derived from a useful visualization: One imagines sampling from a horizontal slice of the region under the probability density curve. Let x_0 be the initial sample. Three steps are required for every subsequent sample:

1. Evaluate $f(x_t)$ and pick a uniform random number y in $(0, f(x_t)]$.
2. Find horizontal slice of values x^* with $f(x^*) \geq y$.
3. Sample x_{t+1} uniformly from horizontal slice.

Finding the bounds of the horizontal slice can be difficult and may require several evaluations of f in a stepping-out procedure. If the horizontal slice is unbounded in one direction, the algorithm aborts with an error. This can happen in a monotonically decreasing function on $\mathbb{R}^+$ only implicitly bounded by zero.

[2] In this example, there are eight times as many earthquakes as burglaries in a given house. While this seems like a very harmonic or tectonically active place, each earthquake affects many houses, while burglaries usually affect only one.

2.4 Other sampling methods

Standard algorithms exist to generate samples for most well-known distributions if uniformly distributed random numbers are available. We used the Box-Muller-transform to sample from normal distributions.[3].

2.4.1 Inverse Transform Sampling

When for a probability distribution A the cumulative distribution function $\mathrm{cdf}_A(x)$ is known and can be inverted, samples can be generated using inverse transform sampling. A random number r between 0 and 1 is drawn from a uniform distribution, $\mathrm{cdf}_A^{-1}(r)$ is then a sample from distribution A. In practice, the inversion or even the cumulative distribution function itself is often unknown for distributions of interest. We discretize the probability density function by choosing a range where most of the probability mass is contained and evaluating it at fixed intervals d. The cumulative distribution function of a discrete distribution is a simple partial sum and straightforward to invert. $\mathrm{cdf}_{A-\mathrm{discrete}}^{-1}(r)$ is used to approximate a sample from A. Accuracy increases with smaller intervals, with $A-\mathrm{discrete}$ approaching A for $d \to 0$. The time complexity depends on the range and the step size d.

$$\mathrm{Time}(\mathrm{discretizedInverseTransform}, \mathrm{range}, d) \in O(\frac{\mathrm{range}}{d})$$

For example, to sample from a Gaussian distribution between -10 and 10 and discretization interval 0.01, we need $20/0.01 = 2000$ evaluations of the probability density function.

2.5 Stochastic Processes

In this section we present two stochastic processes resulting in prior distributions. The Indian buffet process presented in Section 2.5.2 results in a distribution over feature presence while the Chinese restaurant process in Section 2.5.1 gives us a distribution over partitions. These prior distributions are especially apt for use in Gibbs sampling as they are available in the form of conditional probabilities.

We also use the gamma distribution family, the Gaussian distribution family and the inverse gamma distribution family as priors for single parameters.

2.5.1 Chinese Restaurant Process

The Chinese restaurant process is a stochastic process to partition a number of objects into an unbounded number of clusters. It is related to a Dirichlet distribution and was initially presented by Aldous [1]. The name is derived from an analogy used in its description: One imagines a Chinese restaurant with an infinite number of tables, each table being of potentially infinite size. When a new customer enters the restaurant, he has to decide whether to sit next to previous customers or at a new table. In the one-parameter variant, these probabilities are controlled by a parameter called λ.[3] The set of tables (clusters) is given by B and the number of customers (objects) already sitting at a table t is denoted by $|t|$. An existing table t is chosen with a probability proportional to the number of customers seated there:

$$\mathcal{P}(\mathrm{table} = t) = |t| / (\lambda + \sum_{\mathrm{table}\ b \in B} |b|)$$

The probability of choosing a new table is $\lambda / (\lambda + \sum_{\mathrm{table}\ b \in B} |b|)$.

[3] Mostly called α in the literature, we call it λ to avoid confusion with the parameter of the Indian buffet process to follow.

Example. For example, let there be an empty Chinese restaurant with λ set to 1. Alice is the first customer to enter, choosing a new table with probability $\lambda/(\lambda+0) = 1$. After a while, Bob comes in. The probability of Bob choosing a new table is $\lambda/(\lambda+1) = 0.5$. In this example, Bob chooses to sit next to Alice, so they can talk to each other. Charlie enters shortly after, sitting at a new table with probability $\lambda/(\lambda+2) = 1/3$. For the purpose of this example, he sits at a new table. Finally, Dora enters and now has three tables to choose from: She could either sit at the table with Bob and Alice, she could sit next to Charlie, or sit at a new table. She sits next to Bob or Alice with a probability of $2/(3+\lambda) = 0.5$, next to Charlie with a probability of $1/(3+\lambda) = 0.25$ and at an empty table with $1/(3+\lambda) = 0.25$.

Distribution Properties. The Chinese restaurant process induces a partition on objects: Customers correspond to objects, each table corresponds to one cluster. The order in which customers sit around each table is ignored. The order in which customers enter does not influence the final probabilities of the seating. The probability of a particular partition is shown in Equation 2.7, where B_n is a random variable over partitions with n objects and B is a specific partition. In contrast to the family of probability distributions also written with Γ, the Γ in Equation 2.7 denotes the gamma function, a generalization of the factorial.

$$\mathcal{P}(B_n = B) = \frac{\Gamma(\lambda)\lambda^{|B|}}{\Gamma(\lambda+n)} \prod_{b \in B} \Gamma(|b|) \tag{2.7}$$

The expected number of tables for n objects is:

$$\sum_{k=1}^{n} \frac{\lambda}{\lambda+k-1} \tag{2.8}$$

2.5.2 Indian Buffet Process

The Indian buffet process is a generalization of the Chinese restaurant process, presented by Griffiths et al [12]. The restriction of each object being assigned to exactly one cluster is relaxed. Instead, each object can have zero, one or several of infinitely many features. One imagines an Indian buffet with an infinite number of dishes. Each customer can eat from a number of dishes and does so proportionally to their popularity.

The number of dishes each customer chooses from is governed by a parameter α.

The first customer chooses Poisson(α) new dishes, untried by everyone.[4] Each subsequent customer chooses some of the dishes others have already tried and some new ones. The ith customer eats from dish m with probability $|m|/i$, where $|m|$ is the number of times dish m has been tried by other customers. Since m as at most $i-1$, the probability to eat from a tried dish is always below 1. Afterwards, the ith customer tries Poisson(α/i) new dishes.

Distribution Properties. The Indian buffet process parametrized with α and n induces a probability distribution on features. As with the Chinese restaurant process, the probability of each feature distribution is independent of the order in which objects are considered. For each object i, Poisson(α/i) new features are added. The sum of Poisson distributions is again a Poisson distribution, the shape parameter being the sum of parameters. Thus the number of resulting non-empty features follows a Poisson distribution:

[4] The Poisson distribution is often used to model the sum of discrete rare events, such as Prussian cavalrymen dying of horse-kicks, see [34].

$$f \propto Poisson(\sum_{i=1}^{n} \frac{\alpha}{i}) \tag{2.9}$$

The expected number of features is $\alpha \sum_{i=1}^{n} 1/i = \alpha H_n$, where H_n is the harmonic number. The harmonic number grows about as fast as the natural logarithm: $H_n \in O(\log n)$. Therefore, the expected number of features is linear in α and logarithmic in n:

$$\mathbb{E}(f) \in O(\alpha \log n) \tag{2.10}$$

2.5.3 Use of Methods in this Thesis

We first define a generative model in Section 3 using, among others, the Indian buffet process and the Chinese restaurant process. This model gives us a prior distribution over model instances and a likelihood function for the solutions given input data. With the prior distribution and the likelihood function, we thus obtain a posterior probability distribution. Since this posterior has no closed form solution, we use a MCMC sampling method to sample from it.

3. Generative Model

The model we use is an extension of a model developed by Palla [27]. We first discuss their model, which will be called *base model*. Afterwards, we present our extension in Section 3.2.

3.1 Base Model

The base model was developed to model relationship networks. The observable events are whether two objects are connected or not, given in the form of a binary connectivity matrix R. The hidden (also called latent) variables are a potentially infinite set of features in which objects can participate or not. Within each feature, objects are clustered In each feature, a weight matrix describes the relationships of the feature's clusters.

Given n objects and k features, the model consists of:

- one $n \times k$ *feature matrix* Z indicating whether object i has feature m, for all $1 \leq i \leq n$, $1 \leq m \leq k$;
- one $n \times k$ *cluster matrix* C showing which cluster in feature m object i is assigned to, if $Z(i, m) = 1$;
- k *weight matrices* W^m describing relationships between the clusters in feature m;
- a bias term $b \in \mathbb{R}$.

An object i will only be clustered in feature m if it participates in that feature. A tuple $s = (Z, C, W, b)$ will be called a *model instance*. When iterating through features, we will use the variable m, while $n_f(s)$ denotes the number of features a solution s has. The number of clusters in feature m is indicated by $n_{cf}(s, m)$. The total number of objects is $n(s)$. In summary, $Z \in \{0, 1\}^{n(s) \times n_f(s)}$ and $C \in \mathbb{N}^{n(s) \times n_f(s)}$, $W^m \in \mathbb{R}^{n_{cf}(s,m) \times n_{cf}(s,m)}$ for $1 \leq m \leq n_f(s)$. When no model instance is defined, the number of objects is referenced by n.

Within a feature $f^m(s)$ of instance s, $c_c^m(s)$ is the cluster c of feature m. The number of objects in cluster c of feature m is denoted by $n_{oc}(s, m, c)$, while $n_{of}(s, m)$ describes the same for feature m.

All objects of a feature participate in its clustering and hence $\sum_{c=1}^{n_{cf}(s,m)} n_{oc}(s, m, c) = n_{of}(s, m)$ for every feature m.

3.1.1 Variables and Priors

Z. The binary feature matrix Z with dimensions $n \times k$ is generated using the Indian buffet process (IBP). k is not known in advance and may be zero or even higher than n. As a result of the IBP, Z is usually sparse, with the majority of features only active in a small subset of the objects.

Alpha. In the Indian buffet process, the value of α governs the number of features an object participates in. We kept the prior distribution used by Palla [27] and use a gamma prior: $\alpha \propto \Gamma(k_\alpha, \theta_\alpha)$. The shape parameter k_α and scale parameter θ_α were set to 1.

C. After Z is generated, each object needs to be assigned to a cluster for each feature it participates in. This is done using the Chinese restaurant process: The parameter λ controls the tendency to create new clusters vs. choosing existing ones. The probability of choosing a new cluster is $\lambda/(\lambda + \sum_{\text{cluster } l \in m} |l|)$, where $|l|$ is the number of objects in cluster l. An existing cluster c is chosen with probability $|c|/(\lambda + \sum_{\text{cluster } l \in m} |l|)$.

Lambda. The value of λ governs the number of clusters in each feature, created by the Chinese restaurant process. λ is also sampled from a gamma distribution: $\lambda \propto \Gamma(k_\lambda, \theta_\lambda)$.

W. A weight matrix W^m is created for each feature. It models a connection between every pair of clusters: $W^m(s) \in \mathbb{R}^{n_{cf}(s,m) \times n_{cf}(s,m)}$. Each weight is sampled from a Gaussian distribution: $W^m_{ij}(s) \propto \mathcal{N}(\mu_w, \sigma_w)$.

Bias. The bias term is also sampled from a Gaussian distribution: $b \propto \mathcal{N}(\mu_b, \sigma_b)$.

3.1.2 Observation matrix R

The hidden variables (Z, C, W, b) are combined to form the link probability. To obtain the probability for two objects i and j being connected, all features are considered in which both objects participate. In each feature m, the weight between the cluster of i (c_i^m) and the cluster of j (c_j^m) is summed. Finally, the bias b is added to the sum and a sigmoid function $\sigma(x) = 1/(1 + e^{-x})$ applied to obtain a result between 0 and 1. For each pair of objects i and j, we apply a Bernoulli distribution:

$$\mathcal{P}(R(i,j) = 1 | Z, C, W, b) = \sigma(\sum_{m=1}^{k} Z(i,m) Z(j,m) W^m(c_i^m, c_j^m) + b). \tag{3.1}$$

These probabilities are calculated independently for every pair (i, j). Like the burglar state and the earthquake state in the burglary example resulted in a Bernoulli distribution over alarm states, the random variables in the model of Palla [27] result in a distribution over binary matrices R.

Figure 3.1 shows the graphical model. We will omit the hyperparameters for the prior distributions in later sketches. The observation matrix R is marked in light gray.

T. The sum $\sum_{m=1}^{k} Z(i,m) Z(j,m) W^m(c_i^m, c_j^m) + b$ is an important intermediate step to generate R, it already determines the distribution on R_{ij}. We call the matrix of these sums $T \in \mathbb{R}^{n \times n}$:

$$T(i,j) = \sum_{m=1}^{k} Z(i,m) Z(j,m) W^m(c_i^m, c_j^m) + b$$

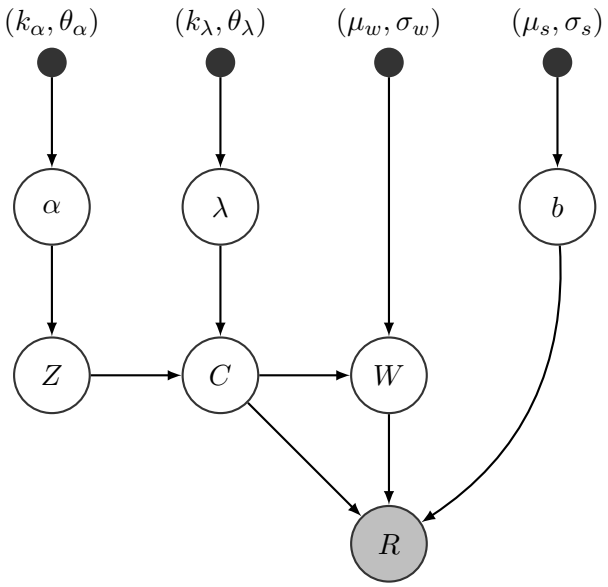


Figure 3.1: Base model of Palla. α, λ are drawn from a Γ distribution, b from a Gaussian distribution. Feature matrix Z is generated by IBP(α). The cluster matrix C is generated by the CRP and depends on Z and λ. The weight matrix W is sampled from a Gaussian distribution. For each pair of objects (i, j), the result $R(i, j)$ is drawn from a Bernoulli distribution.

Likelihood. When inverting the model, the likelihood is of interest. The likelihood $\mathcal{L}(s|R)$ of a model instance s given R is the product of edge probabilities. Since the distribution on R is also determined by T, we can write:

$$\mathcal{L}(s|R) = \mathcal{L}(T|R) = \mathcal{P}(R|T) = \prod_{i=1}^{n}\prod_{j=1}^{n} \sigma(T(i,j))^{R(i,j)} \cdot \big(1 - \sigma(T(i,j))\big)^{1-R(i,j)}$$

3.2 Extension to Real-Valued *R*

In many applications, connections between objects are not binary and forcing them into a binary form by thresholding might cause precision losses. To model non-binary links, we extended the base model from binary matrices to real valued matrices. This enables us to use the model for similarity matrices, which can be created from many kinds of input data.

In our adaption, R is a real valued matrix. Instead of using a sigmoid function and creating Bernoulli distributions, we generate a Gaussian distributions for each $R(i, j)$ with $T(i, j)$ as the mean.[1] The Gaussian distribution can be understood as physiological or measurement noise. We add an estimation for the noise variance σ to the model: $s =(Z, C, W, b, \sigma)$. A graphical model is shown in Figure 3.2. The noise is independent for each value $R(i, j)$.

Since we used normally distributed noise, the final probability distribution for each link is a Gaussian distribution whose mean is again the sum of Gaussian distributions – the weights and the bias. This simplifies sampling since now bias and weights have *conjugate priors*. The other conditional probabilities are unchanged.

[1] A graph analogy comes to mind: The base model infers clusters from the adjacency matrices of unweighted graphs, we extend it to weighted graphs. However, the analogy stops being useful after this point, which is why we stop using it further.

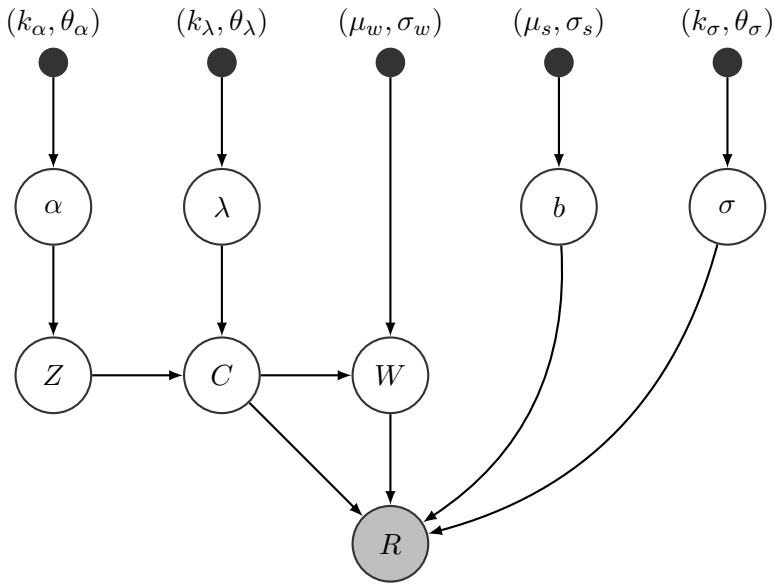


Figure 3.2: Extended model. We add the noise variance σ, drawn from an Inverse Γ distribution. The entries in R are real valued instead of binary and are normally distributed.

3.2.1 Likelihood

The likelihood of the extended model has a normal distribution and is shown in Equation 3.2:

$$\mathcal{L}(s|R) = \mathcal{L}(T|R) = p(R|T) = \prod_{i=1}^{n}\prod_{j=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(R(i,j)-T(i,j))^2}{2\sigma^2}} \tag{3.2}$$

3.2.2 Example

In this example, we sample a model instance from the prior distribution and generate R.

Sampling Z. Let $n = 4$, $\alpha = 1$.

- Object 1 chooses Poisson(1) features, let it be 0.
- Object 2 has no existing features to choose, chooses Poisson($\frac{1}{2}$) new features, let it be 1
- Object 3 chooses feature 1 with probability $\frac{|m|}{i} = \frac{1}{3}$. In our example, it is chosen. Afterwards, Poisson($\frac{1}{3}$) new features are created, in this case 2.
- Object 4 chooses:
  - $P(z_{4,1} = 1) = \frac{|m|}{i} = \frac{2}{4}$. Let $z_{4,1}$ be 1.
  - $P(z_{4,2} = 1) = \frac{|m|}{i} = \frac{1}{4}$. Let $z_{4,2}$ be 0.
  - $P(z_{4,3} = 1) = \frac{|m|}{i} = \frac{1}{4}$. Let $z_{4,3}$ be 1.

  Poisson($\frac{1}{4}$) new features are chosen for Object 4, in this case 0.

After completing the Indian buffet process, k is 3 and the resulting matrix can be seen in Figure 3.3(a).

Object \ Feature	1	2	3
1	0	0	0
2	1	0	0
3	1	1	1
4	1	0	1

(a) Z matrix

Object \ Feature	1	2	3
1			
2	1		
3	1	1	1
4	2		2

(b) C matrix

Figure 3.3: Example feature and cluster matrices with four objects and three features. Only the objects contained in a feature are partitioned into that feature's clusters.

Sampling C. The cluster matrix C is influenced by the feature matrix Z and the hyperparameter λ. We use the Z matrix from the previous example and let λ be 1.

- Object 1 has no active features. No clusters need to be sampled.
- Object 2 has active feature 1. ($z_{2,1} = 1$). Since no previous clusters exist in feature 1, the probability of choosing a new one is $\lambda/(\lambda + \sum_{clusterl \in m} |l|) = \lambda/(\lambda + 0) = 1$. Therefore $c_{2,1} = 1$.
- Object 3: Feature 1 is active. Probability of choosing existing cluster 1 is $\frac{1}{1+1}$. Let Object 3 choose cluster 1. In feature 2 and 3, no previous clusters exist and new ones are created.
- Object 4: Features 1 and 3 are active. For feature 1, the probabilities are $\frac{2}{\lambda+2}$ and $\frac{\lambda}{\lambda+2}$. Let Object 4 choose new clusters in features 1 and 3.

The resulting cluster matrix can be seen in Figure 3.3(b).

Sampling W. Weights between clusters are sampled from a Gaussian distribution. Let $\mu_w = 1$ and $\sigma_w = 2$.

- Feature 1 has 2 clusters. A 2×2 weight matrix W^1 needs to be created. Let W^1 be $\begin{pmatrix} 1 & 0.1 \\ 0.1 & 2 \end{pmatrix}$
- Feature 2 has only one cluster. Let W^2 be $\begin{pmatrix} 3 \end{pmatrix}$.
- Feature 3 has two clusters. Let W^3 be $\begin{pmatrix} 0.5 & -1 \\ -1 & 1 \end{pmatrix}$.

Sampling Bias. The bias b is sampled from a Gaussian distribution. Let b be 0.

Sampling R. For clarity, we first sum the weights and the bias. Finally, noise is added. Let $T \in \mathbb{R}^{n \times n}$ be the sum of weights and bias. Since b is 0, we only need to sum the weights.

- $T(1, x) = T(x, 1) = 0, \forall x$ (Object 1 doesn't have any features)
- $T(2, 2) = T(2, 3) = T(3, 2) = W^1[1][1] = 1$
- $T(2, 4) = T(4, 2) = W^1[1][2] = 0.1$
- $T(3, 3) = W^1[1][1] + W^2[1][1] + W^3[1][1] = 1 + 3 + 0.5 = 4.5$
- $T(3, 4) = W^1[1][2] + W^3[1][2] = 0.1 - 1 = -0.9$

- $T(4,4) = W^1[2][2] + W^3[2][2] = 2 + 1 = 3$

Equation 3.3 shows the resulting edge array.

$$T(i,j) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0.1 \\ 0 & 1 & 4.5 & -0.9 \\ 0 & 0.1 & -0.9 & 3 \end{pmatrix} \tag{3.3}$$

Finally, noise is added to obtain R. Let σ_{noise} be 0.5 and sample noise from $\mathcal{N}(0, 0.5)$. We sampled values 0, -0.65, -0.56, -0.27, 0.96, -0.2, -0.4, -0.3, 0.34 and 0.08. We add these values to the edge array in Equation 3.4.

$$R = T + \text{noise} = \begin{pmatrix} 0 & -0.65 & -0.56 & -0.27 \\ -0.65 & 1.96 & 0.8 & -0.3 \\ -0.56 & 0.8 & 4.2 & 0.24 \\ -0.27 & -0.3 & 0.24 & 3.08 \end{pmatrix} \tag{3.4}$$

3.2.3 Inherent Ambiguity

Each input dataset R can be fitted with an infinite amount of possible models. The example models shown in Figures 3.4 and 3.5 result in the exact same T (Figure 3.6) and thus the same distribution on R. When inverting the model from a given R, both of them have the same likelihood and it is impossible to tell which of these (or other) models were the hidden variables. Other cases of ambiguity exist: The order of features and clusters is lost in summing the weights. We later define equivalence classes on model instances to resolve some of this ambiguity. However, the previous example would not be resolved by the equivalence classes, but could be removed by fixing the bias to 0.

Object	
1	1
2	1
3	1
4	1

(a) Z

Object	
1	1
2	1
3	2
4	2

(b) C

1	1
1	4

(c) W^1

Figure 3.4: Example model A, Bias: 0

Object	
1	0
2	0
3	1
4	1

(a) Z

Object	
1	
2	
3	1
4	1

(b) C

3

(c) W^1

Figure 3.5: Example model B, Bias: 1

		Object			
		1	2	3	4
Object	1	1	1	1	1
	2	1	1	1	1
	3	1	1	4	4
	4	1	1	4	4

Figure 3.6: T of example model A and B

4. Algorithm, Implementation and Engineering

This chapter contains the *generative* algorithms to create test data when the hidden variables are known and the *inversion* algorithm to infer the hidden variables when they are indeed hidden.

We discuss the generation in Section 4.1. The Gibbs sampler, main part of the inversion algorithm, is presented in Section 4.2. The result of the inversion is a distribution of samples approximating the posterior distribution, which was described in Section 2.5.3. Getting the mode of this distribution of samples requires discretization and a high number of samples. A mean is yet undefined, as we need an average operation on model instances. We define such an operation in 4.3.

One of the problems of unsupervised MCMC clustering is the interpretation of the resulting clusters. We present a variant for supervised learning in Section 4.4.

Time and memory complexity of each algorithm are discussed after the algorithm itself, a summary is found in Section 4.5.

The implementation was done in C++, for reasons of speed and accessibility with free software.[1] We discuss the code structure in Section 4.6, optimizations in Section 4.7 and finally parallelization in Section 4.8. The source code is available online along with a manual.[2]

4.1 Model Generation

The generative process is not required for the inversion, we implemented it to generate simulated data and random starting states. Algorithm 1 shows how model instances are sampled from the prior distribution. When discussing pseudocode and implementation, we use array notation for Z, C, W, T and R. This is used to differentiate the data structures from the mathematical concepts. For example, $C(i, m)$ is undefined if $Z(i, m)$ is zero, but

[1] Several packages for Bayesian methods and Gibbs samplers exist, for example the WinBUGS and OpenBUGS project[17]. The distributions used in our model were not covered by them, requiring a new implementation.

[2] `https://bitbucket.org/mektah/ila-hierarchical`

$C[i][m]$ may be assigned a value. We often fixed the noise in practice instead of sampling it. After generating a model instance s, we can sample similarity matrices R as test data or use s as a random starting point for Markov chains.

Algorithm 1: Generating a model instance (Gen)

Input: n, k_α, θ_α, k_λ, θ_λ, μ_b, σ_b, μ_{weights} σ_{weights}, k_{v}, θ_{v}
Output: model instance s
`/* sample hyperparameters */`
sample α from $\Gamma(k_\alpha, \theta_\alpha)$;
sample λ from $\Gamma(k_\lambda, \theta_\lambda)$;
`/* sample` Z `with the Indian buffet process */`
Z = binary matrix with n rows and zero columns;
for $i = 1$ **to** n **do**
  for $m = 1$ **to** *columns(Z)* **do**
    set $Z[i][m]$ to 1 with probability $n_{of}(s, m)/i$;
  end
  add Poisson(α/i) columns to Z, set $Z[i][m] = 1$ for new columns m;
end
`/* sample` C `with the Chinese restaurant process */`
for *feature* m *in* Z **do**
  for $i = 1$ **to** n **do**
    if $Z[i][m]$ **then**
      `/* cluster needs to be selected for */`
      select existing cluster c with probability $\big(n_{oc}(s, m, c)/(n_{of}(s, m) + \lambda)\big)$;
      OR select new cluster with probability $\big(\lambda/(n_{of}(s, m) + \lambda)\big)$;
      set $C[i][m]$ to selection;
    end
  end
end
`/* sample weights */`
foreach *feature* m *in* Z **do**
  foreach *pair* c, d *of clusters in feature* m **do**
    sample $W^m[c][d]$ from $\mathcal{N}(\mu_{\text{weights}}, \sigma_{\text{weights}})$;
    $W^m[d][c] = W^m[c][d]$;
  end
end
`/* sample other parameters */`
sample b from $\mathcal{N}(\mu_b, \sigma_b)$;
sample σ_{noise} from **Inv-Gamma**(k_{v}, θ_{v});
return *model*(α, λ, Z, C, W, b, σ_{noise})

4.1.1 Memory Consumption

The generative algorithm needs memory to store the resulting model instance and to keep the number of objects in each feature and cluster. Memory required to store the result dominates the memory required for the number of objects.

Most memory is used to store matrices Z, C and the weight matrices W. We used the summed sizes of the weight matrices as a complexity measure, calling it the *total link count*

(TLC):

$$\mathrm{TLC}(s) = \sum_{m=1}^{n_f(s)} n_{cf}(s,m)^2 \tag{4.1}$$

The total memory needed is linear in the number of objects $n(s)$, the number of features $n_f(s)$ and the total link count. We use $T(\text{algorithm}, s)$ and $M(\text{algorithm}, s)$ for the time and memory complexity of an algorithm with input n.

$$M(\mathrm{Gen}, s) \in \Theta\big(n(s) \cdot n_f(s) + \mathrm{TLC}(s)\big) \tag{4.2}$$

The memory needed to store parameters α, λ, b and σ is constant in $n(s)$ and $n_f(s)$.

4.1.2 Time Complexity

Every matrix entry needs to be modified exactly once. In several steps the number of objects in a feature or a cluster is needed for the Indian buffet process and Chinese restaurant process. If these numbers are stored during the process, the matrices do not need to be read at all. In this case, the time complexity is equal to the summed size of all matrices:

$$T(\mathrm{Gen}, s) \in \Theta\big(n(s) \cdot n_f(s) + \mathrm{TLC}(s)\big) \tag{4.3}$$

This assumes a random number generator of constant time complexity is available for all desired distributions.

4.2 Inversion

In this section, we present the main sampling algorithm. As discussed in Section 2.5.3 and 2.3.3, we use a Gibbs sampler to generate samples from the posterior probability distribution. Each random variable is sampled from the conditional distribution given all others. We treat every field of Z, C and W as a separate random variable for this purpose.

4.2.1 Input Data

When input data is not in the form of a similarity matrix R, it needs to be transformed. We assume n objects with d measurements each. The input dataset D can be interpreted as a set of coordinates in d-dimensional space, $D \in \mathbb{R}^{n \times d}$.[3] To get a symmetric similarity matrix, we took the negative Euclidean distance of each pair of points.[4] While some data is lost, the distances allow reconstruction of the original data except for translation and rotation. Translations and rotations are not of interest as they do not change the clusterings.

We also used the dot product to create R, which resulted in somewhat slower convergence. Any kind of kernel [11] could be used, enabling a wide range of applications.

4.2.2 Likelihood Calculation

To compute the likelihood of a solution s, we need to create the matrix $T(s)$. Each link strength needs to be computed, which requires iteration over all non-empty features. Assuming random access, the time complexity to compute the likelihood is quadratic in the number of objects and linear in the number of features, as shown in Equation 4.4. $T(\mathrm{L}, s)$ is used for the time complexity of the likelihood computation. The memory consumption is constant.

$$T(\mathrm{L}, s) \in O\big(n(s)^2 \cdot n_f(s)\big) \tag{4.4}$$

[3] For the fMRI data analyzed in Section 5.5, n was 83 and d was 12.

[4] This could be interpreted as a power kernel with the exponent set to 1.

4.2.3 Initialization

We used three different ways to choose starting points for the sampling chains. The first method is to generate a random model instance with Algorithm 1. When iterating through several chains in parallel, it is useful to have overdispersed starting states. Starting with random model instances enables generating overdispersed starting states by choosing different values for parameters α and λ. The second method uses a model instance of minimal complexity without any features or cluster. This was especially useful for debugging and comparing the influence of tuning parameters. The third method is called sequential initialization and starts with a small subset of the objects. Z and C are sampled and the remaining objects gradually added, with some sampling steps between the object additions. While the sampling chains should converge to the same limiting distribution regardless of initialization, the mixing speeds differ. Chains initialized sequentially approach the stationary distribution faster.

4.2.4 Complete Iteration

One iteration consists of sampling each random variable in turn, as shown in Algorithm 2. To keep the algorithm interfaces short, we assume the model elements are passed within s and extracted when needed.

If thinning is not used, every model instance is saved as a sample. This is useful for initial analysis and convergence tests.

The runtime is dominated by Algorithm 3, the sampling of Z and C. The total time complexity will be revisited in Section 4.5.

Algorithm 2: Iteration (Iter)

```
   Input: model instance s, R
1  (sample σ);
2  for i = 1 to n(s) do
3  |  for m = 1 to n_f(s) do
4  |  |  sample Z and C(s,R,i,m);
5  |  end
6  |  add new features;
7  end
8  sample W(s, R);
9  sample b;
10 sample α;
11 sample λ;
```

4.2.5 Sampling Z and C

This sampling step is called from the main algorithm for every object i and every feature m. We sample whether feature m is active for object i and if it is, which cluster object i as assigned to. The probability of object i activating feature m depends on the present population of m and the likelihood given R. As a result from Bayes' rule, the probability of turning $f^m(s)$ on for object i is proportional to the likelihood when the feature is turned on times the prior probability of it being turned on. Both the likelihood and the prior probability depend on the remaining random variables, denoted by $s\backslash(i,m)$. $Z\backslash(i,m)$ is

the Z matrix without field (i,m), $C\backslash(i,m)$ the C matrix.

$$\mathcal{P}(Z(i,m)=1|R,s\backslash(i,m)) = \frac{p(R|Z(i,m)=1,s\backslash(i,m))\cdot\mathcal{P}(Z(i,m)=1|s\backslash(i,m))}{p(R|Z(i,m)=1,s\backslash(i,m))+p(R|Z(i,m)=0,s\backslash(i,m))} \tag{4.5}$$

Fortunately, the prior distribution generated by the Indian buffet process is interchangeable, which is why we can treat each object like it has been the last object arriving. This results in the prior probability of $\mathcal{P}(Z(i,m)=1|s\backslash(i,m))$ being $(n_{of}(s,m)-Z(i,m))/n$, the popularity of feature m without object i.[5]

Equation 4.6 gives the likelihood of activating feature m for object i.

$$\mathcal{P}(Z(i,m)=1|s\backslash(i,m),R) \propto \frac{n_{of}(s\backslash(i,m),m)}{n} p(R|Z(i,m)=1,s\backslash(i,m)) \tag{4.6}$$

To calculate the likelihood of R when the feature is active, one needs to integrate over possible cluster choices. This includes new clusters not yet present. Analytically solving this integral is infeasible, which is why we approximate it using samples of temporary empty clusters. This is done by the `CP` (*Cluster Probabilities*) algorithm, which is discussed in the next section and uses the auxiliary variable method introduced in Section 2.3.3. The sampling step of Z is shown in Algorithm 3, which mostly consists of calculating the right-hand side of Equation 4.5. Line 1 shows calling the `CP` algorithm, the sum of the result vector is an approximation of $p(R|Z(i,m)=1,s\backslash(i,m))$. This conditional probability is multiplied with the prior $\mathcal{P}(Z(i,m)=1|s\backslash(i,m))$ in Line 2 to obtain the numerator in Equation 4.5. Line 3 calculates the likelihood of s with $Z(i,m)$ set to 0. These values are combined in Line 4 to finally obtain the likelihood of $Z(i,m)$ being 1. If it is set, a cluster has to be selected. It is chosen from the possible clusters returned by the `CP` call. The probability of selecting each cluster is proportional to its value in clusterProbs, which is the posterior probability computed by the `CP` algorithm. Afterwards, all empty clusters – those existing previously and temporary new ones – are discarded. Empty features are discarded as well.

Algorithm 3: Sample$Z$and$C$ (SZ)

Input: s, R, i, m

1 clusterProbs = CP(s, i, m);

2 prOn = $\sum_{c=1}^{n_{cf}(s,m)+m_{\text{aux}}} \text{clusterProbs}[c]\cdot\frac{n_{of}(s\backslash(i,m),m)}{n}$;

3 prOff = $p(R|Z(i,m)=0,s\backslash(i,m))\cdot(1-\frac{n_{of}(s\backslash(i,m),m)}{n})$;

4 set $Z(i,m)$ to 1 with probability $\frac{\text{prOn}}{\text{prOn+prOff}}$;

5 **if** $Z(i,m)$ = *1* **then**

6 select cluster c with probability $\frac{\text{clusterProbs}[c]}{\sum_d^{n_{cf}(s,m)+m_{\text{aux}}}\text{clusterProbs}[d]}$;

7 $C(i,m)=c$;

8 **end**

9 removeEmptyClustersAndFeatures;

[5] Because feature m is deactivated for object i before calculating the popularity of m, the prior will always be below 1. This follows from the Indian buffet process and it is easy to see why it is important: If the popularity of feature m is calculated without subtracting $Z(i,m)$ and m is active for every object, the prior would evaluate to 1 and the feature could never be deselected. The Markov chain could get stuck in a local region and no longer have a single limiting distribution.

4.2.6 Calculating Cluster Probabilities

To determine the likelihood of a feature m being active for object i, we need to approximate the integral over possible cluster assignments. Algorithm 4 shows how the probabilities are gathered. In addition to the existing clusters, object i could be assigned to a new cluster. A number $m_{\text{aux}}$of temporary new clusters are created, representing possible values for the new cluster. These temporary clusters are empty. This mirrors the auxiliary variable algorithm discussed in Section 2.3.3: The new clusters are variables not belonging to the present state. We keep the prior from the Chinese restaurant process, which assigns a probability of λ / $(\lambda + n_{of}(s,m))$ to new clusters. With several possible new clusters, the probability mass has to be divided among them. For each of the new clusters, weights are sampled from the weight prior.

Probabilities are calculated for each of the existing and new clusters. If one of the new clusters is chosen, it is kept.

Algorithm 4: Cluster assignment probabilities (CP)

Input: s, i, m
Output: array of $\mathcal{P}(C(i,m)=c)$ for each possible c

add m_{aux} clusters to feature m, extend W^m with new weights sampled from prior;
for $c = 1$ **to**$n_{cf}(s,m)$ **do**
  prob$[c] = \frac{n_{oc}(s,m,c)}{n_{of}(s,m)+\lambda} \cdot p(R|C(i,m)=c, Z(i,m)=1, s\backslash(i,m))$.
end
for $c = n_{cf}(s,m)+1$ **to** $n_{cf}(s,m)+m_{\text{aux}}$ **do**
  prob$[c] = \frac{\lambda/m_{\text{aux}}}{n_{of}(s,m)+\lambda} \cdot p(R|C(i,m)=c, Z(i,m)=1, s\backslash(i,m))$.
end
return *prob*

Likelihood Δ. The likelihood $p(R|C[i][m]=c, s\backslash(i,m))$ could be computed from scratch as described in Section 3.2. However, if the cluster membership of i changes, only values $T[i][x]$ or $T[x][i]$ are affected, where $1 \leq x \leq n$. By only recomputing the values in one row and column of T, we can reduce the computational complexity of the likelihood calculation from $O(n^2 \cdot f)$ to $O(n \cdot f)$.

Computational Complexity. Sampling the cluster probabilities of one object i in feature m requires as many likelihood calculations as there are clusters in m, see Equation 4.7.

$$T\big(\text{CP}, (s,m,m_{\text{aux}})\big) \in O\big((n_{cf}(s,m)+m_{\text{aux}}) \cdot T(\text{LT}, s)\big) \tag{4.7}$$

The time needed for L(s) is in $O(n(s)^2 \cdot n_f(s))$. If a likelihood difference is computed and only n connections need to be evaluated, the complexity is linear in n:

$$T(\text{L}\Delta, s) \in O(n(s) \cdot n_f(s))$$

[6]

4.2.7 Sampling Weights

Weights are sampled successively for each element of the weight matrices while keeping them symmetrical. The prior for each weight is a normal distribution. The posterior

[6] One could store even more information and reduce the complexity even further to $O(n)$. Since f is small in most of the cases, this optimization would have been of little benefit and remains unimplemented.

Algorithm 5: Sample$W$ (SW)

Input: s, R

```
1 foreach feature m in s do
2 |   foreach pair of clusters c,d in m do
3 |   |   W^m[c][d] = sampleWeight(m, c, d, s, R);
4 |   |   W^m[d][c] = W^m[c][d];
5 |   end
6 end
```

density $f_{m,c,d}(weight)$ is proportional to the likelihood of $weight = W^m[c][d]$ given R, multiplied with the prior.

$$f_{m,c,d}(weight) \propto p(R|W^m[c][d] = weight, s) \cdot \mathcal{N}(weight|\mu_w, \sigma_w) \tag{4.8}$$

In the above equation, $\mathcal{N}(weight|\mu_w, \sigma_w)$ is the probability density of the normal distribution $\mathcal{N}(\mu_w, \sigma_w)$ at $weight$. The posterior is a Gaussian distribution. One way to sample from it involves creating a histogram or using slice sampling. A faster approach is to calculate μ and σ of the posterior from s and R and subsequently sample from it using the Box-Muller-method[3]. In the following two sections, we describe both approaches in detail.

Naive Approach. The probability can be computed from scratch as described in Equation 3.2. To sample from the weight posterior in the naive approach, we discretize the posterior probability density function, create a histogram and sample from it.

The weight between two clusters c and d only affects the connections between objects in these clusters. Analogous to Section 8, the process can be accelerated by only computing the likelihood difference Δ.

As mentioned earlier in Equation 3.2, the likelihood of a model instance s is the product of the likelihoods of each edge. When only some parts of T change, the likelihood of the unchanged part can be kept. Algorithm 6 shows how the likelihood density is computed. It accepts as input a model instance s , the feature m, clusters c_1 and c_2, the new weight and the likelihood of s without changes. First the likelihood of the affected values $T[i][j]$ is computed in the old version. In Line 7, the likelihood of the unchanged part is computed. A backup of the previous weight is made in Line 9. For most sampling purposes, only a function proportional to the probability density is required. In this case, Algorithm 6 can be made quicker by only computing newpartl and omitting the computation of oldpartl and baseLikelihood. As an alternative to creating a histogram of probabilities, slice sampling can also be used with the probability density function. It was not noticeably faster.

Using Conjugate Priors. Fortunately, an analytical solution is available. Let the weight in question be in feature m between clusters c and d. We first set it to 0 and recompute T. Because of the conjugate prior, the posterior parameters $\mu_{\text{posterior}}$ and σ are then given by Equations 4.9 and 4.10[22]. μ_w and σ_w are the prior mean and variance for weight sampling, σ is the noise variance.

$$\mu_{\text{posterior}} = \Big(\frac{\mu_w}{\sigma_w^2} + \frac{\sum_{i\in c}\sum_{j\in d} R[i][j] - T[i][j]}{\sigma^2}\Big) \Big/ \Big(\frac{1}{\sigma_w^2} + \frac{n_{oc}(s,m,c)n_{oc}(s,m,d)}{\sigma^2}\Big) \tag{4.9}$$

$$\sigma^2_{\text{posterior}} = \Big(\frac{1}{\sigma_w^2} + \frac{n_{oc}(s,m,c)n_{oc}(s,m,d)}{\sigma^2}\Big)^{-1} \tag{4.10}$$

Algorithm 6: Probability Density function of weight sampling

Input: R, s, m, c_1, c_2, weight, oldLikelihood
Output: Likelihood density of $(s, W^m[c][d])$ at *weight*

```
/* compute likelihood of changed part in old version */
```
oldpartl = 1;
foreach *object i in cluster* c_1 **do**
  foreach *object j in cluster* c_2 **do**
    oldpartl *= $(\mathcal{N}(R[i][j]|T[i][j], \sigma))$
  end
end
```
/* likelihood of unchanged part is old total likelihood divided by
   likelihood of changed part */
```
unchangedPartl = oldLikelihood / oldpartl;
```
/* compute likelihood of changed part in new version */
```
newpartl = 1;
previousWeight = $W^m[c_1][c_2]$;
$W^m[c_1][c_2] = weight$;
$W^m[c_2][c_1] = weight$;
foreach *object i in cluster* c_1 **do**
  foreach *object j in cluster* c_2 **do**
    update $T[i][j]$;
    newpartl *= $\mathcal{N}(R[i][j] \mid T[i][j], \sigma)$
  end
end
```
/* new total likelihood is likelihood of unchanged part multiplied by
   likelihood of changed part */
```
newl = unchangedPartl * newpartl;
$W^m[c_1][c_2]$ = previousWeight;
$W^m[c_2][c_1]$ = previousWeight;
return *newl*;

Algorithm 7 computes the posterior distribution for each weight matrix entry. We can sample from this distribution directly using the Box-Muller-transform if a uniform random number generator is available, which is contained in almost all execution environments. This process is much faster than creating a histogram or using slice sampling, as we will see in Section 5.1.

Algorithm 7: Parameters for posterior function, the conjugate way

Input: R, s, m, c, d
Output: μ and σ of posterior

```
set W^m[c][d] = W^m[d][c] = 0 ;
sum = 0;
foreach object i ∈ c do
    foreach object j ∈ d do
        diff = R[i][j] − T[i][j];
        sum += diff;
    end
end
lc = n_oc(s, m, c) · n_oc(s, m, d);
σ_posterior = 1/(1/(σ_w^2 + lc/s.σ^2);
μ_posterior = (μ_w/(σ_w^2) + sum/s.σ^2) · σ_posterior;
return (μ_posterior, σ_posterior)
```

Computational Complexity. If a histogram of the posterior likelihood density function is created, one likelihood calculation is needed for each histogram step. The complexity thus depends on the range and the step size of the histogram, see the following equation.

$$T(\mathrm{SW}, s, range, stepsize) \in O\Big(\sum_{f=0}^{n_f(s)} n_{cf}(s,f)^2 \cdot T(\mathrm{L}\Delta, s) \cdot \big(\frac{range}{stepsize}\big)\Big) \tag{4.11}$$

In the naive approach, we used the interval $[-10, 10]$ and a step size of 0.1, requiring 200 $L\Delta$ evaluations for each weight.

If the distribution parameters are calculated and the Box-Muller-method is used, the complexity is:

$$T(\mathrm{SW}, s) \in O\Big(\sum_{f=0}^{n_f(s)} n_{cf}(s,f)^2 \cdot T(\mathrm{L}\Delta, s)\Big).$$

4.2.8 Sampling Hyperparameters

Bias. The bias b has a Gaussian prior and the sampling is very similar to the weight sampling in Algorithms 6 and 7. We first used a histogram-based sampling and then a direct sampling. Complexity wise, sampling the bias is comparable to sampling a single weight.

Noise Variance. It was often suitable to set the noise variance to a fixed value when using real world data. In cases where we sampled it, a histogram was used. We chose the inverse gamma distribution as the prior distribution for the variance, which is conjugate.

Alpha. The value of α does not directly influence R but governs the number of features in Z. Thus we only need to evaluate the likelihood of α with respect to Z, which saves us the computationally expensive step of generating T. Since the rows and columns of Z are interchangeable, we need to invert α from the final number of features. These follow a Poisson distribution, for which the Gamma prior is conjugate. The posterior density is proportional to $\Gamma\big(k_\alpha + n_f(s), \theta_\alpha/(\theta_\alpha + \sum_{i=1}^{n} 1/i)\big)$.

The computational complexity is constant if a suitable random number generator is available.

Lambda. The value of λ influences C but not directly R. Alas, the prior is not conjugate and we need to use either a histogram or slice sampling.

The time required depends on the range and step size of the constructed histogram:

$$T(S\lambda, range, stepsize) \in O(\frac{range}{stepsize}) \tag{4.12}$$

When using slice sampling, the time depends on the burn-in period.

4.3 Aggregation

After completing the inversion in the previous section, we have samples from the posterior distribution. The aim of this section is to obtain a representative solution, the *expected value* of this distribution, which is otherwise a bit daunting to interpret. First, we remove redundant features within model instances. A feature $f^l(s)$ is called a *refinement* of $f^k(s)$ if every cluster in $f^l(s)$ is contained in a cluster of $f^k(s)$.[7] Not participating in a feature is handled as being assigned to a dummy cluster. Thus a feature $f^l(s)$ can be a refinement of $f^k(s)$ even if not all objects participating in $f^k(s)$ also participate in $f^l(s)$. If in instance s feature $f^l(s)$ is a refinement of feature $f^k(s)$, we can omit feature $f^k(s)$ and add the weights to the appropriate clusters in $f^l(s)$. This step is discussed in Section 4.3.1. Together with reordering features and clusters to a canonical form, this groups model instances into *equivalence classes*, presented in Section 4.3.2.[8] Finally, we define an aggregation function on equivalence classes which resembles a mean, see Section 4.3.3.

4.3.1 Merging of Features

Given a model instance s with at least two features $f^j(s)$ and $f^k(s)$, it is possible to merge $f^j(s)$ and $f^k(s)$ into a single feature $f^l(s)$ without changing the link sums $T(s)$.

If a feature $f^l(s)$ has less clusters than a feature $f^k(s)$, it cannot be a refinement of $f^k(s)$. If a feature $f^l(s)$ has n non-empty clusters, it is a refinement of all other features. In Figure 4.1, feature 2 is a refinement of feature 1. Merging these features results in the model instance shown in Figure 4.2. If $f^j(s)$ is a refinement of $f^k(s)$ or vice versa, the number of non-empty clusters in the result $f^l(s)$ is equal to $\max(n_{cf}(s,j), n_{cf}(s,k))$. It is also possible to merge features where none is a refinement of the other, but this will result in a higher number of clusters. In this case, we construct all of the $(n_{cf}(s,j)+1)\cdot(n_{cf}(s,k)+1)$ possible cluster combinations and keep those which actually contain objects.

If $f^j(s)$ and $f^k(s)$ are completely orthogonal, one cluster has to be created for every combination: If an object does not participate in one of the features, we assign it to a

[7] Alternative definition: Every pair of objects i, j assigned to different clusters in $f^k(s)$ is also assigned to different clusters in $f^l(s)$.

[8] Equivalence classes among feature matrices are also discussed in [12]

Object		
1	1	1
2	1	1
3	1	1
4	1	1

(a) Z

Object		
1	1	1
2	1	1
3	2	2
4	2	3

(b) C

1	1
1	4

(c) W^1

1	2	3
2	4	5
3	5	6

(d) W^2

Figure 4.1: The partition of feature 2 is a refinement of the partition of feature 1. Each cluster in feature 2 is contained in a cluster of feature 1.

Object	
1	1
2	1
3	1
4	1

(a) Z

Object	
1	1
2	1
3	2
4	3

(b) C

1+1	2+1	3+1
2+1	4+4	5+4
3+1	5+4	6+4

(c) W^1

Figure 4.2: Result of merge. The cluster assignment is taken from the partition with more clusters, the weights are added to result in the same T.

dummy cluster. This leads to $(n_{cf}(s,j)+1)\cdot(n_{cf}(s,k)+1)$ possible combinations . Since objects not participating in either cluster are omitted from the result, the combination of both dummy clusters is discarded. This leads to at most $(n_{cf}(+,1))\cdot(n_{cf}(+,1))-1$ clusters in the result, as seen in the following equation.

$$n_{cf}(s,l) = \min\big\{(n_{cf}(s,j)+1)\cdot(n_{cf}(s,k)+1)-1, n\big\} \tag{4.13}$$

4.3.2 Equivalence Classes

Two model instances s_1, s_2 are called *equivalent* if s_1 can be transformed into s_2 through applications of the following transformations:

- Reordering features
- Merging features if one is a refinement of the other
- Reordering clusters within features

We use Algorithm 8 to transform each solution to a representative of it's equivalence class. The first step consists of reordering the features so Z is left-ordered:[12] Features are represented by columns of Z. Each column is a binary vector and can be interpreted as a number. We sort features according to their binary value, which is also a lexicographic ordering. The second step consists of merging all redundant features into their refinements. If a feature has more than one refinement and they are not refinements of each other, it will be merged into the earlier one. Because of the first sorting step, this merging order is unambiguous. The third step consists of reordering clusters: Cluster $c_c^f(s)$ is sorted before $c_d^f(s)$ if the lowest object index in $c_c^f(s)$ is smaller than the lowest object index of $c_d^f(s)$. Since the clusterings are disjoint, this order is unambiguous. Finally, the features in Z are sorted again to make sure the result is left-ordered.

4.3.3 Aggregation of Samples

After grouping samples into equivalence classes, we want to aggregate the equivalence classes into a single representative solution. The aggregation consists of three steps: The

Algorithm 8: Mapping to Equivalence Classes (equiv)

Input: model instance s
Output: representative of equivalence class of s

```
1  sort(Z, C, W) ;                                  /* sort so Z is left-ordered */
2  for feature k do
3  |  for feature l do
4  |  |  if k is a refinement of l then
5  |  |  |  merge l into k;
6  |  |  end
7  |  end
8  |  sort clusters in k;
9  end
10 sort(Z, C, W) ;                                  /* sort so Z is left-ordered */
11 return (Z, C, W, b)
```

first step consists of transforming each sample into a common form, which is independent of the number and order of clusters in each feature. The common forms are averaged in the second step, this average is finally transformed into a model instance, which we call *collapsing the common form.* We only discuss aggregating Z, C and W, as the parameters b, σ, α and λ are straightforward to average.

Common Form. Each column m of the cluster matrix C is extended to a binary $n \times n$-matrix E^m. $E^m[i][j]$ stores whether objects i and j are in the same cluster in feature m. Objects not participating in feature m are not in any cluster, especially not in the same as any other objects. From this form, C can be reconstructed except for relabelings. $X^m[i][j]$ stores the weight between objects i and j in feature m. If both participate in feature m, it is taken from the weight matrices. If at least one object does not participate in feature m, $X^m[i][j]$ is 0. The pseudocode can be seen in Algorithm 9.

Algorithm 9: Construct common form (Construct)

Input: model instance s

```
1 foreach feature m in s do
2 |  for i = 1 to n do
3 |  |  for j = 1 to n do
4 |  |  |  set E^m[i][j] to 1 if C[i][m] = C[j][m], 0 otherwise;
5 |  |  |  set X^m[i][j] to W^m[C[i][m]][C[j][m]];
6 |  |  end
7 |  end
8 end
9 return Z, E and X
```

Average of Common Forms. Two common forms can be averaged directly with Algorithm 10. First, features are reordered to average those which are already similar to each other. Two possibilities exist to reorder the features before averaging them: One can make Z left-ordered as done when we defined the equivalence classes or one can reorder them to maximize some similarity measure between them. The next nested loop simply computes the average of Z_1, Z_2, E_1, E_2 and X_1, X_2 The averages of different Z or E contain real values between 0 and 1 instead of binary values. Counting the proportion of samples in

which two objects are in the same cluster is a commonly used measure in MCMC clustering evaluations[21]. If we average two common forms with the same number of features, the algorithm ends at Line 12. Should one of the forms have more features, the excess features are copied to the average, but divided by two. This pseudocode shows the average of

Algorithm 10: Averaging common forms (Av)

Input: (Z_1, E_1, X_1) and (Z_2, E_2, X_2)
Output: Z, E, X

```
1  n_f(Z) = max(n_f(Z_1), n_f(Z_2));
2  reorder features of Z_1 and Z_2 to maximize similarity of E_1^m and E_2^m for all m;
3  for m = 0 to min(n_f(Z_1), n_f(Z_2)) do
4  |  for i = 0 to n do
5  |  |  Z[i][m] = (Z_1[i][m] + Z_2[i][m])/2;
6  |  |  for j = 0 to n do
7  |  |  |  E^m[i][j] = (E_1^m[i][j] + E_2^m[i][j])/2;
8  |  |  |  X^m[i][j] = (X_1^m[i][j] + X_2^m[i][j])/2;
9  |  |  end
10 |  end
11 end
   /* copy unmatched content                                              */
12 g = arg max_x n_f(Z_x);
13 for m = min f(Z_1), f(Z_2) to max(f(Z_1), f(Z_2)) do
14 |  for i = 0 to n do
15 |  |  Z[i][m] = Z_g[i][m]/2;
16 |  |  for j = 0 to n do
17 |  |  |  E^m[i][j] = E_g^m[i][j];
18 |  |  |  X^m[i][j] = X_g^m[i][j];
19 |  |  end
20 |  end
21 end
22 return Z, E, X
```

only two instance. We implemented it with weighting factors allowing more instances or a skewed average, but describe the simpler version for clarity.

Collapsing Common Form to Model Instance. We set out to create an aggregation resembling a mean. For this, the result needs to be of the same form as the samples to be aggregated. Algorithm 11 collapses a common form back to a model instance. Each feature can be considered separately. At first, the feature matrix Z is created by thresholding the averaged values. If the average value for feature m in object i is over the threshold, $Z(i, m)$ is set to 1.

A similar thresholding is done with the extended clusters E. If $E^m[i][j]$ is higher than the cluster threshold, object i is put in the same cluster as object j.[9] After feature presence and cluster assignments are determined, the weights are averaged. The resulting weight between clusters c and d is the average of all entries $X^m[i][j]$, where object i is contained in cluster c and object j is contained in cluster d.

[9] One could also frame it as a problem in graph theory: After thresholding, we have a binary adjacency matrix and identify one cluster with each connected component.

Algorithm 11: Collapsing common form to model instance (Collapse)

Input: (Z, E, X), featureThreshold, clusterThreshold
Output: model instance s

foreach *feature* $m \in Z$ **do**
  `/* collapse` Z `*/`
  for $i = 1$ **to** n **do**
    if $Z[i][m] \geq$*featureThreshold* **then**
      $Z[i][m] = 1$;
    end
    else
      $Z[i][m] = 0$;
    end
  end
  `/* collapse` C `*/`
  currentCluster = 1;
  for $i = 1$ **to** n **do**
    if $Z[i][f] = 1$ **then**
      if i *not yet assigned to cluster in* f **then**
        $C[i][f]$ = currentCluster;
        currentCluster++;
      end
      for $j = 1$ **to** n **do**
        if $E^m[i][j] \geq$ *clusterThreshold* $\wedge$ j *not yet assigned to cluster in* f **then**
          $C[j][f] = C[i][f]$;
        end
      end
    end
  end
  foreach *pair of clusters* c *and* d **do**
    W^m[c][d] $= \frac{1}{n_{oc}(s,m,c) \cdot n_{oc}(s,m,d)} \sum_{i \in c,\ j \in d} X^m[i][j]$;
  end
end
return Z, C, W

Object 1	1
Object 2	1
Object 3	2
Object 4	2

(a) Cluster Matrix of A

Object 1	2
Object 2	2
Object 3	1
Object 4	

(b) Cluster Matrix of B

Object 1	1
Object 2	2
Object 3	
Object 4	1

(c) Cluster Matrix of D

Figure 4.3: Initial cluster matrices of three samples

1	1	0	0
1	1	0	0
0	0	1	1
0	0	1	1

(a) E^1 of A

1	1	0	0
1	1	0	0
0	0	1	0
0	0	0	0

(b) E^1 of B

1	0	0	1
0	1	0	0
0	0	0	0
1	0	0	1

(c) E^1 of D

Figure 4.4: Cluster matrix of common forms

The thresholds influence how many features and clusters are created during the collapse phase. A higher feature threshold will cause less feature membership, as an object needs to be in a feature in a higher proportion of the samples for it to be activated. A higher cluster threshold will cause more clusters in each feature, since two objects need to be in the same cluster for a higher proportion of the samples to get sorted into the same resulting cluster.

Example. Let A, B and D be samples with one feature each. Figure 4.3 shows the initial cluster matrices of each sample. The single feature contains every object in sample A, while it lacks object four in sample B and object three in sample D. We show the common cluster matrices in Figure 4.4. These three $n \times n$-matrices are averaged by Algorithm 10, resulting in the single matrix shown in Figure 4.5(a).

We assume a feature and cluster threshold of 0.5. The averages in Figure 4.5(a) result in the binary matrix of Figure 4.5(b). Figure 4.5(c) shows the final cluster matrix of the mean.

Calibrating Thresholds. We implemented an automatic calibration of the thresholds so the result roughly matches the averaged α and λ. This uses a brute force approach: The interval $[0, 1]$ of possible thresholds is divided into steps of length 0.01. For each possible feature threshold, a temporary feature matrix Z_{temp} is created and a posterior mean of α is sampled. The threshold which results in the closest match of α is kept. Afterwards, the same is repeated for the cluster and γ threshold. Since α only depends on the number of features, there is some room for optimization.

1	$\frac{2}{3}$	0	$\frac{1}{3}$
$\frac{2}{3}$	1	0	0
0	0	$\frac{2}{3}$	$\frac{1}{3}$
$\frac{1}{3}$	0	$\frac{1}{3}$	$\frac{2}{3}$

(a) Averaged E^1

1	1	0	0
1	1	0	0
0	0	1	0
0	0	0	1

(b) Thresholded E of mean

Object 1	1
Object 2	1
Object 3	2
Object 4	3

(c) Final cluster matrix C

Figure 4.5: Averaging and Collapsing several instances

Time and Memory Complexity. Creating the common form increases the memory consumption by a factor of n in the worst case:

$$M(\text{Com}, s) \in O(n_f(s) \cdot n(s)^2) \tag{4.14}$$

When many samples need to be aggregated, it is better to aggregate them sequentially, always having at most two common forms in memory. The time complexity is equal to the memory complexity:

$$T(\text{Com}, s) \in O(n_f(s) \cdot n(s)^2)$$

4.4 Supervised Learning Variant

In addition to the unsupervised inversion algorithm, we created a variant for supervised learning. In supervised learning, training data with known solutions is used to prepare a system, which can then be used to get solutions on real data. The features and clusters created in supervised learning correspond to known attributes of the objects used in the learning process, which makes them easier to interpret. This is useful if a ground truth is available for only some of the objects and we want to predict it for the rest of them. With a complete ground truth, one can omit parts of it and try to validate the clustering with the missing links. We use it to validate the inversion on the fMRI dataset, see Section 5.5.5.

Example. Alice has bought a box with 20 pieces of fireworks. After a labeling mix up, no one knows the color they burn. A matrix of similarities R is created and half of them are tested destructively. They burn with different colors (blue, green and gilden) and durations (10-20 seconds). A method is needed to infer the posterior distributions of burn color and duration for the remaining pieces of firework. Similar problems often occur: Measurements are incomplete or only present for a part of the data.

Algorithm. The input of the supervised learning variant consists of the similarity matrix R and a ground truth for a subset of the objects. Let n be the total number of objects, l the number of objects with a known ground truth and k the number of ground truth features.[10] The feature matrix Z and cluster matrix C are created for the subset: $Z \in \{0,1\}^{l \times k}, C \in \mathbb{N}^{l \times k}$, R is given for the full set of objects: $R \in \mathbb{R}^{n \times n}$. The columns of C and Z map to the features present in the ground truth. If a feature is categorical, one cluster is created for each value. If it is numerical, thresholding is used. After this preprocessing step is finished, Algorithm 12 is used. Z and C are constructed manually, R is reduced to the objects with a known ground truth and only W, b and σ are inverted. Since this reduced model is quite simple, the sampling process converges quickly. In the next loop, each unknown object o is assigned to features and clusters using the inversion step in Algorithm 3. It is removed afterwards to make sure the assignments of each object only depends on the known data, not on other unknown objects. The result is a distribution of feature and cluster assignments for each object.

Result of Example. The pieces of firework had two features: color and burn duration. Three clusters are created for the first feature, representing color. The durations are thresholded into 4 clusters: 10-12.5, 12.5-15, 15-17.5 and 17.5-20. After the algorithm ran, interpreting the cluster assignments is simple: If an untested piece of firework is clustered with the blue-burning pieces 90% of the time, it will probably burn blue.

[10] In the example, n was 20, l was 10 and k was 2.

Algorithm 12: Supervised learning (Sv)

Input: l objects with ground truth, R, $n - l$ unknown objects
construct Z and C manually using partial ground truth;
construct $subR \in \mathbb{R}^{l \times l}$;
construct k weight matrices of appropriate size;
for *sufficiently many iterations* **do**
    sample$W(subR)$;
    sample$b$;
    sampleσ;
end
$s = (Z, C, W, b, \sigma)$;
foreach *unknown object* o **do**
    add row representing o to Z and C;
    extend $subR$ to include o;
    for *sufficiently many iterations* **do**
        foreach *feature* m *of ground truth* **do**
            sample$Z$and$C(s,subR,o,m)$;
        end
    end
    remove o from Z, C and $subR$;
end
return *assignment distribution of unknown objects*

4.5 Time and Space Complexity

Table 4.1 shows the memory and time complexity for all algorithms presented in this chapter. We assume the most efficient version is used for sampling cluster assignments and weights. The time complexity of an iteration is dominated by sampling Z and C. In this table, "Sampling Z" means the whole Z matrix, while the Sample$Z$and$C$ algorithm sampled one entry of the Z matrix. When samples are stored, additional memory is required. The number of features and clusters appear as arguments in the time complexity. To simplify the time complexity, we take a look at the expected number of features and clusters discussed in Sections 2.5.1 and 2.5.2. By Equation 2.10, the expected number of features is in $O(\alpha \log n)$ We thus replace $n_f(s)$ with $\alpha \log n$ in asymptotical bounds.

The expected number of clusters in each feature depends on λ. If λ is 1, the expected

Step	Symbol	Time	Space
Likelihood	L	$O(n^2 \cdot f)$	$O(1)$
Likelihood Δ	LΔ	$O(n \cdot f)$	$O(1)$
Cluster Prob.	CP	$O\big((n_{cf}(s,l) + m_{\text{aux}}) \cdot T(\text{L}\Delta, s)\big)$	$O(m_{\text{aux}})$
Sampling Z	SZ	$O\big(n(s) \cdot \sum_{f=0}^{n_f(s)} T(\text{CP}, s, f, m_{\text{aux}})\big)$	$O(m_{\text{aux}})$
Sampling W	SW	$O\big(\sum_{f=0}^{n_f(s)} n_{cf}(s,f)^2 \cdot T(\text{L}\Delta, s)\big)$	$O(1)$
Iteration	Iter	$O\big(n^2 \cdot n_f(s) \cdot (\sum_{f=0}^{n_f(s)} n_{cf}(s,f) + m_{\text{aux}}))\big)$	$O(m_{\text{aux}})$
Aggregation	Agg	$O(n_f(s) \cdot n^2)$	$O(n_f(s) \cdot n^2)$
Supervised	Sv	$O(SW)$	$O(SW)$

Table 4.1: Space and Time complexity for different steps

number of clusters is the harmonic number H_n, which can be approximated with $\log n$. As λ grows, the number of clusters approaches n.

$$\mathbb{E}(n_{cf}(s,m)) \in \begin{cases} O(1) & \text{if } \lambda = 0 \\ O(\log n) & \text{if } \lambda = 1 \\ O(n) & \text{if } \lambda = \infty \end{cases}$$

For a high λ, we get the time complexities shown in Table 4.2. The feature count is replaced by $\alpha \log n$ and the cluster count per feature by n. Sampling the cluster assignment of an object i in feature m takes $O((n + m_{\text{aux}}) \cdot \mathrm{L}\Delta)$ steps, which evaluates to

$$O(n^2 \alpha \log n + n m_{\text{aux}}).$$

One iteration of the whole Z and C matrices then takes

$$O\big(n \cdot \alpha \log n \cdot (n^2 \alpha \log n + n m_{\text{aux}})\big)$$

steps, which can be simplified to

$$O\big(n^3 \cdot \alpha^2 (\log n)^2 + (n \cdot \alpha \log n \cdot n m_{\text{aux}})\big).$$

If we consider $\alpha n \log n$ to be a reasonable upper bound for m_{aux}, we get $O(n^3 \cdot \alpha^2 \log^2 n)$ as the time complexity for a single iteration of Z and C.

Sampling the weights requires

$$O\big(\sum_{f=0}^{n_f(s)} n_{cf}(s,f)^2 \cdot \mathrm{L}\Delta(s)\big)$$

steps, the expected complexity is in

$$O(\alpha \log n \cdot n^2 \cdot \alpha n \log n) = O(n^3 \cdot \alpha^2 \log^2 n).$$

One sampling iteration of the whole model consists of sampling Z, C, W and the hyperparameters. Thus, for large λ, the expected time complexity is:

$$\mathbb{E}\big(T(\text{Iter}, s)\big) \in O\big(\mathbb{E}(T(SZ,s)) + \mathbb{E}(T(SW,s)) + T(\mathrm{L},s) + \mathbb{E}(T(S\lambda))\big) = O(n^3 \cdot \alpha^2 \log^2 n). \quad (4.15)$$

If λ has values near 1, the expected number of clusters in each feature is in $O(\log n)$. The complexity of a cluster sampling step in this case is in $O\big((\log n + m_{\text{aux}}) \cdot T(\mathrm{L}\Delta) + T(\mathrm{L})\big)$. If we consider $m_{\text{aux}} + \log n$ to be bounded by n, we get an expected number of steps in $O(\alpha n^2 \log n)$ for each iteration..

4.6 Code Structure

An overview of the main compilation units is shown in Figure 4.6. Model instances are represented by the `Features` class, holding Z, C, W and the parameters as well as handling the likelihood calculation. The Gibbs sampler is contained in `Sampling`. Where the priors are not conjugate, discretized inverse transform sampling or slice sampling is used. The probability density functions for that purpose are defined in `PDFunctor`. The unit `Generative` provides prior distributions for use in `Features` and `Sampler`. During sampling, the samples generated from the Markov chains are stored for later analysis with the eponymous class. The figures in this work are a product of the `Plotting` class while the test data was parsed by the `Input` class.

Not shown are several smaller classes:

Step	Symbol	Time	Space
Likelihood	L	$O(n^2 \cdot \alpha \log n)$	$O(1)$
Likelihood Δ	LΔ	$O(n \cdot \alpha \log n)$	$O(1)$
Cluster Assignment	CP	$O\big((n + m_{\text{aux}}) \cdot T(\text{L}\Delta, s)\big)$	$O(m_{\text{aux}})$
Sampling Z and C	SZ	$O(n^3 \cdot \alpha^2 (\log n)^2)$	$O(m_{\text{aux}})$
Sampling W	SW	$O(n^3 \cdot \alpha^2 (\log n)^2)$	$O(1)$
Iteration	Iter	$O(n^3 \cdot \alpha^2 \log^2 n)$	$O(m_{\text{aux}})$
Aggregation	Agg	$O(\alpha \log n \cdot n^2)$	$O(\alpha \log n \cdot n^2)$
Supervised	Sv	$O(SW)$	$O(SW)$

Table 4.2: Expected time and space complexity for high λ

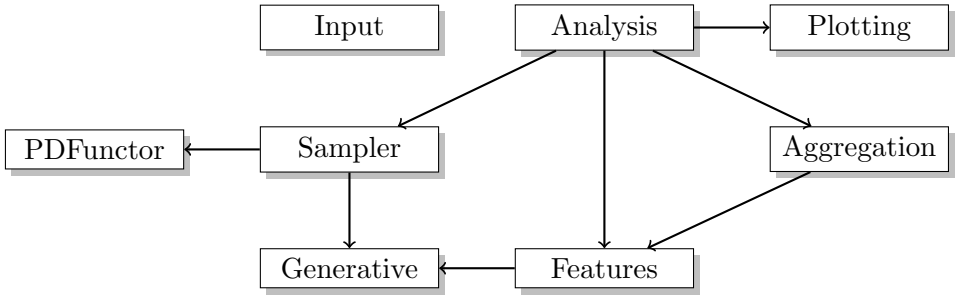


Figure 4.6: Main compilation units of implementation

- `Graph` contains the graph R.
- `Settings` stores all settings to keep the method interfaces simple.
- `TestChamber` handles test runs. The data for inversion plots in Section 5.3 was generated here.
- `Util` provides matrix manipulation, permutations and cluster quality evaluation methods.

External Libraries. We used several libraries from the BOOST project. Pseudo-random numbers from various distributions are essential to a MCMC method and we used the *Random* library to create them. The *Serialization* library is used to write logs of Markov chains and read them later. The command line interface relies on the *Program Options* library. The *Bind* and *String Algorithm* libraries are used to enable sorting of model instances and to parse file names of stored markov chains.

Data Structures. The matrices in use are stored as nested vectors. The feature presence in matrix Z is implemented as a `vector<vector<bool> >`. Cluster assignment matrix C is a `vector<vector<int> >` and the weights W a `vector<vector<vector<double> > >`. If object i does not participate in feature m and hence $Z[i][m] = 0$, $C[i][m]$ is set to -1. In the Aggregation, averages of feature presence are calculated, Z is a `vector<vector<double> >`. All probabilities are stored as their natural logarithms. This not only allows greater precision with small probabilities but also simplifies most equations considerably.

4.7 Optimization

In this section, we present engineering efforts to improve the time consumption of our implementation. An analysis of asymptotical complexity is found in Section 4.5, time measurements are found in the Evaluation, Section 5.1. Unless noted otherwise, the speedups

in this section were measured with 50 objects and 50 chains with 100 iterations each on an Intel P8600 clocked at 2.4 Ghz.

4.7.1 Calculating the Likelihood

Calculating the likelihood requires multiple iterations over the whole feature and cluster matrices to create the predicted matrix and compare it with the actual input. It is therefore quite expensive and dominates the time consumption in a naive implementation, presenting an ideal target for optimization efforts. Fortunately, in many cases only some of the link strengths between objects are affected while the others remain unchanged. Since these improvements affected the asymptotical complexity, we already discussed them in Section 4.2.

In addition to reducing the instances where a likelihood density needs to be evaluated, we can simplify the calculation itself. Back in Section 3.2, we defined the likelihood of a model instance as the multiplied edge likelihoods:

$$\mathcal{L}(s|\mathcal{R}) \propto \prod_{i=1}^{n}\prod_{j=1}^{n}\mathcal{N}(R[i,j],T[i,j],\sigma_s) \tag{4.16}$$

The probability density function of the normal distribution is:

$$f_{\mathcal{N}}(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

As discussed in Section 4.6, the implementation uses the logarithmic form to store all probabilities. Equation 4.16 thus becomes:

$$\log\big(\mathcal{L}(s|\mathcal{R})\big) = \sum_{i=1}^{n}\sum_{j=1}^{n}\Big(-\big(\log(\sigma)+2\cdot\log(\pi)\big) - \frac{(R[i,j]-T[i,j])^2}{2\sigma^2}\Big).$$

The logarithms of π and σ can be moved in front of the sum:

$$\log\big(\mathcal{L}(s|\mathcal{R})\big) = -\big(\log(\sigma)+2\cdot\log(\pi)\big)\cdot n^2 - \big(\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{(R[i,j]-T[i,j])^2}{2\sigma^2}\big).$$

Just moving the logarithms out of the loop brought a surprising speedup of about 15%, as logarithms are expensive to compute. For sampling purposes, we only need a function proportional to the likelihood. We can omit the common factors and simplify the equation even further:

$$\log\big(\mathcal{L}(s|\mathcal{R})\big) = \big(\sum_{i=1}^{n}\sum_{j=1}^{n} -\frac{\big(R[i,j]-T[i,j]\big)^2}{2\sigma^2}\big).$$

4.7.2 Feature Compression

A significant amount of time was spent in the likelihood calculation iterating through all features to figure out which of them were shared by the two objects in question. Since the feature matrix Z is quite sparse, most of the features were not shared by both objects. When limiting the number of features to 32 or 64 (a reasonable assumption) we can store a row of Z in a single *int* respective *long*. We obtain the shared clusters with a simple bitwise and-operation and subsequent bit shifting. Algorithm 13 shows how a single entry of T is computed in the optimized version. Saving the boolean matrix entries in a single

Algorithm 13: Computing T[i][j]

Input: compressedFeatures(i), compressedFeatures(j), ClusterMatrix c, Weights w, bias b

```
   /* getting shared features                                          */
1  int common = compressedFeatures(i) & compressedFeatures(j);
2  if !common then
3  |  return b
4  end
5  double result = b;
6  for m = 0 to f do
   |  /* see if current feature is shared                             */
7  |  if common & 1 then
8  |  |  ic= C[i][m];
9  |  |  jc= C[j][m];
10 |  |  result += W(m, ic, jc);
11 |  end
12 |  else if !common then
13 |  |  return result
14 |  end
   |  /* shift feature int to the right                               */
15 |  common = common ≫ 1;
16 end
17 return result
```

field greatly improves cache locality. Memory accesses are reduced to a single field instead of four pointer evaluations each iteration. Depending on which other optimizations were active, compressing the Feature Matrix decreased runtime between 0 and 50%. In model instances with a high amount of features, De Bruijn sequences [15] could be used to skip unused features.

4.7.3 Slice Sampling

Instead of creating a histogram for sampling λ, we could use slice sampling. It was used in the original algorithm by Palla, but didn't create samples sufficiently proportional to the given distribution after our adaptions. If using a longer burn-in period, it was no longer faster than creating a histogram. No likelihood calculation of R is necessary for sampling λ, therefore it is satisfyingly fast even without optimizations.

4.8 Parallelization

Only minor changes were necessary to make different markov chains run in parallel, as they have no data dependencies between them. We used OpenMP, a library for shared memory parallelization [2].

Within one chain, several steps exist where parallelization can be used. However, we used enough different markov chains to saturate the processing cores available. As intra-chain parallelization would have been of little benefit, these considerations remain theoretical.

Model Generation. The Chinese restaurant process is done independently for each feature and could in theory be parallelized. We omitted the parallelization because the runtime is dominated by other steps.

Creating Histograms. If histograms are used, parallelization might be useful. Since we replaced histogram creation with a faster direct sampling in most cases, there is nothing left to parallelize. Only the parameter λ is sampled with a histogram, but it does not need a full likelihood calculation and thus very little runtime. The probability density functor of λ is threadsafe and a parallelization is possible.

Calculating cluster probabilities (CP). In Algorithm 4, the likelihood is computed for each possible cluster. These computations are independent and could be parallelized.

Calculating the Likelihood. The likelihood for a solution s considering a graph R_{input} is a product of single link likelihoods. These can be computed independently and multiplied afterwards. The whole model instance is required to calculate the likelihood, resulting in large data transfers.

Aggregation. Each field of two common forms can be averaged independently. This can be done in up to $n^2 \cdot f$ independent threads. In the current implementation, aggregating l samples with n objects each and f features is in $O(l \cdot n^2 \cdot f)$. Since the average is associative, one can average l samples in sets of k, obtaining a k-ary tree with l leaves and a height of $\log_k l$. With l/k processing cores, the time complexity would be reduced to $O(k \cdot \log_k l \cdot n^2 \cdot f)$, the total work remaining asymptotically the same. The memory complexity would be increased to $l \cdot n^2 \cdot f$, as more common forms have to be kept in memory simultaneously. If memory consumption is a limiting factor, one could reduce the number of concurrent threads to $c \leq l/k$. This results in a time complexity of $O(H_{\log_k l,k} l/c \cdot k \cdot n^2 \cdot f)$ and a memory complexity of $O(c \cdot n^2 \cdot f)$, where $H_{n,m}$ is the generalized harmonic number.

Collapsing common forms. Each feature can be collapsed independently. When calibrating the thresholds, Algorithm 11 was called multiple times to get expected means for α and γ for each threshold value. These calls can be parallelized easily.

5. Evaluation

In this chapter, we test our algorithm on synthetic and real data. In Section 5.1, we evaluate the time consumption. How does the practical runtime compare to the theoretical time complexity discussed in Section 4.5?

After discussing convergence in Section 5.2, we measure how well the model is inverted in Section 5.3. This is straightforward for parameters α, λ and b but will need a measure of quality for matrices Z, C and W, discussed in Section 5.3.5. These preparations will be used in the inversion of the model with real data in Section 5.5.

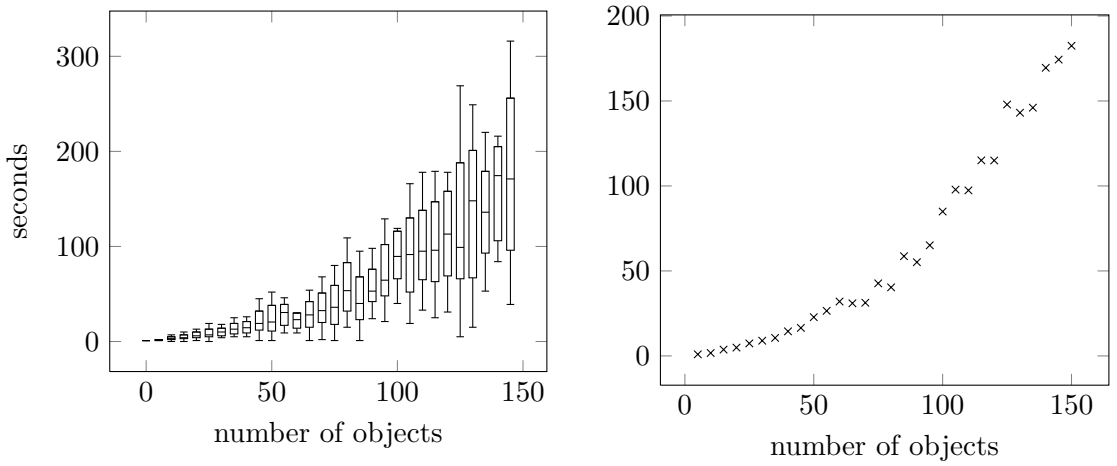
5.1 Time

For the time measurements, we set α and λ to 1 and generated matrices Z, C, W randomly with the generative process described in Section 4.1. From this model, the predicted matrix R was created with a variance of 0. The inference algorithm then tries to recover Z, C and W from R. We used 1000 sampling steps on an E5430 Intel Xeon CPU, clocked at 2.66GHz. The operating system was Suse Linux, running on kernel 3.4.47-2.38. We compiled the binary with GCC version 4.6.3 on Ubuntu 12.04. While 1000 iterations are too short for convergence in most cases, the required time is proportional to the time needed for longer chains, which allows us to test runtime predictions.

The resulting runtimes of the fully optimized version can be seen in Figure 5.1. The time needed to recover a model increases with the number of objects n. The data points are noisy, which is to be expected: The time depends on the complexity of the model. In Table 4.1, we saw that time scales quadratically with the number of objects:

$$T(Iter, s) \in O\big(n^2 \cdot n_f(s) \cdot (\sum_{f=0}^{n_f(s)} n_{cf}(s, f) + m_{\text{aux}}))\big)$$

Since the ground truth was randomly generated each time, the infered number of clusters varies and the data points are not perfectly smooth. Repeating the measurements sufficiently often would average out these variances. The whiskers in the boxplot extend to the most extreme datapoints.



(a) Median and quartiles, in seconds

(b) Average time in seconds

Figure 5.1: Time for 1000 iterations of optimized sampling process with random ground truth. The runtime grows about quadratically in the number of objects.

5.1.1 Effects of Optimizations

We measured the speedup of the four main optimizations in the cluster selection, bias sampling, weight sampling and feature compression. The first optimization in all three sampling steps was to improve the likelihood calculation of a possible change by only computing a likelihood delta. In the bias and weight sampling phase, additional improvements were possible by exploiting the conjugate prior:

The experiments were done on a single core of a Intel Core P8600 clocked at 2.4 Ghz. 4 GiB of RAM were equipped along with 3 MiB cache. We used GCC 4.6.3 to compile the implementation and Ubuntu 12.04 to run it. The histograms required for the naive and Δ implementation were created for the interval $[-10, 10]$ with a step size of 0.1. This is a rather conservative choice. For most applications, a larger histogram would be required, resulting in even higher speedups when it can be omitted.[1] Palla [27] also replaced the full likelihood calculation with a likelihood delta in the weight and cluster sampling steps. Because of the different likelihood function, their priors were not conjugate and they were not able to use direct sampling. They used slice sampling instead, whose complexity is constant with respect to the input value but might take several sampling steps to obtain good samples. Table 5.1 shows the runtimes for each combination of optimizations enabled and disabled. The columns indicate the state of cluster optimization and feature compression. Rows show whether bias and weights were sampled naively with a histogram and full likelihood calculation, with a histogram and a Δ likelihood calculation or using the conjugate prior and the Box-Muller-method. The times are seconds for 50 runs with 100 iterations each. When all optimizations are active, a speedup of 30 compared to the naive implementation is visible.

Direct comparison with the original MATLAB implementation of Palla [27] is difficult, because the real valued test data cannot be used for this. We used thresholding on the distances of random points in $\mathbb{R}^2$ in one experiment and an input consisting only of zeros in the second. 785 seconds were needed for 100 iterations of the first experiment, 104 seconds for 100 iterations of the second. 104 seconds can thus be used as a lower bound for the

[1] The most extreme bias we encountered with real data was $-1.5 \cdot 10^5$, slowing down a histogram-based approach to the point of unusability.

		Naive Clusters		Delta Clusters	
		Normal Z	Compressed Z	Normal Z	Compressed Z
Naive Weights	Naive Bias	1816	1400	1346	1307
	Δ Bias	1403	1384	1313	1557
	Conj. Bias	1486	1343	1405	1255
ΔWeights	Naive Bias	377	254	164	131
	Δ Bias	398	223	116	88
	Conj. Bias	379	218	125	102
Conj. Weights	Naive Bias	381	217	109	95
	Δ Bias	325	161	93	56
	Conj. Bias	336	186	80	58

Table 5.1: Total runtime in seconds for all combinations of optimizations enabled and disabled. 50 runs with 100 iterations each. The optimizations resulted in a speedup of 30 against the naive implementation.

runtime of 100 iterations. Our implementation used an average time of $56/50 = 1.12$ seconds for 100 iterations, achieving a speedup of 92 to 700.

5.2 Convergence

Markov Chain Monte Carlo (MCMC) methods use Markov chains whose limiting distribution is the desired distribution. To have a limiting distribution, the sampling chains need to actually converge. To make sure the sampling process doesn't get stuck in local optima, it is useful to start several chains with overdispersed starting parameters. If they still converge against the same values, they likely have the same limiting distribution which is indeed the desired one. It is also easier to estimate the burn-in period when several chains approach the same distribution.

Example. As an example, we started 7 different runs with the same random ground truth and let them run for 200000 iterations.[2] Figure 5.2(a) shows the cumulative means of alpha for each chain. They seem to converge to about 0.72, with a burn-in period of about 25000 iterations. Figure 5.2(b) shows the cumulative means of lambda for the same set of chains. As with α, the chains converge with a burn-in period of about 25000 iterations. The true values were 1 for α, which differs from the inversion result by 30% and 1 for λ, which is recovered quite well. The distributions of samples are shown in Appendix A, Figure A.1 and A.2.

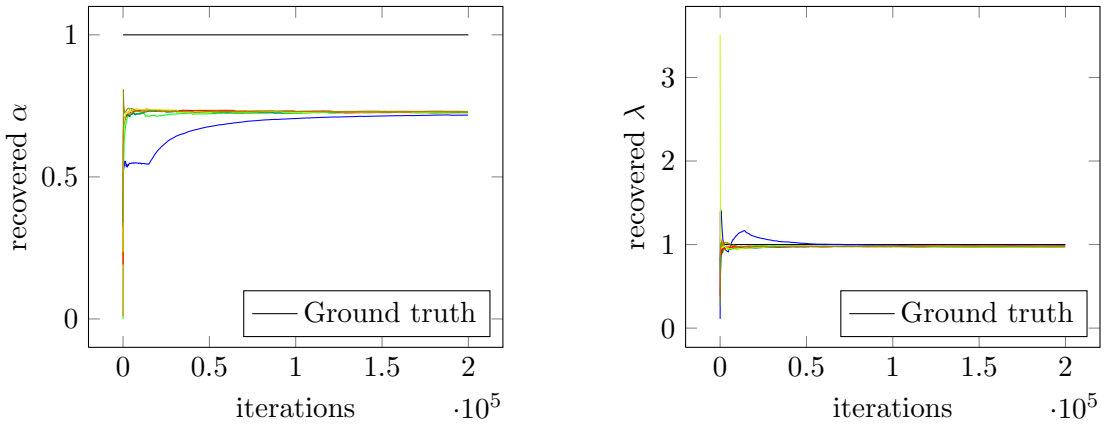
5.3 Inversion

After seeing the chains converging in the previous section, we need to evaluate the limiting distribution. Does it recover the hidden features and parameters? In other words, is the model suitably *inverted*?

In this section, we discuss the inversion of each part first separately and then together.

To grasp the performance in different regions of parameter space, we will use averages over many runs with different ground truths, such as in Figure 5.17 or 5.1. To illustrate typical results and to show how we arrived at our conclusions, we sometimes show plots of single runs, such as in Figure 5.2.

[2]The required command was "ILA --randomgraph --endlesschains --iterations 200000 --seed 1375366548".



(a) Recovered α after 200000 steps

(b) Recovered λ after 200000 steps

Figure 5.2: Convergence of example chains. Both α and λ approach the final value after about 25000 iterations, the cumulative means are indistinguishable from this point.

5.3.1 Alpha

The results of single parameter estimations are the easiest to validate. We first invert only the connection of α and Z as shown in Figure 5.3(a), where the inversion direction is marked with a red arrow. We only evaluate the inversion step of α from Z. The other random variables λ, b, Z, C and W are fixed. In this case, the sampling chain is independent even from its present state, and we have a Bernoulli scheme.

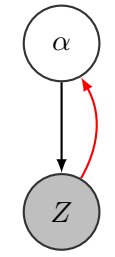
Afterwards, we invert the whole model starting from R, see Figure 5.4(a). In both cases, we assume a Gamma prior: $\alpha \propto \Gamma(1, 1)$.

Example Run. Figure 5.5 shows an example inversion of α from Z. Since the gamma prior was conjugate to the likelihood in the Indian Buffet Process, we can compute the posterior distribution directly without sampling. The original α is 2.83 and the posterior mean is 1.81, both are marked with vertical lines. Some difference is to be expected because of the discretization happening when constructing Z.

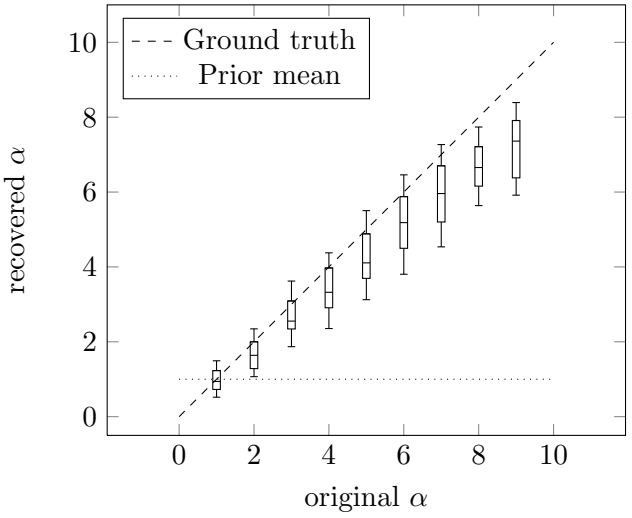
Average Results. In Figures 5.3(b) and 5.4(b), we show how well α is recovered for ground truth values in $(0.5, 10)$. These values were aggregated over 50 different random ground truths after 100 sampling steps. Figure 5.3(b) shows the average recovered α if inverted from Z. Figure 5.4(b) shows the same but for inversion of the complete model directly from R. A small underestimation is visible in both cases and expected because we use an exponential distribution with a mean of 1 as the prior distribution. This fits well with the inversion of α visible in the convergence plot in Figure 5.2(a). Getting a infered value of 0.7 for a ground truth of 1 is a typical result when compared with Figure 5.4(b). The inversion of α is more accurate when inverted from Z than if inverted from R. In the second case, the amount of structure needed to fit the data has to be infered as well. The additional inversion step leads to reduced precision. In conclusion, the inversion from Z works well and the inversion from R has an increasingly large margin of error when α increases.

5.3.2 Lambda

Similar to the inversion of α, we first invert λ from C, see Figure 5.6 and then through C from R, see Figure 5.7. When inverting λ from C, the other random variables are fixed to

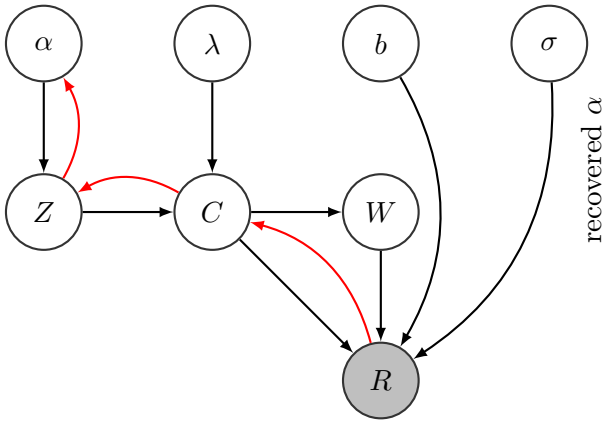


(a) Sketch

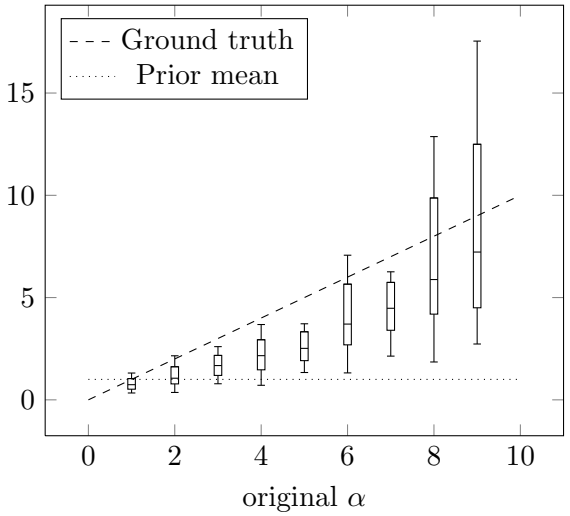


(b) Result

Figure 5.3: Inversion of α from Z



(a) Sketch: α from R



(b) Result: α from R

Figure 5.4: Inversion from R

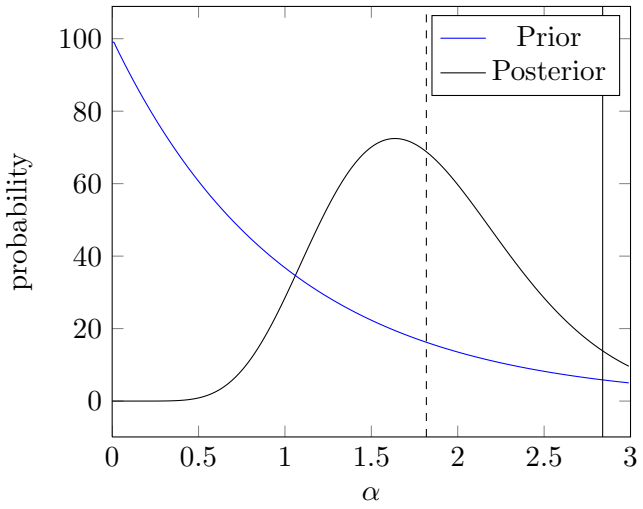


Figure 5.5: Example inversion of α. Solid line at 2.83: True α. Dashed line at 1.81: estimated α

the ground truth values. We assume a Gamma prior: $\lambda \propto \Gamma(1, 1)$.

Example Run. An example inversion of λ from C is shown in Figure 5.8. The ground truth value is 1.04, shown as a solid vertical line and the posterior mean is 1.76, shown as a dashed line. The prior distribution is shown as the blue line, samples from the posterior distribution are marked as crosses. At first, this example seems unusual in that λ is estimated too high. However, λ is only estimated too low for values higher than 2, see Figure 5.6.

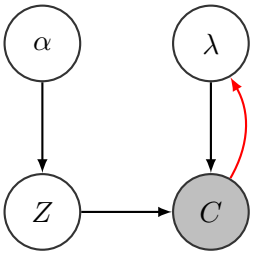
Average Results. Averaged recovered values of λ are shown in Figures 5.6 and 5.7. As with α, these values were accumulated over 50 runs with 100 iterations each. The other values including α were sampled from the prior. For ground truth values near 1, the infered values are also close to 1, which matches the convergence plot in Figure 5.2(b). For higher values, the inversion is noisier than when inverting α. The value of λ is consistently estimated to low when inverting from C and even more so when inverting from R. This effect was already visible in the original implementation of Palla [27].

Both the poor inversion quality and the underestimation are probably caused by the higher amount of discretization: When α is low, only very few objects participate in each feature, offering few possibilities for different cluster partitions. If, for example, only two objects participate in the first feature, they can either be in the same cluster or not, even a very high λ will result in the same partition as a medium one.

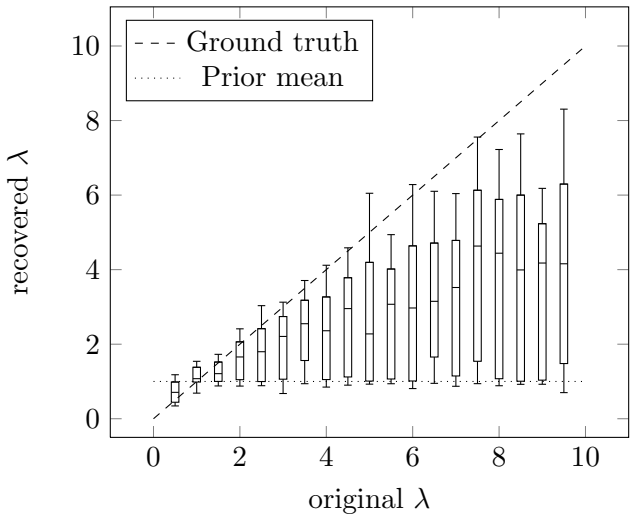
If we assume a feature with 10 objects and a λ value of 5, the expected number of clusters is $\approx$5.8. This increases to $\approx$ 7.18 for $\lambda = 10$ and 7.86 for $\lambda = 15$. With the discretization to natural numbers, these will be difficult to separate. The higher n and α, the easier it is to recover λ from C. Inverting λ from R works best if n is high and α is low.

5.3.3 Bias

Figure 5.9 shows the inversion of b (the bias term) when all matrices R, W, and C are known. With this information, estimating b is a simple average operation and gives flawless results, see Figure 5.9(b). When the whole model is inverted simultaneously, the inversion quality of b is reduced. Figure 5.10 shows the recovered values for b when Z, C, W, α and λ

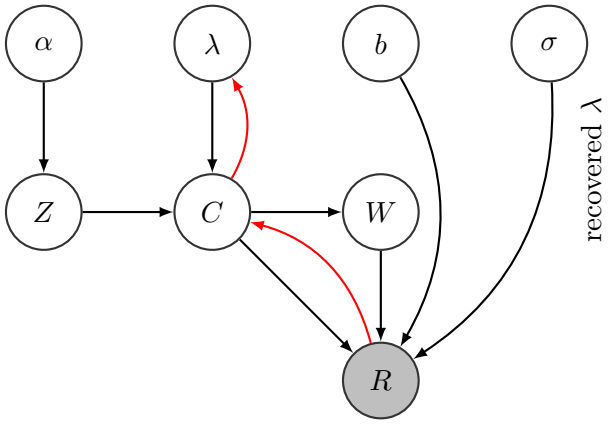


(a) Sketch

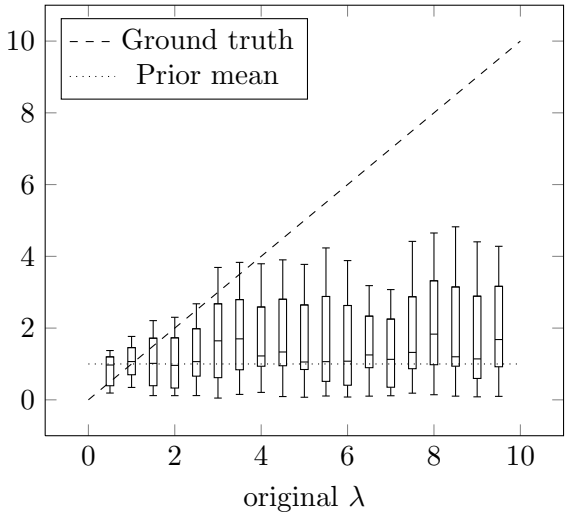


(b) Recovered λ

Figure 5.6: Inversion of λ from C



(a) Sketch: λ from R



(b) Result: λ from R

Figure 5.7: Inversion of λ from R

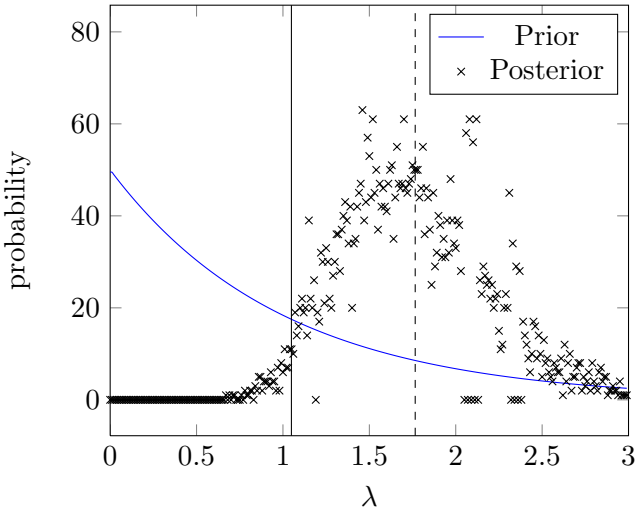
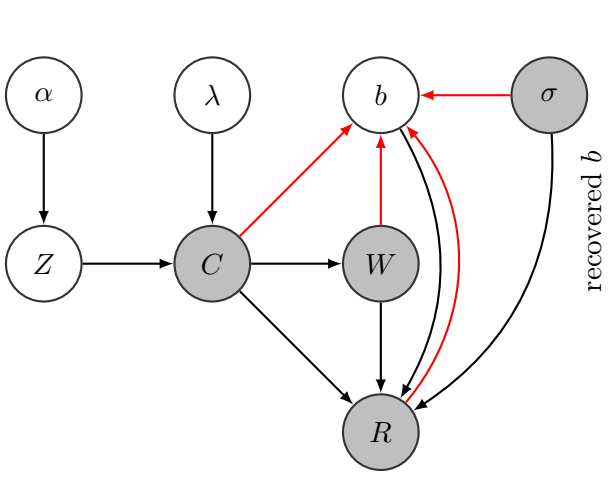
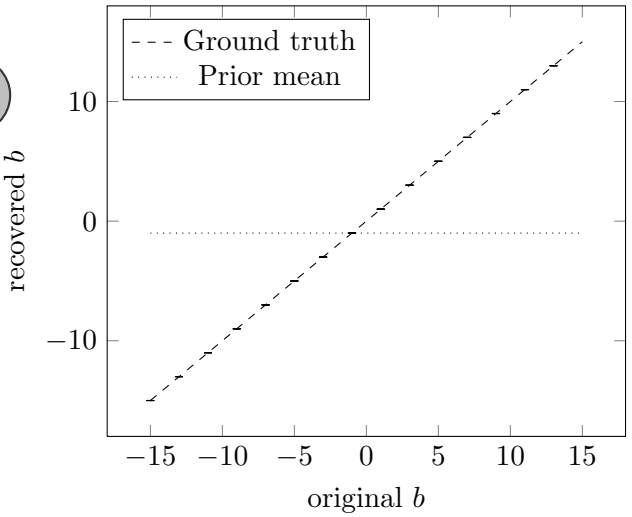


Figure 5.8: Example inversion of λ. Solid line at 1.04: true λ. Dashed line at 1.76: estimated λ

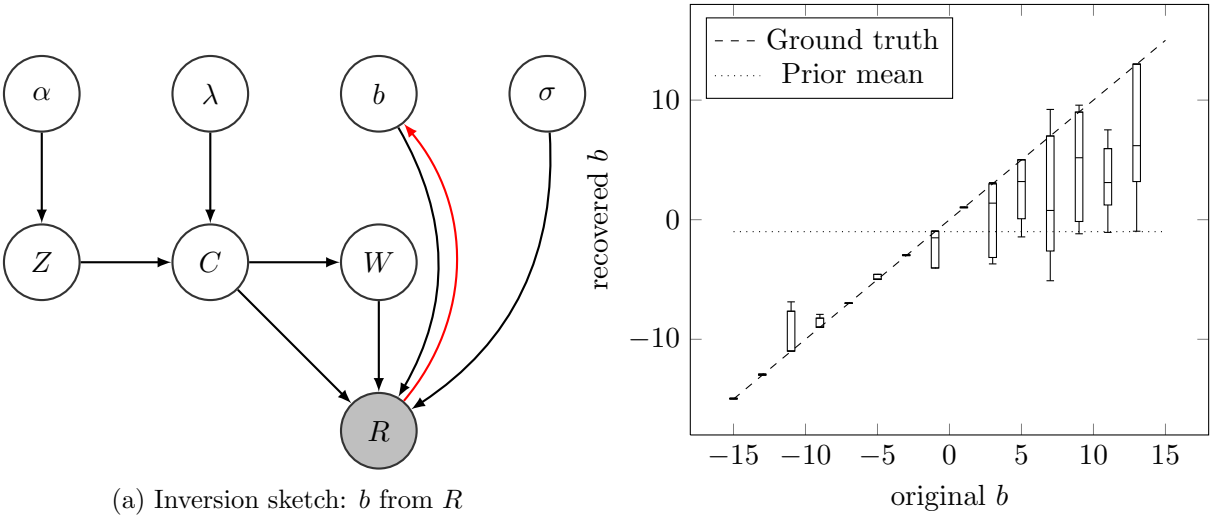


(a) Inversion sketch: b from R, W, C, σ



(b) Recovered value of b

Figure 5.9: Inversion of b from R, W, C, σ



(a) Inversion sketch: b from R

(b) Recovered value of b

Figure 5.10: Inversion of b from R

are sampled as well. The bias is recovered well for values below zero but is underestimated for values above zero. This is caused by the different priors: In our example, the prior for the bias is -1 whereas the prior for the weight matrices is 2. If the edge weight between two objects i and j is very low, it is unlikely that i and j share a feature. In this case, the connection is modeled only with the bias, which is therefore recovered quite well. For higher values of b, features are created to account for them and the bias is estimated too low. This problem might be reduced if the variance b_σ of the prior distribution for the b is set to a higher value in exploratory analysis.

5.3.4 Noise

As with the bias, the noise estimate σ is quite easy to invert if the correct model is supplied along with the input data. Figure 5.11 shows σ inverted from R, Z, C and W. It was correctly recovered in all test cases. If the whole model is inverted simultaneously, the inversion quality is reduced, as seen in Figure 5.12. While a correlation exists between the original σ and the recovered σ, the spread is higher and outliers exist. Especially for higher values of σ an underestimation occurs: The sampling process creates features and clusters to fit the noise and the noise estimate shrinks. During inversion, we mostly set σ fixed to 1.

5.3.5 Feature Matrix Z and Cluster Matrix C

While it is straightforward to compare parameter estimates for α and λ, matrices Z and C may be permuted and comparing the multiple features and clusterings requires some considerations. Fortunately, methods to compare two clusterings with each other exist in the literature. We can use these as components for an evaluation strategy.

Balanced Purity. Measures commonly used to evaluate clusterings against a known ground truth are the *balanced purity* (BP)[19] and the *normalized mutual information* (NMI)[19]. BP and NMI are both measures of clustering quality and result in a value between 0 and 1. The higher the value, the better the clustering.

Let s and g be clusterings of objects, where s is a solution to evaluate and g is the ground truth. The balanced purity of s and g is calculated using equations 5.2, 5.1 and 5.3.

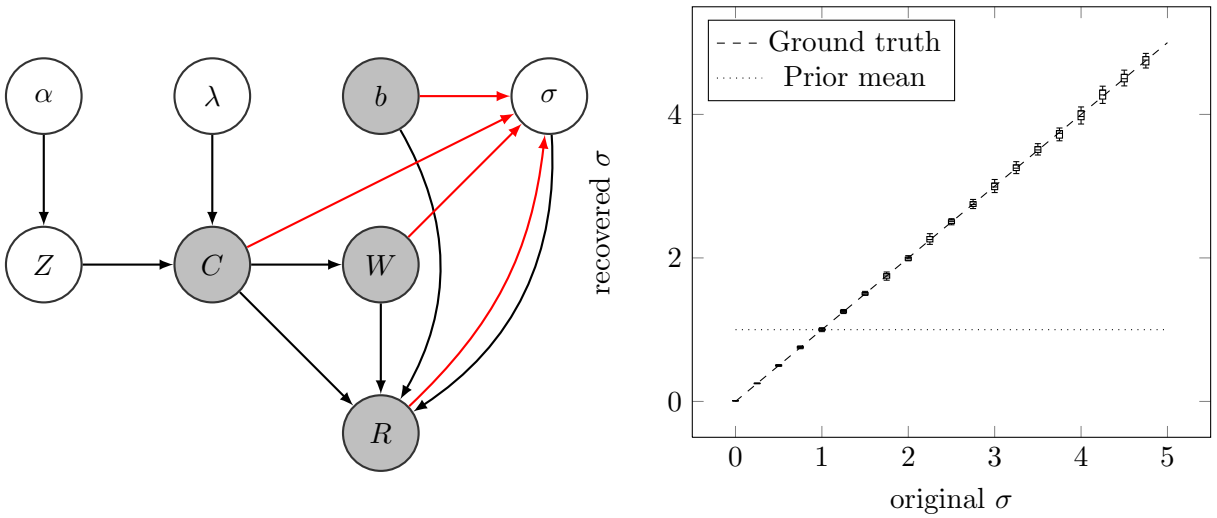


Figure 5.11: Inversion of σ from R, W, C, b

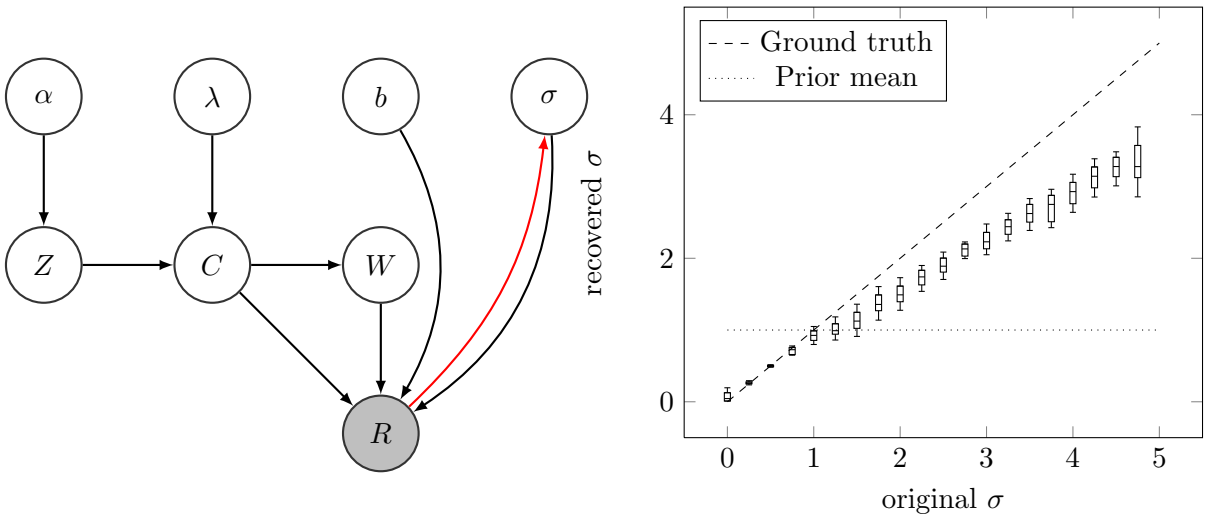


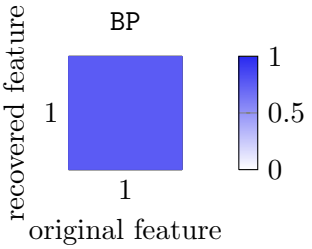
Figure 5.12: Inversion of σ from R

Object	
1	1
2	1
3	1
4	2
5	2

(a) Clusters of s_1

Object	
1	3
2	2
3	2
4	1
5	1

(b) Clusters of s_2



(c) Purity plot of Figure 5.13(a) and 5.13(b)

Figure 5.13: Example of purity visualization

In the purity calculation, we iterate over all clusters of s and assign each of them to the cluster of g it has the most objects in common with. The cardinalities of the intersections between each cluster of s and its corresponding cluster in g are summed and divided by n.

$$\mathrm{P}(s,g) = \frac{1}{n}\sum_{k=1}^{n_{cf}(s)} \max_{1\le l\le n_{cf}(g)} |c(s,k)\cap c(g,l)| \tag{5.1}$$

The ratio is needed for the normalization. It only depends on the ground truth and is the cardinality of the biggest original cluster divided by the number of objects.

$$\mathrm{r} = \frac{\max_l |c(g,l)|}{n} \tag{5.2}$$

Finally, we combine the ratio, the purity and the number of clusters to obtain the balanced purity, which is normalized to the interval $[0,1]$.

$$\mathrm{BP}(s,g) = (1-\frac{1}{n_{cf}(g)})\cdot\frac{P(s,g)-r}{1-r}+\frac{1}{n_{cf}(g)} \tag{5.3}$$

Example. Let s_1 and s_2 be two clustering solutions of size 5 with one feature each: $n(s_1) = n(s_2) = 10$, $n_f(s_1) = n_f(s_2) = 1$. The cluster assignments are shown in Figure 5.13.

The balanced purity of the first feature of s_1 against the first feature of s_2 is shown in Equations 5.4, 5.5 and 5.6.

$$r = \frac{2}{5} \tag{5.4}$$

$$P(f^1(s_1), f^1(s_2)) = \frac{1}{5}\cdot(2+2) = \frac{4}{5} \tag{5.5}$$

$$BP(f^1(s_1), f^1(s_2)) = (1-\frac{1}{3})\cdot\frac{0.8-0.4}{1-0.4}+\frac{1}{3} = \frac{2}{3}\cdot\frac{2}{3}+\frac{1}{3} \simeq 0.77 \tag{5.6}$$

The balanced purity in this example is about 0.77. It is clearly asymmetric:

$$BP(f^1(s_1), f^1(s_2)) \simeq 0.77 \neq 1 = BP(f^1(s_2), f^1(s_1))$$

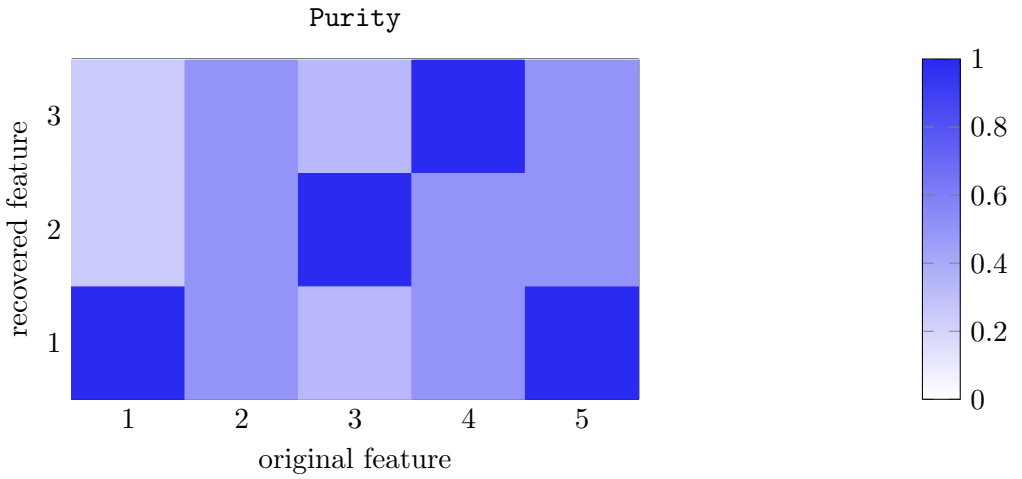


Figure 5.14: Purity matrix plot of example run with random ground truth

Application to solutions with multiple features. The BP allows us to evaluate the accuracy of a single feature. However, we have multiple features to evaluate and thus need to extend the BP. Let s_1, s_2 be clustering solutions with $n_f(s_1) \geq 1, n_f(s_2) \geq 1$ For each pair of features $i, j \in F(s_1) \times F(s_2)$, we calculate the balanced purity of feature $f^i(s_1)$ against feature $f^j(s_2)$ and obtain a matrix. We call this matrix $BP(s_1, s_2)$ and it's dimensionality depends on the feature count of s_1 and s_2:

$$BP(s_1, s_2) \in [0, 1]^{n_f(s_1) \times n_f(s_2)} \tag{5.7}$$

In the previous example, this matrix would have consisted of a single value, as seen in Figure 5.13(c).

Since the generated data matrix R is invariant under feature reordering, we have to permit permutations in our evaluation. The results of an example run with a random ground truth can be seen in Figure 5.14.

Had the model been recovered perfectly, we would see a purity of 1 at the diagonal. In our example, we see that the second original feature was not recognized and the first feature of the inversion corresponds with the first and fifth original feature. Note that a higher purity is easy to achieve if the number of found clusters increases. If each subject gets its own cluster, the purity would be 1 – even though this is a very poor clustering solution! Thus, we cannot use the purity alone to evaluate the algorithm, since we would invariably cause it to have a higher number of clusters than desired.

Normalized Mutual Information. The normalized mutual information (NMI) is a symmetric measure of cluster quality, it automatically penalizes a high number of clusters. When we want to obtain the NMI of $f^i(s_1)$ in solution s_1 and $f^j(s_2)$ in solution s_2, we use equations 5.8 and 5.10. I is the mutual information:

$$I\big(f^i(s_1), f^j(s_2)\big) = \sum_{k}^{n_{cf}(i,s_1)} \sum_{l}^{n_{cf}(j,s_2)} |c^i_{s_1}(k) \cap c^j_{s_2}(l)| \log \frac{|c^i_{s_1}(k) \cap c^j_{s_2}(l)|}{|c^i_{s_1}(k)||c^j_{s_2}(l)|} \tag{5.8}$$

To normalize the mutual information, we need the entropy H of $f^i(s_1)$ and $f^j(s_2)$. It is defined by the following equation:

$$H(f^i(s)) = -\sum_{\text{cluster } c \in f^i(s)} \frac{n_{oc}(s, i, c)}{n(s)} \log \frac{n_{oc}(s, i, c)}{n(s)} \tag{5.9}$$

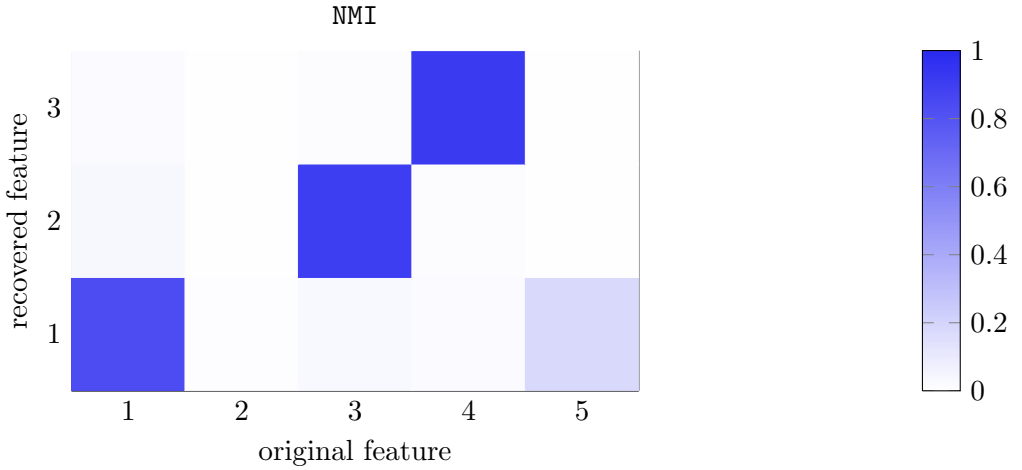


Figure 5.15: NMI of each feature of the recovered solution from each feature of the ground truth

The mutual information is then normalized using the entropy H, which results in a value between 0 and 1.

$$\mathrm{NMI}\big(f^i(s_1), f^j(s_2)\big) = \frac{I\big(f^i(s_1), f^j(s_2)\big)}{\Big(H\big(f^i(s_1)\big) + H\big(f^j(s_2)\big)\Big)/2} \tag{5.10}$$

For Figure 5.15, we used the same example but computed the NMI of each feature pair instead of the balanced purity. In general, feature pairs with a high purity also have a high NMI, but exceptions exist.

P-values. Some of the BP and NMI values shown in Figure 5.14 and 5.15 are close to chance level. For example, if the ground truth only contains two clusters, the balanced purity will always be above 0.5. To check whether the values for balanced purity and NMI we obtained are significant, we calculate the p-values by permuting the objects sufficiently often.[3] The ground truth is kept fixed and the cluster assignments of the solution are permuted randomly 1000 to 10000 times, the BP and NMI values are computed for each permutation. This gives a distribution of samples approximating the distribution of BP and NMI values. We compute p-values by comparing the inversion results with the sample distribution and see which proportion of samples has a value at least as good as the inversion result. The p-values for the balanced purity are shown in Figure 5.16. For better visibility, we capped the plotted values at 0.1. Otherwise, the color range would have been distributed over the whole interval $(0, 1)$ and the values of interest between 0 and 0.05 would have been colored too similar to discern. In many publications, a p-value of 0.05 marks the threshold of significant results[5]. The best values in Figure 5.16 for the first three features are certainly below that. Since we make multiple comparisons [20] and use $6 \cdot 7 = 35$ BP values in our example, we need to reduce the threshold. A conservative approach is to divide the desired level of significance – 0.05 in our case – by the number of values. In our example, this is 0.0014. The p-values for the first three features are below 0.001, so we can conclude the algorithm does recover the underlying structure in this example.

[3] A reader might be surprised to find p-values in a work so far concerned with a Bayesian approach. Unfortunately, a Bayesian way to estimate the probability of our algorithm recovering the underlying structure would require a prior. Giving a prior for the methodology being sound and the coding being bug-free strikes us as too self-conscious.

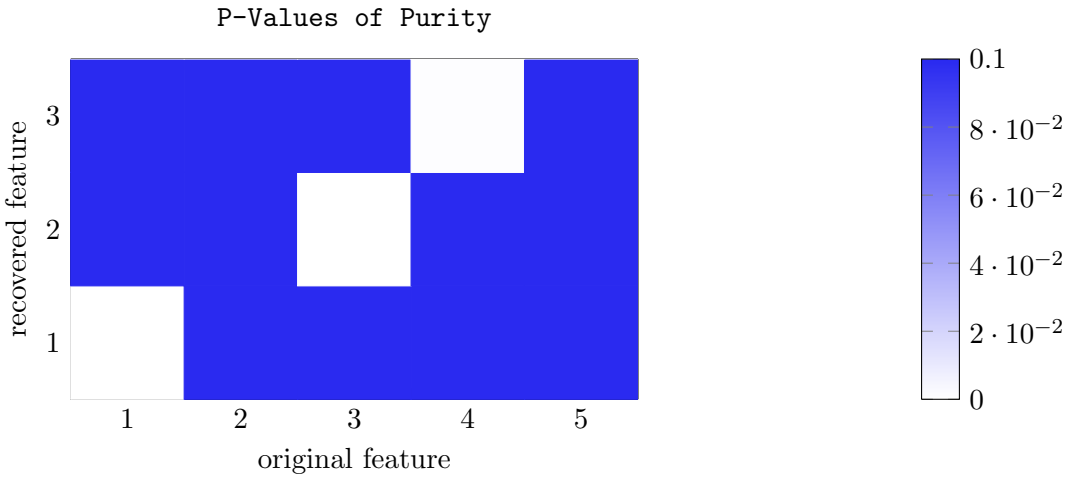


Figure 5.16: P-values of purity, capped at 0.1

Reduction to Real Number. While the BP and NMI matrices discussed in the previous paragraphs allow a detailed comparison, their detailedness can be a drawback. If, for example, we want to measure the influence of noise on the resulting quality, comparing purity matrices is cumbersome.

For this reason, we aimed at reducing the similarity of two clustering solutions to a single value.

Reducing NMI and purity matrices. When we have NMI and BP matrices, we can reorder the columns to put the highest values on the diagonal. We then take the average of the values of the diagonal, obtaining a single value. This measure is called *average NMI* for the reduction of a NMI matrix and used to create Figure 5.17, where we see the average NMI for different noise variances. For each noise value, we repeated the run with 45 different randomly generated models and averaged the results. One can see a relatively high NMI average with small noise and deteriorating quality with higher noise levels. The exact values of course depends on our prior distributions and scaling. If the prior mean for the weights and the bias are multiplied with a factor of k, Figure 5.17 would stretch accordingly. For this experiment, the means were 2 for the weights and -1 for the bias. As with all synthetic experiments, n was 50.

5.3.6 Influence of Alpha and Lambda on Average of NMI

How well features and clusters are recovered depends on how many of them exist in input data. If three features exist in a ground truth of 50 objects, these three are easier to recover than if 40 overlapping features exist. To test this intuitive reasoning, we divided the interval $[0, 20]$ into intervals of equal length. For each combination of α and $\lambda \in [0, 20]$, we started 50 chains with 5000 iterations. In Figure 5.18, we see the average NMI for each combination averaged over 50 runs each. We see that the value of α has a strong effect on the result quality, but λ has not.

5.3.7 Memory Complexity and Likelihood

It seems sensible a model with a higher memory footprint can fit an input matrix R easier. To measure the memory complexity of a model, we defined the total link count (TLC) in Section 4.1.1. Figure 5.19 shows the TLC of the seven example chains encountered in

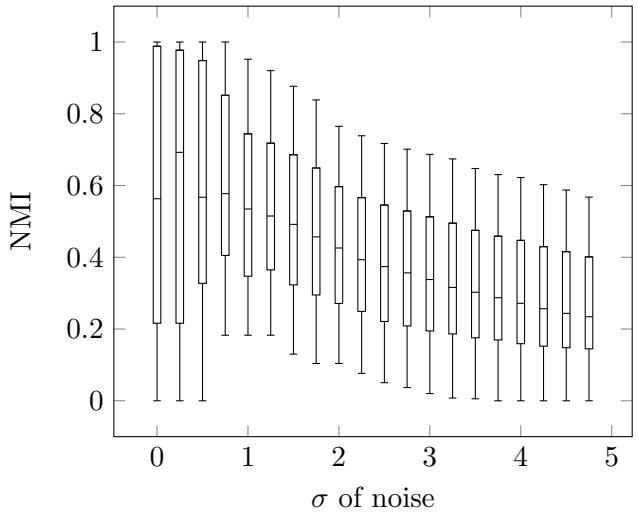


Figure 5.17: NMI average of synthetic test data with different levels of noise

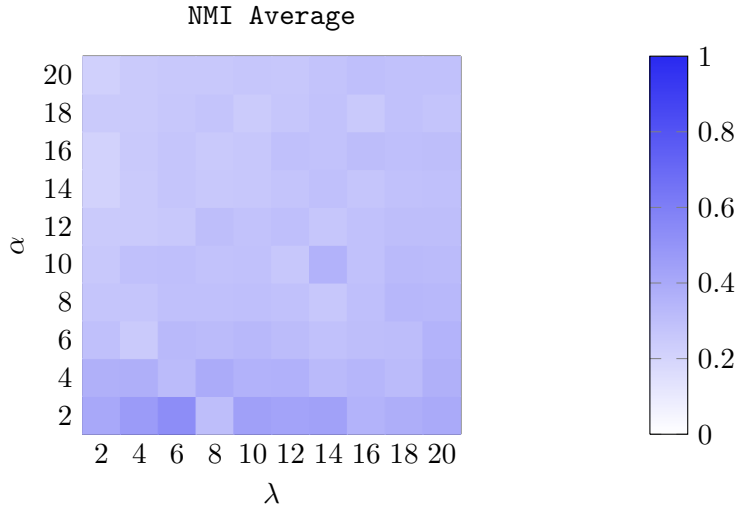


Figure 5.18: Average of NMI depending on alpha and lambda

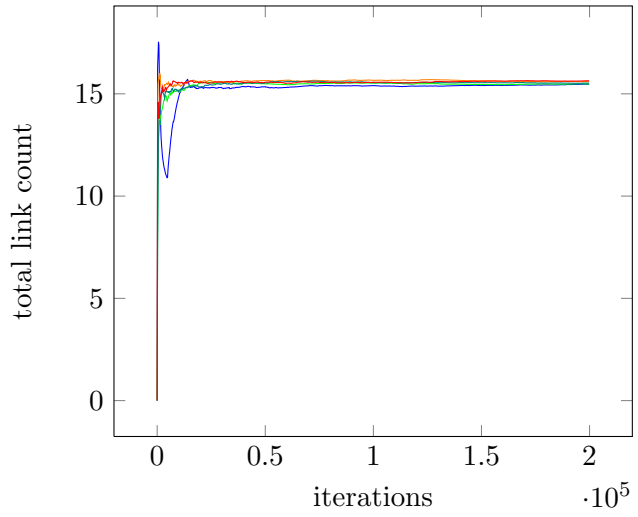


Figure 5.19: Total Link Count of example chains

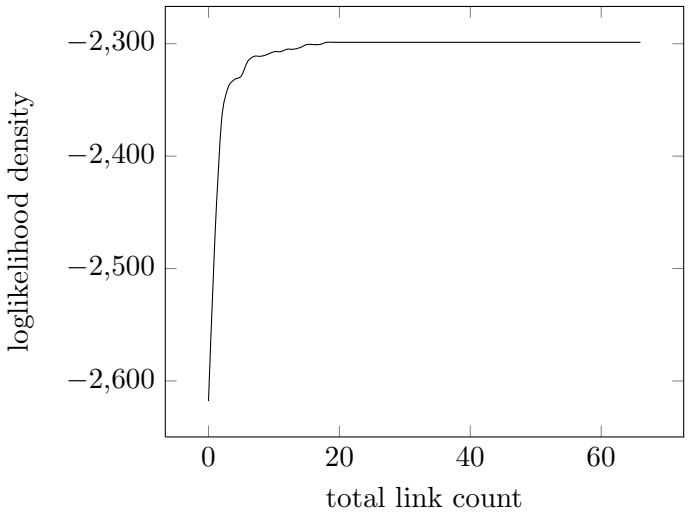


Figure 5.20: Likelihood and TLC

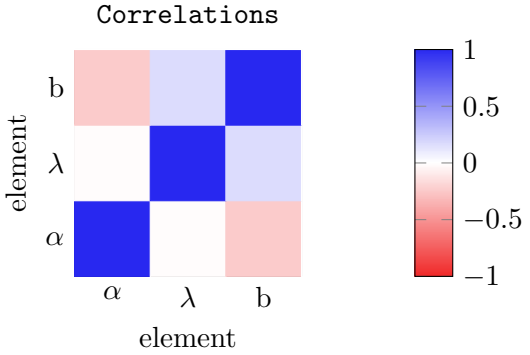


Figure 5.21: Correlations between Parameters

the convergence discussion. Surprisingly, a higher TLC is not strongly correlated with a better fit. In Figure 5.20, we plot for each TLC the likelihood of the best solution with at most that many links. The best fit for this example is achieved by a solution with a TLC of only 18.

5.3.8 Correlations between Model Elements

When the whole model is inverted, the random variables may influence each other. Figure 5.21 shows the correlation between α, λ and the bias. In this experiment, α was negatively correlated with the bias. This seems sensible: The weights have a positive expected prior and more features will need a lower bias to result in the same mean.

5.4 Aggregation

We evaluated the aggregation presented in Section 4.3 with fixed and automatic thresholds. Three sampling chains were run with a synthetic ground truth and 5000 iterations each. Figure 5.22 shows the NMI average of each trace. Each sample is evaluated against the ground truth, a NMI matrix is created and reduced to a real value, as presented in Section 5.3.5. For each iteration in the chains, all samples up to the current sample are aggregated. This cumulative aggregation is called the *aggregated mean* of a sampling chain. Figure 5.23 shows the NMI average of the aggregated means.

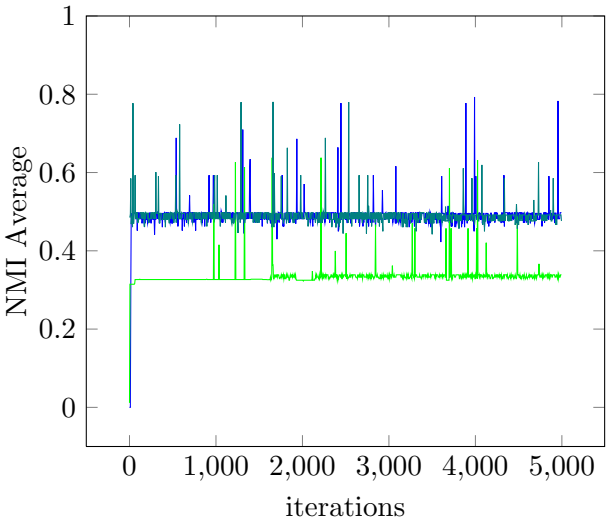


Figure 5.22: Trace of NMI Average

This aggregation is again evaluated against the ground truth, a NMI matrix is created and reduced. The plots in Figure 5.23 thus represent a cumulative mean of the whole model. Figure 5.23(a) shows the aggregation with fixed thresholds. The NMI averages of the cumulative aggregations fluctuate less than those of the traces and are even higher. While the trace rarely reaches values above 0.6, the aggregated means are consistently above 0.6. Results of the aggregation with adaptive thresholds are visible in Figure 5.23(b). While the maximum NMI reached is higher, the values fluctuate even more than those of the traces and are on average lower. This indicates that while the automatic threshold calibration still needs work, the aggregation with fixed values works well on synthetic data.

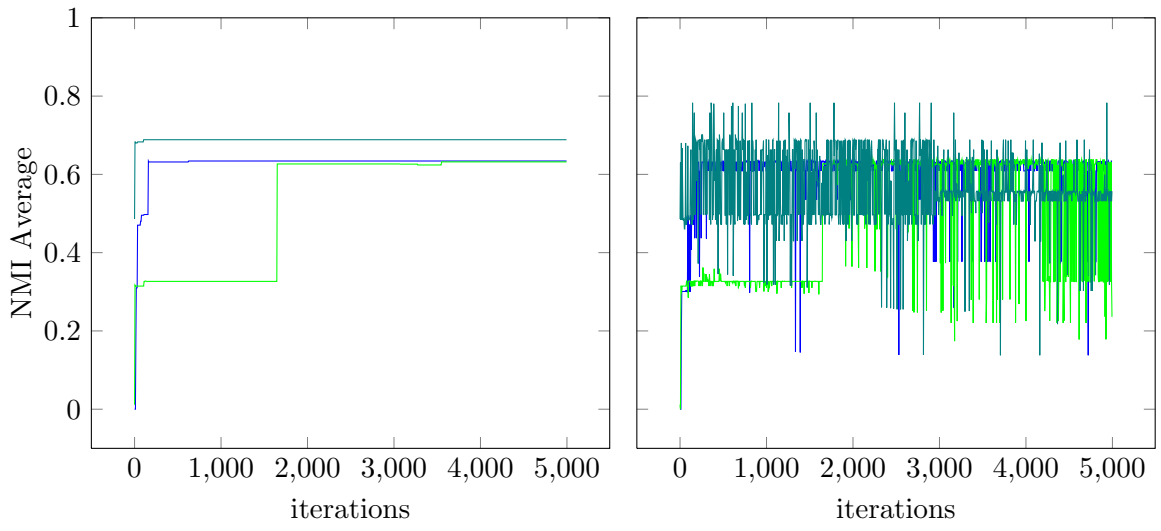
5.5 Applications and fMRI Data

After evaluating the results of synthetic data in Section 5.3, let us turn to some applications. We focus here on the fMRI dataset of 83 human subjects, 41 of which were diagnosed with some form of schizophrenia. It was kindly provided by Desorno [6]. The data was acquired using a working memory task, where deficits in schizophrenia are well-known [6]. Dynamic causal modeling [9] was used to interpret the imaging data, resulting in 12 values for each subject. From this dataset, we built the required quadratic input matrix by using the negative Euclidean distance[4] or the dot product. We observed better results with the Euclidean distance, which is why we use it for most of the following tests.

5.5.1 Convergence

We started 8 Markov chains with different random starting values for α, λ, b, Z, C, and W. The cumulated means of α are shown in Figure 5.24. After about 10^5 iterations, the values are distributed between 0.5 and 0.7, but mixing is much slower than with synthetic data (Figure 5.2). After $8 \cdot 10^5$ iterations, the cumulative means of α still differ by about 0.2. More iterations might be necessary, which makes waiting for full convergence cumbersome. This implies that either the model is not a suitable fit for this dataset or other parameters need to be used. Figure 5.25 shows the cumulative means of λ, which have similar convergence issues. A negative correlation between α and λ is visible: With

[4] A power kernel with the exponent set to 1.



(a) Aggregation with fixed thresholds

(b) Aggregation with adaptive thresholds

Figure 5.23: NMI average is noisier with adaptive thresholds.

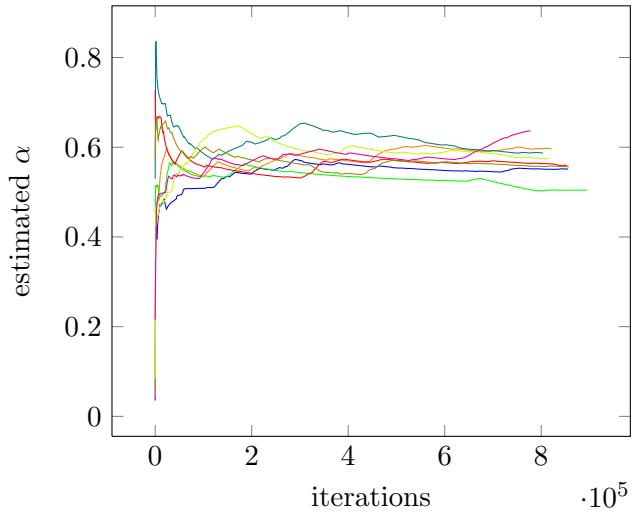


Figure 5.24: Cumulative mean of α for chains from fMRI data

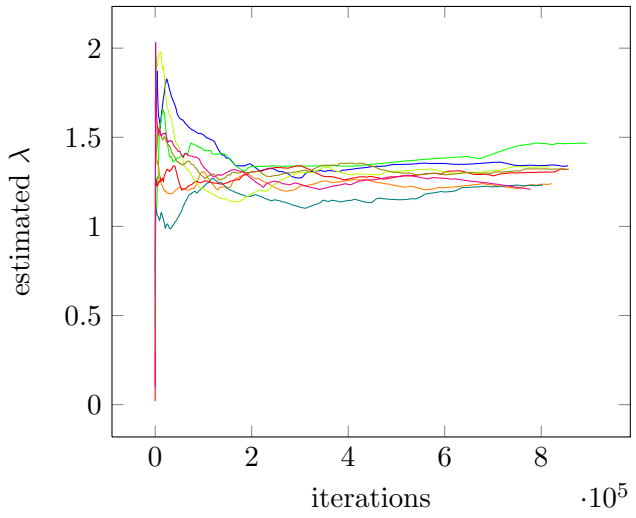


Figure 5.25: Cumulative mean of λ for chains from fMRI data

more features and more clusters, it is easier to have a good fit of the data. The value of α influences the number of features, while λ controls the number of clusters within features. A good fit of the data can be achieved by having a high number of features or a high number of clusters within few features. If one of them is high, the data is likely fitted closely already and the likelihood gain by increasing the other is small. Other parameter estimates mix much faster: Figure 5.26 shows the bias of each chain, distributed around -5 for most of the chains and Figure 5.27 shows the cumulative mean of the total link count.

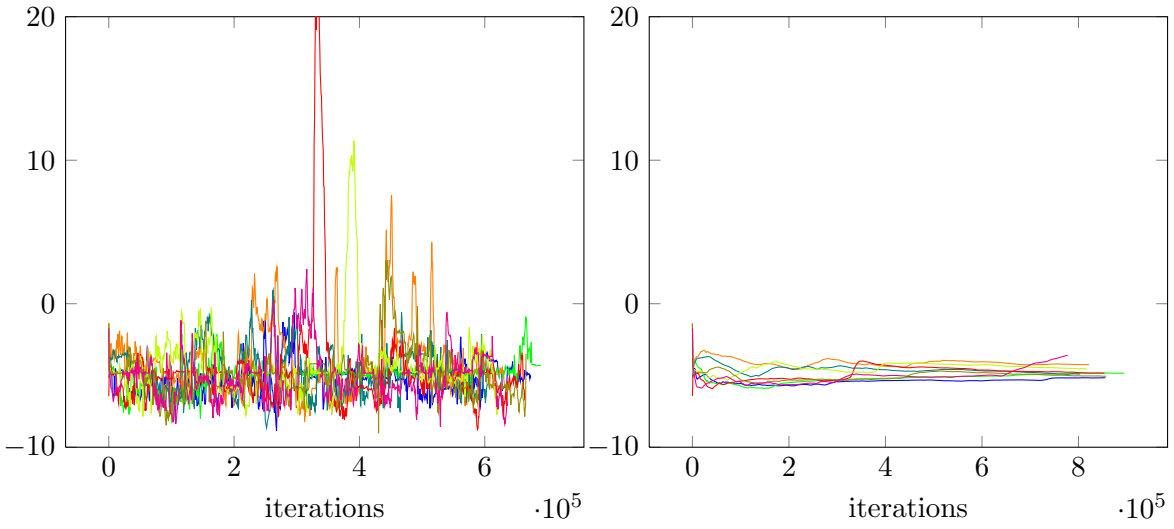
5.5.2 Correlations between Model Elements

Figure 5.28 shows all correlations between parameters. A small negative correlation between α and λ is visible, as well as a small positive correlation between α and the bias. The first correlation was visible in the convergence plots and is to be expected, whereas the second one is somewhat surprising, as it is exactly the opposite correlation as seen with synthetic data. Maybe both correlation figures still reflect random effects and more samples are needed.

5.5.3 Comparison with Ground Truth

The ground truth in the schizophrenia dataset consisted of several aspects: group (patient / control), performance, gender, handedness, age, 2-back-task, 0-back task, age of onset, duration in years, number of episodes, medication type, equivalent dose in Chlorpromazine, PANSS (positive symptoms), PANSS (negative symptoms) and PANSS (total score). Some of them were categorical, such as group, performance, gender, handedness and medication type. The other categories were continuous. PANSS, the *Positive and negative symptom score* is a common measure to evaluate the severity of a schizophrenic illness. We used the NMI to compare our results to the categorical aspects and analysis of variance (ANOVA[5]) for continuous data. In one-way ANOVA, a partition is evaluated with a continuous ground truth by computing the mean of each cluster and comparing the variance of means with the total variance of samples.

NMI. We aggregated all the chains shown in Section 5.5.1 to a single result and compared it to the categorical ground truth features. Figure 5.29 shows the NMI matrix



(a) Trace of bias

(b) Cumulative mean of bias

Figure 5.26: Trace and cumulative mean of bias for 8 sampling chains, started with the fMRI data and using the Euclidean distance. The bias is mostly between -2.5 and -7.5, but shows occasional spikes.

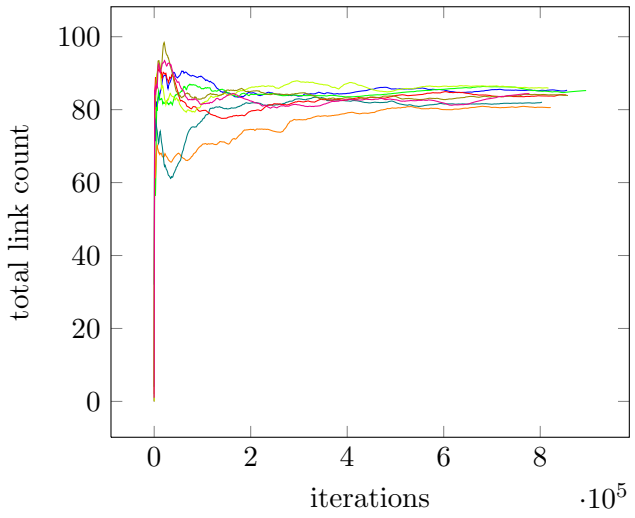


Figure 5.27: Cumulative mean of total link count

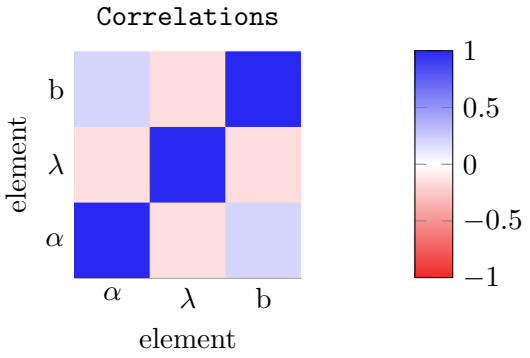


Figure 5.28: Correlations between samples

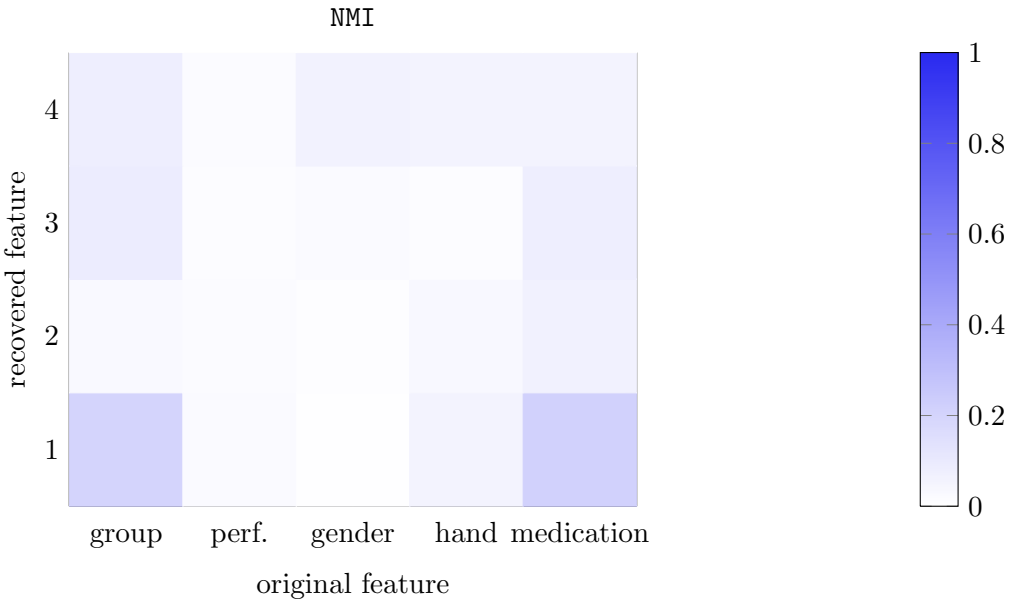


Figure 5.29: NMI matrix of aggregated samples inverted with an Euclidean distance measure

of this aggregation. The balanced purity and the p-values of the purity can be seen in Figures B.6 and B.7 in the appendix. An interesting combination consists of the first feature of the aggregated mean and the group, with an NMI value of 0.2. While the combination of the first feature and the medication has also a relatively high NMI value, the medication choice is highly correlated with the group attribute, since the control group is unmedicated. These results are not as clear as with synthetic data, compare Figure 5.15. They aren't attributable to pure noise either: The p-values for NMI(f^1(trace), group) and NMI(f^1(trace), medication) were below 0.001, as shown in Figure B.5

To see whether the convergence issues encountered influence the NMI between the final result and the ground truth, we plotted the NMI(f^1(trace), group) of each chain against the number of iterations in Figure 5.30. While the values fluctuate, there is no clear trend towards a higher NMI for a higher number of iterations.

Comparison with prior. To interpret the NMI values in Figure 5.29, we sampled 50 C matrices from the CRP prior and computed the NMI of these against the ground truth. Figure 5.31 shows the NMI matrix of the aggregated results while Figure 5.32 shows the distribution of average NMI values of the individual samples. As expected, these values are much smaller than our results.

Anova. The analysis of variance is shown in Figure 5.33. While the second feature seems to have good results, the high Anova values are a result of too many clusters: The second feature has 59 clusters for 83 objects, clearly too many for a meaningful clustering. During inversion, the number of clusters per feature was much lower, as seen in Figure A.4. This indicates the aggregation phase needs to be improved.

5.5.4 Other Similarity Measures

In addition to the Euclidean distance, we used the dot product to create another version of the input matrix R. This resulted in somewhat poorer mixing, as shown in Figure 5.34. If the kernel in use returns a strict subset of $\mathbb{R}$, the prior distributions might be changed

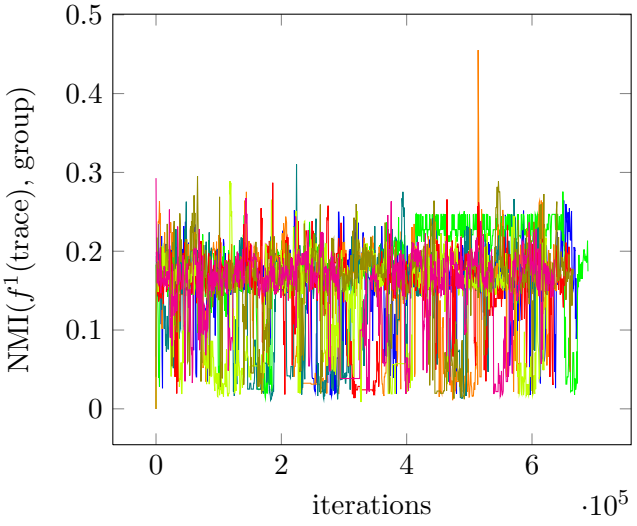


Figure 5.30: Trace of NMI of first feature against patients/control group clustering

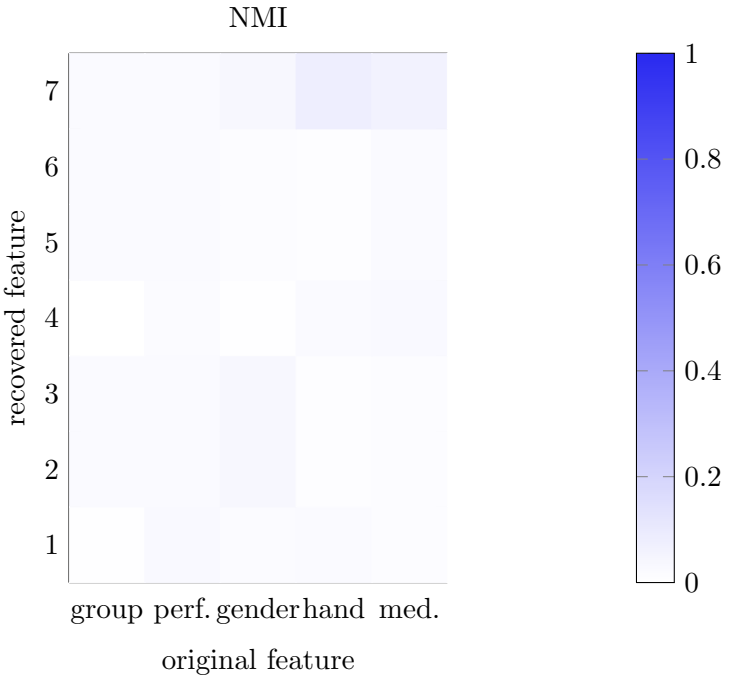


Figure 5.31: Aggregated results sampled from the prior, evaluated against the fMRI ground truth. The NMI values (≤ 0.08) are much smaller than the inversion results in Figure 5.29 (≈ 0.2). This implies the inversion results are significantly above chance level.

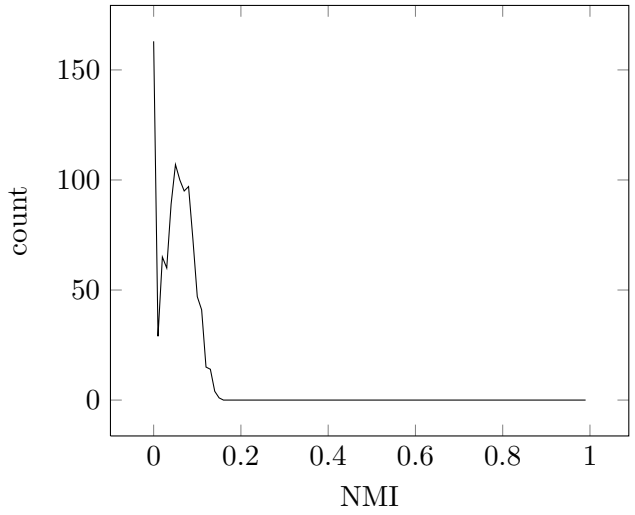


Figure 5.32: Histogramm: Averaged NMI of prior. 1000 model instances were sampled from the prior distribution and the NMI average against the schizophrenia ground truth was computed for each sample.

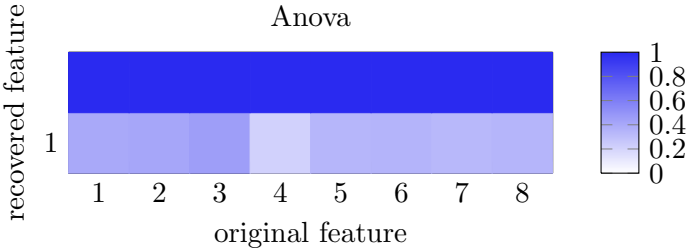


Figure 5.33: Analysis of variance (Anova) of aggregated samples. The input data were the fMRI measurement transformed with the Euclidean distance measure. 1 = age, 2 = age of onset, 3 = duration, 4 = number of episodes, 5 = equivalent dose in Chlorpromazine, 6 = PANSS positive symptom score, 7 = PANSS negative symptom score, 8 = PANSS total score

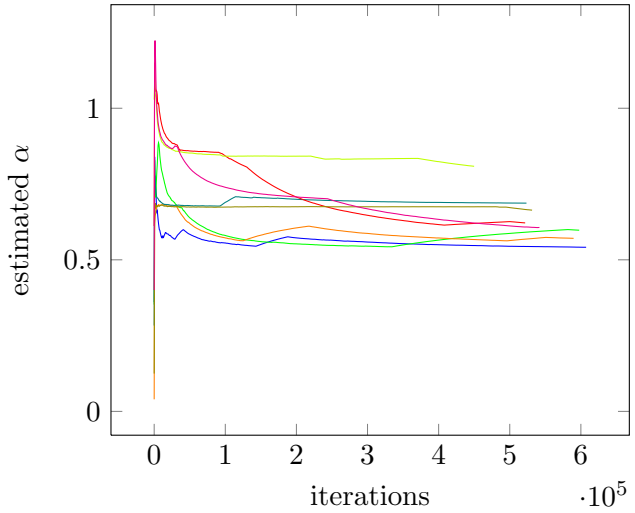


Figure 5.34: Cumulative mean of α for chains created with the dot product of fMRI data

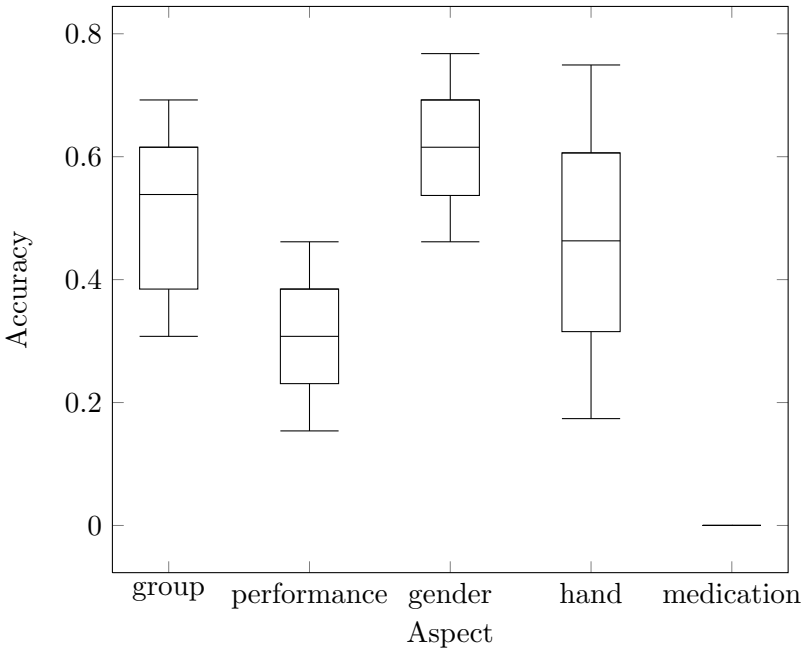


Figure 5.35: Clustering accuracy of remaining objects in supvervised learning

to reflect this. The negative Euclidean distance results in values ≤ 0 so we could have used the truncated Gaussian distribution instead of the normal Gaussian distribution. We kept the normal Gaussian distribution because of the conjugacy, which greatly improved sampling speed.

5.5.5 Supervised Learning

We described a variant for supervised learning in Section 4.4. When only a partial ground truth is available, it can be used to assign subjects to clusters manually and predict missing values.

In each test run, we first used the ground truth to assign 70 objects to clusters manually. One feature was created for each categorical aspect of the ground truth. For each category, a cluster was created. Afterwards, we sampled the weight matrices for 100 iterations and finally added the remaining objects. Figure 5.35 shows the results of 1000 test runs. Unfortunately, the results were not significant. Group and performance are at chance level, only the gender is slightly more accurate. Curiously, the medication was guessed incorrectly in every single case. The validation results on the schizophrenia dataset are of lower quality than with synthetic data. This suggests that either the latent structure in this dataset does not fit well to our generative model or the physiological or measurement noise is higher than assumed.

5.6 Visualization

To visualize the clustering results, we used the GraphViz library of graph drawing programs.[7] One graph drawing is created for each feature. The node colors indicate the cluster assignments and the distances between nodes i and j are roughly proportional to $1 + max - R(i, j)$, where max is the maximum of R. The latter is achieved by using a force-directed model[10]. Figure 5.36 shows the first feature of an inversion of the fMRI dataset. The 83 subjects are represented by 83 colored shapes. Subjects diagnosed with

schizophrenia are marked with anellipseand members of the control group are marked with asquare. We see some clusters shaped like shell segments, for example the red, green and teal cluster. This is characteristic of the generative model: Objects are clustered together not only if they are close to each other, but also if their distance to other clusters is similar.

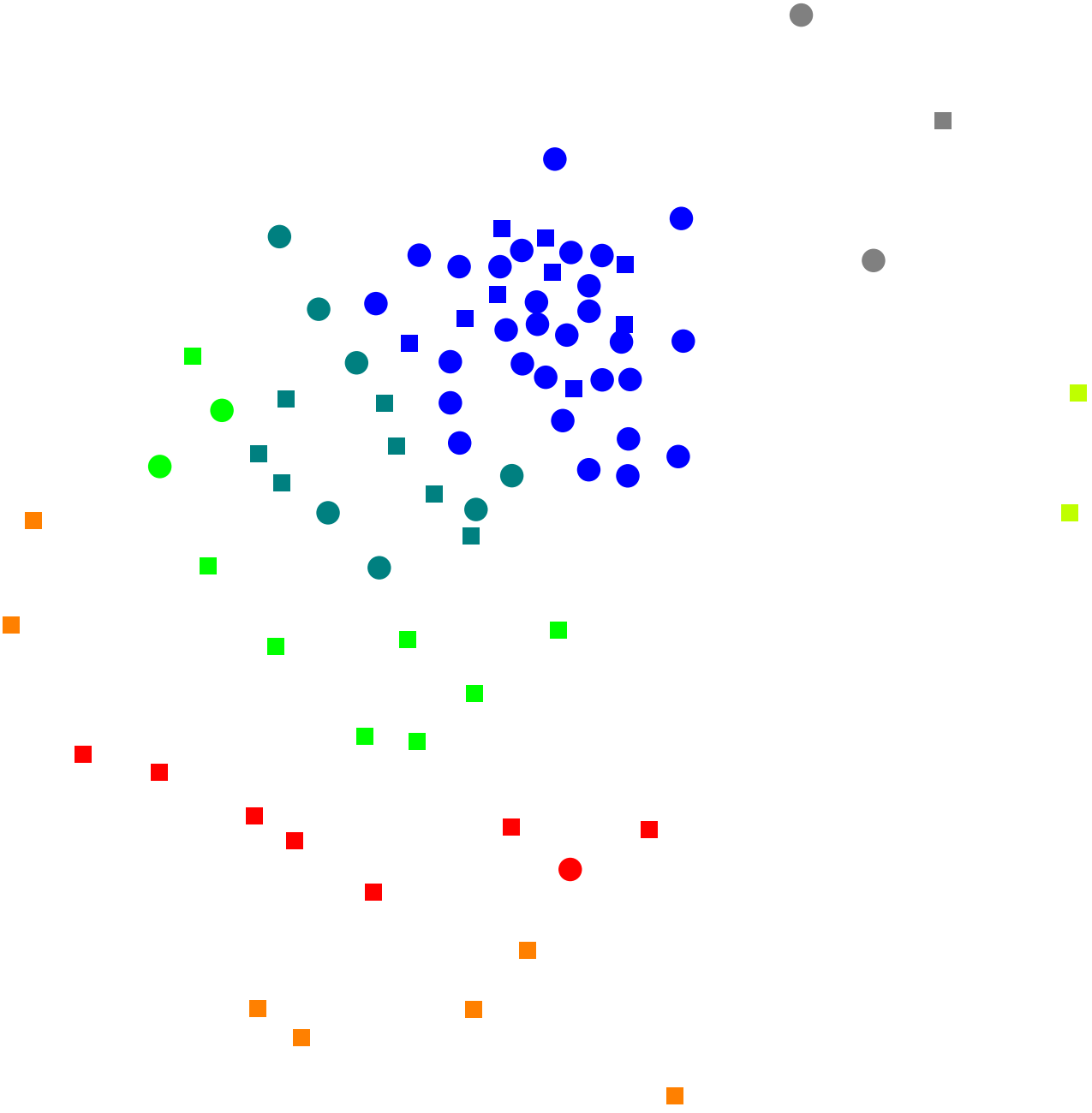


Figure 5.36: Visualization of first feature with GraphViz. The colors indicate the clustering returned by the inversion, the shapes correspond to the diagnosis: Patients are marked with anellipse, members of the control group are marked with asquare. The position is created with a force-directed layout, creating a node placement so the distances in $\mathbb{R}^2$ roughly match the original distances in $\mathbb{R}^{12}$. Cluster colors are blue, teal, green, orange, red, gray and lime.

6. Conclusion

Summary. This work is about a hierarchical generative model for Bayesian clustering on similarity data. The model has multiple latent features, with each feature partitioning a subset of the objects into clusters. Psychiatric diseases inspired this approach, especially spectrum diseases. In schizophrenia, for example, multiple underlying dysfunctions can be present in the same subject. Symptoms seem to not map to these dysfunctions clearly, which makes diagnosis and medication choice difficult. For the eventual goal of redefining psychiatric diseases due to the hidden causes, an exploratory analysis on physiological data is necessary.

In Chapter 3, we described the generative model. It consists of a binary feature matrix Z, an integer cluster matrix C and a set of real valued weight matrices W^m. The number of features and clusters are governed by parameters α and λ. A model from Palla [27] to model binary matrices serves as the foundation. We extended their model to real valued matrices, enabling the use of kernel methods and making several priors conjugate.

In Chapter 4, we presented the model inversion algorithm, a Markov chain Monte Carlo (MCMC) method. This inversion results in a distribution of samples. Many of these samples are very similar except for reordering of features and clusters. To simplify interpretation, we grouped similar samples into equivalence classes. We also described an aggregation algorithm based on consensus clustering to obtain a representative point estimate. This aggregation uses thresholds for feature selection and cluster assignment of the point estimate. Apart from fixed thresholds, we discussed a way to automatically select threshold values. In addition to the unsupervised inversion algorithm, we discussed a variant for supervised learning. This variant can be used when a partial ground truth is available and inference on the remaining unknown objects is desired. The asymptotical complexity of a complete iteration is $O\big(n^2 \cdot n_f(s) \cdot (\sum_{f=0}^{n_f(s)} (n_{cf}(s,f) + m_{\text{aux}}))\big)$, where n is the number of objects, $n_f(s)$ is the number of features and $n_{cf}(s,f)$ the number of clusters in feature f. As discussed in Section 4.5, the number of features and clusters are governed by α and γ. The complexity of a complete iteration for a high value of λ is $O(n^3 \cdot \alpha^2 \log^2 n)$. A low value of λ results in an expected complexity of $O(n^2 \cdot \alpha^2 \log^3 n)$. Since λ is estimated too low in most inversions, the latter complexity term is a better description of practical runtime.

Optimizations. Inverting a complex model with a MCMC method may require a high number of iterations until the sampling process has converged sufficiently. To avoid an

unfeasibly long runtime, it is vital to reduce the time required for a single iteration. We implemented the inversion algorithm in C++ and included three optimizations which resulted in a considerable speedup. In almost all steps of the sampling process, an expensive likelihood calculation is required. When only a part of the model changes, the calculation can be hastened by caching intermediate steps and only recomputing the changed part. Since we extended the generative model from binary matrices to real valued matrices, we changed the final Bernoulli distribution to a Gaussian distribution, making several priors conjugate. This improves sampling speed and precision. Finally, we compressed each row of the binary feature matrix Z into a single `int` using bit shifting. Finding out which features are shared between two objects is used very often in the likelihood calculation. This operation is now possible with a single bitwise and-operation, saving memory accesses and loop iterations. With this set of optimizations, we achieved a speedup of 30 over a naive C++ implementation and 92 to 700 over the Matlab implementation of the base model.

To judge convergence in MCMC methods, it is often useful to start multiple parallel sampling chains. The benefit of additional intra-chain parallelization would have been small. We outlined possible parallelization targets, but did not implement them.

Evaluation. We evaluated runtime, convergence and inversion quality in Chapter 5. The runtime largely depends on the number of features and clusters and is about quadratic in the number of nodes. 1000 iterations of 50 nodes took on average 22 seconds. After about $2{\cdot}10^4$ iterations, the sampling chains with synthetic data of 50 objects had very similar distributions. After $2{\cdot}10^5$ iterations, the cumulative means were indistinguishable.

The inversion of parameters consisting of single values were straightforward to validate. α and σ were approximately recovered, the bias b was recovered quite well and λ was mostly estimated too low. To evaluate the inversion quality of Z and C, we discussed the commonly used clustering quality measures Balanced Purity (BP) and Normalized Mutual Information (NMI). Since we had several features in the inversion result and ground truth, we evaluated each feature of the result against each feature of the ground truth. We thus obtained a matrix of BP and NMI values, which we reduced to a real value for easier comparison. With this reduction, it was possible to illustrate the influence of noise and different priors on the inversion quality.

fMRI Data. We finally tested the inference on a dataset of fMRI values consisting of schizophrenia patients and a control group. Convergence was much slower than with synthetic data. After nearly 10^6 iterations, the sampling chains were still noticeably different. However, the inference produced several features which correspond roughly with the diagnosis (NMI 0.20) and medication (NMI 0.21) of the ground truth.

6.1 Discussion

During the development and implementation, some ideas and issues appeared. They are discussed briefly and informally below.

6.1.1 Feature Merging and Equivalence Classes

We defined equivalence classes in Section 4.3.2. When the partition in one feature m was a refinement of the partition in another feature l, we discard feature l and add the weights to feature m. In other words, should a set of features model a hierarchical clustering, we only keep the finest level. This is done after the sampling and does not bias the sampler, but might lose information about the hierarchy. However, feature merging occurred rarely in practice. In over 95% of samples, no features were merged.

6.1.2 Theoretical Motivation of Aggregation

The aggregation algorithm described in Section 4.3.3 was introduced to get a point estimate of the posterior distribution. It consists of three steps: First a common form is created from each model instance, the common forms are then averaged and finally the average is collapsed back to a model instance. The aggregation was designed to resemble an arithmetic mean. We briefly discuss which properties it fulfills and outline proof ideas. Since the aggregation was defined on equivalence classes and not on samples, we allow reorderings among features and clusters.

Idempotence. When given a sample A, the aggregated result of A with itself should be equivalent to A. We would need to show that the result of creating the common form of A, averaging it with itself and collapsing the average back to a model instance is equivalent to A. In other words: $\text{Collapse}(\text{Av}(\text{Construct}(A), \text{Construct}(A))) \approx A$ needs to hold. Since Algorithm 10 simply computes an arithmetic average, the result of two identical common forms is the form itself. We thus need to show that collapsing the common form of A results in a model instance equivalent to A. This also seems to hold: The feature matrix Z remains unchanged from the original and the cluster similarity matrices E are binary, thus collapse back to the original for all thresholds between 0 and 1. The resulting clusters may have a different order than the original sample. However, this is covered by the equivalence classes.

Associativity. An operation is called associative if the order of operations does not influence the result. We need to obtain the same result irrelevant whether we first aggregate samples A and B, then aggregate the result with C or first aggregate B and C and finally aggregate this result with A. This can be interpreted in two ways. The stronger version includes the collapse step:

$$\begin{aligned}\text{Av}\Big(\text{Collapse}\big(\text{Av}(\text{Construct}(A), \text{Construct}(B))\big), \text{Construct}(C)\Big) = \\ \text{Av}\Big(\text{Av}\big(\text{Construct}(A), \text{Construct}(\text{Collapse}(\text{Construct}(B), \text{Construct}(C)))\big)\Big)\end{aligned} \quad (6.1)$$

This version seems not to hold. The weaker version only demands associativity before collapsing.

$$\begin{aligned}\text{Av}(\text{Av}(\text{Construct}(A), \text{Construct}(B)), \text{Construct}(C)) = \\ \text{Av}(\text{Av}(\text{Construct}(A), (\text{Construct}(B), \text{Construct}(C)))\end{aligned} \quad (6.2)$$

Even this does not hold with the simple version presented in Algorithm 10.When using the generalized version with weighting factors, Equation 6.2 holds if the correct weighting factors (1/3 for each) are used.

Non-negative Influence. The aggregation of A, A and B should be as least as close to A as the aggregation of A and B. To determine this, we need a distance measure. We chose the Rand index [31], since it mirrors the consensus clustering used in the aggregation. Because the values in cluster similarity matrix E are indeed at least as close to A when averaging A, A and B as when averaging A and B, it should be possible to prove that the collapsed versions have the desired property with respect to the Rand index.

6.1.3 Issues in Estimation

The noise variance σ controlled the influence of prior and likelihood on the posterior distribution. When σ was very small, the sampling process approximated a maximum-likelihood estimation. A higher σ resulted in a stronger influence of the prior. When the

σ used in the inversion was lower than the true noise in the input data, runaway feature creation could happen. New features were selected in the Metropolis-Hastings step, even when they did not improve the likelihood of the solution. The resulting number of features was even larger than the number ob objects and the sampling chains did not converge. For this reason, we mostly fixed σ to 1. When sampling σ with the rest of the model, it was often estimated too low as features and clusters were created to fit the noise.

When inverting the model with synthetic input data without any noise, even a very low σ did not result in a large number of features. Instead, the sampling process recovered the original solution after about 10^3 iterations. From this point, the σ was estimated very low and almost no changes in feature selection and cluster assignment were accepted.

6.1.4 Metropolis-Hastings Step for Feature Removal

The present sampling algorithm has the same overall structure as the one used by Palla [27]. New features are added in a Metropolis-Hastings step in Algorithm 2. Existing features are removed when no objects participates in them. However, due to the IBP prior, an existing feature with many objects is very unlikely to be removed again. In our test runs, we never observed an established feature with a large number of objects to eventually become empty and removed. This might contribute to the runaway feature creation discussed in the previous section. One could add a Metropolis-Hastings step to remove even non-empty features to balance this effect.

6.1.5 Removal of Bias

The bias term b might be rendered obsolete by the extension to real valued matrices. Removing it would simplify the model. We fixed the bias to 0 in several short test runs, it did not seem to have a negative effect on the inversion quality.

6.1.6 Overlapping Feature Views

Niu [26] also present a non-parametric Bayesian clustering method using the Indian buffet process and the Chinese restaurant process. They focus on giving multiple independent clustering views, each of them based on some aspects of a multi-faceted input dataset. In contrast to our approach, they focus on selecting the aspects of input data relevant for each clustering view.

6.2 Further Work

Several avenues for future work exist. They are roughly grouped into optimizing the existing approach, trying it on more test data and further extensions or changes in the model.

- Slow convergence was an issue with the schizophrenia dataset. Approaches to achieve faster convergence collapsed or blocked Gibbs sampling.
- Because of the autocorrelation found in MCMC methods, the Z and C matrices of a sample are often similar or even equal to the matrices of the immediate predecessor. One could greatly improve memory consumption by storing only the current sample in full and computing the delta to previous versions.
- The Euclidean distance and the dot product was used to generate similarity matrices from input data. Many other measures are possible [11]. For example, dynamic causal modeling gives a posterior distribution on parameters and one could use a Fisher kernel with different distributions.

- Prior distributions for features were generated using the Indian buffet process, which was built to generate sparse feature matrices. The ground truth in our schizophrenia dataset is not sparse. A two-parameter version of the Indian buffet process exists, providing finer control over the density of feature matrices.

7. Appendix

A Histograms

Figures A.1 to A.4 are from the inversion with synthetic data discussed in Section 5.2.

B BP, NMI and Correlations

C Recreating Figures

All figures in this work were created with the implementation available at `https://bitbucket.org/mektah/ila-hierarchical`. Due to changes during the writing phase, not all tests were run with the same version. Table C.1 shows how to recreate each plot.

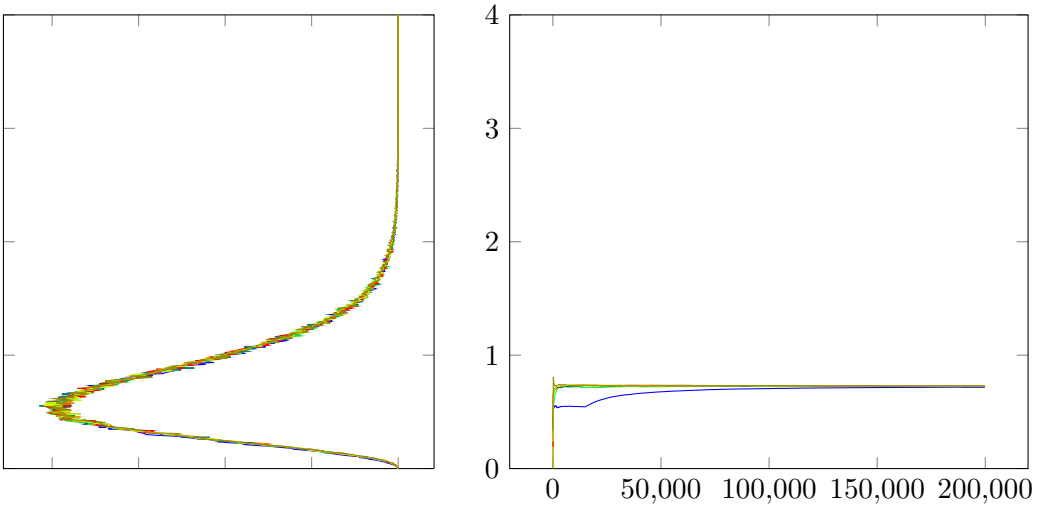


Figure A.1: Histogram and cumulative means of α. This inversion was done with 7 chains, a synthetic ground truth and 200000 iterations. The cumulative means were indistinguishable after 10^5 iterations and the final histograms are nearly equal, with a mean of ≈ 0.7. All but one chain are very similar much earlier, at about 10^4 iterations.

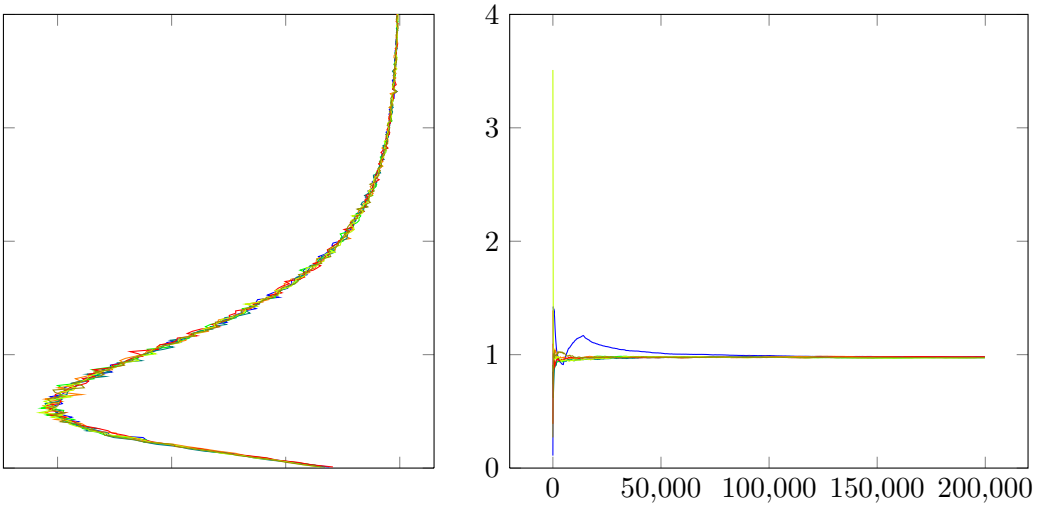


Figure A.2: Histogram and cumulative means of γ of the same inversion as in the previous plot. The cumulative means converge after about 10^5 iterations. All but one chain are very similar much earlier, at about 10^4 iterations.

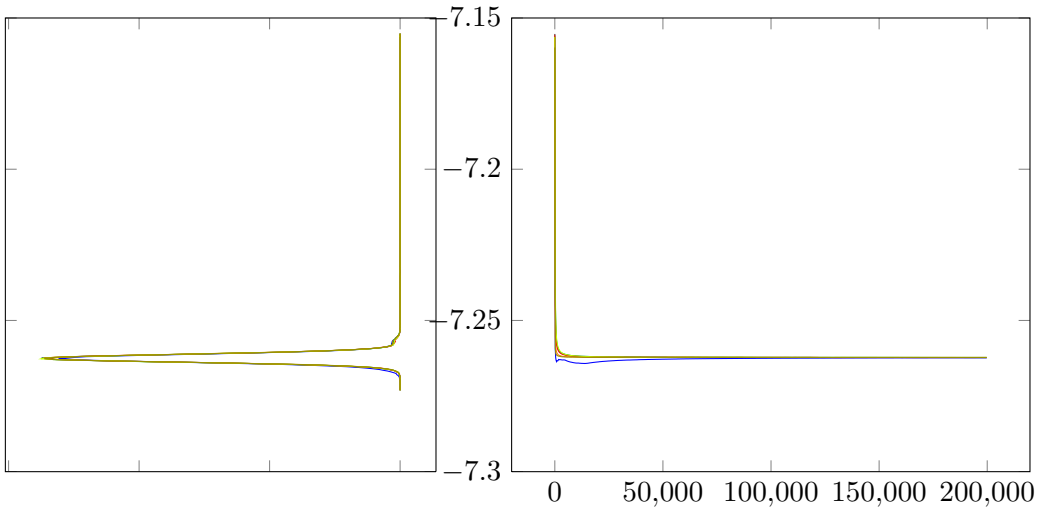


Figure A.3: Histogram and cumulative means of bias b. All cumulative means are between -7.25 and -7.27 after only 10^3 iterations. The histogram shows a spike at about -7.26.

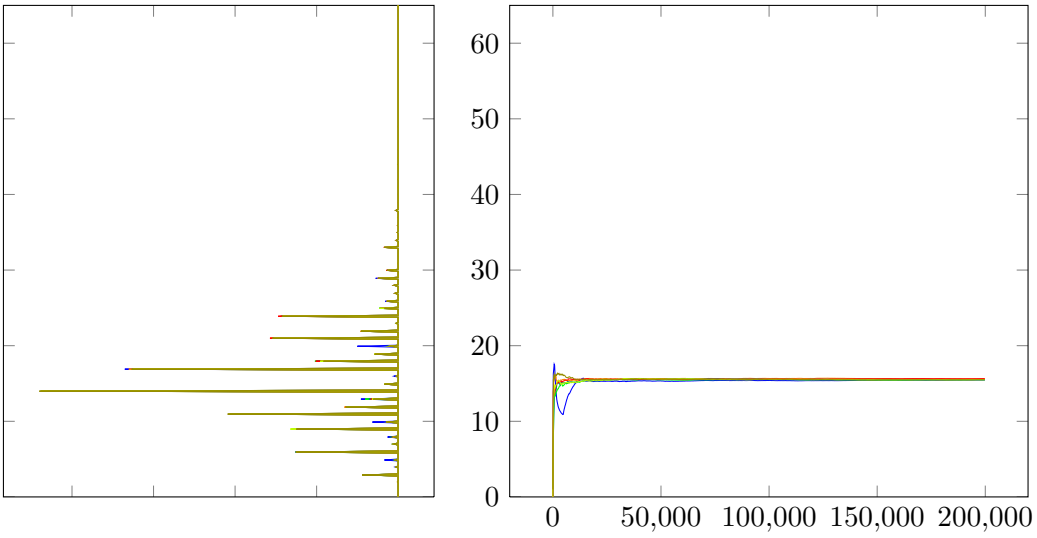


Figure A.4: Histogram and cumulative mean of the total link count. The cumulative means are at 15 after about $2 \cdot 10^4$ iterations. This does not appear to be a prominent value in the histogram, as the total link count is a sum of squares. Samples are distributed between 2 and 34, with no single value representing more than half of the samples.

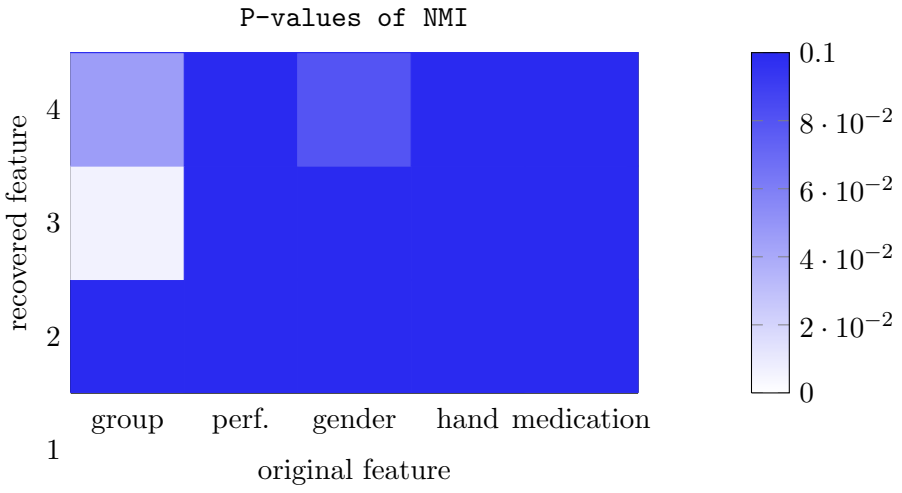


Figure B.5: P-Values of NMI.

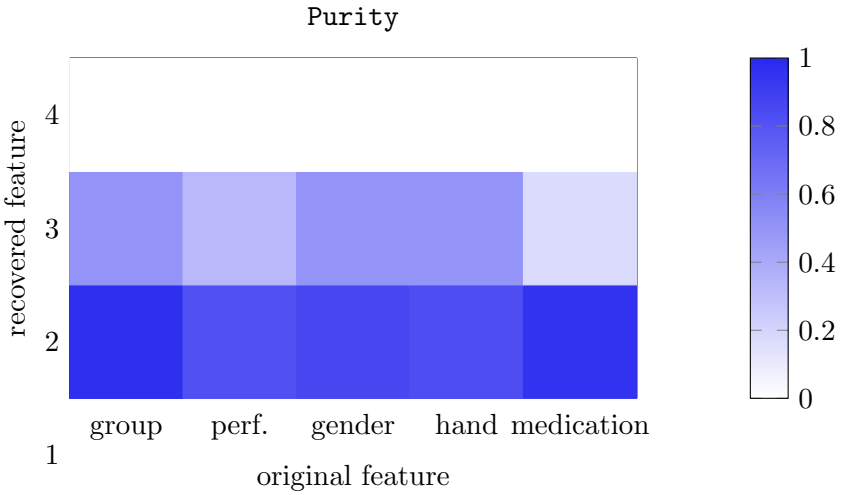


Figure B.6: Purity of aggregated chains from euclidean distance

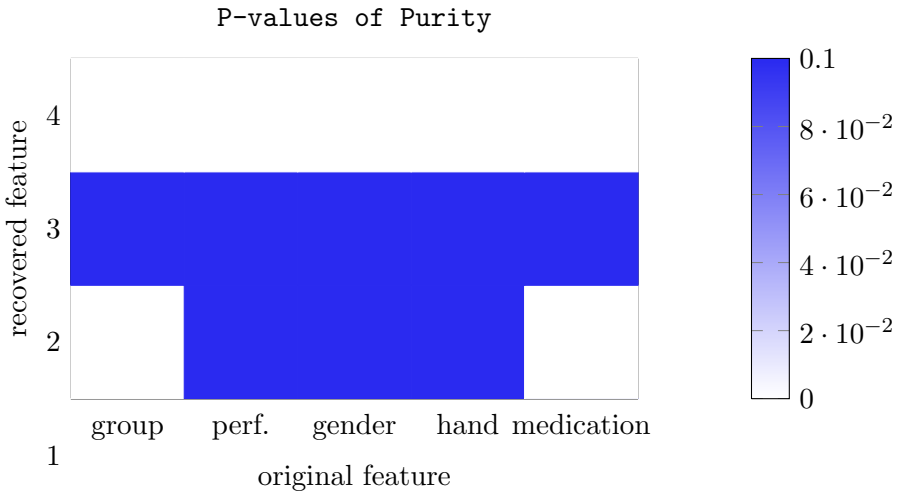


Figure B.7: P-values of Purity

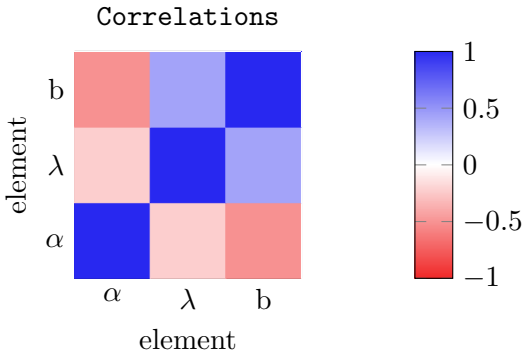


Figure B.8: Another correlation plot of an inversion from synthetic data.

Figure	Command
5.1	--timeplot --iterations 500
5.1	--optimap --randomgraph --repeat 50 --seed 0
5.2(a)	--randomgraph --endlesschains --iterations 200000 --seed 1375366548
5.3, 5.4(b)	--alphamap
5.5, 5.8	--partlyfixed
5.6, 5.7	--gammamap
5.9, 5.10	--biasmap
5.11, 5.12	--noisemap
5.17	--exacting
5.18	--hypermap
5.22	--randomgraph --iterations 5000 --seed 1379606950
5.23, 5.24 - 5.30	*1379606950* --euclid --endless --num_threads 8
5.31, 5.32	--priorBaseline <model>
5.34	--endless --num_threads 8
5.35	--seededrun --euclid

Table C.1: Command list

Bibliography

[1] David J. Aldous. "Exchangeability and related topics". In: *École d'Été de Probabilités de Saint-Flour XIII — 1983*. Ed. by P.L. Hennequin. Vol. 1117. Lecture Notes in Mathematics. Springer Berlin Heidelberg, 1985, pp. 1–198. ISBN: 978-3-540-15203-3. DOI: 10.1007/BFb0099421. URL: http://dx.doi.org/10.1007/BFb0099421.

[2] OpenMP Architecture Review Board. *OpenMP Application Program Interface Version 3.0*. 2008. URL: http://www.openmp.org/mp-documents/spec30.pdf.

[3] George EP Box and Mervin E Muller. "A note on the generation of random normal deviates". In: *The Annals of Mathematical Statistics* 29.2 (1958), pp. 610–611.

[4] Mary Kathryn Cowles and Bradley P Carlin. "Markov chain Monte Carlo convergence diagnostics: a comparative review". In: *Journal of the American Statistical Association* 91.434 (1996), pp. 883–904.

[5] Gerard Dallal. *The Little Handbook of Statistical Practice*. 2012.

[6] Lorenz Deserno et al. "Reduced Prefrontal-Parietal Effective Connectivity and Working Memory Deficits in Schizophrenia". In: *The Journal of Neuroscience* 32.1 (2012), pp. 12–20. DOI: 10.1523/JNEUROSCI.3405-11.2012. eprint: http://www.jneurosci.org/content/32/1/12.full.pdf+html. URL: http://www.jneurosci.org/content/32/1/12.abstract.

[7] J. Ellson et al. "Graphviz and Dynagraph – Static and Dynamic Graph Drawing Tools". In: *Graph Drawing Software*. Ed. by M. Junger and P. Mutzel. Springer-Verlag, 2004, pp. 127–148.

[8] Thomas S Ferguson. "A Bayesian analysis of some nonparametric problems". In: *The annals of statistics* (1973), pp. 209–230.

[9] Karl J Friston, Lee Harrison, and Will Penny. "Dynamic causal modelling". In: *Neuroimage* 19.4 (2003), pp. 1273–1302.

[10] Thomas MJ Fruchterman and Edward M Reingold. "Graph drawing by force-directed placement". In: *Software: Practice and experience* 21.11 (1991), pp. 1129–1164.

[11] Marc G Genton. "Classes of kernels for machine learning: a statistics perspective". In: *The Journal of Machine Learning Research* 2 (2002), pp. 299–312.

[12] Thomas L. Griffiths and Zoubin Ghahramani. *Infinite Latent Feature Models and the Indian Buffet Process*. Tech. rep. 2005.

[13] W. K. Hastings. "Monte Carlo sampling methods using Markov chains and their applications". In: *Biometrika* 57.1 (1970), pp. 97–109. DOI: 10.1093/biomet/57.1.97. eprint: http://biomet.oxfordjournals.org/content/57/1/97.full.pdf+html. URL: http://biomet.oxfordjournals.org/content/57/1/97.abstract.

[14] E. T. Jaynes. *Probability Theory*. Ed. by G. Larry Bretthorst. Cambridge University Press, 2003. ISBN: 9780521592710.

[15] Charles E Leiserson, Harald Prokop, and Keith H Randall. "Using de Bruijn sequences to index a 1 in a computer word". In: (1998). URL: http://supertech.csail.mit.edu/papers.html.

[16] David Lunn et al. *The BUGS Book: A Practical Introduction to Bayesian Analysis*. A Chapman & Hall book. CRC Press, 2013. ISBN: 9781584888499.

[17] David Lunn et al. "The BUGS project: Evolution, critique and future directions". In: *Statistics in medicine* 28.25 (2009), pp. 3049–3067.

[18] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003. URL: http://www.inference.phy.cam.ac.uk/itprnn/book.html.

[19] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Vol. 1. Cambridge University Press Cambridge, 2008.

[20] JR Miller. *RG (1981): Simultaneous Statistical Inference*.

[21] Stefano Monti et al. "Consensus clustering: a resampling-based method for class discovery and visualization of gene expression microarray data". In: *Machine learning* 52.1-2 (2003), pp. 91–118.

[22] Kevin P Murphy. *Conjugate Bayesian analysis of the Gaussian distribution*. Tech. rep. $2\sigma 2$. 2007, p. 16.

[23] Radford M Neal. "Markov chain sampling methods for Dirichlet process mixture models". In: *Journal of computational and graphical statistics* 9.2 (2000), pp. 249–265.

[24] Radford M Neal. *Probabilistic inference using Markov chain Monte Carlo methods*. Tech. rep. 1993.

[25] Radford M Neal. "Slice Sampling". In: *Annals of Statistics* 31 (2000), pp. 705–767.

[26] Donglin Niu, Jennifer G. Dy, and Zoubin Ghahramani. "A Nonparametric Bayesian Model for Multiple Clustering with Overlapping Feature Views". In: *International Conference on Artificial Intelligence and Statistics*. 2012, pp. 814–822.

[27] Konstantina Palla, David A. Knowles, and Zoubin Ghahramani. "An Infinite Latent Attribute Model for Network Data". In: *Proceedings of the 29th International Conference on Machine Learning*. ICML 2012. Edinburgh, Scotland, GB, 2012.

[28] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausble Inference*. Morgan Kaufmann Publishers Inc., 1988.

[29] Cross-Disorder Group of the Psychiatric Genomics Consortium. "Identification of risk loci with shared effects on five major psychiatric disorders: a genome-wide analysis". In: *The Lancet* 381 (2013), pp. 1371 –1379. DOI: doi:10.1016/S0140-6736(12)62129-1. URL: http://www.thelancet.com/journals/lancet/article/PIIS0140-6736%2812%2962129-1/fulltext.

[30] Howard Raiffa and Robert Schlaifer. "Applied Statistical Decision Theory (Harvard Business School Publications)". In: (1961).

[31] William M Rand. "Objective criteria for the evaluation of clustering methods". In: *Journal of the American Statistical association* 66.336 (1971), pp. 846–850.

[32] Klaas E Stephan, Karl J Friston, and Chris D Frith. "Dysconnection in schizophrenia: from abnormal synaptic plasticity to failures of self-monitoring". In: *Schizophrenia bulletin* 35.3 (2009), pp. 509–527.

[33] Eric A Suess, Bruce E Trumbo, and Bruce E Trumbo. *Introduction to Probability Simulation and Gibbs Sampling with R*. Springer New York, 2010. URL: `http://algebra.sci.csueastbay.edu/~esuess/psgs/120119PSGSAnsMast.pdf`.

[34] *The Poisson Distribution*. University of Massachusetts. URL: `http://www.umass.edu/wsp/statistics/lessons/poisson/index.html`.

[35] Frank Wood, Thomas Griffiths, and Zoubin Ghahramani. "A Non-Parametric Bayesian Method for Inferring Hidden Causes". In: *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence*. 2006, pp. 536–543.

[36] Eliezer Yudkowsky. *An Intuitive Explanation of Bayes' Theorem*. English. Machine Intelligence Research Institute. URL: `http://yudkowsky.net/rational/bayes`.