Parameterwahl beim ITERATIVE SIGNATURE ALGORITHMUS

Bachelorarbeit

im Studienfach Statistik

Julia Kammerer

Betreuung: Dipl. Stat. Sebastian Kaiser – Prof. Dr. Friedrich Leisch

Institut für Statistik – Ludwig-Maximilians-Universität München

10. August 2010

Inhaltsverzeichnis

1 Motivation und Einführung

1.1 Motivation

Ein fiktives Beispiel sollte den Kern meiner Arbeit exemplarisch aufzeigen: Ein Wissenschaftler möchte mit einer DNA-Microarray-Analyse ein menschliches Gewebe untersuchen. Daher führt er zahlreiche Experimente durch, deren Ergebnisse die Genexpressionslevel von Tausenden von Genen unter den verschiedenen experimentellen Bedingungen aufzeichnen. Nach Beendigung seiner Untersuchungen hat er eine Genexpressionsdatenmatrix von sehr großem Umfang erhalten und will diese nun genauer betrachten und auf eventuelle Besonderheiten prüfen. Er möchte die einzelnen Gene und experimentellen Bedingungen nach bestimmten Kriterien klassifizieren, doch aufgrund der enormen Datenmenge steht er vor einem Problem. Wie soll er bei der Analyse der Expressionsdatenmatrix, die Millionen von Werten enthält, vorgehen?

Eine mögliche Lösung für die genannte Problemstellung wäre die Anwendung von standardmäßigen Cluster-Algorithmen, um die Gene und Bedingungen zu klassifizieren. In der Arbeit von Madeira und Oliveira [2004] wird jedoch erklärt, dass normale Cluster Algorithmen nicht geeignet sind, um sie auf große zweidimensionale Datensätze anzuwenden, da diese gesondert entweder nur auf die Zeilen (Gene) oder auf die Spalten (Bedingungen) einer Datenmatrix angewendet werden können. Bergmann, Ihmels und Barkai [2003] beschreiben die Problematik der Anwendung von normalen Cluster-Algorithmen anhand von Genexpressionsdaten, genauer: Gewöhnliche Cluster-Methoden würden jedes Gen nur einem einzigen Cluster zuordnen, obwohl diese an mehreren Zellprozessen beteiligt sein könnten und demnach in mehr als einem Clustern enthalten sein sollten. Ein weiterer Nachteil ist, dass die Expression von Genen unter Berücksichtigung aller experimentellen Bedingungen analysiert wird, obwohl viele Zellprozesse nur von einer Teilmenge der Bedingungen beeinflusst werden und die unbeteiligten Bedingungen die Ergebnisse nur verzerren. Allerdings kann die Verwendung von standardmäßigen Cluster-Methoden zu interessanten Ergebnissen führen, wenn relativ kleine Datensätze mit etwa zehn experimentellen Bedingungen und höchstens mehreren hundert Genen vorliegen.

Um die genannten Probleme zu umgehen, wurde eine interessante Methodik entwickelt, die in beiden Dimensionen einer Matrix nach Clustern sucht. Diese Erfindung nennt sich Biclustering oder auch Two-Way-Clustering und stammt ursprünglich von Hartigan [1972]. Es handelt sich dabei um eine Technik, bei der Clustering gleichzeitig auf die Zeilen und Spalten von Matrizen angewendet wird. Man möchte damit die ursprüngliche Datenmatrix in Submatrizen zerlegen, also in Teilmengen von Zeilen und Spalten.

Ab dem Jahr 2000 kamen Bicluster-Methoden wieder häufiger zum Einsatz, speziell in der biologischen Datenanalyse. Cheng und Church [2000] führten diesen Begriff im Bereich der Microarray (Genexpressions-) Daten ein und von dem Zeitpunkt an wurde Biclustering als bevorzugte Methode in diesem Bereich verwendet. Ziel ist es dabei, Teilmengen von Genen und Teilmengen von Bedingungen durch simultanes Clustern der Zeilen und der Spalten zu finden. Man möchte also eine Gruppe von Genen identifizieren, die unter einer speziellen Teilmenge von experimentellen Bedingungen ein ähnliches Expressionsverhalten aufweisen.

1.2 Einführung

Auf der Suche nach Biclustern in großen Datensätzen werden spezielle Algorithmen eingesetzt. Diese beinhalten oftmals wichtige Parameter und je nach Einstellung werden unterschiedliche Bicluster-Ergebnisse geliefert. Der Iterative Signature Algorithmus (kurz ISA) ist ein solcher Bicluster-Algorithmus und wird in dieser Bachelorarbeit genauer betrachtet. Es wird im Folgenden untersucht, welche seiner vielen Parameter für das Finden möglichst guter Bicluster-Ergebnisse verantwortlich sind und welche eher weniger relevant sind. Dabei wird der ISA anhand von mehreren Durchläufen auf simulierte Daten mit versteckten Biclustern analysiert. Die erhaltenen Bicluster-Objekte werden anschließend mit den versteckten Biclustern verglichen. Ziel dieser Arbeit ist es möglichst viele verschiedene Parametersettings auszuprobieren um Regeln für eine möglichst optimale Parameterwahl beim ISA aufstellen zu können.

In dieser Arbeit ist die Analyse des Iterative Signature Algorithmus in 6 Kapitel unterteilt. In Kapitel 2 werden zunächst die Begriffe Clustering und Biclustering eingeführt, bevor in Kapitel 3 die Funktionsweise des ISA erläutert wird, was einen der Schwerpunkte dieser Arbeit

darstellt. Als Vorarbeit für die Simulationen des ISA in Kapitel 5 werden in Kapitel 4 das R-package `isa2` und die darin enthaltenen R-Befehle erläutert. Kapitel 5 behandelt einen weiteren wesentlichen Schwerpunkt und zwar wird der Algorithmus unter Verwendung der R-packages `biclust` [Kaiser et al., 2009] und `isa2` [Csárdi, 2009a] auf simulierte Datensätze angewendet. Ziel ist es, im Datensatz versteckte Bicluster durch Einsatz des ISA zu finden. Dabei sollen die Bicluster-Ergebnisse anhand unterschiedlicher Mengen von Parametersettings berechnet werden. In Kapitel 6 werden die Simulationsergebnisse aus dem vorhergehenden Kapitel anhand von zwei generalisierten linearen Regressionsmodells analysiert und Gesetzmäßigkeiten entwickelt, welche die bestmögliche Parameterwahl für den ISA liefern. Verglichen werden dabei ein lineares und ein quasibinomialverteiltes Regressionsmodell. Kapitel 7 enthält eine Zusammenfassung der wichtigsten Ergebnisse zur Parameterwahl und zusätzlich eine kurze Diskussion mit anschließendem Ausblick auf weitere Möglichkeiten den ISA zu analysieren.

2 Biclustering

Vor der Einführung des Begriffs Biclustering wird zunächst der Begriff Clustering geklärt. Definiert wird Clustering als eine Methode um Klassen zu bilden, also eine große Menge von Objekten in kleinere Mengen, sogenannte Teilmengen zu unterteilen. Dabei ist es wichtig, dass Objekte innerhalb einer Teilmenge so homogen wie möglich sind und sich die Teilmengen untereinander möglichst klar unterscheiden [vgl. Fahrmeir et al., 1996].

2.1 Zweidimensionaler Datensatz

Der Ausgangspunkt für die Anwendung von Biclustering ist ein zweidimensionaler Datensatz in Form einer $n \times m$ Matrix E, die bis auf mögliche NAs ausschließlich numerische Werte enthalten darf.

	y_1	$\cdots$	y_1	$\cdots$	y_i
x_1	e_{11}	$\cdots$	e_{i1}	$\cdots$	e_{m1}
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
x_j	e_{1j}	$\cdots$	e_{ij}	$\cdots$	e_{mj}
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
x_n	e_{1n}	$\cdots$	e_{in}	$\cdots$	e_{mn}

Abbildung 1: Zweidimensionaler Datensatz in Form einer Matrix.

Die Matrix $E = (X, Y)$ wird definiert durch eine Menge von Zeilen $X = \{x_1, \dots, x_n\}$ und eine Menge von Spalten $Y = \{y_1, \dots, y_m\}$ und enthält Elemente e_{ij}. Aus dieser Matrix möchte man nun eine Teilmenge E_{IJ} identifizieren, welche sich aus einer Menge von Zeilen $I = \{i_1, \dots, i_k\}$ mit $k \leq n$ und $I \subset X$ und einer Menge von Spalten $J = \{j_1, \dots, j_s\}$ mit $s \leq m$ und $J \subset Y$ zusammensetzt. Betrachtet man die Datenmatrix E, ist ein Zeilencluster $E_{IY} = (I, Y)$ eine Teilmenge von Zeilen, die durch alle Spalten Y definiert wird. Gemäß obiger Definition von Teilmengen kann dieses Cluster als eine $k \times m$ Submatrix bezeichnet werden. Analog dazu ist ein Spaltencluster $E_{XJ} = (X, J)$ eine Teilmenge von Spalten, die durch alle Zeilen X bestimmt wird und somit als $n \times s$ Submatrix bezeichnet werden kann [vgl. Madeira und Oliveira, 2004].

Im Gegensatz zum Cluster-Prinzip weist eine Teilmenge von Zeilen in einem Bicluster $E_{IJ} = (I, J)$ nur unter einer Teilmenge von Spalten ein ähnliches Verhalten auf. Das Gleiche gilt umgekehrt für eine Teilmenge von Spalten. Ein solches Bicluster wird als eine $k \times s$ Submatrix bezeichnet [vgl. Madeira und Oliveira, 2004].

2.2 Bicluster-Typen

Das Ziel aller Bicluster-Algorithmen ist das Finden von Teilmengen in einer Datenmatrix. Dabei sollen die Elemente in den gefundenen Biclustern eine gewisse Art der Gleichartigkeit erfüllen. Beispiele für die verschiedenen Typen von Biclustern sind in Abbildung 2 dargestellt.

1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0

(a) Konstante Bicluster

1.0	1.0	1.0	1.0
2.0	2.0	2.0	2.0
3.0	3.0	3.0	3.0
4.0	4.0	4.0	4.0

(b) Konstante Zeilen

1.0	2.0	3.0	4.0
1.0	2.0	3.0	4.0
1.0	2.0	3.0	4.0
1.0	2.0	3.0	4.0

(c) Konstante Spalten

1.0	2.0	5.0	0.0
2.0	3.0	6.0	1.0
4.0	5.0	8.0	3.0
5.0	6.0	9.0	4.0

(d) Zueinander in Beziehung stehende Werte - Additives Modell

1.0	2.0	0.5	1.5
2.0	4.0	1.0	3.0
4.0	8.0	2.0	6.0
3.0	6.0	1.5	4.5

(e) Zueinander in Beziehung stehende Werte - Multiplikatives Modell

S1	S1	S1	S1
S1	S1	S1	S1
S1	S1	S1	S1
S1	S1	S1	S1

(f) Ähnliche Wertentwicklung

S1	S1	S1	S1
S2	S2	S2	S2
S3	S3	S3	S3
S4	S4	S4	S4

(g) Ähnliche Wertentwicklung in den Zeilen

S1	S2	S3	S4
S1	S2	S3	S4
S1	S2	S3	S4
S1	S2	S3	S4

(h) Ähnliche Wertentwicklung in den Spalten

70	13	19	10
29	40	49	35
40	20	27	15
90	15	20	12

(i) Ähnliche Wertentwicklung in den Spalten

Abbildung 2: Beispiele für verschiedene Typen von Biclustern. Abbildung basiert auf Madeira und Oliveira [2004].

Je nachdem welchen Algorithmus man wählt, erhält man unterschiedliche Arten von Biclustern. Madeira und Oliveira [2004] definieren die folgenden vier Hauptklassen:

1. Bicluster mit einem konstanten Wert
2. Bicluster mit konstanten Werten in den Spalten oder in den Zeilen
3. Bicluster mit – in den Zeilen und/oder in den Spalten – zueinander in Beziehung stehenden Werten
4. Bicluster mit ähnlicher Wertentwicklung

Die erste Klasse beschäftigt sich mit dem Finden von konstanten Biclustern. Um in einer Datenmatrix ein konstantes oder mehrere konstante Bicluster zu entdecken, werden die Zeilen und Spalten in der Matrix so umgeordnet, dass ähnliche Zeilen und ähnliche Spalten gruppiert werden. Auf diese Weise erhält man Teilmengen von Zeilen und Teilmengen von Spalten mit ähnlichen Werten. In Abbildung 3 sieht man auf der linken Seite eine ursprüngliche Datenmatrix mit den verschiedenen ungeordneten Werten und auf der rechten Seite diese Datenmatrix mit den gefundenen Biclustern, nach einer geeigneten Umstellung der Zeilen und Spalten.

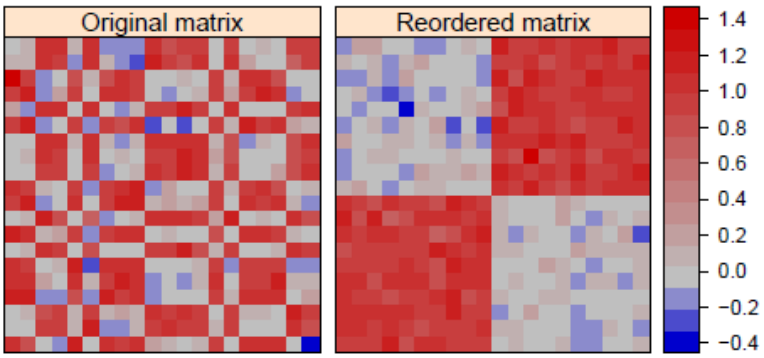

Abbildung 3: Datenmatrix vor (links) und nach (rechts) der Anwendung eines Bicluster-Algorithmus. Abbildung entnommen aus Csárdi [2009b].

Das Bicluster in Abbildung 2 (a) ist ein Beispiel für ein Bicluster mit konstanten Werten. Es handelt sich dabei sogar um ein perfektes konstantes Bicluster. Darunter versteht man eine Submatrix (I, J), in der alle Werte e_{ij} für alle $i \in I$ und $j \in J$ gleich sind:

$$e_{ij} = \mu$$

Dieser Idealfall kommt bei echten Daten allerdings sehr selten vor, da die Elemente in konstanten Biclustern normalerweise eine gewisse Streuung enthalten. Das heißt die Werte e_{ij} in einem konstanten Bicluster werden durch die Gleichung

$$e_{ij} = \eta_{ij} + \mu$$

dargestellt, wobei η_{ij} die vom gewünschten Wert abweichende Streuung darstellt [vgl. Madeira und Oliveira, 2004]. Der CC-Algorithmus von Cheng und Church [2000] aus dem R-package `biclust` ist ein Beispiel für eine Methode, die nach solchen Biclustern sucht.

In Klasse zwei definieren Madeira und Oliveira [2004] Bicluster mit konstanten Werten in den Zeilen oder in den Spalten. Die Abbildungen 2 (b) und 2 (c) sind Beispiele für solche Bicluster. Hätte man eine Genexpressionsdatenmatrix vorliegen, würde ein Bicluster mit konstanten Werten in den Zeilen einer Teilmenge an Genen entsprechen, die sich unter einer Teilmenge an Bedingungen ähnlich ausprägen, wobei sich das entsprechende Expressionslevel von Gen zu Gen unterscheiden kann. Das gleiche Prinzip gilt für Bicluster mit konstanten Werten in den Spalten: Dabei werden Teilmengen an Bedingungen gefunden, unter denen sich eine Teilmenge an Genen in ihrer Ausprägung ähnlich verhält. Auch hier variiert das Expressionslevel von Bedingung zu Bedingung. Ein solches perfektes Bicluster, mit konstanten Zeilen ist eine Submatrix (I, J), in der alle enthaltenen Werte durch eine der folgenden Ausdrücke erreicht werden können:

$$e_{ij} = \mu + \alpha_i$$

$$e_{ij} = \mu \times \alpha_i$$

In beiden Formeln ist μ der gewünschte Wert im Bicluster und α_i die dementsprechende Anpassung für Zeile $i \in I$. Der angepasste Wert e_{ij} kann, wie in den beiden Ausdrücken zu erkennen, durch eine Addition oder durch eine Multiplikation erreicht werden.
Im Fall eines Biclusters mit konstanten Werten in den Spalten gilt:

$$e_{ij} = \mu + \beta_j$$

$$e_{ij} = \mu \times \beta_j$$

Neben μ als gewünschter Wert im Bicluster ist hier β_j die entsprechende Anpassung für Spalte $j \in J$. Als Beispiel für einen nach solchen Biclustern suchenden Algorithmus sei die Plaid-Methode von Lazzeroni und Owen [2002] aus dem R-package `biclust` genannt.

Gemäß Madeira und Oliveira [2004] gibt es noch weitere Algorithmen, die in Klasse drei nach Biclustern mit in den Zeilen und/oder in den Spalten zueinander in Beziehung stehenden Werten suchen. Im Fall einer Genexpressionsdatenmatrix würden solche Algorithmen nach Biclustern suchen, in denen Teilmengen von Genen und Teilmengen von Bedingungen, in Zeilen und in Spalten, zueinander in Beziehung stehende Werte aufweisen. Dabei wird jede Zeile und jede Spalte entweder durch Addition oder Multiplikation der anderen Zeilen und Spalten mit einer Konstante erreicht. Im Grunde führen hierfür zuständige Algorithmen das, was in Klasse zwei in zwei separaten Schritten passiert, in einem Schritt durch. Klasse zwei ist also ein Spezialfall von Klasse drei, wenn $\alpha_i = 0$, bzw. $\beta_i = 0$ gesetzt wird. Solch ein perfektes Bicluster (I, J) ist definiert durch eine Teilmenge von Zeilen und eine Teilmenge von Spalten, dessen Werte e_{ij} mit folgenden Ausdrücken prognostiziert werden können:

$$e_{ij} = \mu + \alpha_i + \beta_j$$

$$e_{ij} = \mu \times \alpha_i \times \beta_j$$

Das Beispiel in Abbildung 2 (d) zeigt die Klasse drei für eine Addition und das Beispiel in Abbildung 2 (e) entsprechend für eine Multiplikation. Der Algorithmus Spectral [Kluger et al., 2003] kann hier verwendet werden, um nach Biclustern mit in den Zeilen und/oder in den Spalten zueinander in Beziehung stehenden Werten zu suchen.

Die letzte Klasse von Algorithmen sucht nach Biclustern mit ähnlicher Wertentwicklung. Dabei werden die tatsächlichen Einträge in den Matrizen nur als symbolische Werte angesehen und bei der Suche nach Biclustern nicht berücksichtigt. Man könnte sich beispielsweise bei Genexpressionsmatrizen nur für Gene interessieren, deren Expressionslevel unter einer Teilmenge an Bedingungen ansteigen oder absinken. Die exakten Expressionslevel in der Datenmatrix sind hierbei nicht relevant. Die ähnliche Wertentwicklung kann in den Zeilen und in den Spalten beobachtet werden oder entweder nur in den Zeilen oder nur in den Spalten. Ein Beispiel für eine ähnliche Wertentwicklung in beide Dimensionen zeigt Abbil-

dung 2 (f). Abbildung 2 (g) weist eine ähnliche Wertentwicklung in den Zeilen auf und die Bicluster in Abbildung 2 (h) und Abbildung 2 (i) stellen beide eine ähnliche Wertentwicklung in den Spalten dar [vgl. Madeira und Oliveira, 2004]. Murali und Kasif [2003] entwickelten die Xmotifs-Methode, um nach Biclustern mit einer ähnlichen Wertentwicklung zu suchen.

2.3 Bicluster-Strukturen

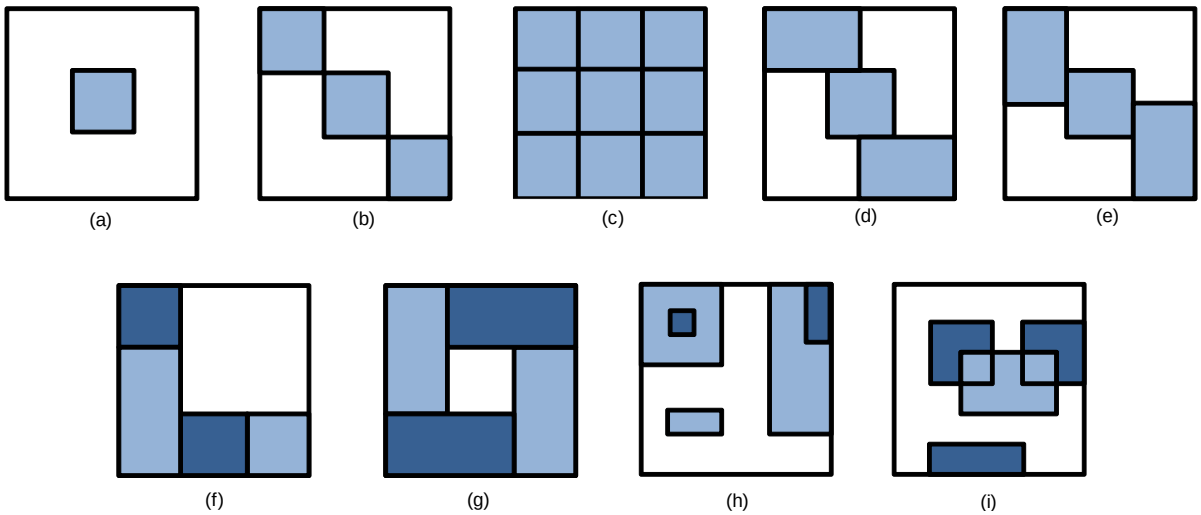

Abbildung 4: Verschiedene Strukturen von Biclustern. Abbildung basiert auf Madeira und Oliveira [2004].

Die unterschiedlichen Bicluster-Algorithmen führen zu verschiedenen Bicluster-Strukturen. Nach Madeira und Oliveira [2004] enthalten Datenmatrizen entweder nur ein Bicluster (siehe Abbildung 3 (a)) oder eine bestimmte Anzahl K an Biclustern (siehe Abbildung 3 (b) - 3 (i)), wobei K schon vorher festgelegt wird. Finden Algorithmen mehrere Bicluster, kann man folgende Strukturen unterscheiden:

1. Zeilen und Spalten gehören nur zu einem einzigen Bicluster (rechteckige Blöcke auf der Diagonalen nach Umstellung der Zeilen und Spalten) (Abbildung 3 (b))
2. Schachbrettmuster mit sich nicht überlappenden Biclustern (Abbildung 3 (c))
3. Zeilen gehören nur zu einem Bicluster (Abbildung 3 (d))
4. Spalten gehören nur zu einem Bicluster (Abbildung 3 (e))
5. Baumstruktur mit sich nicht überlappenden Biclustern (Abbildung 3 (f))

6. sich nicht überlappende Bicluster, deren Zeilen und Spalten auch in anderen Biclustern enthalten sind (Abbildung 3 (g))

7. sich überlappende Bicluster mit hierarchischer Struktur (Abbildung 3 (h))

8. willkürlich angeordnete sich überlappende Bicluster (Abbildung 3 (i))

In Abbildung 3 (b) werden die Zeilen und Spalten so umgeordnet, dass ähnliche Zeilen und ähnlich Spalten gruppiert werden. Dabei entstehen auf der Diagonale eckige Blöcke aus Teilmengen von Zeilen und Spalten mit ähnlichen Einträgen. Jede Zeile und jede Spalte gehört hier nur zu genau einem Bicluster. Dieser Fall kommt allerdings selten bei der Verwendung von echten Daten vor. Es ist also sinnvoller anzunehmen, dass die Zeilen und Spalten zu mehr als nur einem einzigen Bicluster gehören können.

Abbildung 3 (c) beschreibt ein Schachbrettmuster, d.h. es existieren K sich nicht überlappende Bicluster, bei denen jede Zeile und jede Spalte zu genau K Biclustern gehören darf.

Neben den bisher genannten Algorithmen, gibt es noch andere, die festlegen, dass die ausgewählten Zeilen bzw. Spalten nur zu einem einzigen Bicluster gehören dürfen. Dagegen können die dazu gehörigen Spalten bzw. Zeilen durchaus auch noch in anderen Biclustern enthalten sein (siehe Abbildung 3 (d) und 3 (e)).

Weitere sich nicht überlappende Strukturen sind in Abbildung 3 (f) (die sogenannte Baumstruktur) und in Abbildung 3 (g) dargestellt.

Durch die bisher in Abbildung 3 (b) - 3 (g) präsentierten Strukturen geht man davon aus, dass jede Zeile und jede Spalte zu mindestens einem Bicluster gehört. Es gibt jedoch auch Datensätze in denen es vorkommen kann, dass einige Zeilen und einige Spalten zu gar keinem Bicluster gehören und dass sich gefundenen Bicluster möglicherweise an manchen Stellen überlappen. Beides ist bei Datenmatrizen mit echten Werten am wahrscheinlichsten. Die Struktur in Abbildung 3 (h) besagt, dass die gefundenen Bicluster entweder disjunkt sind, oder dass ein Bicluster ein anderes enthält. Eine generellere Struktur erlaubt die willkürliche Anordnung und das Überlappen von Biclustern, wobei man diese dann nicht direkt in der Datenmatrix durch Umstellung der Zeilen und Spalten beobachten kann. Zudem kann es

auch wieder Zeilen und Spalten geben, die zu keinem der gefundenen Bicluster gehören. Ein Beispiel hierfür zeigt Abbildung 3 (i).

3 Funktionsweise des ISA

Der Iterative Signature Algorithmus ist ein weiterer Bicluster-Algorithmus, der vor allem bei der Analyse von sehr großen Genexpressionsdatensätzen zum Einsatz kommt [vgl. Bergmann et al., 2003]. Aus diesem Grund wird in dieser Arbeit die Funktionsweise des ISA beispielhaft anhand von Genexpressionsdaten erläutert.

Der Unterschied des Iterative Signature Algorithmus zu anderen Bicluster-Algorithmen liegt vor allem darin, dass er eine Grenzfunktion besitzt, die den Grad der Ähnlichkeit innerhalb eines Biclusters bestimmt. Dafür werden zwei Grenzparameter verwendet, von denen einer die Menge an ausgewählten Zeilen und einer die Menge an ausgewählten Spalten für das Bicluster festlegt [vgl. Csárdi, 2009b].

Vor der Veranschaulichung der Anwendung des Iterative Signature Algorithmus auf große Datensätze in Kapitel 4, erfolgt erst einmal eine detaillierte Beschreibung der Funktionsweise des ISA anhand von Genexpressionsdaten.

3.1 Normalisierung des Datensatzes

Gegeben sei eine große $n \times m$ Genexpressionsdatenmatrix E. Dabei sind die Gene zeilenweise und die experimentellen Bedingungen spaltenweise angeordnet. Die Matrix E enthält die Expressionen der einzelnen Gene bezüglich der entsprechenden Bedingungen, d.h. Element e_{ij} steht für die Expression des Gens g_i unter der Bedingung c_j. Bevor der ISA auf diese Matrix angewendet wird, ist es notwendig die Datenmatrix zu normalisieren, um die einzelnen Genexpressionen vergleichbar zu machen. Betrachtet man nur die „rohe“ Datenmatrix, ist es schwierig, zwei Gene $(g_C \text{ und } g_{C'})$ oder zwei Bedingungen $(c_G \text{ und } c_{G'})$ zu vergleichen, da verschiedene experimentelle Bedingungen die Expressionslevel in unterschiedlichem Ausmaß beeinflussen. Außerdem erhält man mit dem ISA durch geeignete Standardisierung bessere Ergebnisse [vgl. Bergmann et al., 2003].

Die Matrix *E* wird einmal zeilenweise bezüglich der Gene und einmal spaltenweise, also hinsichtlich der Bedingungen standardisiert. Zur Standardisierung wird folgende Formel [Bergmann et al., 2003] verwendet:

$$\tilde{x}_i = \left(\frac{x_i - \mu(x)}{\sigma(x)} \right)$$

Bei der zeilenweisen Normalisierung wird die Matrix *E* transponiert und obige Formel wird auf E^T angewendet. Es wird für jeden Genvektor der Mittelwert und die Standardabweichung gebildet und dann die entsprechenden Werte für jedes x_i der jeweiligen Zeile eingesetzt. Die daraus entstehenden Werte liegen dann zwischen -1 und 1. Man erhält also eine neue normalisierte Matrix E_G (hier: *G* für (engl.) gene). Bei der Normalisierung der Spalten, bzw. der Bedingungen, geht man im Grunde gleichermaßen vor. Einziger Unterschied ist, dass man *E* nicht zunächst transponieren muss. Daher wird die Formel gleich auf die Zeilen von *E* angewendet. Es ergeben sich wieder neue Werte zwischen -1 und 1 und man erhält die normalisierte Matrix E_C (hier: *C* für (engl.) condition) [vgl. Csárdi, 2009b].

3.2 Inputvektoren

Um einen Algorithmus zu starten wird ein Input benötigt. Es muss also zunächst eine genügend große Menge an Inputvektoren generiert werden. Dabei entsteht eine Matrix, die ausschließlich die Werte Null und Eins enthält. Als Startvektoren können die Zeilen- oder Spaltenvektoren, sowie beide parallel verwendet werden. Betrachtet man jetzt den Fall, dass der ISA mit einem Spaltenvektor g^0 (Genvektor) der Länge n gestartet wird, so definieren die Einsen in diesem Vektor diejenigen Gene, mit denen der Algorithmus beginnen soll [vgl. Csárdi, 2009b]. Werden zusätzlich auch die Spaltenvektoren der 0/1 Matrix verwendet, läuft die Prozedur des ISA parallel für beide Inputmengen ab. Genaueres zur Anzahl der Einsen in den Vektoren folgt in Kapitel 6 bei der Simulation.

3.3 „Scores“

Auf der Suche nach einem Bicluster in der Datenmatrix ordnet der ISA den einzelnen Genen und Bedingungen sogenannte „Scores“ zu. Diese liegen zwischen -1 und 1 und sind für diejenigen Gene und Bedingungen Null, die nicht im Bicluster enthalten sind. Je weiter der „Score“ für ein Gen bzw. eine Bedingung von Null entfernt ist, desto stärker die Verbindung zwischen dem Gen bzw. der Bedingung und dem Bicluster. Falls die Vorzeichen zweier Gene bzw. zweier Bedingungen gegensätzlich sind, sind diese unkorreliert [vgl. Csárdi, 2009b].

3.4 Signature Algorithmus

Jede einzelne Iteration des ISA besteht selbst aus einem Algorithmus und zwar dem Signature Algorithmus. Grundlegendes Ziel jeder Iteration ist es, zwei Fixpunkte zu finden, anhand derer man die Datenmatrix in Bicluster zerlegen kann. Alle Iteration laufen in zwei Schritten ab, wobei jeder der Schritte die Berechnung einer Formel [Bergmann et al., 2003] beinhaltet:

$$c^{(n+1)} = f_{t_C}\left(E_G \cdot g^{(n)}\right) \tag{1}$$

$$g^{(n+1)} = f_{t_G}\left(E_C^T \cdot c^{(n+1)}\right) \tag{2}$$

Für Formel (1) wird der Inputvektor $g^{(0)}$ genutzt, um eine Teilmenge an Bedingungen zu identifizieren, unter denen die Gene koreguliert sind. Dabei werden nur die Bedingungen berücksichtigt, unter denen die Gene aus dem Inputvektor die höchste durchschnittliche Expressionsveränderung aufweisen, also nur Bedingungen, deren „Score“ die Grenzfunktion zulässt [d’Enfert und Hube, 2007]. Hierfür wird die normalisierte Datenmatrix E_G mit dem Inputvektor $g^{(0)}$ multipliziert und als Output erhält man einen Ergebnisvektor auf den anschließend die Grenzfunktion $f_{t_C}(\cdot)$ angewendet wird. Der sogenannte „Score“ entspricht dem Wert, der für jedes Element im Vektor nach Anwendung der Grenzfunktion entsteht.

Nach Bergmann, Ihmels und Barkai [2003] lässt sich die Grenzfunktion folgendermaßen berechnen: Es werden der Mittelwert und die Standardabweichung des Ergebnisvektors berechnet und anschließend setzt der ISA alle Elemente des Vektors auf Null, deren durch die Anwendung der Grenzfunktion berechneter „Score“ nicht um mindestens $t\sigma(X)$ größer oder

kleiner ist als der Mittelwert $\mu(X)$. Das in diesem Fall im Index stehende t_C entspricht dem vorgegebenen Grenzparameter t für die Zeilen bzw. Bedingungen im Bicluster. Des Weiteren kann durch einen Richtungsparameter entschieden werden, ob Werte („Scores“) beibehalten werden, die deutlich größer oder kleiner als der Mittelwert sind. Es können auch beide Richtungen gleichzeitig zur Auswahl verwendet werden. Nach Anwendung der Grenzfunktion bekommt man schließlich als Output den Bedingungsvektor $c^{(1)}$.

Anschließend wird der aus Formel (1) entstandene Vektor $c^{(1)}$ in Formel (2) eingesetzt, um eine Teilmenge von Genen zu identifizieren, die unter den gegebenen Bedingungen des Vektors $c^{(1)}$ koreguliert sind. In diesem Fall wird die standardisierte Datenmatrix E_C zunächst transformiert und mit dem Bedingungsvektor $c^{(1)}$ multipliziert. Als Output erhält man wieder einen Ergebnisvektor, auf den die Grenzfunktion $f_{t_G}(\cdot)$ angewendet wird. Die Anwendung der Grenzfunktion im 2. Schritt erfolgt analog zu Schritt 1, wobei t in diesem Fall t_G ist, also der festgelegte Grenzparameter für die Spalten bzw. Gene im Bicluster. Letztendlich erhält man als Output den Genvektor $g^{(1)}$.

Anhand Abbildung. 5 wird das Prinzip des Signature Algorithmus nochmal graphisch veranschaulicht. D’Enfert und Hube [2007] erläutern die Problematik dabei wie folgt:

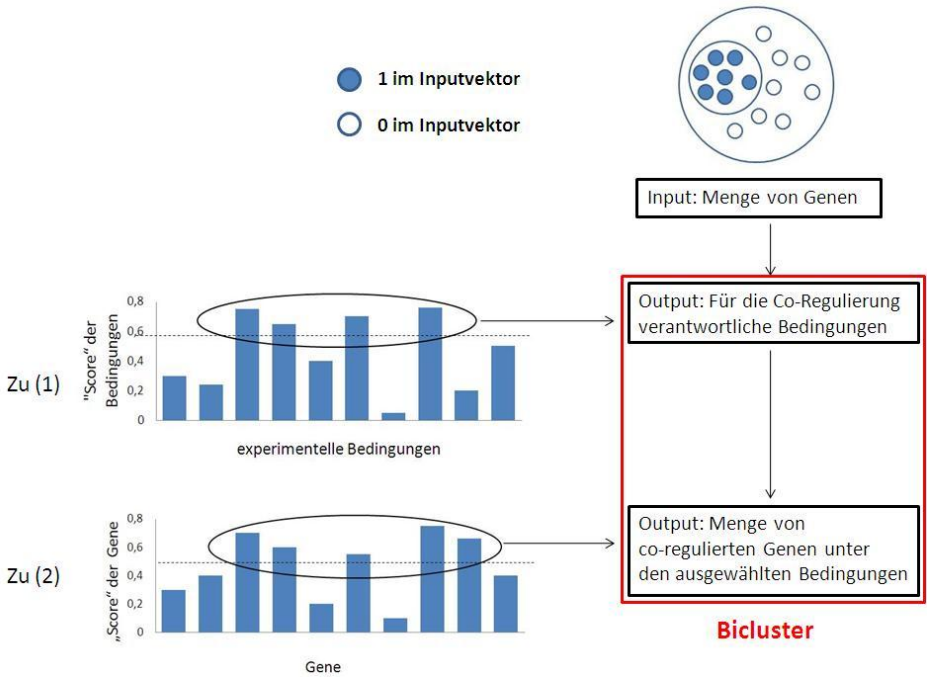

Abbildung 5: Prinzip des Signature Algorithmus. Abbildung basiert auf d’Enfert und Hube [2007].

Die blau eingekreisten Punkte entsprechen den Genen, die im Inputvektor mit einer Eins definiert sind und manchmal so gewählt werden, dass sie bereits koreguliert sind. In den Diagrammen zu den Formeln (1) und (2) werden jeweils anhand des Inputs die relevanten Bedingungen bzw. Gene identifiziert, deren „Score“ die Grenzfunktion auswählt. Die gestrichelten Linien bilden die durch die Parameter definierten Grenzen, d.h. nur die über der Linie eingekreisten Balken werden als Output ausgegeben. Die aus beiden Formeln entstandenen Outputs könnte man im vereinfachten Fall bereits verwenden, um ein Bicluster zu bilden.

Abbildung 6 [D'Enfert und Hube, 2007] zeigt, dass der Output nach Durchführung des Signature Algorithmus nicht nur den koregulierten Teil aus dem Input enthält, sondern auch weitere neu hinzugekommene Gene, die ein ähnliches Verhalten, in ihrer Expression, zu den anderen Genen aufweisen.

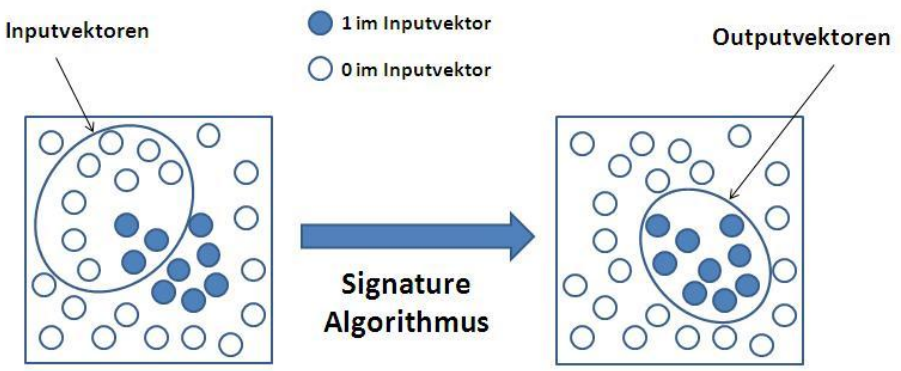

Abbildung 6: Input und Output beim Signature Algorithmus. Abbildung basiert auf d'Enfert und Hube [2007].

3.5 Iterationen

Der in Abschnitt 3.4 beschriebene Ablauf des Signature Algorithmus läuft für den Iterative Signature Algorithmus bei jeder Wiederholung ab. Dabei wird der Output der n-ten Iteration für den nächsten Durchlauf als neue Inputmenge eingesetzt. Der Genvektor $g^{(1)}$ wird also als Input wieder in die Formel (1) des 1. Schritts eingesetzt und nach erneutem Ablauf des Signature Algorithmus erhält man $\{c^{(2)},\ g^{(2)}\}$ als Ergebnismenge. Für das erneute Einsetzen von $g^{(2)}$ würde man $\{c^{(3)},\ g^{(3)}\}$ erhalten usw. Das Prinzip setzt sich iterativ solange fort, bis die Reihe $\{g^{(0)},\ g^{(1)},\ g^{(2)},\ g^{(3)}, \dots\}$ gegen einen Fixpunkt $g^{(*)}$ konvergiert, d.h. der Algorithmus terminiert [vgl. Bergman et al., 2003].

Nach jeder Iteration wird der neue Outputvektor $g^{(\cdot)}$ mit einem Konvergenzkriterium auf eine mögliche Erfüllung der Fixpunkt-Eigenschaft getestet. Automatisch erhält man den dazugehörigen Bedingungsvektor $c^{(\cdot)}$ als zweiten Fixpunkt. Das Konvergenzkriterium wird standardmäßig mit der Korrelation nach Pearson berechnet.

Sind $c^{(n-1)}$ und $c^{(n)}$, bzw $g^{(n-1)}$ und $g^{(n)}$ ähnlich genug, konvergiert die Iteration. Bergmann, Ihmels und Barkai [2003] berechnen das Konvergenzkriterium nach folgender Formel:

$$\frac{\left|g^{(*)} - g^{(n)}\right|}{\left|g^{(*)} + g^{(n)}\right|} < \varepsilon$$

Hätte man den ISA anstatt mit den Spaltenvektoren mit den Zeilenvektoren gestartet, sähe die dazugehörige Formel für die Fixpunktberechnung folgendermaßen aus:

$$\frac{\left|c^{(*)} - c^{(n)}\right|}{\left|c^{(*)} + c^{(n)}\right|} < \varepsilon$$

In diesem Fall wäre der Ablauf der Formeln (1) und (2) umgekehrt und man würde $g^{(*)}$ automatisch dazu erhalten. Werden jedoch Spalten- sowie Zeilenvektoren gleichzeitig für den Input verwendet, laufen beide Prozesse parallel ab, d.h. $g^{(*)}$ und $c^{(*)}$ berechnen sich unabhängig voneinander.

Nach Erhalten der Fixpunkte wird nun durch eine Art Multiplikation der beiden Vektoren das Bicluster aus Genen und Bedingungen gebildet. Die daraus berechnete Datenmatrix, bestehend aus Nullen und „Scores“, wird dann mit der ursprünglichen Datenmatrix verglichen. Auf diese Weise kann man diejenigen Gene herausfiltern, deren „Scores“ unter den entsprechenden Bedingungen ungleich Null sind und daher für das Bicluster in Frage kommen. Damit man letztendlich alle Bicluster erhält, werden der Algorithmus mit der gesamten Inputmenge (bzw. den gesamten Inputmengen) durchgeführt, die jeweils entstehenden Fixpunkte gesammelt und zur Zerteilung der Datenmatrix in Submatrizen verwendet [vgl. Bergman et al., 2003].

Um die Multiplikation der beiden Fixpunkte zu veranschaulichen, sei folgendes Beispiel gegeben. Der Einfachheit halber sind beide Fixpunkte reine 0/1 Vektoren:

$$g^{(*)} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \; c^{(*)} = (1 \; 1 \; 0 \; 0 \; 1)$$

Die Multiplikation des Spaltenvektors mit dem Zeilenvektor ergibt folgende Matrix:

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

Würde man diese Matrix nun über die ursprüngliche Matrix legen, könnte man gut erkennen, welche Gene und Bedingungen wegfallen würden.

3.6 Parameter

In diesem Abschnitt soll nun noch einmal dargestellt werden, was die wichtigsten Parameter bei der Verwendung des Iterative Signature Algorithmus sind, weshalb diese Parameter so bedeutend sind und was für einen Einfluss unterschiedliche Werte auf die Bicluster-Ergebnisse haben.

Wie bereits zuvor erwähnt, besitzt der ISA zwei wichtige Grenzparameter: Einer für die Auswahl der Gene und einer für die experimentellen Bedingungen unter denen die Gene koreguliert sind. Grundsätzlich bestimmen sie die Gleichartigkeit der Elemente in den Biclustern. Wählt man nun die Grenzparameter durch kleine Werte sehr streng, gelangen weniger Gene in das Bicluster und die Übereinstimmung in der Genexpression ist demnach höher. Das Gleiche trifft für die Bedingungen im Bicluster zu. Stellt man allerdings die Parameter mit großen Werten sehr großzügig ein, enthalten die Bicluster mehr Gene und auch mehr Bedingungen. Die Ähnlichkeit im Bicluster wird dadurch geringer [vgl. Csárdi, 2009b].

Neben diesen zwei wichtigsten Parametern gibt es noch die Richtungsparameter, die festlegen, ob man Werte („Scores“) beibehält, die deutlich höher („`up`“) oder niedriger („`down`“) sind als der Mittelwert der Ergebnisvektoren nach jedem Iterationsschritt. Eine Kombination aus beiden („`updown`“) ist ebenfalls möglich. Da man nach jeder Iteration zwei neue Output-Vektoren erhält, gibt es auch für jeden der beiden einen eigenen Richtungsparameter. Die drei Möglichkeiten können beliebig kombiniert werden. Wählt man nun den Richtungsparameter „`up`“, gelangen Genexpressionen in das Bicluster, die unter den entsprechenden Bedingungen hochreguliert wurden. Im Gegensatz dazu kommen beim Richtungsparameter „`down`“ nur Genexpressionen in das Bicluster, die herunterreguliert wurden. Bei der Wahl beider Richtungsparameter, also „`updown`“, gelangen sowohl hochregulierte als auch herunterregulierte Genexpressionen in das Bicluster.

Mit dem Parameter `cor.limit` wird das Korrelationslimit festgelegt, das zur Berechnung der Fixpunkte bei den Iterationen und zur Eliminierung von mehrfach vorkommenden bzw. ähnlichen Biclustern benötigt wird. Gemäß der R-Documentation `isa.iterate()` [vgl. Csárdi, 2009a] kann zwischen zwei Arten der Konvergenz gewählt werden: Entweder wird das Korrelationslimit nach Pearson verwendet oder die „Scores“ innerhalb der Ergebnisvektoren müssen eine gewisse Ähnlichkeit zueinander haben.

4 Das R-package `isa2`

Die Implementierung des Iterative Signature Algorithmus im package `isa2` der Statistik-Software R [R Development Core Team, 2010] wurde von Csárdi [2009a] entwickelt. Das R-package `biclust` von Kaiser et al. [2009] beinhaltet neben weiteren Bicluster-Methoden verschiedene Visualisierungsmöglichkeiten für die Ergebnisse des ISA. Beide R-packages sind kostenlos auf der Homepage http://www.r-project.org/ verfügbar. Die Erläuterungen in diesem Kapitel basieren auf der R-Documentation zum package `isa2`.

4.1 Der Befehl `isa()`

Bevor der ISA angewendet werden kann, muss sichergestellt werden, dass die Datenmatrix – bis auf mögliche NAs – ausschließlich numerische Werte enthält. Die einfachste Möglichkeit den Algorithmus auf die vorhandene Matrix anzuwenden, ist den Befehl `isa()` auszuführen. Dabei ist der Name der Inputmatrix das einzige Argument, das im Befehl übergeben werden muss:

```
isa(data)
```

In diesem Fall werden die für den ISA benötigten Inputvektoren in R nach dem Zufallsprinzip generiert. Man hat die Möglichkeit, Werte für die Parameter `thr.row` und `thr.col` optional zu wählen, aber standardmäßig werden jeweils die Werte 1, 1.5, 2, 2.5 und 3 verwendet. Der ISA läuft demnach für alle möglichen Kombinationen von Grenzparametern ab und für jeden einzelnen Lauf wird die gleiche Menge an Inputvektoren verwendet. Für den Parameter `no.seeds` wird, wenn nicht anders gewählt, der Standardwert 100 eingesetzt. Die nachfolgende Tabelle gibt einen Überblick über die notwendigsten Parameter.

Parameter	Details
`data`	Numerische Inputmatrix. NAs dürfen enthalten sein, verlängern aber die Laufzeit des Algorithmus.
`thr.row`	Numerischer Vektor, der die Grenzparameter für die Zeilen enthält. Der ISA verwendet alle möglichen Kombinationen aus `thr.row` und `thr.col`.
`thr.col`	Numerischer Vektor, der die Grenzparameter für die Spalten enthält. Der ISA verwendet alle möglichen Kombinationen aus `thr.row` und `thr.col`.
`no.seeds`	Anzahl der Inputvektoren, mit denen der ISA gestartet wird.

Tabelle 1: Überblick über die grundlegendsten Parameter des Iterative Signature Algorithmus.

Möchte man diese Parameter selbst wählen, verwendet man folgende Formel (die Standardeinstellungen sind hier entsprechend zu ersetzen):

```
isa(data, thr.row = seq(1, 3, by = 0.5), thr.col = seq(1, 3, by =
                    0.5), no.seeds = 100)
```

Nach der Terminierung des ISA gibt R eine Reihe von Rückgabekomponenten auf der Console aus. Der Output enthält die Elemente `rows`, `columns`, `seeddata` und `rundata`. Details dazu sind in Tabelle 2 nachzulesen.

Rückgabewert	Details
`rows`	Die im Bicluster enthaltenen Zeilen, dargestellt als numerische Matrix. Jede Spalte steht für ein Bicluster. Elemente („Scores“) ungleich Null bedeuten: Zeile ist im Bicluster enthalten. Andernfalls ist sie es nicht. Die „Scores“ liegen zwischen -1 und 1. Konvergiert der entsprechende Inputvektor nicht innerhalb der erlaubten Anzahl an Iterationen, enthält die Spalte NAs.
`columns`	Die im Bicluster enthaltenen Spalten, dargestellt als numerische Matrix. Details analog zu `rows`.
`seeddata`	Liste enthält Informationen zu den Biclustern. Jede Zeile steht für ein Bicluster.
`rundata`	Eine Reihe von Informationen zu den ISA-Iterationen.

Tabelle 2: Überblick über die Rückgabewerte des Iterative Signature Algorithmus.

Das Rückgabeargument `seeddata` beinhaltet eine tabellarische Auflistung von Informationen zu den einzelnen Biclustern. Dazu gehören die Anzahl an Iterationen bis zur Konvergenz, die verwendeten Grenzparameter, die Angabe wie oft das Bicluster gefunden wurde, der Robustheitsscore des Biclusters und das Robustheitslimit, das verwendet wurde, um die stabilsten Bicluster herauszufiltern.

Das Rückgabeargument `rundata` enthält einige Informationen zu den ISA-Iterationen. Die wichtigsten davon sind die Richtungsparameter, die angeben, welche „Scores“ in jedem ISA-Schritt beibehalten wurden, das Konvergenzkriterium für die Iterationen (zusammen mit dem verwendeten Korrelationslimit), die maximale Anzahl an erlaubten Iterationen und die Gesamtzahl an eingesetzten Inputvektoren für alle Grenzparameter.

4.2 Der ISA im Detail

In der vereinfachten Anwendung des Algorithmus durch den Befehl `isa()` werden die einzelnen Schritte des ISA automatisch von R durchgeführt. Im Folgenden wird die ISA-Analyse im Detail betrachtet. Interessant ist dies, wenn man die Parameter in den einzelnen Schritten manuell einstellen möchte.

4.2.1 `isa.normalize()`

Wie bereits in Kapitel 3 erklärt, ist es wichtig, die Datenmatrix zunächst zu standardisieren. Der Befehl in R von Csárdi [2009a] lautet dafür wie folgt:

```
isa.normalize(data, prenormalize = FALSE)
```

Die Funktion isa.normalize() wendet die passende Normalisierung auf den Datensatz an und generiert zwei standardisierte Matrizen E_r und E_c (r für (engl.) rows und c für (engl.) columns). Das Argument prenormalize wird nur dann auf TRUE gesetzt, wenn die zeilenweise Standardisierung nicht auf der Originalmatrix angewendet wird sondern auf der bereits spaltenweise standardisierten Matrix E_c.
Die Dimensionen von E_r. und E_c. können jeweils mit den Befehlen `dim(isanorm$E_r)` und `dim(isanorm$E_c)` angegeben werden. Mit dem Befehl `names(isanorm)` werden die Namen der beiden normalisierten Matrizen ausgegeben [vgl. Csárdi 2009b].

4.2.2 `generate.seeds()`

Bevor nach der Normalisierung der Daten der ISA gestartet werden kann, muss mit dem R-Befehl `generate.seeds()` eine Menge von Inputvektoren per Zufallsgenerator gebildet werden:

```
generate.seeds(length, count = 100, method = c("uni"), sparsity = 2)
```

Durch diese Funktion wird in R eine 0/1 Matrix generiert. Im Argument `length` wird die Länge der Inputvektoren festgelegt. Für Zeilenvektoren als Input (`row.seeds`) muss l`ength` der Zeilenanzahl (`nrow(data)`) in der Datenmatrix entsprechen und für Spaltenvektoren

(`col.seeds`) dementsprechend der Spaltenanzahl (`ncol(data)`). Im Parameter `count` kann die Menge an Inputvektoren eingestellt werden und `method` bestimmt, dass die Anzahl der Einsen in allen Inputvektoren gleich ist. Im letzten Argument `sparsity` wird die Anzahl von Einsen pro Inputvektor gespeichert.

Bei der Funktion `isa()` wird der Algorithmus standardmäßig mit zufällig gewählten Inputvektoren, generiert durch die Funktion `isa.iterate()`, gestartet. Dabei kann der Lauf des ISA nicht kontrolliert werden und ist ein vollkommen zufälliger Prozess, d.h., für verschiedene Durchläufe kann der Algorithmus durchaus verschiedene Ergebnisse liefern. Im Gegensatz zum so genannten „random seeding“ ist es aber genauso gut möglich, nicht-zufällige Inputvektoren zu wählen: Hat man entweder Informationen über die vorhandenen Daten oder ist man lediglich an einer speziellen Information (z.B. einer bestimmten Menge an Genen, bzw. Bedingungen) interessiert, kann man das dementsprechend im Befehl `isa.iterate()` definieren. In diesem Fall spricht man von „smart seeding“ [vgl. Csárdi, 2009b].

4.2.3 `isa.iterate()`

Nach der Standardisierung der Datenmatrix und der Generierung der Inputvektoren kann nun der ISA mit dem R Befehl `isa.iterate()` gestartet werden:

```
isa.iterate(normed.data, row.seeds, col.seeds, thr.row = 2,
            thr.col = 2, direction = c("updown", "updown"),
            convergence = cor, cor.limit = 0.99, maxiter = 100)
```

Die Funktion `isa.iterate()` wird mit den zwei standardisierten Datensätzen und der vorhandenen Inputmenge durchgeführt. Zusätzliche Einstellungen können durch die Wahl der Parameter `thr.row`, `thr.col` und `direction` vorgenommen werden. Mit dem Argument `convergence` werden das Konvergenzkriterium und das entsprechende Korrelationslimit eingestellt. Die maximale Anzahl an möglichen Iterationen kann wahlweise in der Variable `maxiter` festgelegt werden. Schließlich erhält man nach Ausführung die gleichen Rückgabekomponenten wie bei der Funktion `isa()`.

4.2.4 `isa.unique()`

Da die Funktion `isa.iterate()` im Gegensatz zum Befehl `isa()` keine Bicluster aussortiert, enthält das nach Durchführung von `isa.iterate()` entstandene Bicluster-Objekt so viele Bicluster wie vorhandene Inputvektoren. Es kann allerdings vorkommen, dass manche Inputvektoren zum gleichen Bicluster konvergieren. Um also solche Duplikate zu eliminieren, wendet man nach Beendigung der ISA-Iterationen den R-Befehl `isa.unique()` an:

```
isa.unique(normed.data, isaresult, cor.limit = 0.99)
```

R benötigt für diese Funktion als Eingabe den normalisierten Datensatz, das nach Durchführung von `isa.iterate()` entstandene Objekt `isaresult` und den Parameter `cor.limit`, der angibt, ab welchem Korrelationslimit zwei Bicluster als gleich angesehen werden. Der Rückgabewert sieht im Prinzip genauso aus wie die Ausgabe von `isa.iterate()`, nur ohne die mehrfach vorkommenden Bicluster.

4.2.5 `isa.filter.robust()`

Bicluster Algorithmen sollen nicht nur alle in der Datenmatrix enthaltenen Bicluster finden, sondern auch deren Robustheit berücksichtigen. Durch die Berechnung der Robustheit wird sicher gestellt, dass die in den Biclustern enthaltenen Zeilen und Spalten wirklich korreliert sind und nicht nur durch Zufall von den Algorithmen gefunden wurden [vgl. Csárdi, 2009b]. Um herauszubekommen wie korreliert die Zeilen und Spalten innerhalb des Biclusters wirklich sind, wird folgende Formel angewendet:

```
robustness(normed.data, row.scores, col.scores)
```

Neben der Angabe des normierten Datensatzes benötigt die Funktion `robustness()` zudem die `row.scores` und die `col.scores`. Unter den beiden letzten Argumenten versteht man die “Scores” der Zeilenkomponenten bzw. der Spaltenkomponenten der Bicluster. Man erhält diese nach der Durchführung von `isa.unique()` auf der R-Console, durch Eingabe von `isaresult$rows`, bzw. `isaresult$columns`.

Um nun herauszufinden, welche Bicluster wirklich robust sind, wird der Befehl `isa.filter.robust()` ausgeführt. Dabei wird die Datenmatrix durchgemischt und auf die neu entstandene Matrix wird dieselbe Prozedur angewendet, wie bisher auf die Originaldatenmatrix. Dann werden die Robustheits-„Scores“ der Bicluster für beide Matrizen berechnet und verglichen. Anschließend entfernt der ISA alle Bicluster der Originalmatrix, die einen niedrigeren „Score“ haben als mindestens eines der Bicluster der permutierten Matrix. Diese Filterung funktioniert aber nur, wenn beide Bicluster-Mengen mit denselben Grenzparametern gefunden wurden [vgl. Csárdi, 2009b]. Implementiert ist diese Prozedur in folgender Funktion:

```
isa.filter.robust(data, normed.data, perms = 1,
                  row.seeds, col.seeds)
```

Dabei ist es nicht unbedingt notwendig, dieselben Inputvektoren (d.h. `row.seeds` und/oder `col.seeds`) wie für die Originaldatenmatrix zu verwenden. Der Parameter `perm` gibt an, wie oft die Originalmatrix permutiert werden soll, wobei standardmäßig nur einmal permutiert wird [vgl. Csárdi, 2009a]. Zurückgegeben wird letztendlich das Ergebnis aus `isa.unique()` in gefilterter Version [vgl. Csárdi, 2009b].

5 Anwendung auf simulierte Datensätze

In diesem Kapitel werden nun die Funktionsweise des Iterative Signature Algorithmus und damit die Auswirkungen unterschiedlich eingestellter Parameter anhand von sechs Simulationen veranschaulicht. Dabei soll untersucht werden, wie gut der ISA versteckte Bicluster in simulierten Datensätzen identifizieren kann. Um den Algorithmus praktisch durchführen zu können, wird die statistische Software R [R Development Core Team, 2010] verwendet.

Bevor man sich an die eigentliche Programmierung macht, müssen als Vorbereitung die notwendigen R-packages `biclust` [Kaiser et al., 2009] und `isa2` [Csárdi, 2009a] geladen werden. Die dafür notwendigen Kommandos lauten `library(biclust)` und `library(isa2)`.

Für die Nachvollziehbarkeit, der aus dem R-Codes resultierenden Ergebnisse wird mit dem Befehl `set.seed()` ein fester Startwert für die Zufallszahlengenerierung gesetzt. Der Zufallszahlengenerator wird für die Generierung der Datensätze benötigt. Man erhält demnach bei jedem erneuten Durchlauf der ISA-Analysen die gleichen Ergebnisse.

5.1 Datengenerierender Prozess

Um die Simulation des ISA durchführen zu können, werden zunächst sechs verschiedene standardnormalverteilte $(\mu = 0,\ \sigma = 1)$ Datenmatrizen mit jeweils 1000 Zeilen und 500 Spalten generiert. In jeder dieser Matrizen sollen drei sich nicht überlappende Bicluster versteckt werden. Für einen Datensatz (`artdata`) wird jeweils eine Stichprobe der Größe 150 aus der Menge der Zeilen und der Menge der Spalten gezogen. Die gezogenen Zeilen und Spalten werden anschließend gleichmäßig aufgeteilt, so dass drei Teilmatrizen mit jeweils 50 Zeilen und 50 Spalten entstehen. Im nächsten Schritt werden die Verteilungen in den drei Biclustern festgelegt, wobei es sich auch hier wieder um eine Normalverteilung handelt. Die für die Verteilung in den Biclustern notwendigen Parameter (Mittelwert und Standardabweichung) können optional eingestellt werden. Die verschiedenen Datensätze unterscheiden sich dabei lediglich in den Verteilungen der drei versteckten Bicluster, genauer gesagt in

deren Mittelwerten. In den ersten drei Datensätzen liegen die Mittelwerte mit 5, 3, 4, bzw. -5, -3, -4 und 5, 3, -4 weiter entfernt vom globalen Mittelwert μ = 0, dagegen liegen sie in den anderen drei Datensätzen mit 3, 2, 1, bzw. -3, -2, -1 und 3, -2, 1 näher am globalen Mittelwert μ = 0. Für die Standardabweichung hingegen wurde in allen Biclustern der sechs Datensätze σ = 0,1 gewählt. Im Grunde hätte man die Funktionsweise des Algorithmus auch nur anhand einer einzigen Datenmatrix untersuchen können, doch in dieser Bachelorarbeit wurden, hinsichtlich der Erwartung möglichst gute und aussagekräftige Analyseergebnisse zu bekommen, mehrere verschiedene Datensätze verwendet. Der R-Code für die Erstellung der sechs Datensätze befindet sich in Anhang D.1.

5.2 Visualisierung der Daten

Eine Möglichkeit die Datensätze grafisch darzustellen bieten die `drawHeatmap()`- und die `drawHeatmap2()`-Funktionen aus dem package `biclust` [Kaiser et al., 2009]. Beide sind mit ihren Default-Werten angegeben:

```
drawHeatmap(x, bicResult = NULL, number = NA, local = TRUE,...)
drawHeatmap2(x, bicResult = NULL, number = NA, plotAll = FALSE)
```

Dabei kann mit der R-Funktion `drawHeatmap()` die Datenmatrix mit umgeordneten Zeilen und Spalten so dargestellt werden, dass in der linken oberen Ecke das Bicluster erscheint. Möchte man die ungeordnete Datenmatrix darstellen, muss das Argument `bicResult` gleich `NULL` gesetzt werden. Für den Fall dass man mehr als nur ein Bicluster grafisch darstellen möchte, ist die Funktion `drawHeatmap2()` zu verwenden. In beiden R-Befehlen wird mit `x` die Datenmatrix übergeben und falls `bicResult` auf `TRUE` gesetzt wird, wird die Datenmatrix mit umgeordneten Zeilen und Spalten dargestellt. Mit `number` kann die Anzahl der Bicluster angegeben werde, die im Plot erscheinen sollen. Sollen alle Bicluster in der Grafik erscheinen, muss in der `drawHeatmap2()`-Funktion das Argument `plotAll` auf `TRUE` gesetzt werden. Mit `local` kann optional angegeben werden, ob nur die Zeilen und Spalten des Biclusters dargestellt werden sollen.

Die `drawHeatmap2()`-Funktionen wurden speziell mit den folgenden Argumenten ausgeführt [siehe Anhang D.1]:

```
drawHeatmap2(artdata, bicResult = NULL, number = NA,
             plotAll = FALSE)
drawHeatmap2(artdata, bicResult = artres, number = NA,
             plotAll = TRUE)
```

In Abbildung 7 sieht man links eine der ungeordneten Datenmatrizen `artdata` und rechts die gleiche Datenmatrix mit umgeordneten Zeilen und Spalten:

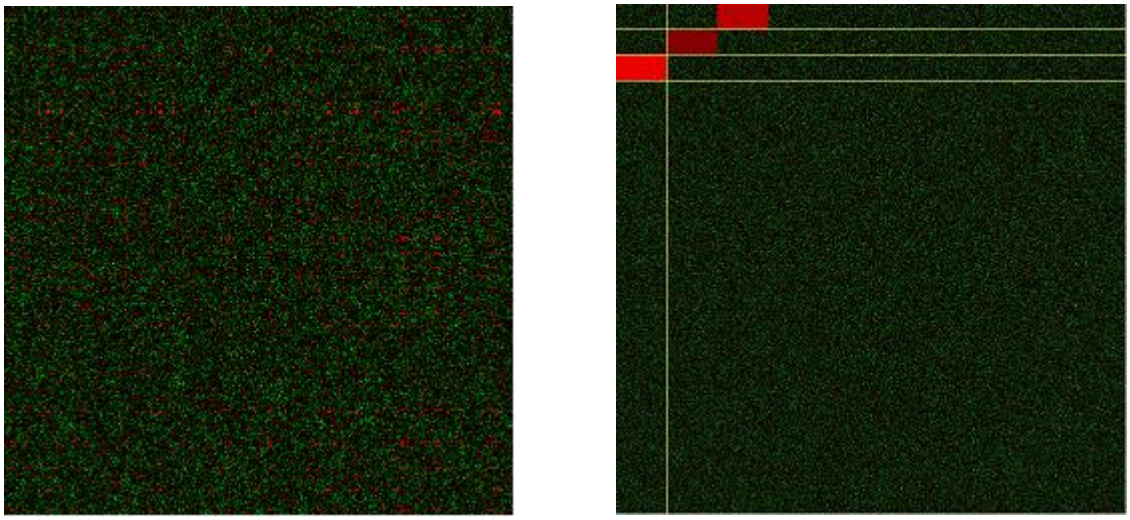

Abbildung 7: Datenmatrix `artdata` vor (links) und nach (rechts) der Umordnung der Zeilen und Spalten.

In der linken Abbildung sieht man die ungeordnete Datenmatrix `artdata` mit den darin versteckten drei Biclustern. Die verschiedenen rot abgestuften Punkte stellen die zu den verschiedenen Biclustern gehörenden Daten dar. Die übrigen grünen Datenpunkte bilden den Rest des Datensatzes. Nach Umstellung der Zeilen und Spalten mit `drawHeatmap2()` werden die drei Bicluster in der linken oberen Ecke sichtbar. In beiden Grafiken ist durch die vielen kleinen Datenpunkte erkennbar, dass große Datensätze zur Analyse des ISA verwendet werden.

5.3 Verschiedene Parametersettings

Bevor im Abschnitt 5.6 der Iterative Signature Algorithmus mit unterschiedlich eingestellten Parameterwerten durchgeführt wird, wird in diesem Abschnitt ein Überblick über diejenigen Parameter gegeben die variiert werden sollen. In Tabelle 3 kann nachgelesen werden, welche verschiedenen Parametersettings beim Lauf des ISA verwendet werden.

Funktion	Parameter	Details	Parameterwerte
ISA-Ablauf	`i`	`i` gibt an, wie oft der ISA durchgeführt werden soll. Pro Lauf wird eine neue Datenmatrix generiert.	`c(1:3)`
`generate.seeds`	`length`	Länge der Inputvektoren. Für Zeilenvektoren (`row.seeds`) entspricht `length` der Zeilenanzahl in der Datenmatrix und für Spaltenvektoren (`col.seeds`) der Spaltenanzahl.	Für alle Simulationen: `nrow(artdata)`
	`count[j]`	Anzahl der Inputvektoren.	`seq(50, 150, by=50)`
	`sparsity[k]`	Anzahl der Einsen in den Inputvektoren.	`seq(100, 900, by=400)`
`isa.iterate`	`seed`	Wahl der Inputvektoren: Zeilenvektoren (`row.seeds`) und (oder) Spaltenvektoren (`col.seeds`)	`row.seeds <- generate.seeds (length = nrow(artdata), count = count[j], method = c("uni"), sparsity = sparsity[k])`
	`thr.row[l]`	Grenzparameter für die Auswahl der Zeilen im Bicluster.	`seq(1.5, 3, by=0.5)`
	`thr.col[m]`	Grenzparameter für die Auswahl der Spalten im Bicluster.	`seq(1.5, 3, by=0.5)`
	`d1[n]`	Richtungsparameter `d1` für die Zeilen.	`c("up", "down", "updown")`
	`d2[o]`	Richtungsparameter `d2` für die Spalten	`c("up", "down", "updown")`
	`cl.iterate[p]`	Korrelationskoeffizient für die ISA Iterationen. Werte liegen zwischen 0 und 1.	`seq(0.1, 0.9, by=0.4)`
`isa.unique`	`cl.unique[q]`	Korrelationskoeffizient für die Entfernung ähnlicher (gleicher) Bicluster. Werte liegen zwischen 0 und 1.	`seq(0.1, 0.9, by=0.4)`

Tabelle 3: Überblick über die verschiedenen Parametersettings.

Wie bereits erwähnt, wurden sechs Datensätze mit verschiedenen Verteilungen in den versteckten Biclustern generiert. Der Parameter `i` gibt an, wie oft der ISA mit den unterschiedlichen Bicluster-Verteilungen durchlaufen werden soll. In unserem Fall sind es jeweils drei Durchläufe, wobei aufgrund des Zufallsgenerators pro Durchlauf immer ein neuer Datensatz produziert wird.

Für die Parameter der Funktion `generate.seeds()` werden nur `count` und `sparsity` verändert. Die Länge der Inputvektoren ist in der gesamten Analyse `nrow(artdata)`, d.h.

es werden nur `row.seeds` erstellt. `Count[j]` nimmt den Werte `j` an mit `j` ∈ {50, 100, 150} und `sparsity[k]` den Wert `k` mit `k` ∈ {100, 500, 900}.

Die für `isa.iterate()` wichtigen Grenzparameter `thr.row[l]` und `thr.col[m]` variieren beide zwischen 1,5, 2 und 3 und für die Richtungsparameter `d1[n]` und `d2[o]` werden alle möglichen Kombinationen aus „`up`", „`down`" und „`updown`" gebildet. Der Korrelationskoeffizient `cl.iterate[p]` für die ISA-Iterationen nimmt die Werte 0,1, 0,5 und 0,9 an. In der Spalte Parameterwert (siehe Tabelle 3) steht für `seed` die R-Funktion mit den entsprechenden zugewiesenen Argumenten und der `length nrow(artdata)`. Mit dem Argument `method` wird sichergestellt, dass die Einsen in den Inputvektoren immer konstant zufällig verteilt werden.

Im Befehl `isa.unique()` werden dem Korrelationskoeffizient `cl.unique[q]` für die Entfernung ähnlicher (gleicher) Bicluster, genau wie für den Korrelationskoeffizient in `isa.iterate()`, die Werte 0,1, 0,5 und 0,9 zugewiesen.

5.4 Der Jaccard-Index

Da nun geklärt ist, welche Parameter in den ISA-Simulationen abgewandelt werden sollen, wird noch eine Validierungsmethode benötigt, um die gefundenen Bicluster pro ISA-Durchlauf mit den versteckten Biclustern zu vergleichen. In dieser Arbeit wird der Jaccard-Index verwendet, um die Ähnlichkeit der Bicluster-Ergebnisse zu untersuchen.

Erfunden und schließlich 1901 veröffentlich wurde er von Paul Jaccard. Folgendes sei zur Theorie gesagt: Der Jaccard-Index vergleicht zwei Mengen und gibt den übereinstimmenden Anteil von Datenpunkten an der Gesamtzahl von vorhandenen Datenpunkten an. Demnach kann der Jaccard Werte aus dem Bereich [0, 1] annehmen. Er nimmt den Wert Null an, falls die zu vergleichenden Mengen überhaupt nicht übereinstimmen und den Wert Eins falls sie vollkommen identisch sind. Es gilt die Formel:

$$JacInd(BC_1, BC_2) = \frac{|BCP_1 \cap BCP_2|}{|BCP_1| + |BCP_2| - |BCP_1 \cap BCP_2|}$$

wobei

$$BCP_i = \{(a, b)|(\exists k|a \in BC_{ik} \wedge b \in BC_{ik})\}.$$

BC_i ist das i-te Bicluster-Ergebnis und BC_{ik} das k-te Bicluster aus dem i-ten Bicluster-Ergebnis. Mit BCP_1 ist die Menge der Datenpunkte (a, b) aus dem k-ten Bicluster des ersten Bicluster-Ergebnisses gemeint. Das gleiche gilt für BCP_2 mit $i = 2$. Von diesen beiden Mengen wird die Schnittmenge gebildet und durch die Vereinigung der beiden Mengen BCP_1 und BCP_2 geteilt. Da mit der Vereinigung nur diejenigen Datenpunkte gemeint sind, die jeweils nur in BCP_1 oder BCP_2 enthalten sein sollen, wird von der Summe der Datenpunkte aus BCP_1 und BCP_2, noch die Schnittmenge von BCP_1 und BCP_2 abgezogen.

5.5 Der data frame

Vor dem Starten des ISA wird zunächst für jeden der sechs Datensätze ein data frame mit so vielen Spalten wie verwendete Parameter und sovielen Zeilen wie vorhandene Ergebnisse erzeugt. Wenn man nun die Parameteraufstellung in Tabelle 3 betrachtet und dabei nur die zu variierenden Parameter berücksichtigt, kommt man auf neun Spalten pro data frame. Also ergeben sich, zusammen mit dem Jaccard-Index, insgesamt zehn Spalten.

Um die Anzahl der Zeilen zu bekommen, müssen alle Möglichkeiten an Parameterwerten miteinander multipliziert werden. Durch eine einfache Rechung ergibt sich die Zeilenanzahl 34992 $(3 \times 3 \times 3 \times 4 \times 4 \times 3 \times 3 \times 3 \times 3)$. Fügt man schließlich alle Ergebnisse der sechs Datesätze zusammen, ergibt sich eine Zeilenanzahl von 209952 (6×34992). Im R-Code für die ISA-Simulation [siehe Anhang D.2] wird der data frame zunächst mit folgender Funktion als Matrix programmiert:

```
res <- matrix(0,34992, 10)
```

Die Matrix besteht zunächst nur aus Nullen und wird nach jedem Durchlauf des ISA mit den entsprechenden Parameterwerten und Jaccard-Resultaten zeilenweise gefüllt. Das zeilenweise Füllen geschieht durch einen Index `z`, der vor dem ersten ISA-Lauf auf Eins gesetzt wird und nach jedem Durchlauf des Algorithmus um Eins erhöht wird. So wird beim zweiten Durchlauf die zweite Zeile des data frames gefüllt und nach dem dritten die dritte Zeile usw.. Auch hier als Veranschaulichung der entsprechende R-Befehl aus dem R-Code [siehe Anhang D.2]:

```
res[z,] <- c(i, count[j], sparsity[k], tr[l], tc[m], d1[n],
             d2[o], cl.iterate[p], cl.unique[q], jaccard)
z <- z+1
```

Als Beispiel sei folgender Ausschnitt aus einem solchen data frame gegeben:

	artdata	count	sparsity	Row	Col	d1	d2	cl_iterate	cl_unique	Jaccard
[1,]	"1"	"50"	"100"	"1.5"	"1.5"	"up"	"up"	"0.1"	"0.1"	"0.908212560386473"
[2,]	"1"	"50"	"100"	"1.5"	"1.5"	"up"	"up"	"0.1"	"0.5"	"0.908212560386473"
[3,]	"1"	"50"	"100"	"1.5"	"1.5"	"up"	"up"	"0.1"	"0.9"	"0.730098476402824"
[4,]	"1"	"50"	"100"	"1.5"	"1.5"	"up"	"up"	"0.5"	"0.1"	"1"
[5,]	"1"	"50"	"100"	"1.5"	"1.5"	"up"	"up"	"0.5"	"0.5"	"1"
[6,]	"1"	"50"	"100"	"1.5"	"1.5"	"up"	"up"	"0.5"	"0.9"	"1"

Abbildung 7: Ausschnitt aus einem als Matrix definierten data frame.

Es handelt sich dabei um die ersten sechs Zeilen aus dem data frame der ISA-Simulation mit dem Datensatz1. Die Zeilen enthalten die entsprechenden neun Parameterwerte und den dazu berechneten Jaccard-Index.

5.6 Die for-Schleife

Um den Iterative Signature Algorithmus, mit den definierten Parametersettings in R zu starten, muss man sich zunächst überlegen, wie man dies am effektivsten programmiert. Anstatt den Algorithmus mit den verschiedenen Startwerten immer wieder von Hand durchzuführen, gibt es die Möglichkeit, mit Schleifen zahlreiche Befehle automatisch zu wiederholen. Dadurch erspart man sich viel Schreibarbeit in R. Bei der ISA-Simulation ist eine solche Methode von Vorteil, da spätere Iterationsschritte von vorherigen abhängig sind [vgl. Ligges, 2008]. Man möchte schließlich die im `isa2`-package enthaltenen Befehle und Funktionen mit allen möglichen Kombinationen von Startwerten laufen lassen.

Nach Ligges [2008] sieht eine for-Schleife folgendermaßen aus:

```
for(i in M){Ausdruck}
```

In einer solchen Schleife nimmt der Index `i` zunächst das erste Element der Menge `M` an und führt hierfür den in geschweiften Klammern definierten `Ausdruck` aus. Dann springt der

Index `i` zum zweiten Element aus `M` und führt mit dem entsprechenden Wert für `i` wieder den `Ausdruck` in den geschweiften Klammern aus, usw.

In dieser Arbeit werden haupsächlich Laufindizes verwendet. Ein beispielhafter Ausschnitt, aus dem dazugehörigen R-Code in Anhang D.2 für die ISA-Simulation sieht wie folgt aus:

```
for(l in 1: length(thr.row))
{
  # R-Befehle
}
```

Die R-Befehle werden für alle `l` im Vektor `length(thr.row)` ausgeführt, wobei der Parameter `thr.row` die Elemente 1,5, 2, 2,5 und 3 annehmen kann (siehe Tabelle 3).

Genausogut können anstatt der R-Befehle im R-Code weitere Schleifen eingefügt werden, wie der nachfolgende Ausschnitt aus dem R-Code aus Anhang D.2 zeigt:

```
for(l in 1: length(thr.row))
  {
    for(m in 1:length(thr.col))
      {
        # R-Befehle (oder weitere Schleifen)
      }
  }
```

Man spricht von geschachtelten Schleifen, wenn sich mehrere Schleifen ineinander befinden. Im dargestellten Beispiel aus der ISA-Simulation würde der Laufindex `l` erst dann von 1,5 auf 2 springen, wenn die innere Schleife alle Werte 1,5, 2, 2,5 und 3 durchlaufen hat.

5.7 Durchführung der Simulationen

Nach der bisherigen Vorarbeit in Kapitel 5, wird nun in diesem Abschnitt die Durchführung der ISA-Analyse anhand des R-Codes in Anhang D.2 beschrieben. Die Analyse wird für jeden der sechs Datensätze separat durchgeführt. Die Ergebnisse werden im Anschluss gespeichert und unter Verwendung von R zu einem großen data frame mit allen Parameterwerten und Jaccard-Ergebnissen zusammengefügt.

Die Simulation des Algorithmus beinhaltet neun verschiedene for-Schleifen, d.h. für jeden Parameter wurde eine programmiert. Bei den for-Schleifen handelt es sich um geschachtelte Schleifen, in denen die zum jeweiligen Parameter gehörenden Werte aus Tabelle 3 durchlaufen werden.

Nach der Normalisierung des Datensatzes mit `isa.normalize()` werden die `row.seeds` als Inputvektoren generiert. Da hierfür die zwei Parameter `count` und `sparsity` mit den Laufindizes `j` und `k` unterschiedlich eingestellt werden sollen, werden zwei ineinander geschachtelte for-Schleifen benötigt. Die `row.seeds` sollen schließlich mit allen möglichen Kombinationen aus Parameterwerten `count[j]` und `sparsity[k]` generiert werden, damit der ISA mit verschiedenen Inputvektoren gestartet werden kann:

```
row.seeds <- generate.seeds (length = nrow(artdata),
             count = count[j], method = c("uni"),
             sparsity = sparsity[k])
```

Im nächsten Schritt wird der ISA mit `isa.iterate()` und den dazugehörigen Inputvektoren gestartet. Dafür werden einige unterschiedliche Parametersettings verwendet, also sind wieder einige geschachtelte for-Schleifen nötig. Bei den abzuwandelnden Parametern handelt es sich um die `thr.row[l]`, `thr.col[m]`, `d1[n]`, `d2[o]` und `cl.iterate[p]`. Diese fünf Parameter durchlaufen mit ihren Laufindizes den definierten Wertebereich und es entstehen unterschiedliche Kombinationen von Parameterwerten für die ISA-Iterationen. Die dazugehörige R-Funktion sieht wie folgt aus:

```
isamodules <- isa.iterate(isanorm, row.seeds = row.seeds,
              thr.row = thr.row[l], thr.col = thr.col[m],
              direction = c(d1[n], d2[o]),
              convergence = "cor", cor.limit = cl.iterate[p],
              oscillation = FALSE, maxiter = 100)
```

Nach den Durchläufen von `isa.iterate()` erhält man mit `isamodules` einige Rückgabewerte auf der R-Console (siehe Tabelle 2).

Um nun mögliche Duplikate oder nicht konvergierte Inputvektoren aus den Bicluster-Ergebnissen zu entfernen, werden diesmal keine geschachtelten for-Schleifen benötigt. Da

für den Befehl `isa.unique()` nur dem Argument `cl_unique[q]` unterschiedlichen Werten `q` zugewiesen werden, reicht eine einzige for-Schleife:

```
for (q in 1:length(cl.unique))
{
  isamodules2 <- isa.unique(isanorm, isamodules,
                 cor.limit = cl.unique[q])
  # R-Befehle
}
```

Das definierte Argument `isamodules2` gibt ein gefiltertes Bicluster-Ergebnis als Rückgabewert aus.

Neben dem Filtern mit `isa.unique()`, werden die Bicluster-Ergebnisse noch auf Robustheit überprüft. Dafür ist keine for-Schleife notwendig, da keine Parameter speziell eingestellt werden müssen. Dies wird ganz einfach mit dem R-Befehl

```
isamodules3 <- isa.filter.robust(artdata, isanorm,
               isamodules2, perms = 1)
```

durchgeführt und man erhält einen neuen Rückgabewert `isamodules3`.

Der letzte wichtige Punkt in der Analyse ist der Vergleich der Bicluster-Ergebnisse mit denen, die im Datensatz versteckt wurden. Da der Jaccard eine Funktion aus dem package `biclust` ist, muss ein ISA-Objekt zunächst in ein `Biclust`-Objekt umgewandelt werden, bevor man beide Resultate vergleichen kann. Dies funktioniert mit dem Befehl:

```
Bc <- isa.biclust(isamodules3)
```

Nachdem der Rückgabewert `isamodules3` mit der Funktion `isa.biclust()` in ein `Biclust`-Objekt umgewandelt wurde, können mit der Funktion

```
jaccard <- jaccardind(artres, Bc)
```

die `Biclust`-Objekte `artres` und `Bc` miteinander verglichen und der Jaccard-Index berechnet werden. Das Umwandeln in ein `Biclust`-Objekt ist allerdings nur möglich, wenn der ISA nach einem Durchlauf mindestens ein Bicluster gefunden hat. Kann der ISA für be-

stimmte Parametersettings keine Ergebnisse liefern, bricht der Algorithmus den Lauf an dieser Stelle ab und man erhält eine Fehlermeldung. Der Jaccard kann somit für solche Resultate nicht berechnet werden, obwohl er nach Definition in solchen Fällen den Wert Null annehmen muss. In R kann das Problem der plötzlichen Terminierung mit einer if-Schleife umgangen werden. In dieser Schleife wird nach jedem ISA-Lauf überprüft, ob die Anzahl der Spalten des ISA-Rückgabewerts `rows` (siehe Tabelle 2) Null ist:

```
if(ncol(isamodules3$rows) == 0)
{
  jaccard <- "0"
}
else
{
# R-Befehle
}
```

Falls dies der Fall ist, wird dem Jaccard automatisch der Wert Null zugewiesen und eine Fehlermeldung kommt nicht zustande. Findet der ISA aber ein Ergebnis, wird der Jaccard wie üblich berechnet.

5.8 Zusammenfassen der Simulationsergebnisse

Bevor im Anschluss an die Durchführung der ISA-Simulationen in Kapitel 6 die Auswertung der Ergebnisse erfolgt, müssen die sechs einzelnen gespeicherten Simulationsergebnisse zu einem großen data frame zusammengefügt werden. Dafür werden die Simulationen mit dem Befehl `load()` in R geladen und die jeweiligen data frames mit folgendem Befehl in richtige data frames umgewandelt [siehe Anhang D.3]:

```
dataframe1 <- data.frame(res)
```

In Abbildung 8 sind die ersten sechs Zeilen aus dem data frame der ISA-Simulation mit dem Datensatz1, nach Umwandlung in einen richtigen data frame dargestellt:

	artdata	count	sparsity	Row	Col	d1	d2	cl_iterate	cl_unique	Jaccard
1	1	50	100	1.5	1.5	up	up	0.1	0.1	0.908212560386473
2	1	50	100	1.5	1.5	up	up	0.1	0.5	0.908212560386473
3	1	50	100	1.5	1.5	up	up	0.1	0.9	0.730098476402824
4	1	50	100	1.5	1.5	up	up	0.5	0.1	1
5	1	50	100	1.5	1.5	up	up	0.5	0.5	1
6	1	50	100	1.5	1.5	up	up	0.5	0.9	1

Abbildung 8: Beispiel für einen data frame nach Umwandlung mit data.frames().

Mit `str(dataframe1)` können die Strukturen, der in daten1 enthaltenen Parameter betrachtet werden. Bei der Umwandlung in einen data frame werden alle Parameter als Faktoren interpretiert. Es wäre allerdings sinnvoller, den Jaccard-Index als numerische Variable zu kodieren:

```
dataframe1$Jaccard <- as.numeric(as.factor(dataframe1$Jaccard))
```

Nach der entsprechenden Bearbeitung der anderen fünf Simulations-Ergebnisse werden alle data frames zeilenweise mit `rbind()` zu einem großen data frame zusammengefügt. Der dazugehörige R-Code befindet sich in Anhang D.3.

6 Analyse der Simulationen

Die Analyse der Bicluster-Ergebnisse erfolgt mit einem generalisierten Regressionsmodell. Dabei soll untersucht werden, wie gut der ISA die versteckten Bicluster anhand der unterschiedlichen Parametersettings findet. Es wird der Zusammenhang zwischen dem Jaccard und den anderen Variablen untersucht und getestet, ob der Jaccard für die eingestellten Parameterwerte ansteigt und Werte nahe Eins bzw. den Wert Eins annimmt oder ob er sinkt bzw. gegen Null geht.

6.1 Das generalisierte lineare Regressionsmodell (GLM)

Zur Auswertung des zusammengefügten data frames, wurde das generalisierte lineare Regressionsmodell herangezogen. Das generalisierte lineare Regressionsmodell ist eine Verallgemeinerung des linearen Regressionsmodells. Nach Fahrmeir et al. [2007a] kommen lineare Modelle vor allem bei Regressionsanalyse von stetigen Zielvariablen, die normalverteilt oder zumindest approximativ normalverteilt sind, zum Einsatz. Besteht allerdings der Fall, dass die Zielvariable binär, bzw. kategorial oder eine Zählvariablen ist, wird das generalisierte lineare Regressionsmodell verwendet.

Das Grundprinzip des GLM gleicht dem klassischen linearen Regressionsmodell. Mit der abhängigen metrischen Variable y und den unabhängigen metrischen oder binär kodierten kategorialen Variablen oder Regressoren hat die Modellgleichung eines linearen Modells die Form:

$$y_i = \beta_0 + \beta_1 x_1 + \ldots + \beta_k x_{ik} + \varepsilon_i \,, \qquad i = 1, \ldots, n$$

Bei einer Regressionsanalyse soll der Einfluss der erklärenden Variablen x auf die abhängige Variable y untersucht werden. Dabei wird der bedingte Erwartungswert $E(y|\, x_1, \ldots, x_k)$ gebildet und mit obiger Gleichung wird eine Funktion berechnet, die die Abhängigkeit der Variablen mit einer Geraden beschreibt. Mit dieser Regressionsgleichung können durch Einsetzen von Werten für $\boldsymbol{x_i}$ Prognosen für die abhängige Zielvariable y berechnet werden. Dabei kann mit der geschätzten linearen Funktion

$$\hat{f}(x_1,\ldots,x_k) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \ldots + \hat{\beta}_k x_k$$

die Schätzung $\hat{E}(y|\, x_1,\ldots,x_k)$ für den bedingten Erwartungswert von y, gegeben die Kovariablen $x_1,\ldots,x_k$ betrachtet werden. Die Prognose von y wird dabei mit $\hat{y}$ bezeichnet [vgl. Fahrmeir et al., 2007a].

Ein lineares Regressionsmodell mit nicht-normal verteilten Zielvariablen besitzt gemäß Fahrmeir et al. [2007a] folgende Eigenschaften:

1. Der lineare Prädiktor

$$\eta = x_i'\beta = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_k x_{ik}$$

wird mit dem Erwartungswert der abhängigen Variable y $\left(\mu = E(y|\, x_1,\ldots,x_k)\right)$ durch eine Responsefunktion h bzw. eine Linkfunktion $g = h^{-1}$ verknüpft:

$$\mu = h(\eta) \text{ bzw. } \eta = g(\mu)\,.$$

2. Die Zielvariablen in einem GLM besitzen Verteilungen, die aus der Familie der Exponentialverteilungen stammen, welche die folgende Form hat:

$$f(y_i|\, \theta_i,\, \phi_i) = \exp\left(\frac{y_i\theta_i - b(\theta_i)}{\phi_i} + c(y_i,\, \phi_i)\right).$$

Der Parameter θ heißt natürlicher oder kanonischer Parameter. Durch die Funktion $b(\theta)$ wird sichergestellt, dass sich $f(y|\, \theta)$ normieren lässt und die Ableitungen $b'(\theta)$ und $b''(\theta)$ existieren. Mit ϕ wird der Dispersionsparameter bzw. die Varianz definiert.

Zusammengefasst ist ein individuelles GLM durch den Verteilungstyp der Exponentialfamilie (z.B. Normalverteilung, Binomialverteilung, Poissonverteilung oder Negative Binomialverteilung), durch die Wahl einer Link- oder Responsefunktion und die entsprechenden Kovariablen definiert. Die Linkfunktion ist definiert durch: $\theta_i = \eta_i = x_i'\beta$ und es gilt $g(\mu) = \theta$. Bei einer normalverteilten Zielvariable gilt einfach der natürliche Link $\mu_i = \eta_i = x_i'\beta$, wogegen bei einer binären Zielvariable das Logit-Modell gilt [vgl. Fahrmeir et al., 2007a].

6.2 Umsetzung – GLM in R

In R wird ein GLM mittels folgender Formelgleichung gefittet:

```
glm(formula, family = gaussian, data)
```

Die wichtigsten Argumente in der Modellgleichung sind die Beschreibung des linearen Prädiktors mittels `formula` und die Festlegung der Verteilungsfamilie mit der entsprechenden kanonischen Linkfunktion durch das Argument `family`. Standardmäßig wird ein normalverteiltes Regressionsmodell, mit der Verteilung `gaussian` und der entsprechenden Linkfunktion „`identity`" verwendet.

Bei der Analyse wird in dieser Arbeit ein normalverteiltes lineares Modell mit einem quasibinomialverteiltem Modell verglichen. Bei der Response-Variable `Jaccard` handelt es sich nämlich nicht um eine binomialverteilte Größe, allerdings ist die Zielgröße auf das Intervall [0, 1] beschränkt, da der Jaccard-Index nur Werte in diesem Intervall annehmen kann. Es bietet sich daher an, ein quasibinomialverteiltes Modell als Vergleich zum normalen linearen Modell zu verwenden.

6.3 Struktur der Parameter

Vor der Durchführung der generalisierten Regression ist es wichtig, sich zu überlegen, wie man die Struktur der Parameter im Modell wählt. Kategorialen Variablen wie z.B. die Richtungsparameter (`d1` und `d2`), die die Ausprägungen „`down`", „`up`" oder „`updown`" annehmen können, sollten als Faktoren betrachtet werden. Dabei wird der Variablen intern eine Nummer zugeordnet, nach außen hin wird der Faktor aber durch den Namen repräsentiert [vgl. Ligges, 2008]. Ebenso sollte `artdata` als Faktor-Variable betrachtet werden. Im Gegensatz dazu ist der Jaccard nicht als Faktor sondern als numerische Variable zu betrachten, da er keine kategorialen, sondern numerische Ausprägungen annimmt. Die übrigen Parameter könnte man anstatt als Faktoren eventuell auch als numerische Variablen ins Modell mit aufnehmen. Ob dies sinnvoll ist, erkennt man nach der Auswertung der Modellparameter. Lassen diese für die einzelnen Parameterwerte einen Trend erkennen, könnte es durchaus passend sein, die Variablen als numerisch zu betrachten. Zunächst werden allerdings alle Parameter, bis auf den `Jaccard` als Faktoren dargestellt.

6.4 Interpretation – GLM ohne Interaktionen

Um nun herauszufinden, wie gut der ISA mit den verschiedenen Startwerten der Parameter die drei versteckten Bicluster identifizieren kann, müssen nach Aufstellung des Regressionsmodells in R die Modell-Parameter interpretiert werden. Zunächst wird ein GLM ohne Interaktionen zwischen den Parametern betrachtet.

6.4.1 Modell mit Normalverteilung

Wie bereits erwähnt, werden ein normalverteiltes und ein quasibinomialverteiltes Modell miteinander verglichen. Beim normalverteilten Modell mit Identitätslink wird folgende Modellgleichung betrachtet [siehe Anhang D.4]:

```
GLM_Jaccard <- glm(Jaccard ~ artdata + count + sparsity + Row +
                   Col + d1 + d2 + cl_iterate + cl_unique,
                   data = gesamt_dataframe)
```

Es muss nichts weiter angegeben werden als das Argument `formular` und der data frame. Bei den Parametern handelt es sich bei allen, bis auf den Jaccard um Faktoren. Führt man diese Funktion aus, erhält man über `summary(GLM_Jaccard)` einen R-Output zum Regressionsmodell. Der entsprechende Output befindet sich in Anhang E.1.

Die Interpretation der Modell-Parameter gestaltet sich genau wie in einem normalen linearen Regressionsmodell. Es geht dabei um die durchschnittliche Veränderung des Regressands im Vergleich zur Referenzkategorie unter festgehaltenen übrigen Variablen. Unter `Coefficients` stehen aufgelistet alle Parameter mit ihren unterschiedlichen Startwerten, wobei die fehlenden Werte die Referenzkategorien bilden. Auf den ersten Blick ist so gut wie alles signifikant, was aber daran liegt, dass ein sehr großer data frame verwendet wurde. Daher wird im Folgenden nicht von Signifikanz, sondern von Relevanz gesprochen.

Wie bereits zu vermuten, spielen die Grenzparameter Row und Col eine entscheidende Rolle für den ISA. `Row` und `Col` sind klar als relevante Einflussgrößen auf den Jaccard interpretierbar. Im Gegensatz zur Referenzkategorie `Row1.5`, verbessern sich mit `Row2` und

Row2.5 die Jaccard-Ergebnisse um durchschnittlich etwa 0,022 bzw. 0,020. Bei Row3 allerdings ist ein negativer durchschnittlicher Einfluss von ca. -0,009 auf den Jaccard erkennbar. Interpretiert werden kann das so, dass der ISA für zu hohe Grenzparameter keine Bicluster bzw. nicht die gesuchten Bicluster finden kann. Für den Grenzparameter Col ist durchgehend ein negativer durchschnittlicher Einfluss auf den Jaccard erkennbar, was aber vermutlich auch an der Wahl der Daten liegen kann. Was aber genau wie beim Grenzparameter Row, deutlich zu erkennen ist, ist der klare negative durchschnittliche Einfluss von Col3 auf den Jaccard. Der Einfluss liegt hier bei ca. -0,477.

Ebenso ist der Parameter artdata als sehr wichtig zu betrachten. An den negativen Vorzeichen der Modell-Parameter für die Datensätze artdata4, artdata5 und artdata6 erkennt man, dass diese den Jaccard bei festgehaltenen anderen Variablen, im Gegensatz zur Referenzkategorie (artdata1), im Durchschnitt um ca. -0,205, -0,204 und -0,194 verschlechtern. Begründet werden kann das durch die Wahl der Mittelwerte in den drei versteckten Biclustern. Dadurch, dass sich die Mittelwerte in diesen Biclustern zu nah am globalen Mittelwert $\mu = 0$ befinden, kann der ISA nicht mehr so gut unterscheiden, was die gesuchten Bicluster sind und was nicht. Im Gegensatz dazu hat artdata2 einen vernachlässigbaren negativen Einfluss auf den Jaccard, was daran zu erkennen ist, dass der Parameter nicht signifikant ist. Artdata3 hat wieder einen leicht positiven durchschnittlichen Einfluss auf den Jaccard und ist demnach genau wie artdata4, artdata5 und artdata6 als relevant zu betrachten.

Weitere wichtige Komponenten sind die beiden Richtungsparameter d1 und d2. Bei den Parametern für die Zeilen, bzw. für die Spalten handelt es sich bei den Referenzkategorien in beiden Fällen um die Richtung „down“. Der Parameter d1up hat im Gegensatz zur Referenzkategegorie einen negativen durchschnittlichen Einfluss von ca. -0,001 auf den Jaccard. Der Einfluss ist hier allerdings nicht als relevant anzusehen. Dagegen hat d1updown einen relevanten positiven durchschnittlichen Einfluss von ca. 0,124 auf den Jaccard. Für d2up und d2updown ergeben die Modell-Parameter in beiden Fällen mit ca. 0,011 und 0,131 einen relevanten positiven durchschnittlichen Einfluss auf den Jaccard. Die besten Ergebnisse erzielt der ISA mit der Wahl von „updown“ für beide Richtungsparameter, denn dabei gelangen sowohl hochregulierte als auch herunterregulierte Gene in die Bicluster.

Betrachtet man die Koeffizienten für die Korrelationslimits, sind cl_unique und cl_iterate klar als ebenfalls relevante Parameter einzustufen. Bei cl_unique0.5 und cl_unique0.9 kommt im Vergleich zur Referenzkategorie cl_unique0.1 mit ca. -0,032 und -0,018 ein klarer negativer durchschnittlicher Einfluss auf den Jaccard zustande. Man kann das so begründen, dass mit zunehmendem Korrelationslimit für die Entfernung von gleichen bzw ähnlichen Biclustern möglicherweise auch gesuchte Bicluster entfernt werden. Dennoch ist es nicht sinnvoll, das Korrelationslimit möglichst niedrig zu halten, da dann viele zusätzliche, evtl doppelte Bicluster gefunden werden.
Cl_iterate0.5 und cl_iterate0.9 mit Referenzkategorie cl_iterate0.1 haben beide mit ca. 0,089 und 0,193 einen klaren relevanten positiven durchschnittlichen Einfluss auf den Jaccard. Das liegt daran, dass je höher das Korrelationslimit bei den ISA-Iterationen ist, desto genauer wird bei der Bildung der Bicluster vorgegangen.

Für den eher weniger wichtigen Parameter sparsity ist mit der Parameterwahl sparsity500 ein deutlich relevanter positiver Einfluss erkennbar. Im Gegensatz zur Referenzkategorie sparsity100, verbessert sich mit der Parameterwahl 500 der Jaccard um durchschnittlich ca. 0,007. Wählt man jedoch sparsity mit sparsity900 sehr groß, verschlechtert sich der Jaccard durchschnittlich um ca. -0,003. Man kann dabei aber nicht von einer klaren Signifikanz sprechen. Wählt man also den Parameter sparsity zu groß, hat das negative Auswirkungen auf den Jaccard.

Am wenigsten relevant ist der Parameter count. Count150 ist im Vergleich zur Referenzkategorie count100 nur leicht signifikant und count50 im Vergleich überhaupt nicht signifikant. Mit count150 liegt zwar ein negativer durchschnittlicher Einfluss von ca. -0,003 auf den Jaccard vor, doch dieser ist sehr schwach. Demnach kann man sagen, dass für die Anwendung des ISA der Parameter count so gut wie keine Rolle spielt.

6.4.2 Modell mit Quasibinomialverteilung

Hier gilt für die Modellgleichung [siehe Anhang D.4]:

```
GLM_Jaccard2 <- glm(Jaccard ~ artdata + count + sparsity + Row +
                    Col + d1 + d2 + cl_iterate + cl_unique,
                    family = quasibinomial(link = „logit"),
                    data = gesamt_dataframe)
```

Auch hier handelt es sich wieder bei allen Parametern – bis auf den Jaccard – um Faktoren. Für ein quasibinomialverteiltes Regressionsmodell muss für die Linkfunktion der „`logit`"-Link verwendet werden. Das Logit-Modell hat folgenden Ansatz:

$$\frac{\pi_i}{1-\pi_i} = \frac{\mathrm{P}(y_i = 1|\, x_i)}{\mathrm{P}(y_i = 0|\, x_i)} = \exp(\beta_0) \cdot \exp(x_{i1}\beta_1) \cdot \ldots \cdot \exp(x_{ik}\beta_k)\,,$$

wobei π_i die Wahrscheinlichkeit ist, dass der Jaccard ansteigt und 1-π_i die Gegenwahrscheinlichkeit, dass er nicht ansteigt. Mit $\frac{\pi_i}{1-\pi_i}$ werden die Chancen (odds) berechnet.

Erhöht man nun z.B. x_{i1} auf $x_{i1} + 1$, gilt für das Verhältnis der Chancen

$$\frac{\mathrm{P}(y_i = 1|\, x_{i1}, \ldots)}{\mathrm{P}(y_i = 0|\, x_{i1}, \ldots)} \Big/ \frac{\mathrm{P}(y_i = 1|\, x_{i1} + 1, \ldots)}{\mathrm{P}(y_i = 0|\, x_{i1} + 1, \ldots)} = \exp(\beta_1).$$

Die Chance, dass der Jaccard ansteigt, verändert sich im Vergleich zur Referenzkategorie multiplikativ um $\exp(\beta_1)$.

Betrachtet man den dazugehörigen R-Output [siehe Anhang D.3], sieht man, dass die Vorzeichen der Modellparameter, außer beim Intercept, genauso sind wie im Modell mit Normalverteilung. Auch die Parameter, die sich als signifikant erweisen, und die Referenzkategorien stimmen überein. Die Koeffizienten der beiden Modelle sind gemäß der Modellinterpretation zwar nur ähnlich, was aber daran liegt, dass in einem Logit-Modell die Parameter als Chancen interpretiert werden. Also haben die Modellparameter im Grunde den gleichen tendenziellen Einfluss wie im Modell mit Normalverteilung. Wegen der Übereinstimmung der Ergebnisse wird im Folgenden nur auf den Parameter mit dem größten Einfluss und auf den mit dem geringsten Einfluss näher eingegangen.

Die Grenzparameter für die Zeilen und die Spalten sind auch hier wieder von großer Wichtigkeit. Im Vergleich zur Referenzkategorie Row1.5 hat `Row` mit zunehmendem Parameterwert, eine absteigende Tendenz in der Chance auf einen Anstieg des Jaccard. Für Row2 und

Row2.5 und Row3 steigen die Chancen multiplikativ um ca. 1,137 und 1,128; für Row3 sinken sie multiplikativ um den Faktor 0,948.
Für den Grenzparameter `Col` haben die Parameter `Col2`, `Col2.5` und `Col3` einen ansteigend negativen multiplikativen Einfluss auf den Jaccard im Gegensatz zu `Col1.5`. Die Chance auf einen Anstieg des Jaccard verringert sich jeweils multiplikativ um ca. 0,948, 0,651 und 0,030.

Auch in diesem Modell ist `count` der am wenigsten relevante Parameter, wie man am Output erkennen kann. Bei `count150` sinkt die Chance auf einen Anstieg des Jaccard um den Faktor 0,983 im Vergleich zu `count100`. `Count50` hingegen hat keinen relevanten Einfluss. Die Chance auf einen Anstieg nimmt nur um 0,004 zu. Trotz der hohen Datendichte hat sich der Parameter nicht als klar relevant herausgestellt.

6.5 Interpretation – GLM mit Interaktionen

Im Weiteren wird noch untersucht, ob möglicherweise Interaktionen zwischen Variablen einen Einfluss auf den Jaccard haben. Dabei werden allerdings nur Interaktionen zwischen zwei Variablen betrachtet.

6.5.1 Modell mit Normalverteilung

Es erfolgt wieder der Vergleich zwischen dem normalverteilten und dem quasibinomialverteilten Modell. Mit dem Operator „`.^2`“ werden zusätzlich zu den Parametern, alle Kombinationen aus Zweier-Interaktionen in das Modell mitaufgenommen (z.B. `artdata : count`, `artdata : sparsity` usw.).

```
GLM_Jaccard3 <- glm(Jaccard ~ .^2, data = gesamt_dataframe)
```

Nach der Durchführung dieser Regression in Anhang D.4, erhält man einen sehr großen R-Output, welcher sich in Anhang E.3 befindet.

Bei Betrachtung des R-Outputs fällt auf, dass so gut wie alle Interaktionen mit den Grenzparametern (`Row` und `Col`), den Richtungsparametern (`d1`, `d2`) und den Korrelationslimits (`cl_iterate`, `cl_unique`) für den ISA als relevant anzusehen sind.

Dagegen sind fast alle Interaktionen des Parameters artdata mit den Parametern count und sparsity kaum relevant. Es bestätigen sich also die vorhergehenden Analysen der Koeffizienten im Modell ohne Interaktionen. Die Einstellungen der Parameter `count` und `sparsity`, in Bezug zum Datensatz, sind für das Erhalten von möglichst guten Bicluster-Ergebnissen nicht sehr von Bedeutung. Ebenso fallen die Interaktionen zwischen `count` und `sparsity` kaum ins Gewicht.

6.5.2 Modell mit Quasibinomialverteilung

Das quasibinomialverteilte Modell ist hinsichtlich des Arguments formular genauso aufgebaut wie das normalverteilten Modell und sieht wie folgt aus [siehe Anhang D.4]:

```
GLM_Jaccard4 <- glm(Jaccard~.^2, family = quasibinomial(link =
                    „logit"), data = gesamt_dataframe)
```

Die Modellparameter im dazugehörigen R-Output in Anhang E.4 weisen wieder ein sehr ähnliches Verhalten zum Modell mit Normalverteilung auf.

Die Wichtigkeit der Parameter `Row`, `Col`, `d1`, `d2`, `cl_iterate` und `cl_unique` kann auch in diesem R-Output wieder herausgelesen werden. Die meisten Interaktionen mit diesen Parametern weisen eine hohe Relevanz für die Parameterwahl beim ISA auf.

Als weniger wichtig anzusehen sind, ebenso wie im Modell mit Normalverteilung, die Interaktionen der Parameter `count` und `sparsity` mit `artdata`, sowie die Interaktionen zwischen den beiden Parametern.

6.6 Modell mit numerischen Parametern

In Abschnitt 6.3 wurde kurz drauf hingewiesen, dass man alle Parameter bis auf die kategorialen Parameter artdata, d1 und d2 auch als numerisch betrachten könnte. Um nun herauszufinden, ob ein Modell mit numerischen Parametern besser ist als eines mit Faktoren, wird mit einem linearen Modell (LM) eine Regressionsanalyse durchgeführt. Da das hier nur als kleine Anmerkung gedacht ist, werden lediglich die beiden Bestimmtheitsmaße betrachtet und verglichen. Das Bestimmtheitsmaß gibt an, wie gut das Modell zu den Daten passt und

nimmt Werte im Intervall [0, 1] an. Je höher der Wert, desto besser der Modell-Fit. Die dazugehörigen Outputs befinden sich in Anhang E.5 und E.6 und der R-Code in Anhang D.5.

Das Bestimmtheitsmaß R^2 liegt für das Modell mit numerischen Parametern bei ca. 0.421 und beim Modell mit Faktoren bei ca. 0,5. Demnach ist das Modell mit numerischen Parametern als das schlechtere einzustufen, da der Wert etwas niedriger ist. Für die Verwendung von numerischen Parametern, müssten die Abstände der Koeffizienten zwischen den Wertabstufungen gleich sein, was aber nicht der Fall ist, wie man am LM mit Faktoren sieht. Demnach sind Faktoren nicht geeignet, da zwischen den einzelnen Parameterwerten kein linearer Zusammenhang besteht.

6.7 Residualanalyse für Modell ohne Interaktionen

Da beide Modelle im Grunde einen ähnlichen Output liefern, werden nun die Residuen analysiert um den Modell-Fit hinsichtlich der Verteilung zu überprüfen. Es wird anhand der Residuen untersucht, ob die Annahmen der Normalverteilung (Standardmodell) erfüllt, bzw. approximativ erfüllt sind. Wichtige Residualplots sind der Plot der beobachteten Residuen gegen die geschätzten Residuen und der Normal-QQ-Plot [vgl. Fahrmeir et al., 2007b]. Der R-Code für die Residualanalyse befindet sich in Anhang D.5.

6.7.1 Plot – beobachtete Residuen gegen geschätzte Residuen

Zunächst wird die Grafik der beobachteten Residuen gegen die geschätzten (gefitteten) Residuen für das normalverteilte Modell betrachtet. In R funktioniert das mit folgender Funktion:

```
plot(predict(GLM_Jaccard,type="response"), residuals(GLM_Jaccard,
     type = "pearson"), main = "Residuen gegen Fitted",
     xlab = "gefittete Werte", ylab = "Pearson-Residuen")
abline(0,0, col = "red", lty = 3)
```

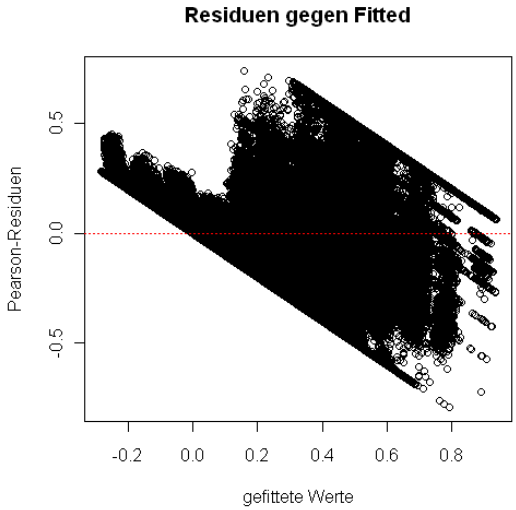

Abbildung 9 : Plot – Residuen gegen Fitted für normalverteiltes Modell.

Die Daten im Plot weisen eine deutliche Struktur auf, was aber im Idealfall nicht sein sollte. Für die Annahme einer Normalverteilung sollten die Varianzen der Residuen gleichmäßig um Null streuen (rote gestrichelte Linie), was hier allerdings nicht erfüllt ist. Es liegt also Heteroskedastizität [vgl. Fahrmeir et al., 2007b] vor.

Zum Vergleich derselbe Plot für das quasibinomialverteilte Modell:

```
plot(predict(GLM_Jaccard2, type="response"), residuals(GLM_Jaccard2,
     type="pearson"), main = "Residuen gegen Fitted",
     xlab="gefittete Werte", ylab="Pearson-Residuen")
abline(0,0, col = "red", lty = 3)
```

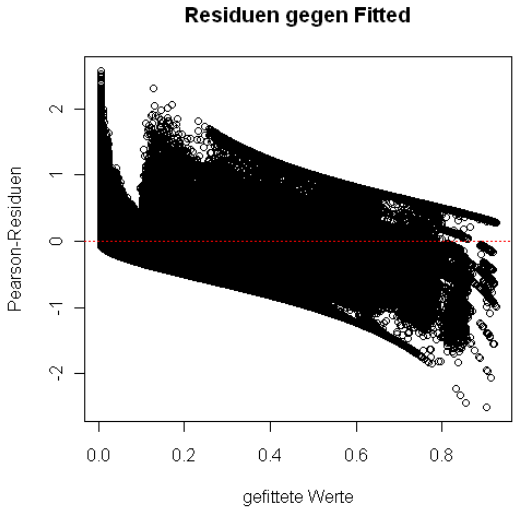

Abbildung 10 : Plot – Residuen gegen Fitted für quasibinomialverteiltes Modell.

Auch hier weisen die Daten eine Struktur auf (Heteroskedastizität), diese ist aber nicht so stark wie im Modell mit Normalverteilung. Demnach würde man bei Betrachtung der Residualplots annehmen, dass ein quasibinomialverteiltes Modell etwas besser zu den Daten passt als ein normalverteiltes.

6.7.2 Normal-QQ-Plot

Neben dem Residualplot gibt es noch den Normal-QQ-Plot, der anzeigt wie gut das Modell einer Normalverteilung folgt. Für das Modell mit Annahme einer Normalverteilung lautet die R-Funktion des Normal-QQ-Plots wie folgt:

```
qqnorm(residuals(GLM_Jaccard))
qqline(residuals(GLM_Jaccard), col = "red", lty = 3)
```

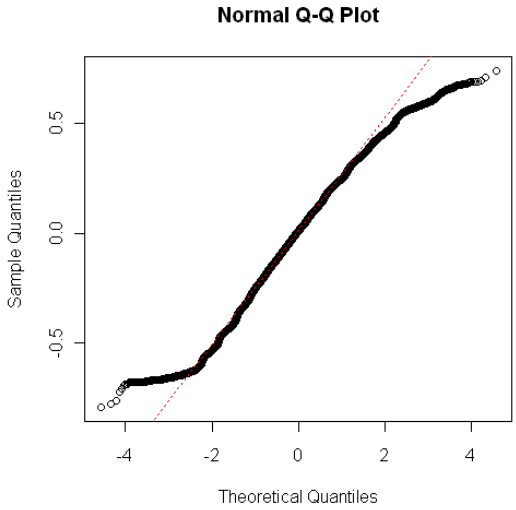

Abbildung 11: Normal-QQ-Plot für das normalverteilte Modell.

Geplottet werden die theoretischen Quantile gegen die empirischen Quantile. Die Grafik zeigt, dass die Datenpunkte bis auf einige wenige größtenteils auf der roten gestrichelten Linie liegen. Daher erfüllt das Modell die Annahme einer Normalverteilung nur annähernd. Doch aufgrund der enorm hohen Anzahl an Daten kann gemäß dem zentralen Grenzwertsatz von einer Normalverteilungsannahme ausgegangen werden [vgl. Fahrmeir et al., 2007b].

Auch hier wieder im Vergleich der Normal-QQ-Plot für das quasibinomialverteilte Modell:

```
qqnorm(residuals(GLM_Jaccard2))
qqline(residuals(GLM_Jaccard2), col = "red", lty = 3)
```

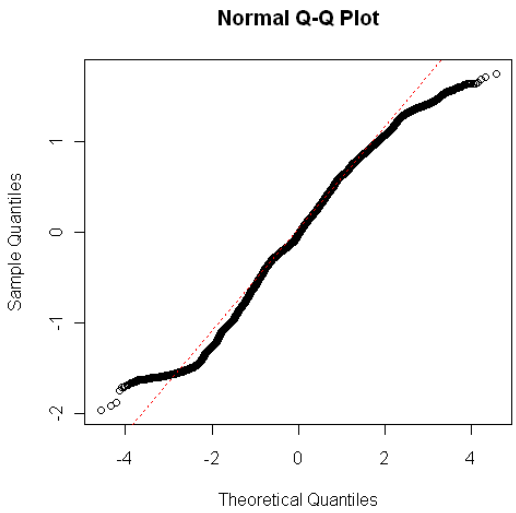

Abbildung 12: Normal-QQ-Plot für das quasibinomialverteilte Modell.

Wie man sieht sind die Normal-QQ-Plots für beide Modelle nahezu identisch. Die Annahme, dass sich für sehr große Datensätze die Verteilungen durch eine Normalverteilung approximieren lassen, gilt auch für das quasibinomialverteilte Modell. Demnach erfüllen beide die Normalverteilungsannahme gleichermaßen gut.

7 Zusammenfassung und Ausblick

Im letzten Kapitel werden nochmal die wichtigsten Ergebnisse aus den ISA-Simulationen in einer kleinen Zusammenfassung dargestellt. Nach dem Vergleich der beiden verwendeten Modelle und der Aufstellung von Gesetzmäßigkeiten zur Parameterwahl werden in einem kleinen Ausblick einige weitere Möglichkeiten präsentiert, anhand derer man den Iterative Signature Algorithmus noch zusätzlich analysieren könnte.

7.1 Vergleich der beiden Modelle

Die Analyse des ISA mittels zweier verschiedener generalisierter Regressionsmodelle hat ergeben, dass beide Methoden von der Adäquatheit in etwa gleich sind. Sowie für das normalverteilte Modell, als auch für das quasibinomialverteilte Modell wurden bis auf den Jaccard alle Parameter als Faktoren kodiert. Durch eine Gegenüberstellung der R-Outputs zu den GLM´s hat sich gezeigt, dass die Modellparameter für das Modell ohne Interaktionen und für das Modell mit Interaktionen den gleichen tendenziellen Einfluss auf den Jaccard-Index haben. Auch eine kleine Residualanalyse zur Überprüfung des Modell-Fits und der Normalverteilungsannahme hat bei Betrachtung der Plots nahezu das gleiche Ergebnis geliefert. Die zwei Modelle sind also im Grunde ganz gut für die Analyse des großen data frames geeignet.

7.2 Gesetzmäßigkeiten zur Parameterwahl

Mit Hilfe der aus den ISA-Simulationen resultierten Ergebnisse, kann man schon einiges über eine geeignete Parameterwahl beim ISA aussagen. In der nachfolgenden Tabelle werden die für den ISA empfohlenen Parameterwerte aufgelistet. Diese Gesetzmäßigkeiten haben sich aus den Analysen, unter Verwendung mehrerer verschiedener Parametersettings, ergeben:

Parameter	empfohlener Wert
Row	Row2, Row2.5
Col	Col2
d1	d1updown
d2	d2updown
cl_iterate	cl_iterate0.9
cl_unique	cl_unique0.9
sparsity	sparsity500
count	count100

Tabelle 4: Übersicht über die empfohlenen Parameterwerte.

Am allerwichtigsten ist es, die Grenzparameter für die Zeilen und für die Spalten weder zu großzügig noch zu streng einzustellen. Die Analysen haben ergeben, dass sich für jeden ISA-Lauf die besten Ergebnisse mit den Werten 2 und 2,5 ergeben haben. Waren die beiden Parameter jedoch auf 3 eingestellt, konnten keine Ergebnisse zurückgeliefert werden. Für Grenzparameter, welche "Scores" beibehalten sollen die 3mal so groß sind wie der Mittelwert im gesamten Datensatz, kann der ISA keine Bicluster mehr finden. Dadurch bestätigen sich auch die Default-Werte für die Grenzparameter in R. Es wird also empfohlen die Werte 2,5 oder 2 als Auswahlkriterium für die Zeilen zu verwenden und für die Spalten den Parameterwert 2.

Als ebenso wichtig hat sich die Wahl der Richtungsparameter für die „Scores“ in den Zeilen bzw. in den Spalten der Bicluster erwiesen. Es ist ratsam für beide Parameter die Richtung „updown“ zu wählen, da hier sowohl Werte, die deutlich über als auch unter dem Mittelwert liegen, in die Bicluster gelangen. Für die Wahl von „up“ und „down“ ist die Auswahl der Werte begrenzter.

Für die ISA-Iterationen ist es von sehr großer Wichtigkeit, das Korrelationslimit möglichst optimal einzustellen. Je näher das Korrelationslimit bei Eins ist, desto genauer wird beim Konvergieren des Algorithmus und bei der Bildung der Bicluster vorgegangen. Am besten ist es stets 0,9 zu verwenden.

Auch das Korrelationslimit für die Entfernung gleicher bzw. zu ähnlicher Bicluster hat sich als klar relevant für den ISA-Lauf erwiesen. Mit zunehmend hohem Limit filtert der ISA immer genauer und stellt sicher, dass keine Duplikate zurückgeliefert werden. Natürlich kann dies ebenso negative Auswirkungen auf den Jaccard haben, da dabei möglicherweise auch wichtige gesuchte Bicluster ausgesiebt werden könnten. Trotzdem ergaben sich die besten Ergebnisse mit einem Limit von 0,9.

Bei der Wahl der Anzahl von Einsen in den Inputvektoren (`sparsity`), war lediglich für sparsity500 eine deutliche Relevanz für den ISA erkennbar. Deshalb kann geschlossen werden, dass man mit einem Wert von 500, sehr gute Ergebnisse erzielt.

Der Parameter `count` hat sich als der unwichtigste Parameter erwiesen und somit ist es ohne Bedeutung, mit wie vielen Inputvektoren der Iterative Signature Algorithmus gestartet wird. Man kann für die Variable `count` einfach den Standardwert 100 einsetzen.

7.3 Ausblick

In dieser Arbeit wurde einiges unternommen um den Iterative Signature Algorithmus hinsichtlich einer optimalen Parameterwahl zu untersuchen. Doch aufgrund der begrenzten Arbeitszeit konnten nicht alle Möglichkeiten ausgeschöpft werden. Man könnte z.B. in weiteren Analysen die Parameterwerte für den ISA in noch viel feinere Abstufungen unterteilen. Wie man aber an den dargestellten Analysen sieht, wurde trotz der recht großzügigen eingestellten Parametersettings ohnehin ein schon ziemlich großer Datensatz mit 6×34992 verwendet. Desweiteren könnte man zusätzlich zu den `row.seeds` als Inputvektoren auch die `col.seeds` oder sogar beide gleichzeitig für die ISA-Iterationen einsetzen. Zusätzlich muss angemerkt werden, dass in dieser Arbeit nur die für den ISA wichtigsten Parameter unterschiedlich eingestellt wurden. Es gibt noch einige andere Parameter die man zusätzlich in die Analysen miteinbauen könnte. Zudem ist es möglich zusätzlich zu den Zweier-Interaktionen, Wechselwirkungen zwischen mehr als zwei Variablen in das GLM mit aufzunehmen. Doch trotz der vielen weiteren hier nicht berücksichtigten Optionen, konnten mit den für diese Bachelorarbeit ausgewählten Untersuchungen bereits interessante und aufschlussreiche Resultate erzielt werden.

A Literaturverzeichnis

S. Bergmann, J. Ihmels and N. Barkai. „Iterative signature algorithm for the analysis of large-scale gene expression data". *Physical Review* E 67, 031902, March 2003.

Y. Cheng und G. M. Church. "Biclustering of Expression Data". *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology,* 1, 2000.

Gábor Csárdi (Oct 26, 2009a). isa2: The Iterative Signature Algorithm. R package version 0.2.1. URL http://CRAN.R-project.org/package=isa2.

G. Csárdi . „The Iterative Signature Algorithm". *Computational Biology Group*, October 5, 2009b.

C. d'Enfert and B. Hube. "Candida, Comparative und Functional Genomics". Great Britain, 2007.

L. Fahrmeir, A. Hamerle, G. Tutz. „Multivariate Statistische Verfahren". Zweite, überarbeitete Auflage, Berlin, 1996.

L. Fahrmeir, T. Kneib, S. Lang. „Regression – Modelle, Methoden und Anwendungen". Zweite Auflage, Berlin Heidelberg, 2007a.

L. Fahrmeir, R. Künstler, I. Pigeot und G. Tutz. „Statistik – Der Weg zur Datenanalyse". Sechste, überarbeitete Auflage, Berlin Heidelberg, 2007b.

J. A. Hartigan. "Direct Clustering of a Data Matrix". *Journal of the American Statistical Association*, 67(337), 1972.

P. Jaccard. „Étude comparative de la distribution florale dans une portion des Alpes et des Jura". *Bulletin de la Société Vaudoise des Sciences Naturelles* 37, 1901.

Sebastian Kaiser, Rodrigo Santamaria, Martin Sill, Roberto Theron, Luis Quintales and Friedrich Leisch, 2009. *biclust: BiCluster Algorithms*. R package version 0.9.1. URL http://CRAN.R-project.org/package=biclust.

Y. Kluger, R. Basri, J. T. Chang, M. Gerstein . „Spectral Biclustering of Microarray Data: Coclustering Genes and Conditions". *Genome Research*, 13, 2003.

L. Lazzeroni, A. Owen . „Plaid Models for Gene Expression Data". Statistica Sinica, 12, 2002.

U. Ligges. „Programmieren mit R". Dritte, überarbeitete und erweiterte Auflage, Berlin Heidelberg, 2008.

S. C. Madeira and A. L. Oliveira. "Biclustering Algorithms for Biological Data Analysis: A Survey". INESC-ID TEC. REP. 1/2004, JAN 2004.

T. Murali, S. Kasif. "Extracting Conserved Gene Expression Motifs from Gene Expression Data". *Pacific Symposium on Biocomputing*, 8, 2003.

R Development Core Team (2010). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

B Abbildungsverzeichnis

C Tabellenverzeichnis

D R-Code

D.1 Datengenerierung_Heatmap.r

Als Beispiel ist hier der R-Code für den Datensatz1 dargestellt. Wie die übrigen Datensätze mit den versteckten Biclustern aussehen, wird als Kommentar beschrieben. Für Datensatz4, Datensatz5 und Datensatz6 gilt der gleiche R-Code, aber mit den Mittelwerten 1, 2 und 3.

```
## Datengenerierung
artdata <- matrix(rnorm(500000), 1000, 500)

## Bicluster verstecken
x <- sample(1:1000, 150)
y <- sample(1:500, 150)
x1 <- 1:1000 %in% x[1:50]
y1 <- 1:500 %in% y[1:50]
x2 <- 1:1000 %in% x[51:100]
y2 <- 1:500 %in% y[51:100]
x3 <- 1:1000 %in% x[101:150]
y3 <- 1:500 %in% y[101:150]

# Zweiter Datensatz: Alle Mittelwerte negativ
# Dritter Datensatz: Zwei Mittelwerte positiv und einer negativ
artdata[x1,y1] <- rnorm(2500, 5, 0.1)
artdata[x2,y2] <- rnorm(2500, 3, 0.1)
artdata[x3,y3] <- rnorm(2500, 4, 0.1)

## Definieren der versteckten Bicluster-Ergebnisse
artres <- BiclustResult(list(), as.matrix(cbind(x1,x2,x3)),
          t(as.matrix(cbind(y1,y2,y3))), 3, list())

## Betrachtung der Daten und Bicluster mit drawHeatmap2()
drawHeatmap2(artdata, bicResult = NULL, number = NA,
             plotAll = FALSE)
drawHeatmap2(artdata, bicResult = artres, number = NA,
             plotAll = TRUE)
```

D.2 ISA-Schleifen-1.r

Die fünf anderen ISA-Schleifen werden nach dem gleichen Prinzip durchgeführt. Es ändern sich lediglich die verwendeten Datensätze bei der Datengenerierung und das Ergebnis muss unter dem passenden Namen abgespeichert werden.

```
### ISA-Schleifen-1
## packages laden
set.seed(1234)
library(isa2)
library(biclust)

## data frame generieren
res <- matrix(0, 34992, 10)

## Parametersettings
# Parameter für generate.seeds()
count <- seq(50, 150, by = 50)
sparsity <-seq(100, 900, by = 400)

# Parameter für isa.iterate()
thr.row <- seq(1.5, 3, by = 0.5)
thr.col <- seq(1.5, 3, by = 0.5)
d1 <- c("up", "down", "updown")
d2 <- c("up", "down", "updown")
cl.iterate <- seq(0.1, 0.9, by = 0.4)

# Parameter für isa.unique()
cl.unique <- seq(0.1, 0.9, by = 0.4)

## Laufindex z auf 1, um erste Zeile des data frames zu definieren
z <- 1

## 3 Durchläufe pro Datensatz
for(i in 1:3)
{
```

```
# Datengenerierung
artdata <- matrix(rnorm(500000), 1000, 500)

# Bicluster verstecken
x <- sample(1:1000, 150)
y <- sample(1:500, 150)

x1 <- 1:1000 %in% x[1:50]
y1 <- 1:500 %in% y[1:50]

x2 <- 1:1000 %in% x[51:100]
y2 <- 1:500 %in% y[51:100]

x3 <- 1:1000 %in% x[101:150]
y3 <- 1:500 %in% y[101:150]

# Zweiter Datensatz: Alle Mittelwerte negativ
# Dritter Datensatz: Zwei Mittelwerte positiv und einer negativ
artdata[x1,y1] <- rnorm(2500, 5, 0.1)
artdata[x2,y2] <- rnorm(2500, 3, 0.1)
artdata[x3,y3] <- rnorm(2500, 4, 0.1)

## Definieren der versteckten Bicluster-Ergebnisse
artres <- BiclustResult(list(), as.matrix(cbind(x1,x2,x3)),
          t(as.matrix(cbind(y1,y2,y3))), 3, list())

## Datensatz mit isa.normalize() standardisieren
isanorm <- isa.normalize(artdata, prenormalize = FALSE)

for (j in 1:length(count))
{
for (k in 1:length(sparsity))
{

## Inputvektoren mit generate.seeds() generieren
row.seeds <- generate.seeds (length = nrow(artdata),
              count = count[j], method = c("uni"),
             sparsity = sparsity[k])
```

```
for(l in 1: length(thr.row))
{
for(m in 1:length(thr.col))
{
for(n in 1:length(d1))
{
for(o in 1:length(d2))
{
for(p in 1:length(cl.iterate))
{

## ISA-Iterationen mit isa.iterate()durchzuführen
isamodules <- isa.iterate(isanorm, row.seeds = row.seeds,
              thr.row = thr.row[l], thr.col = thr.col[m],
              direction = c(d1[n], d2[o]), convergence = "cor",
              cor.limit = cl.iterate[p], oscillation = FALSE,
              maxiter = 100)

for (q in 1:length(cl.unique))
{

## Bicluster mit isa.unique() entfernen
isamodules2 <- isa.unique(isanorm, isamodules,
               cor.limit = cl.unique[q])

## Robustheit mit isa.filter.robust() prüfen
isamodules3 <- isa.filter.robust(artdata, isanorm,
               isamodules2, perms = 1)

## Dokumentation der Simulationsschritte, nach jedem ISA-Durchlauf
print(c(i,j,k,l,m,n,o,p,q))
## Falls kein Bicluster gefunden, Jaccard immer Null
if(ncol(isamodules3$rows) == 0)
{
jaccard <- "0"
}
else
{
```

```
## ISA-Objekt in Biclust-Objekt
Bc <- isa.biclust(isamodules3)
## Vergleich mit Jaccard
jaccard <- jaccardind(artres, Bc)
}

## Zuordnung der Parameterwerte und des Jaccard-Index
res[z,] <- c(i, count[j], sparsity[k], thr.row[l], thr.col[m],
           d1[n], d2[o], cl.iterate[p], cl.unique[q], jaccard)

## Index z um 1 erhöhen
z <- z+1
}
}
}
}
}
}
}
}
}

## Spalten im data frame definieren
colnames(res) <- c("artdata", "count", "sparsity", "Row",              .
"Col", "d1","d2", "cl_iterate", "cl_unique",
                "Jaccard")

## Ausgabe des data frames
res
## Ergebnis der Simulation speichern
save.image("sim-isa1.RData")
```

D.3 ISA-dataframe.r

```
### Simulationsergebnisse zusammenfügen
## Simulation1 laden
load("sim-isa1.RData")

# in richtigen data frame umwandeln
dataframe1 <- data.frame(res)

# Datenstruktur und summary
str(dataframe1)
summary(dataframe1)

# Jaccard von Faktor in numerisch
dataframe1$Jaccard <- as.numeric(as.character(dataframe1$Jaccard))

# Zugehörigkeit der drei Datensätze pro Simulation
dataframe1$artdata <- 1

# Zusammenfassen zu einem großen data frame
gesamt_dataframe <- dataframe1
```

Die gleiche Prozedur wird anschließend für alle weiteren fünf ISA-Simulationen durchgeführt. Der dataframe1 wird unter einer neuen Variable gesamt_dataframe abgespeichert und an diese werden alle data frames mit der Funktion rbind() zeilenweise wie folgt angefügt:

```
## zeilenweises Zusammenfügen der Simulationen
gesamt_dataframe <- rbind(gesamt_dataframe,dataframe2)

## gesamt_dataframe speichern
save.image("gesamt_dataframe.RData")
```

D.4 GLM.r

```
### GLM ohne Interaktionen
## gesamt_dataframe laden
load("gesamt_dataframe.RData")

# artdata von numerisch in Faktor
gesamt_dataframe$artdata <- as.factor(as.numeric(gesamt_dataframe$artdata))

# Datenstruktur und summary
summary(gesamt_dataframe)
str(gesamt_dataframe)

## normalverteiltes Regressionsmodell
# Regression - Zusammenhang zwischen Jaccard und allen Variablen
GLM_Jaccard <- glm(Jaccard ~ artdata + count + sparsity + Row
                   + Col + d1 + d2 + cl_iterate + cl_unique,
                   Family = gaussian(link = "identity"),
                   data = gesamt_dataframe)

## quasibinomialverteiltes Regressionsmodell
# Regression - Zusammenhang zwischen Jaccard und allen Variablen
GLM_Jaccard2 <- glm(Jaccard ~ artdata + count + sparsity + Row
                    + Col + d1 + d2 + cl_iterate + cl_unique,
                    family = quasibinomial(link = "logit"),
                    data = gesamt_dataframe)

### GLM mit Interaktionen
# Um den Output für das normalverteilte Modell mit Interaktionen zu
# erhalten, sollte besser die gespeicherte Datei
# „GaussianInteraktion.RData" geladen werden. Das
# verhindert die lange Laufzeit. Für das quasibinomialverteilte
# Modell entsprechend, die Datei  „QuasiInteraktion.RData" laden.

## normalverteiltes Regressionsmodell
# Regression – Zusammenhang zwischen Jaccard und allen Variablen +
# Zweier-Interaktionen
GLM_Jaccard3 <- glm(Jaccard ~ .^2,
                    family = gaussian(link = "identity"),
                    data = gesamt_dataframe)
```

```
# gespeicherten GLM-Output für normalverteiltes Modell laden
GLM_Interaktion3 <- load("GaussianInteraktion.RData")
GLM_Interaktion3 # [1] "test1"
test1

## quasibinomialverteiltes Regressionsmodell
# Regression – Zusammenhang zwischen Jaccard und allen Variablen +
# Zweier-Interaktionen
GLM_Jaccard4 <- glm(Jaccard ~ .^2,
                    family = quasibinomial(link = "logit"),
                    data = gesamt_dataframe)

# gespeicherten GLM-Output für quasibinomialverteiltes Modell laden
GLM_Interaktion4 <- load("QuasiInteraktion.RData")
GLM_Interaktion4 # [1] "sumtestquasi"
sumtestquasi

### Residualanalyse für Modell ohne Interaktionen
## beobachtete Residuen gegen geschätzte Residuen
plot(predict(GLM_Jaccard, type = "response"),
     residuals(GLM_Jaccard, type = "pearson"),
     main = "Residuen gegen Fitted", xlab = "gefittete Werte",
     ylab = "Pearson-Residuen")
abline(0,0, col = "red", lty = 3)

plot(predict(GLM_Jaccard2, type = "response"),
      residuals(GLM_Jaccard2, type = "pearson"),
      main = "Residuen gegen Fitted", xlab = "gefittete Werte",
      ylab = "Pearson-Residuen")
abline(0,0, col = "red", lty = 3)

## Normal-QQ-Plot
qqnorm(residuals(GLM_Jaccard))
qqline(residuals(GLM_Jaccard), col = "red", lty = 3)

qqnorm(residuals(GLM_Jaccard2))
qqline(residuals(GLM_Jaccard2), col = "red", lty = 3)
```

D.5 LM.r

```
### LM mit Normalverteilung (Vergleich numerisch mit Faktor)
## Parameter als numerische Variablen (bis auf Jaccard und artdata)
gesamt_dataframe$count <- as.numeric(as.character
                          (gesamt_dataframe$count))
gesamt_dataframe$sparsity <- as.numeric(as.character
                             (gesamt_dataframe$sparsity))
gesamt_dataframe$Row <- as.numeric(as.character
                        (gesamt_dataframe$Row))
gesamt_dataframe$Col <- as.numeric(as.character
                        (gesamt_dataframe$Col))
gesamt_dataframe$cl_iterate <- as.numeric(as.character
                               (gesamt_dataframe$cl_iterate))
gesamt_dataframe$cl_unique <- as.numeric(as.character
                              (gesamt_dataframe$cl_unique))

## data frame neu abspeichern
gesamt_dataframe_numerisch <- gesamt_dataframe

## Datenstruktur und summary
summary(gesamt_dataframe_numerisch)
str(gesamt_dataframe_numerisch)

## Regression - Zusammenhang zwischen Jaccard und numerischen
## Variablen
LM_Jaccard <- lm(Jaccard ~ artdata + count + sparsity + Row
               + Col + d1 + d2 + cl_iterate + cl_unique,
               data = gesamt_dataframe_numerisch)

## Regression – Zusammenhang zwischen Jaccard und Faktor-Variablen
LM_Jaccard2 <- lm(Jaccard ~ artdata + count + sparsity + Row
               + Col + d1 + d2 + cl_iterate + cl_unique,
               data = gesamt_dataframe)
```

E R-Output

E.1 GLM_Jaccard

```
Call:
glm(formula = Jaccard ~ artdata + count + sparsity + Row + Col
    + d1 + d2 + cl_iterate + cl_unique,
    family = gaussian(link = "identity"),
    data = gesamt_dataframe)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.7936  -0.1631   0.0108   0.1857   0.7425

Coefficients:
               Estimate Std. Error  t value Pr(>|t|)
(Intercept)    0.4463904  0.0026677  167.334  < 2e-16 ***
artdata2      -0.0005095  0.0018863   -0.270   0.7871
artdata3       0.0119499  0.0018863    6.335 2.38e-10 ***
artdata4      -0.2054187  0.0018863 -108.899  < 2e-16 ***
artdata5      -0.2043665  0.0018863 -108.341  < 2e-16 ***
artdata6      -0.1935800  0.0018863 -102.623  < 2e-16 ***
count150      -0.0029025  0.0013338   -2.176   0.0295 *
count50        0.0006910  0.0013338    0.518   0.6044
sparsity500    0.0067356  0.0013338    5.050 4.43e-07 ***
sparsity900   -0.0028724  0.0013338   -2.153   0.0313 *
Row2           0.0216377  0.0015402   14.049  < 2e-16 ***
Row2.5         0.0201946  0.0015402   13.112  < 2e-16 ***
Row3          -0.0089526  0.0015402   -5.813 6.15e-09 ***
Col2          -0.0113843  0.0015402   -7.392 1.46e-13 ***
Col2.5        -0.0901987  0.0015402  -58.564  < 2e-16 ***
Col3          -0.4774691  0.0015402 -310.009  < 2e-16 ***
d1up          -0.0009377  0.0013338   -0.703   0.4820
d1updown       0.1241771  0.0013338   93.098  < 2e-16 ***
d2up           0.0107882  0.0013338    8.088 6.09e-16 ***
d2updown       0.1313749  0.0013338   98.494  < 2e-16 ***
cl_iterate0.5  0.0893688  0.0013338   67.002  < 2e-16 ***
cl_iterate0.9  0.1934592  0.0013338  145.040  < 2e-16 ***
cl_unique0.5  -0.0315661  0.0013338  -23.666  < 2e-16 ***
cl_unique0.9  -0.0175051  0.0013338  -13.124  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.06225455)

    Null deviance: 26136  on 209951  degrees of freedom
Residual deviance: 13069  on 209928  degrees of freedom
AIC: 12907

Number of Fisher Scoring iterations: 2
```

E.2 GLM_Jaccard2

```
Call:
glm(formula = Jaccard ~ artdata + count + sparsity + Row + Col
    + d1 + d2 + cl_iterate + cl_unique,
    family = quasibinomial(link = "logit"),
    data = gesamt_dataframe)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.96890  -0.34611  -0.02125   0.41535   1.75553

Coefficients:
              Estimate Std. Error  t value Pr(>|t|)
(Intercept)   -0.351189   0.013699  -25.636  < 2e-16 ***
artdata2      -0.002904   0.009705   -0.299   0.7647
artdata3       0.068367   0.009724    7.031 2.06e-12 ***
artdata4      -1.182731   0.010006 -118.205  < 2e-16 ***
artdata5      -1.176153   0.010001 -117.607  < 2e-16 ***
artdata6      -1.109346   0.009951 -111.481  < 2e-16 ***
count150      -0.017287   0.007015   -2.464   0.0137 *
count50        0.004113   0.007012    0.587   0.5575
sparsity500    0.040069   0.007011    5.715 1.10e-08 ***
sparsity900   -0.017121   0.007017   -2.440   0.0147 *
Row2           0.128623   0.008095   15.890  < 2e-16 ***
Row2.5         0.120076   0.008095   14.833  < 2e-16 ***
Row3          -0.053586   0.008120   -6.599 4.15e-11 ***
Col2          -0.053907   0.007223   -7.464 8.45e-14 ***
Col2.5        -0.428637   0.007265  -58.998  < 2e-16 ***
Col3          -3.516429   0.012855 -273.539  < 2e-16 ***
d1up          -0.005633   0.007045   -0.800   0.4239
d1updown       0.727529   0.007046  103.256  < 2e-16 ***
d2up           0.064802   0.007045    9.198  < 2e-16 ***
d2updown       0.770381   0.007059  109.127  < 2e-16 ***
cl_iterate0.5  0.534990   0.007081   75.551  < 2e-16 ***
cl_iterate0.9  1.134273   0.007171  158.175  < 2e-16 ***
cl_unique0.5  -0.187837   0.007017  -26.767  < 2e-16 ***
cl_unique0.9  -0.103862   0.007003  -14.831  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for quasibinomial family taken to be 0.2890842)

    Null deviance: 141598  on 209951  degrees of freedom
Residual deviance:  72284  on 209928  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 6
```

E.3 GLM_Jaccard3

Anzumerken ist hier, dass die Datei `gesamtdaten` im R-Code dem Datensatz `gesamt_dataframe` entspricht.

```
Call:
glm(formula = Jaccard ~ .^2, data = gesamtdaten)

Deviance Residuals:
      Min         1Q     Median         3Q        Max
-0.892821  -0.112215   0.002531   0.112764   0.840790

Coefficients:
                           Estimate Std. Error t value Pr(>|t|)
(Intercept)                3.957e-01  7.517e-03  52.647  < 2e-16 ***
artdata2                   1.552e-02  7.108e-03   2.183 0.029016 *
artdata3                   5.472e-02  7.108e-03   7.698 1.38e-14 ***
artdata4                  -1.749e-01  7.108e-03 -24.608  < 2e-16 ***
artdata5                  -1.512e-01  7.108e-03 -21.276  < 2e-16 ***
artdata6                  -1.202e-01  7.108e-03 -16.918  < 2e-16 ***
count150                  -7.963e-03  5.408e-03  -1.472 0.140925
count50                    4.166e-02  5.408e-03   7.703 1.33e-14 ***
sparsity500                2.223e-03  5.408e-03   0.411 0.681071
sparsity900               -6.898e-04  5.408e-03  -0.128 0.898505
Row2                       2.228e-02  6.101e-03   3.652 0.000261 ***
Row2.5                     3.781e-02  6.101e-03   6.198 5.74e-10 ***
Row3                       1.300e-01  6.101e-03  21.311  < 2e-16 ***
Col2                      -2.940e-02  6.101e-03  -4.819 1.45e-06 ***
Col2.5                    -5.445e-02  6.101e-03  -8.925  < 2e-16 ***
Col3                      -3.085e-01  6.101e-03 -50.557  < 2e-16 ***
d1up                      -3.303e-02  5.408e-03  -6.108 1.01e-09 ***
d1updown                   1.876e-01  5.408e-03  34.698  < 2e-16 ***
d2up                      -1.729e-02  5.408e-03  -3.197 0.001387 **
d2updown                   2.175e-01  5.408e-03  40.225  < 2e-16 ***
cl_iterate0.5              8.498e-02  5.408e-03  15.714  < 2e-16 ***
cl_iterate0.9              2.319e-01  5.408e-03  42.888  < 2e-16 ***
cl_unique0.5              -1.573e-01  5.408e-03 -29.079  < 2e-16 ***
cl_unique0.9              -1.109e-01  5.408e-03 -20.498  < 2e-16 ***
artdata2:count150          1.090e-03  3.994e-03   0.273 0.784862
artdata3:count150          2.302e-03  3.994e-03   0.576 0.564332
artdata4:count150          8.436e-03  3.994e-03   2.112 0.034671 *
artdata5:count150          6.753e-04  3.994e-03   0.169 0.865740
artdata6:count150         -1.283e-02  3.994e-03  -3.213 0.001314 **
artdata2:count50          -4.372e-03  3.994e-03  -1.095 0.273649
artdata3:count50          -3.725e-03  3.994e-03  -0.933 0.350984
artdata4:count50          -8.350e-03  3.994e-03  -2.091 0.036558 *
artdata5:count50          -1.145e-02  3.994e-03  -2.868 0.004134 **
artdata6:count50          -1.198e-02  3.994e-03  -3.000 0.002704 **
artdata2:sparsity500       3.985e-03  3.994e-03   0.998 0.318375
artdata3:sparsity500      -3.974e-03  3.994e-03  -0.995 0.319790
artdata4:sparsity500       1.581e-02  3.994e-03   3.960 7.51e-05 ***
artdata5:sparsity500       2.392e-03  3.994e-03   0.599 0.549327
artdata6:sparsity500       3.076e-03  3.994e-03   0.770 0.441166
artdata2:sparsity900      -1.086e-02  3.994e-03  -2.718 0.006561 **
artdata3:sparsity900      -1.067e-03  3.994e-03  -0.267 0.789379
artdata4:sparsity900       1.189e-02  3.994e-03   2.976 0.002921 **
artdata5:sparsity900      -1.180e-03  3.994e-03  -0.295 0.767746
artdata6:sparsity900      -1.076e-03  3.994e-03  -0.269 0.787635
artdata2:Row2              1.121e-03  4.612e-03   0.243 0.807929
artdata3:Row2             -4.190e-03  4.612e-03  -0.909 0.363593
artdata4:Row2             -1.357e-02  4.612e-03  -2.942 0.003264 **
artdata5:Row2             -1.251e-02  4.612e-03  -2.713 0.006678 **
artdata6:Row2             -1.475e-02  4.612e-03  -3.197 0.001388 **
artdata2:Row2.5            3.820e-03  4.612e-03   0.828 0.407506
artdata3:Row2.5           -1.325e-02  4.612e-03  -2.872 0.004078 **
artdata4:Row2.5           -1.963e-02  4.612e-03  -4.257 2.08e-05 ***
artdata5:Row2.5           -1.926e-02  4.612e-03  -4.176 2.97e-05 ***
artdata6:Row2.5           -3.451e-02  4.612e-03  -7.483 7.29e-14 ***
artdata2:Row3             -3.640e-03  4.612e-03  -0.789 0.429956
```

```
artdata3:Row3             -3.461e-02  4.612e-03  -7.505 6.15e-14 ***
artdata4:Row3             -6.181e-02  4.612e-03 -13.401  < 2e-16 ***
artdata5:Row3             -6.432e-02  4.612e-03 -13.947  < 2e-16 ***
artdata6:Row3             -7.626e-02  4.612e-03 -16.535  < 2e-16 ***
artdata2:Col2              1.113e-03  4.612e-03   0.241 0.809229
artdata3:Col2              5.892e-03  4.612e-03   1.278 0.201423
artdata4:Col2             -7.975e-03  4.612e-03  -1.729 0.083791 .
artdata5:Col2             -7.727e-03  4.612e-03  -1.675 0.093873 .
artdata6:Col2             -1.922e-02  4.612e-03  -4.167 3.08e-05 ***
artdata2:Col2.5           -1.196e-03  4.612e-03  -0.259 0.795427
artdata3:Col2.5           -5.325e-03  4.612e-03  -1.154 0.248301
artdata4:Col2.5           -4.339e-02  4.612e-03  -9.408  < 2e-16 ***
artdata5:Col2.5           -4.695e-02  4.612e-03 -10.181  < 2e-16 ***
artdata6:Col2.5           -4.972e-02  4.612e-03 -10.780  < 2e-16 ***
artdata2:Col3              3.686e-04  4.612e-03   0.080 0.936298
artdata3:Col3             -1.784e-02  4.612e-03  -3.869 0.000109 ***
artdata4:Col3              2.084e-01  4.612e-03  45.187  < 2e-16 ***
artdata5:Col3              2.038e-01  4.612e-03  44.196  < 2e-16 ***
artdata6:Col3              1.868e-01  4.612e-03  40.509  < 2e-16 ***
artdata2:d1up             -1.216e-02  3.994e-03  -3.044 0.002332 **
artdata3:d1up             -7.959e-03  3.994e-03  -1.993 0.046311 *
artdata4:d1up              4.200e-04  3.994e-03   0.105 0.916245
artdata5:d1up             -2.109e-02  3.994e-03  -5.280 1.29e-07 ***
artdata6:d1up             -1.029e-02  3.994e-03  -2.577 0.009973 **
artdata2:d1updown         -1.182e-02  3.994e-03  -2.959 0.003090 **
artdata3:d1updown         -3.473e-02  3.994e-03  -8.694  < 2e-16 ***
artdata4:d1updown         -9.306e-02  3.994e-03 -23.300  < 2e-16 ***
artdata5:d1updown         -9.950e-02  3.994e-03 -24.911  < 2e-16 ***
artdata6:d1updown         -1.113e-01  3.994e-03 -27.875  < 2e-16 ***
artdata2:d2up             -6.947e-03  3.994e-03  -1.739 0.081985 .
artdata3:d2up             -2.217e-03  3.994e-03  -0.555 0.578792
artdata4:d2up              2.549e-04  3.994e-03   0.064 0.949105
artdata5:d2up              5.382e-03  3.994e-03   1.347 0.177843
artdata6:d2up              8.130e-03  3.994e-03   2.036 0.041800 *
artdata2:d2updown         -3.467e-03  3.994e-03  -0.868 0.385341
artdata3:d2updown         -2.241e-02  3.994e-03  -5.611 2.02e-08 ***
artdata4:d2updown         -8.754e-02  3.994e-03 -21.917  < 2e-16 ***
artdata5:d2updown         -8.796e-02  3.994e-03 -22.023  < 2e-16 ***
artdata6:d2updown         -1.043e-01  3.994e-03 -26.121  < 2e-16 ***
artdata2:cl_iterate0.5    -3.908e-03  3.994e-03  -0.978 0.327861
artdata3:cl_iterate0.5    -2.237e-02  3.994e-03  -5.602 2.13e-08 ***
artdata4:cl_iterate0.5    -7.241e-02  3.994e-03 -18.129  < 2e-16 ***
artdata5:cl_iterate0.5    -7.278e-02  3.994e-03 -18.221  < 2e-16 ***
artdata6:cl_iterate0.5    -8.531e-02  3.994e-03 -21.360  < 2e-16 ***
artdata2:cl_iterate0.9    -6.311e-03  3.994e-03  -1.580 0.114085
artdata3:cl_iterate0.9    -3.466e-02  3.994e-03  -8.677  < 2e-16 ***
artdata4:cl_iterate0.9    -3.277e-02  3.994e-03  -8.204 2.33e-16 ***
artdata5:cl_iterate0.9    -3.192e-02  3.994e-03  -7.991 1.35e-15 ***
artdata6:cl_iterate0.9    -5.752e-02  3.994e-03 -14.400  < 2e-16 ***
artdata2:cl_unique0.5      2.900e-03  3.994e-03   0.726 0.467791
artdata3:cl_unique0.5      2.455e-02  3.994e-03   6.146 7.97e-10 ***
artdata4:cl_unique0.5      5.514e-02  3.994e-03  13.806  < 2e-16 ***
artdata5:cl_unique0.5      5.483e-02  3.994e-03  13.727  < 2e-16 ***
artdata6:cl_unique0.5      7.740e-02  3.994e-03  19.380  < 2e-16 ***
artdata2:cl_unique0.9      2.591e-03  3.994e-03   0.649 0.516485
artdata3:cl_unique0.9      2.995e-02  3.994e-03   7.499 6.43e-14 ***
artdata4:cl_unique0.9      6.410e-02  3.994e-03  16.049  < 2e-16 ***
artdata5:cl_unique0.9      6.337e-02  3.994e-03  15.867  < 2e-16 ***
artdata6:cl_unique0.9      9.178e-02  3.994e-03  22.978  < 2e-16 ***
count150:sparsity500       1.298e-03  2.824e-03   0.460 0.645839
count50:sparsity500       -1.156e-03  2.824e-03  -0.409 0.682421
count150:sparsity900      -3.250e-03  2.824e-03  -1.151 0.249814
count50:sparsity900       -1.356e-02  2.824e-03  -4.802 1.58e-06 ***
count150:Row2              3.232e-03  3.261e-03   0.991 0.321665
count50:Row2              -7.788e-03  3.261e-03  -2.388 0.016934 *
count150:Row2.5            7.966e-03  3.261e-03   2.443 0.014581 *
count50:Row2.5            -2.373e-02  3.261e-03  -7.276 3.45e-13 ***
count150:Row3              1.871e-02  3.261e-03   5.736 9.69e-09 ***
count50:Row3              -4.866e-02  3.261e-03 -14.920  < 2e-16 ***
count150:Col2             -2.351e-03  3.261e-03  -0.721 0.470916
count50:Col2              -1.536e-02  3.261e-03  -4.710 2.47e-06 ***
count150:Col2.5            2.676e-02  3.261e-03   8.205 2.31e-16 ***
count50:Col2.5            -4.576e-02  3.261e-03 -14.031  < 2e-16 ***
count150:Col3              1.674e-02  3.261e-03   5.132 2.87e-07 ***
count50:Col3              -3.794e-02  3.261e-03 -11.635  < 2e-16 ***
```

```
count150:d1up              -8.122e-05  2.824e-03  -0.029 0.977059
count50:d1up                5.181e-03  2.824e-03   1.835 0.066568 .
count150:d1updown          -3.062e-03  2.824e-03  -1.084 0.278242
count50:d1updown            1.779e-02  2.824e-03   6.301 2.97e-10 ***
count150:d2up              -6.616e-03  2.824e-03  -2.343 0.019152 *
count50:d2up                3.147e-03  2.824e-03   1.114 0.265113
count150:d2updown          -1.092e-02  2.824e-03  -3.866 0.000111 ***
count50:d2updown            8.126e-03  2.824e-03   2.877 0.004011 **
count150:cl_iterate0.5      1.229e-03  2.824e-03   0.435 0.663360
count50:cl_iterate0.5      -8.234e-04  2.824e-03  -0.292 0.770625
count150:cl_iterate0.9      1.744e-02  2.824e-03   6.176 6.59e-10 ***
count50:cl_iterate0.9      -2.965e-02  2.824e-03 -10.497  < 2e-16 ***
count150:cl_unique0.5      -1.880e-02  2.824e-03  -6.656 2.82e-11 ***
count50:cl_unique0.5        2.326e-02  2.824e-03   8.235  < 2e-16 ***
count150:cl_unique0.9      -1.518e-02  2.824e-03  -5.376 7.61e-08 ***
count50:cl_unique0.9        1.914e-02  2.824e-03   6.778 1.22e-11 ***
sparsity500:Row2            1.506e-03  3.261e-03   0.462 0.644130
sparsity900:Row2           -5.684e-04  3.261e-03  -0.174 0.861636
sparsity500:Row2.5          5.285e-03  3.261e-03   1.621 0.105108
sparsity900:Row2.5          1.347e-03  3.261e-03   0.413 0.679622
sparsity500:Row3            1.079e-02  3.261e-03   3.309 0.000936 ***
sparsity900:Row3            2.578e-03  3.261e-03   0.790 0.429256
sparsity500:Col2           -1.286e-03  3.261e-03  -0.394 0.693416
sparsity900:Col2            2.342e-03  3.261e-03   0.718 0.472737
sparsity500:Col2.5          4.126e-04  3.261e-03   0.127 0.899321
sparsity900:Col2.5          3.233e-03  3.261e-03   0.991 0.321529
sparsity500:Col3           -9.924e-03  3.261e-03  -3.043 0.002341 **
sparsity900:Col3            6.335e-03  3.261e-03   1.943 0.052063 .
sparsity500:d1up           -3.702e-03  2.824e-03  -1.311 0.189931
sparsity900:d1up           -4.868e-04  2.824e-03  -0.172 0.863146
sparsity500:d1updown        7.152e-04  2.824e-03   0.253 0.800079
sparsity900:d1updown       -3.618e-03  2.824e-03  -1.281 0.200161
sparsity500:d2up           -1.163e-03  2.824e-03  -0.412 0.680432
sparsity900:d2up           -6.582e-03  2.824e-03  -2.330 0.019787 *
sparsity500:d2updown        6.183e-03  2.824e-03   2.189 0.028584 *
sparsity900:d2updown        1.037e-04  2.824e-03   0.037 0.970718
sparsity500:cl_iterate0.5   6.008e-04  2.824e-03   0.213 0.831532
sparsity900:cl_iterate0.5   1.131e-03  2.824e-03   0.401 0.688704
sparsity500:cl_iterate0.9  -6.203e-03  2.824e-03  -2.196 0.028061 *
sparsity900:cl_iterate0.9   3.649e-03  2.824e-03   1.292 0.196294
sparsity500:cl_unique0.5    4.645e-04  2.824e-03   0.164 0.869351
sparsity900:cl_unique0.5    2.984e-03  2.824e-03   1.057 0.290651
sparsity500:cl_unique0.9    7.652e-04  2.824e-03   0.271 0.786454
sparsity900:cl_unique0.9    2.778e-03  2.824e-03   0.984 0.325284
Row2:Col2                  -7.420e-03  3.766e-03  -1.970 0.048789 *
Row2.5:Col2                -1.039e-02  3.766e-03  -2.759 0.005796 **
Row3:Col2                  -2.344e-02  3.766e-03  -6.225 4.83e-10 ***
Row2:Col2.5                -7.117e-03  3.766e-03  -1.890 0.058760 .
Row2.5:Col2.5              -3.065e-02  3.766e-03  -8.140 3.96e-16 ***
Row3:Col2.5                -2.259e-01  3.766e-03 -59.992  < 2e-16 ***
Row2:Col3                  -6.514e-02  3.766e-03 -17.298  < 2e-16 ***
Row2.5:Col3                -1.375e-01  3.766e-03 -36.519  < 2e-16 ***
Row3:Col3                  -1.932e-01  3.766e-03 -51.316  < 2e-16 ***
Row2:d1up                  -6.222e-03  3.261e-03  -1.908 0.056424 .
Row2.5:d1up                -5.561e-03  3.261e-03  -1.705 0.088163 .
Row3:d1up                  -1.035e-02  3.261e-03  -3.174 0.001504 **
Row2:d1updown               7.219e-03  3.261e-03   2.214 0.026852 *
Row2.5:d1updown             1.860e-02  3.261e-03   5.702 1.18e-08 ***
Row3:d1updown               3.730e-02  3.261e-03  11.436  < 2e-16 ***
Row2:d2up                  -1.586e-02  3.261e-03  -4.862 1.16e-06 ***
Row2.5:d2up                -2.020e-02  3.261e-03  -6.195 5.84e-10 ***
Row3:d2up                  -1.826e-02  3.261e-03  -5.600 2.15e-08 ***
Row2:d2updown               2.563e-02  3.261e-03   7.858 3.93e-15 ***
Row2.5:d2updown             3.592e-02  3.261e-03  11.014  < 2e-16 ***
Row3:d2updown               5.118e-02  3.261e-03  15.693  < 2e-16 ***
Row2:cl_iterate0.5          4.876e-02  3.261e-03  14.953  < 2e-16 ***
Row2.5:cl_iterate0.5        7.146e-02  3.261e-03  21.913  < 2e-16 ***
Row3:cl_iterate0.5          3.081e-02  3.261e-03   9.448  < 2e-16 ***
Row2:cl_iterate0.9         -1.750e-02  3.261e-03  -5.366 8.06e-08 ***
Row2.5:cl_iterate0.9       -4.983e-02  3.261e-03 -15.278  < 2e-16 ***
Row3:cl_iterate0.9         -1.381e-01  3.261e-03 -42.338  < 2e-16 ***
Row2:cl_unique0.5           2.358e-02  3.261e-03   7.230 4.86e-13 ***
Row2.5:cl_unique0.5         4.509e-02  3.261e-03  13.828  < 2e-16 ***
Row3:cl_unique0.5           5.744e-02  3.261e-03  17.614  < 2e-16 ***
Row2:cl_unique0.9           1.779e-02  3.261e-03   5.454 4.92e-08 ***
```

```
Row2.5:cl_unique0.9         3.613e-02  3.261e-03  11.079  < 2e-16 ***
Row3:cl_unique0.9           4.189e-02  3.261e-03  12.844  < 2e-16 ***
Col2:d1up                   2.720e-03  3.261e-03   0.834 0.404331
Col2.5:d1up                -8.019e-03  3.261e-03  -2.459 0.013934 *
Col3:d1up                   2.382e-03  3.261e-03   0.730 0.465180
Col2:d1updown               1.420e-02  3.261e-03   4.355 1.33e-05 ***
Col2.5:d1updown             2.056e-02  3.261e-03   6.305 2.89e-10 ***
Col3:d1updown              -1.244e-01  3.261e-03 -38.158  < 2e-16 ***
Col2:d2up                   1.044e-02  3.261e-03   3.202 0.001364 **
Col2.5:d2up                 4.405e-03  3.261e-03   1.351 0.176799
Col3:d2up                  -9.966e-03  3.261e-03  -3.056 0.002244 **
Col2:d2updown               1.131e-02  3.261e-03   3.469 0.000522 ***
Col2.5:d2updown             7.210e-03  3.261e-03   2.211 0.027041 *
Col3:d2updown              -1.457e-01  3.261e-03 -44.679  < 2e-16 ***
Col2:cl_iterate0.5          9.363e-03  3.261e-03   2.871 0.004093 **
Col2.5:cl_iterate0.5        1.502e-02  3.261e-03   4.606 4.11e-06 ***
Col3:cl_iterate0.5         -1.097e-01  3.261e-03 -33.639  < 2e-16 ***
Col2:cl_iterate0.9          3.319e-03  3.261e-03   1.018 0.308841
Col2.5:cl_iterate0.9       -1.160e-02  3.261e-03  -3.557 0.000375 ***
Col3:cl_iterate0.9         -2.416e-01  3.261e-03 -74.083  < 2e-16 ***
Col2:cl_unique0.5           3.234e-02  3.261e-03   9.917  < 2e-16 ***
Col2.5:cl_unique0.5         7.871e-02  3.261e-03  24.134  < 2e-16 ***
Col3:cl_unique0.5           8.526e-02  3.261e-03  26.143  < 2e-16 ***
Col2:cl_unique0.9           3.190e-02  3.261e-03   9.780  < 2e-16 ***
Col2.5:cl_unique0.9         7.288e-02  3.261e-03  22.347  < 2e-16 ***
Col3:cl_unique0.9           6.767e-02  3.261e-03  20.749  < 2e-16 ***
d1up:d2up                   9.259e-02  2.824e-03  32.785  < 2e-16 ***
d1updown:d2up               5.561e-02  2.824e-03  19.689  < 2e-16 ***
d1up:d2updown               4.841e-02  2.824e-03  17.142  < 2e-16 ***
d1updown:d2updown          -2.149e-01  2.824e-03 -76.085  < 2e-16 ***
d1up:cl_iterate0.5         -1.121e-03  2.824e-03  -0.397 0.691314
d1updown:cl_iterate0.5      7.222e-02  2.824e-03  25.571  < 2e-16 ***
d1up:cl_iterate0.9          1.728e-04  2.824e-03   0.061 0.951204
d1updown:cl_iterate0.9      1.547e-01  2.824e-03  54.773  < 2e-16 ***
d1up:cl_unique0.5           5.878e-04  2.824e-03   0.208 0.835125
d1updown:cl_unique0.5      -4.033e-02  2.824e-03 -14.278  < 2e-16 ***
d1up:cl_unique0.9          -9.480e-04  2.824e-03  -0.336 0.737139
d1updown:cl_unique0.9      -3.441e-02  2.824e-03 -12.183  < 2e-16 ***
d2up:cl_iterate0.5         -4.013e-03  2.824e-03  -1.421 0.155393
d2updown:cl_iterate0.5      2.623e-02  2.824e-03   9.288  < 2e-16 ***
d2up:cl_iterate0.9         -1.925e-02  2.824e-03  -6.815 9.45e-12 ***
d2updown:cl_iterate0.9      9.512e-02  2.824e-03  33.681  < 2e-16 ***
d2up:cl_unique0.5           2.693e-03  2.824e-03   0.954 0.340241
d2updown:cl_unique0.5      -2.886e-02  2.824e-03 -10.220  < 2e-16 ***
d2up:cl_unique0.9           2.598e-03  2.824e-03   0.920 0.357569
d2updown:cl_unique0.9      -2.431e-02  2.824e-03  -8.607  < 2e-16 ***
cl_iterate0.5:cl_unique0.5  1.881e-02  2.824e-03   6.661 2.72e-11 ***
cl_iterate0.9:cl_unique0.5  6.705e-02  2.824e-03  23.742  < 2e-16 ***
cl_iterate0.5:cl_unique0.9 -2.201e-02  2.824e-03  -7.795 6.48e-15 ***
cl_iterate0.9:cl_unique0.9  2.455e-02  2.824e-03   8.692  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.04651894)

    Null deviance: 26136.5  on 209951  degrees of freedom
Residual deviance:  9754.9  on 209697  degrees of freedom
AIC: -48036

Number of Fisher Scoring iterations: 2
```

E.4 GLM_Jaccard4

Auch hier gilt für die Datei `gesamtdaten` dasselbe wie in Anhang E.3.

```
Call:
glm(formula = Jaccard ~ .^2, family = quasibinomial(), data = gesamtdaten)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.19488  -0.27605  -0.02207   0.32513   2.05776

Coefficients:
                           Estimate Std. Error t value Pr(>|t|)
(Intercept)               -0.4724829  0.0403468 -11.711  < 2e-16 ***
artdata2                   0.1005122  0.0391631   2.567 0.010274 *
artdata3                   0.3079602  0.0388838   7.920 2.39e-15 ***
artdata4                  -0.6294814  0.0398325 -15.803  < 2e-16 ***
artdata5                  -0.4770708  0.0397229 -12.010  < 2e-16 ***
artdata6                  -0.2970307  0.0392186  -7.574 3.64e-14 ***
count150                  -0.0398967  0.0297296  -1.342 0.179602
count50                    0.2444581  0.0298394   8.192 2.57e-16 ***
sparsity500                0.0028889  0.0298049   0.097 0.922785
sparsity900                0.0075823  0.0297922   0.255 0.799106
Row2                       0.0782564  0.0326870   2.394 0.016661 *
Row2.5                     0.1299432  0.0335105   3.878 0.000105 ***
Row3                       0.6524892  0.0346336  18.840  < 2e-16 ***
Col2                      -0.1490342  0.0305924  -4.872 1.11e-06 ***
Col2.5                    -0.3076964  0.0309717  -9.935  < 2e-16 ***
Col3                      -2.0399782  0.0519852 -39.242  < 2e-16 ***
d1up                      -0.2167196  0.0295025  -7.346 2.05e-13 ***
d1updown                   0.9712812  0.0305223  31.822  < 2e-16 ***
d2up                      -0.1210583  0.0295045  -4.103 4.08e-05 ***
d2updown                   1.0994119  0.0304868  36.062  < 2e-16 ***
cl_iterate0.5              0.3633132  0.0295335  12.302  < 2e-16 ***
cl_iterate0.9              1.1889718  0.0306779  38.757  < 2e-16 ***
cl_unique0.5              -0.8788525  0.0300803 -29.217  < 2e-16 ***
cl_unique0.9              -0.6170268  0.0297912 -20.712  < 2e-16 ***
artdata2:count150          0.0071841  0.0227365   0.316 0.752024
artdata3:count150          0.0158238  0.0225922   0.700 0.483672
artdata4:count150          0.0569955  0.0229189   2.487 0.012889 *
artdata5:count150          0.0071832  0.0229331   0.313 0.754113
artdata6:count150         -0.0755175  0.0226932  -3.328 0.000876 ***
artdata2:count50          -0.0294816  0.0229428  -1.285 0.198792
artdata3:count50          -0.0261565  0.0228206  -1.146 0.251723
artdata4:count50          -0.0828629  0.0231194  -3.584 0.000338 ***
artdata5:count50          -0.1046172  0.0231089  -4.527 5.98e-06 ***
artdata6:count50          -0.1076658  0.0228174  -4.719 2.38e-06 ***
artdata2:sparsity500       0.0260495  0.0228949   1.138 0.255210
artdata3:sparsity500      -0.0264514  0.0227586  -1.162 0.245131
artdata4:sparsity500       0.0998652  0.0230492   4.333 1.47e-05 ***
artdata5:sparsity500       0.0116735  0.0230346   0.507 0.612307
artdata6:sparsity500       0.0150186  0.0227833   0.659 0.509773
artdata2:sparsity900      -0.0701847  0.0228213  -3.075 0.002102 **
artdata3:sparsity900      -0.0062062  0.0226948  -0.273 0.784498
artdata4:sparsity900       0.0808055  0.0230548   3.505 0.000457 ***
artdata5:sparsity900      -0.0051806  0.0230403  -0.225 0.822098
artdata6:sparsity900      -0.0036390  0.0227857  -0.160 0.873114
artdata2:Row2              0.0048848  0.0251758   0.194 0.846154
artdata3:Row2             -0.0285132  0.0250549  -1.138 0.255111
artdata4:Row2             -0.1071912  0.0256725  -4.175 2.98e-05 ***
artdata5:Row2             -0.1024153  0.0256856  -3.987 6.69e-05 ***
artdata6:Row2             -0.1187199  0.0253396  -4.685 2.80e-06 ***
artdata2:Row2.5            0.0222002  0.0262156   0.847 0.397089
artdata3:Row2.5           -0.0947182  0.0260872  -3.631 0.000283 ***
artdata4:Row2.5           -0.2293988  0.0264325  -8.679  < 2e-16 ***
artdata5:Row2.5           -0.2316115  0.0264374  -8.761  < 2e-16 ***
artdata6:Row2.5           -0.3265776  0.0261553 -12.486  < 2e-16 ***
artdata2:Row3             -0.0326543  0.0272857  -1.197 0.231404
```

```
artdata3:Row3             -0.2526241  0.0270719  -9.332  < 2e-16 ***
artdata4:Row3             -0.6443572  0.0276453 -23.308  < 2e-16 ***
artdata5:Row3             -0.6697304  0.0276666 -24.207  < 2e-16 ***
artdata6:Row3             -0.7306759  0.0273423 -26.723  < 2e-16 ***
artdata2:Col2              0.0062655  0.0241107   0.260 0.794970
artdata3:Col2              0.0328235  0.0239623   1.370 0.170751
artdata4:Col2             -0.0346886  0.0236153  -1.469 0.141860
artdata5:Col2             -0.0334071  0.0236001  -1.416 0.156908
artdata6:Col2             -0.0852561  0.0233843  -3.646 0.000267 ***
artdata2:Col2.5           -0.0031261  0.0239369  -0.131 0.896095
artdata3:Col2.5           -0.0094330  0.0237683  -0.397 0.691460
artdata4:Col2.5           -0.2324299  0.0240777  -9.653  < 2e-16 ***
artdata5:Col2.5           -0.2528053  0.0240848 -10.496  < 2e-16 ***
artdata6:Col2.5           -0.2434991  0.0237809 -10.239  < 2e-16 ***
artdata2:Col3              0.0117461  0.0369091   0.318 0.750301
artdata3:Col3             -0.0752638  0.0368512  -2.042 0.041116 *
artdata4:Col3              0.2704258  0.0435027   6.216 5.10e-10 ***
artdata5:Col3              0.1900242  0.0441367   4.305 1.67e-05 ***
artdata6:Col3              0.1219011  0.0435406   2.800 0.005115 **
artdata2:d1up             -0.0736478  0.0220874  -3.334 0.000855 ***
artdata3:d1up             -0.0478993  0.0219544  -2.182 0.029128 *
artdata4:d1up              0.0028883  0.0227271   0.127 0.898873
artdata5:d1up             -0.1429520  0.0227446  -6.285 3.28e-10 ***
artdata6:d1up             -0.0691626  0.0224132  -3.086 0.002030 **
artdata2:d1updown         -0.0867217  0.0238101  -3.642 0.000270 ***
artdata3:d1updown         -0.2524553  0.0236284 -10.684  < 2e-16 ***
artdata4:d1updown         -0.7247420  0.0236556 -30.637  < 2e-16 ***
artdata5:d1updown         -0.7699920  0.0235801 -32.654  < 2e-16 ***
artdata6:d1updown         -0.8496177  0.0234004 -36.308  < 2e-16 ***
artdata2:d2up             -0.0419953  0.0221216  -1.898 0.057647 .
artdata3:d2up             -0.0160056  0.0219728  -0.728 0.466354
artdata4:d2up              0.0050770  0.0228154   0.223 0.823904
artdata5:d2up              0.0454432  0.0228222   1.991 0.046461 *
artdata6:d2up              0.0524950  0.0224831   2.335 0.019552 *
artdata2:d2updown         -0.0276796  0.0237213  -1.167 0.243265
artdata3:d2updown         -0.1655174  0.0235576  -7.026 2.13e-12 ***
artdata4:d2updown         -0.6538129  0.0235569 -27.755  < 2e-16 ***
artdata5:d2updown         -0.6490606  0.0236001 -27.503  < 2e-16 ***
artdata6:d2updown         -0.7667055  0.0233707 -32.806  < 2e-16 ***
artdata2:cl_iterate0.5    -0.0279612  0.0221192  -1.264 0.206190
artdata3:cl_iterate0.5    -0.1498375  0.0219370  -6.830 8.49e-12 ***
artdata4:cl_iterate0.5    -0.4325706  0.0228331 -18.945  < 2e-16 ***
artdata5:cl_iterate0.5    -0.4371576  0.0228347 -19.144  < 2e-16 ***
artdata6:cl_iterate0.5    -0.5237603  0.0224927 -23.286  < 2e-16 ***
artdata2:cl_iterate0.9    -0.0527871  0.0238897  -2.210 0.027133 *
artdata3:cl_iterate0.9    -0.2541681  0.0236881 -10.730  < 2e-16 ***
artdata4:cl_iterate0.9    -0.4655695  0.0237711 -19.586  < 2e-16 ***
artdata5:cl_iterate0.9    -0.4614489  0.0237812 -19.404  < 2e-16 ***
artdata6:cl_iterate0.9    -0.6367911  0.0235004 -27.097  < 2e-16 ***
artdata2:cl_unique0.5      0.0230910  0.0231270   0.998 0.318065
artdata3:cl_unique0.5      0.1754934  0.0229750   7.638 2.21e-14 ***
artdata4:cl_unique0.5      0.4218776  0.0233245  18.087  < 2e-16 ***
artdata5:cl_unique0.5      0.4213028  0.0233338  18.055  < 2e-16 ***
artdata6:cl_unique0.5      0.5677683  0.0231311  24.546  < 2e-16 ***
artdata2:cl_unique0.9      0.0205718  0.0230388   0.893 0.371900
artdata3:cl_unique0.9      0.2047760  0.0229049   8.940  < 2e-16 ***
artdata4:cl_unique0.9      0.4975141  0.0231327  21.507  < 2e-16 ***
artdata5:cl_unique0.9      0.4940500  0.0231453  21.346  < 2e-16 ***
artdata6:cl_unique0.9      0.6673829  0.0229503  29.079  < 2e-16 ***
count150:sparsity500       0.0083548  0.0160249   0.521 0.602114
count50:sparsity500       -0.0073062  0.0161242  -0.453 0.650464
count150:sparsity900      -0.0212532  0.0160202  -1.327 0.184627
count50:sparsity900       -0.0882955  0.0161285  -5.474 4.39e-08 ***
count150:Row2              0.0249767  0.0178834   1.397 0.162521
count50:Row2              -0.0567724  0.0179490  -3.163 0.001562 **
count150:Row2.5            0.0630744  0.0183979   3.428 0.000607 ***
count50:Row2.5            -0.1839747  0.0184632  -9.964  < 2e-16 ***
count150:Row3              0.1500912  0.0190877   7.863 3.76e-15 ***
count50:Row3              -0.3999261  0.0192175 -20.811  < 2e-16 ***
count150:Col2             -0.0122817  0.0164783  -0.745 0.456076
count50:Col2              -0.0780618  0.0164286  -4.752 2.02e-06 ***
count150:Col2.5            0.1436887  0.0166991   8.605  < 2e-16 ***
count50:Col2.5            -0.2597126  0.0167914 -15.467  < 2e-16 ***
count150:Col3              0.1868302  0.0280035   6.672 2.54e-11 ***
count50:Col3              -0.6092965  0.0301114 -20.235  < 2e-16 ***
```

```
count150:d1up                0.0005829  0.0158048   0.037 0.970577
count50:d1up                 0.0322370  0.0159206   2.025 0.042883 *
count150:d1updown           -0.0237577  0.0162114  -1.465 0.142787
count50:d1updown             0.1407221  0.0163348   8.615  < 2e-16 ***
count150:d2up               -0.0398179  0.0158452  -2.513 0.011974 *
count50:d2up                 0.0185010  0.0159586   1.159 0.246330
count150:d2updown           -0.0723380  0.0162339  -4.456 8.35e-06 ***
count50:d2updown             0.0757849  0.0163612   4.632 3.62e-06 ***
count150:cl_iterate0.5       0.0089321  0.0158495   0.564 0.573058
count50:cl_iterate0.5        0.0039078  0.0158944   0.246 0.805792
count150:cl_iterate0.9       0.1102012  0.0162966   6.762 1.36e-11 ***
count50:cl_iterate0.9       -0.1698445  0.0164053 -10.353  < 2e-16 ***
count150:cl_unique0.5       -0.1274634  0.0161684  -7.883 3.20e-15 ***
count50:cl_unique0.5         0.1638416  0.0162873  10.059  < 2e-16 ***
count150:cl_unique0.9       -0.1010828  0.0160312  -6.305 2.88e-10 ***
count50:cl_unique0.9         0.1350550  0.0161852   8.344  < 2e-16 ***
sparsity500:Row2             0.0074310  0.0179336   0.414 0.678611
sparsity900:Row2            -0.0042248  0.0179367  -0.236 0.813789
sparsity500:Row2.5           0.0315468  0.0184457   1.710 0.087221 .
sparsity900:Row2.5           0.0078412  0.0184508   0.425 0.670853
sparsity500:Row3             0.0780641  0.0191641   4.073 4.63e-05 ***
sparsity900:Row3             0.0168528  0.0191761   0.879 0.379489
sparsity500:Col2            -0.0061346  0.0164675  -0.373 0.709499
sparsity900:Col2             0.0119262  0.0164638   0.724 0.468828
sparsity500:Col2.5           0.0084603  0.0167539   0.505 0.613579
sparsity900:Col2.5           0.0178455  0.0167691   1.064 0.287245
sparsity500:Col3            -0.0346735  0.0291239  -1.191 0.233831
sparsity900:Col3             0.0524794  0.0290148   1.809 0.070497 .
sparsity500:d1up            -0.0229502  0.0158730  -1.446 0.148218
sparsity900:d1up            -0.0023050  0.0158892  -0.145 0.884656
sparsity500:d1updown         0.0065636  0.0162986   0.403 0.687160
sparsity900:d1updown        -0.0291988  0.0162917  -1.792 0.073095 .
sparsity500:d2up            -0.0076194  0.0159142  -0.479 0.632093
sparsity900:d2up            -0.0415318  0.0159307  -2.607 0.009134 **
sparsity500:d2updown         0.0404647  0.0163230   2.479 0.013176 *
sparsity900:d2updown        -0.0049759  0.0163059  -0.305 0.760243
sparsity500:cl_iterate0.5    0.0026723  0.0158819   0.168 0.866376
sparsity900:cl_iterate0.5    0.0035975  0.0159029   0.226 0.821035
sparsity500:cl_iterate0.9   -0.0383412  0.0163657  -2.343 0.019142 *
sparsity900:cl_iterate0.9    0.0135840  0.0163716   0.830 0.406693
sparsity500:cl_unique0.5     0.0015449  0.0162449   0.095 0.924236
sparsity900:cl_unique0.5     0.0215265  0.0162476   1.325 0.185204
sparsity500:cl_unique0.9     0.0017978  0.0161272   0.111 0.911237
sparsity900:cl_unique0.9     0.0201739  0.0161267   1.251 0.210948
Row2:Col2                   -0.0372945  0.0188937  -1.974 0.048393 *
Row2.5:Col2                 -0.0523677  0.0189419  -2.765 0.005699 **
Row3:Col2                   -0.1212641  0.0191368  -6.337 2.35e-10 ***
Row2:Col2.5                 -0.0365329  0.0189692  -1.926 0.054118 .
Row2.5:Col2.5               -0.1599595  0.0190487  -8.397  < 2e-16 ***
Row3:Col2.5                 -1.2939630  0.0199563 -64.840  < 2e-16 ***
Row2:Col3                   -0.5874474  0.0267138 -21.990  < 2e-16 ***
Row2.5:Col3                 -2.1271775  0.0372702 -57.075  < 2e-16 ***
Row3:Col3                   -5.7335023  0.1478792 -38.772  < 2e-16 ***
Row2:d1up                   -0.0314819  0.0177275  -1.776 0.075754 .
Row2.5:d1up                 -0.0240531  0.0181742  -1.323 0.185679
Row3:d1up                   -0.0624870  0.0189509  -3.297 0.000976 ***
Row2:d1updown                0.0847126  0.0181397   4.670 3.01e-06 ***
Row2.5:d1updown              0.2138389  0.0187206  11.423  < 2e-16 ***
Row3:d1updown                0.4260968  0.0194705  21.884  < 2e-16 ***
Row2:d2up                   -0.0949397  0.0177922  -5.336 9.51e-08 ***
Row2.5:d2up                 -0.1257263  0.0182392  -6.893 5.47e-12 ***
Row3:d2up                   -0.1093560  0.0190090  -5.753 8.79e-09 ***
Row2:d2updown                0.1845134  0.0181528  10.164  < 2e-16 ***
Row2.5:d2updown              0.3043613  0.0187371  16.244  < 2e-16 ***
Row3:d2updown                0.5045277  0.0195181  25.849  < 2e-16 ***
Row2:cl_iterate0.5           0.2831515  0.0177730  15.932  < 2e-16 ***
Row2.5:cl_iterate0.5         0.4481402  0.0182583  24.544  < 2e-16 ***
Row3:cl_iterate0.5           0.2531688  0.0189661  13.349  < 2e-16 ***
Row2:cl_iterate0.9          -0.0679384  0.0182320  -3.726 0.000194 ***
Row2.5:cl_iterate0.9        -0.2258649  0.0188206 -12.001  < 2e-16 ***
Row3:cl_iterate0.9          -0.7651894  0.0194876 -39.265  < 2e-16 ***
Row2:cl_unique0.5            0.1431565  0.0180980   7.910 2.58e-15 ***
Row2.5:cl_unique0.5          0.2937084  0.0186187  15.775  < 2e-16 ***
Row3:cl_unique0.5            0.4120824  0.0193570  21.289  < 2e-16 ***
Row2:cl_unique0.9            0.1063830  0.0179240   5.935 2.94e-09 ***
```

```
Row2.5:cl_unique0.9          0.2373005  0.0184610  12.854  < 2e-16 ***
Row3:cl_unique0.9            0.3133198  0.0192220  16.300  < 2e-16 ***
Col2:d1up                    0.0115674  0.0160661   0.720 0.471531
Col2.5:d1up                 -0.0484914  0.0165245  -2.935 0.003341 **
Col3:d1up                    0.0457400  0.0306970   1.490 0.136213
Col2:d1updown                0.0684669  0.0168065   4.074 4.63e-05 ***
Col2.5:d1updown              0.1023525  0.0169996   6.021 1.74e-09 ***
Col3:d1updown               -0.2023567  0.0299649  -6.753 1.45e-11 ***
Col2:d2up                    0.0518285  0.0161098   3.217 0.001295 **
Col2.5:d2up                  0.0309710  0.0165756   1.868 0.061698 .
Col3:d2up                   -0.0932043  0.0305092  -3.055 0.002251 **
Col2:d2updown                0.0534104  0.0168257   3.174 0.001502 **
Col2.5:d2updown              0.0475023  0.0170399   2.788 0.005309 **
Col3:d2updown               -0.3380409  0.0296199 -11.413  < 2e-16 ***
Col2:cl_iterate0.5           0.0454259  0.0161590   2.811 0.004937 **
Col2.5:cl_iterate0.5         0.0860285  0.0165388   5.202 1.98e-07 ***
Col3:cl_iterate0.5          -0.5235094  0.0298810 -17.520  < 2e-16 ***
Col2:cl_iterate0.9           0.0157998  0.0168354   0.938 0.347994
Col2.5:cl_iterate0.9        -0.0752831  0.0171505  -4.390 1.14e-05 ***
Col3:cl_iterate0.9          -1.4429517  0.0296826 -48.613  < 2e-16 ***
Col2:cl_unique0.5            0.1678649  0.0166066  10.108  < 2e-16 ***
Col2.5:cl_unique0.5          0.4498859  0.0169442  26.551  < 2e-16 ***
Col3:cl_unique0.5            0.8191713  0.0298540  27.439  < 2e-16 ***
Col2:cl_unique0.9            0.1661943  0.0164879  10.080  < 2e-16 ***
Col2.5:cl_unique0.9          0.4271296  0.0168242  25.388  < 2e-16 ***
Col3:cl_unique0.9            0.7654796  0.0295383  25.915  < 2e-16 ***
d1up:d2up                    0.5779990  0.0158250  36.524  < 2e-16 ***
d1updown:d2up                0.3401002  0.0161680  21.035  < 2e-16 ***
d1up:d2updown                0.3085656  0.0161740  19.078  < 2e-16 ***
d1updown:d2updown           -1.2675725  0.0164425 -77.091  < 2e-16 ***
d1up:cl_iterate0.5          -0.0029295  0.0158450  -0.185 0.853321
d1updown:cl_iterate0.5       0.4866811  0.0159984  30.421  < 2e-16 ***
d1up:cl_iterate0.9           0.0090196  0.0160260   0.563 0.573565
d1updown:cl_iterate0.9       1.1567975  0.0169777  68.136  < 2e-16 ***
d1up:cl_unique0.5            0.0035663  0.0160275   0.223 0.823917
d1updown:cl_unique0.5       -0.3430636  0.0165001 -20.792  < 2e-16 ***
d1up:cl_unique0.9           -0.0053268  0.0159098  -0.335 0.737767
d1updown:cl_unique0.9       -0.3063160  0.0163737 -18.708  < 2e-16 ***
d2up:cl_iterate0.5          -0.0272785  0.0158764  -1.718 0.085765 .
d2updown:cl_iterate0.5       0.2132457  0.0160344  13.299  < 2e-16 ***
d2up:cl_iterate0.9          -0.1220280  0.0161017  -7.579 3.51e-14 ***
d2updown:cl_iterate0.9       0.7750947  0.0168929  45.883  < 2e-16 ***
d2up:cl_unique0.5            0.0177883  0.0160704   1.107 0.268339
d2updown:cl_unique0.5       -0.2626477  0.0165116 -15.907  < 2e-16 ***
d2up:cl_unique0.9            0.0130505  0.0159437   0.819 0.413055
d2updown:cl_unique0.9       -0.2405928  0.0163864 -14.682  < 2e-16 ***
cl_iterate0.5:cl_unique0.5   0.0779024  0.0160675   4.848 1.25e-06 ***
cl_iterate0.9:cl_unique0.5   0.3326847  0.0165685  20.079  < 2e-16 ***
cl_iterate0.5:cl_unique0.9 -0.1803854  0.0158781 -11.361  < 2e-16 ***
cl_iterate0.9:cl_unique0.9   0.0608020  0.0163835   3.711 0.000206 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for quasibinomial family taken to be 0.232989)

    Null deviance: 141598  on 209951  degrees of freedom
Residual deviance:  57460  on 209697  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 9
```

E.5 LM_Jaccard

```
Call:
lm(formula = Jaccard ~ artdata + count + sparsity + Row + Col +
    d1 + d2 + cl_iterate + cl_unique, data = gesamt_dataframe_numerisch)

Residuals:
      Min        1Q    Median        3Q       Max
-0.810380 -0.188678 -0.008648  0.178498  0.892943

Coefficients:
             Estimate Std. Error  t value Pr(>|t|)
(Intercept)  9.765e-01  4.363e-03  223.797  < 2e-16 ***
artdata2    -5.095e-04  2.030e-03   -0.251   0.8018
artdata3     1.195e-02  2.030e-03    5.886 3.96e-09 ***
artdata4    -2.054e-01  2.030e-03 -101.181  < 2e-16 ***
artdata5    -2.044e-01  2.030e-03 -100.663  < 2e-16 ***
artdata6    -1.936e-01  2.030e-03  -95.350  < 2e-16 ***
count       -3.594e-05  1.436e-05   -2.503   0.0123 *
sparsity    -3.590e-06  1.794e-06   -2.001   0.0454 *
Row         -5.660e-03  1.048e-03   -5.399 6.71e-08 ***
Col         -3.022e-01  1.048e-03 -288.292  < 2e-16 ***
d1up        -9.377e-04  1.436e-03   -0.653   0.5136
d1updown     1.242e-01  1.436e-03   86.500  < 2e-16 ***
d2up         1.079e-02  1.436e-03    7.515 5.72e-14 ***
d2updown     1.314e-01  1.436e-03   91.514  < 2e-16 ***
cl_iterate   2.418e-01  1.794e-03  134.761  < 2e-16 ***
cl_unique   -2.188e-02  1.794e-03  -12.194  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2685 on 209936 degrees of freedom
Multiple R-squared: 0.4208,      Adjusted R-squared: 0.4207
F-statistic: 1.017e+04 on 15 and 209936 DF,  p-value: < 2.2e-16
```

E.6 LM_Jaccard2

```
Call:
lm(formula = Jaccard ~ artdata + count + sparsity + Row + Col +
    d1 + d2 + cl_iterate + cl_unique, data = gesamt_dataframe)

Residuals:
    Min      1Q  Median      3Q     Max
-0.7936 -0.1631  0.0108  0.1857  0.7425

Coefficients:
               Estimate Std. Error  t value Pr(>|t|)
(Intercept)   0.4463904  0.0026677  167.334  < 2e-16 ***
artdata2     -0.0005095  0.0018863   -0.270   0.7871
artdata3      0.0119499  0.0018863    6.335 2.38e-10 ***
artdata4     -0.2054187  0.0018863 -108.899  < 2e-16 ***
artdata5     -0.2043665  0.0018863 -108.341  < 2e-16 ***
artdata6     -0.1935800  0.0018863 -102.623  < 2e-16 ***
count150     -0.0029025  0.0013338   -2.176   0.0295 *
count50       0.0006910  0.0013338    0.518   0.6044
sparsity500   0.0067356  0.0013338    5.050 4.43e-07 ***
sparsity900  -0.0028724  0.0013338   -2.153   0.0313 *
Row2          0.0216377  0.0015402   14.049  < 2e-16 ***
Row2.5        0.0201946  0.0015402   13.112  < 2e-16 ***
Row3         -0.0089526  0.0015402   -5.813 6.15e-09 ***
Col2         -0.0113843  0.0015402   -7.392 1.46e-13 ***
Col2.5       -0.0901987  0.0015402  -58.564  < 2e-16 ***
Col3         -0.4774691  0.0015402 -310.009  < 2e-16 ***
d1up         -0.0009377  0.0013338   -0.703   0.4820
d1updown      0.1241771  0.0013338   93.098  < 2e-16 ***
d2up          0.0107882  0.0013338    8.088 6.09e-16 ***
d2updown      0.1313749  0.0013338   98.494  < 2e-16 ***
cl_iterate0.5  0.0893688  0.0013338   67.002  < 2e-16 ***
cl_iterate0.9  0.1934592  0.0013338  145.040  < 2e-16 ***
cl_unique0.5  -0.0315661  0.0013338  -23.666  < 2e-16 ***
cl_unique0.9  -0.0175051  0.0013338  -13.124  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2495 on 209928 degrees of freedom
Multiple R-squared:   0.5,Adjusted R-squared: 0.4999
F-statistic:  9126 on 23 and 209928 DF,  p-value: < 2.2e-16
```

F CD-ROM

Die CD-ROM zur Bachelorarbeit beinhaltet alle, für die Durchführung der ISA-Analysen, erstellten und verwendeten Dateien. Im Folgenden wird ein kleiner Überblick über den Inhalt der CD-Rom gegeben:

- R-Code (.r)
- R-Daten (.RData)
- R-Grafiken (.pdf)
- R-Outputs (.pdf)
- Bachelorarbeit als Textdatei (.docx)

Eidesstattliche Erklärung

Hiermit erkläre ich, Julia Kammerer, dass ich die vorliegende Bachelorarbeit ohne fremde Hilfe angefertigt und nur die vorliegenden Quellen und Hilfsmittel benutzt habe.

Holzkirchen, den 10. August 2010

Julia Kammerer