**<https://ranalytics.github.io/data-mining/105-Cohonen-Maps.html>**

**Самоорганизующиеся карты Кохонена**

Самоорганизующиеся карты (SOM, Self Organizing Maps), разработанные Т. Кохоненом (Kohonen, 1982), представляют собой мощный инструмент, объединяющий две важные парадигмы анализа данных – кластеризацию и проецирование, т.е. визуализацию многомерных данных на плоскости. В отличие от рассмотренной в разделах [7.7](https://ranalytics.github.io/data-mining/105-Cohonen-Maps.html#sec_7_7) и [8.2](https://ranalytics.github.io/data-mining/082-NN-with-Caret.html#sec_8_2) нейронной сети обратного распространения, процедура настройки SOM относится к алгоритмами обучения без учителя.

Сеть Кохонена имеет всего два слоя: входной и выходной, составленный из радиальных нейронов упорядоченной структуры (выходной слой называют также слоем топологической карты, или “экраном”). Нейроны выходного слоя располагаются в узлах двумерной сетки с прямоугольными или шестиугольными ячейками. Количество нейронов в сетке *p*

определяет степень детализации результата работы алгоритма, и, в конечном счете, от этого зависит точность обобщающей способности карты.

Самоорганизующиеся карты в ходе своего обучения анализируют характер расположения точек входного слоя в m-мерном пространстве и стремятся воспроизвести на выходе нейронной сети топологический порядок и определенную степень регулярности исходных данных (т.е. метрическую близость векторов). Подгонка SOM заключается в итеративной настройке вектора весовых коэффициентов ***w****j*

каждого нейрона, *j*=1,2…,*p*, для чего используется модифицированный алгоритм соревновательного обучения Хебба, который учитывает не только вклад нейрона-победителя, но и ближайших его соседей, расположенных в *R*

-окрестности:

1. На стадии инициализации всем весовым коэффициентам присваиваются небольшие случайные значения *w*0*ij*,*i*=1,2,…,*m*

 .

 На выходы сети подаются последовательно в случайном порядке образы **y** объектов входного слоя и для каждого из них выбирается “нейрон-победитель” (BMU, Best Matching Unit) с минимальным расстоянием ∑*mi*=1(*yi*−*wtij*)

 - см. рис. [10.19](https://ranalytics.github.io/data-mining/105-Cohonen-Maps.html#fig:fig-10-19).

 Определяется подмножество “ближайшего окружения” BMU, радиус которого *R* уменьшается с каждой итерацией *t*

 .

 Пересчитываются веса ***w****tj* выделенных узлов с учетом их расстояний до нейрона-победителя и близости к вектору **y**

1. .

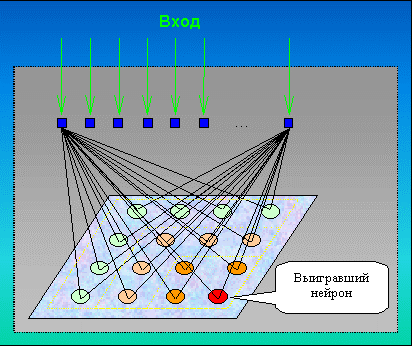


Рисунок 10.19: Схема активации нейронов в сети Кохонена

Шаги 2-4 алгоритма повторяются, пока выходные значения сети не будут стабилизированы с заданной точностью. При этом качество проецирования многомерных данных на плоскость достигается в SOM на нескольких уровнях: *сохранение топологии* (т.е. на множествах точек исходных данных и нейронов обученной сети структура соседства одинакова), *сохранение порядка* (т.е. расстояния между эквивалентными парами точек пропорциональны) и *сохранение метрических свойств при сжатии пространства*.

“Проекционный экран” в результате обучения приобретает свойства упорядоченной структуры, в которой величины синапсов нейронов плавно меняются вдоль двух измерений. Цвет и расположение фрагментов двумерной решетки используется для анализа закономерностей, связываемых с компонентами набора данных. В частности, с каждым узлом (нейроном) могут ассоциироваться локальные сгущения исходных объектов, которые могут служить потенциальными центрами кластеров.

Для обучения сети обычно используются функции somgrid() и som() из пакета kohonen. По завершении итерационного процесса функции plot() становится доступным для визуализации следующий комплект карт:

* "codes" - показывается распределение по решетке соотношение долей участия отдельных исходных переменных;
* "counts" - число исходных объектов в каждом узле сети;
* "mapping" - координаты исходных объектов на сформированной карте;
* "property", "quality", "dist.neighbours" - различными цветами изображается целый набор свойств каждого узла: доли участия отдельных исходных переменных, меры парных или средних расстояний между нейронами и т.д.

Опять воспользуемся в качестве примера набором Boston из пакета MASS. Чтобы график "codes" был более лаконичен, из исходного набора признаков выделим 7 переменных, предположительно наиболее значимых для кластеризации (см. рис. [10.13](https://ranalytics.github.io/data-mining/104-Other-Clustering-Methods.html#fig:fig-10-13)). Смысл сокращенных наименований переменных приведен в разделе [10.3](https://ranalytics.github.io/data-mining/103-Clustering-Quality.html#sec_10_3). Выполним предварительно стандартизацию данных.

Укажем функции somgrid() создать для проекционного экрана гексагональную решетку 6×9, т.е. 506 земельных участков Бостона будут “самоорганизовываться” на 54 нейронах выходного слоя:

data(Boston, package = "MASS")

VarName = c("indus", "dis", "nox", "medv", "lstat", "age", "rad")

# отбор переменных для обучения SOM

data\_train <- Boston[, VarName]

data\_train\_matrix <- as.matrix(scale(data\_train))

library(kohonen)

set.seed(123)

som\_grid <- somgrid(xdim = 9, ydim = 6, topo = "hexagonal")

som\_model <- som(data\_train\_matrix, grid = som\_grid, rlen = 100,

alpha = c(0.05,0.01), keep.data = TRUE)

plot(som\_model, type = "changes")

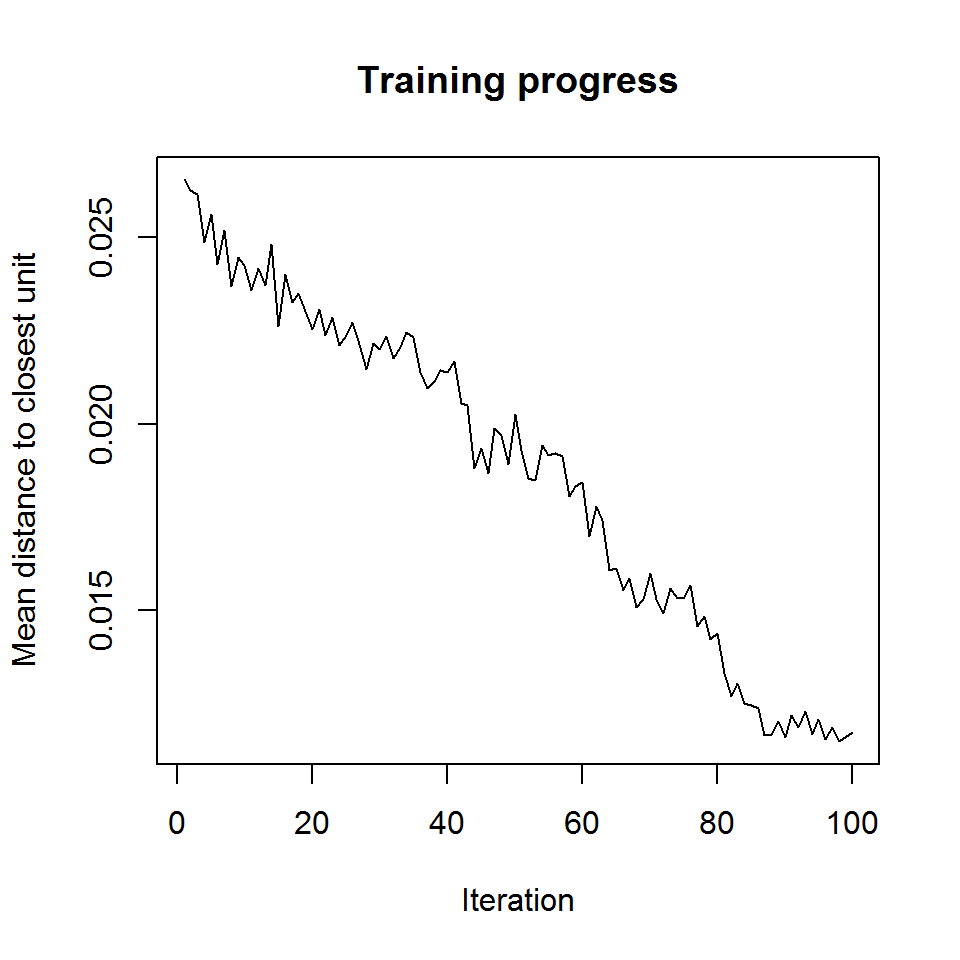


Рисунок 10.20: Снижение среднего расстояния до ближайших нейронов в ходе 100 итераций (rlen) обучения сети SOM при заданных значениях гипер-параметра alpha

Выполним теперь визуализацию комплекта карт Кохонена с разными управляющими параметрами функции plot().

# Зададим палитру цветов

coolBlueHotRed <- function(n, alpha = 1) {

rainbow(n, end = 4/6, alpha = alpha)[n:1]

}

par(mfrow = c(2, 1))

# Сколько объектов связано с каждым узлом?

plot(som\_model, type = "counts", palette.name = coolBlueHotRed)

# Каково среднее расстояние объектов узла до его прототипов?

plot(som\_model, type = "quality", palette.name = coolBlueHotRed)

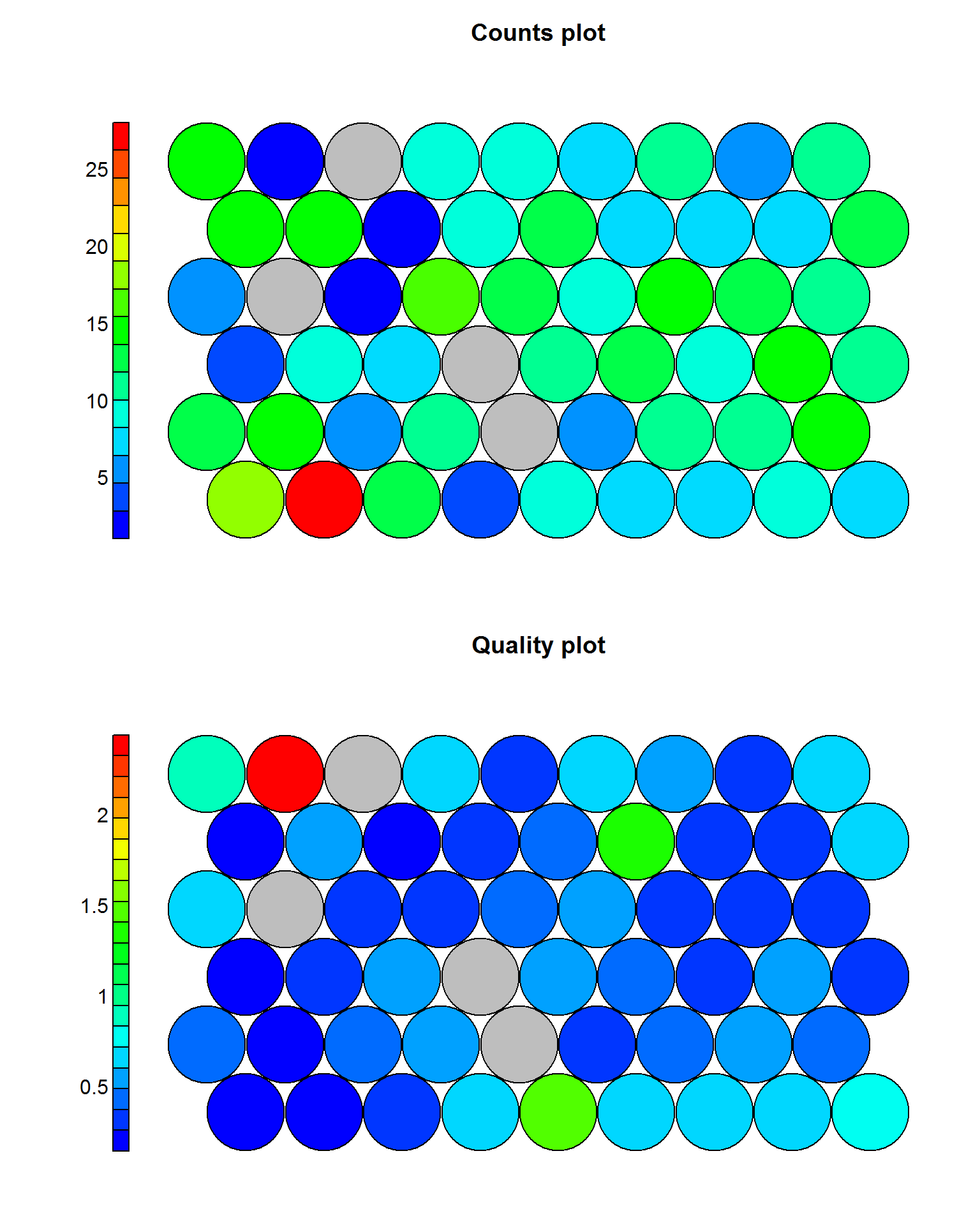


Рисунок 10.21: Карты SOM типа "counts" и "quality"

Тип карты "mapping" позволяет получить распределение объектов по узлам, удовлетворяющее любому заданному условию. Например, мы желаем выделить участки с низкой долей афроамериканцев (black - признак, который в настройке сети не участвовал). А также показать, как при этом распределяются доли участия отдельных исходных переменных:

colB <- ifelse(Boston$black <= 100, "red", "gray70")

par(mfrow = c(2, 1))

plot(som\_model, type = "mapping", col = colB, pch = 16)

plot(som\_model, type = "codes")

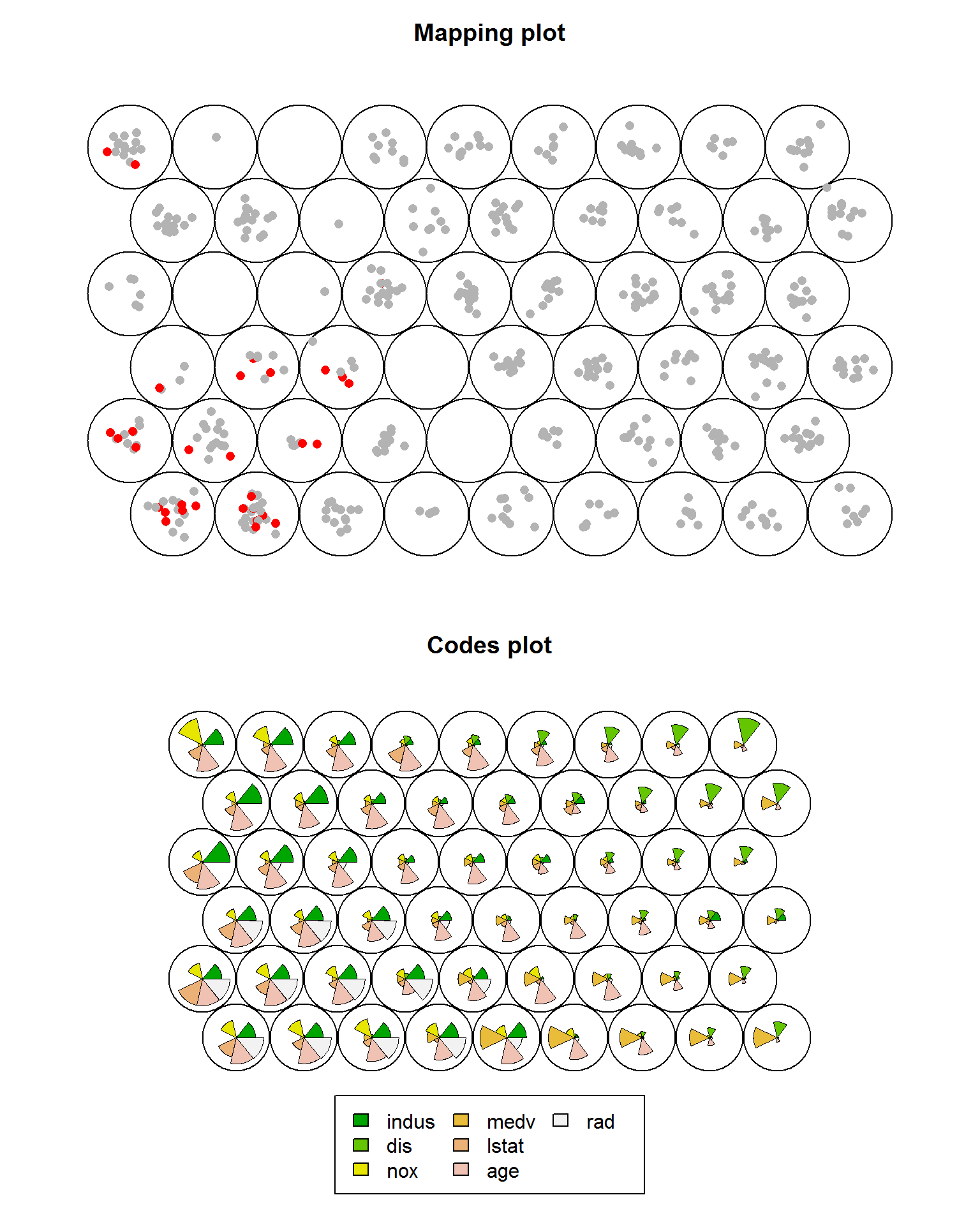


Рисунок 10.22: Карты SOM типа "mapping" и "codes"

Поскольку объединенная карта "codes" не всегда бывает хорошо интерпретируемой, можно получить карту распределения любого показателя в его стандартизованной или натуральной шкале:

par(mfrow = c(2, 1))

plot(som\_model, type = "property",

property = som\_model$codes[[1]][,1],

main = "indus - доля домов, продаваемых в розницу",

palette.name = coolBlueHotRed)

var\_unscaled <- aggregate(as.numeric(data\_train[, 3]),

by = list(som\_model$unit.classif),

FUN = mean, simplify = TRUE)[, 2]

plot(som\_model, type = "property", property = var\_unscaled,

main = "nox - содержание окислов азота",

palette.name = coolBlueHotRed)

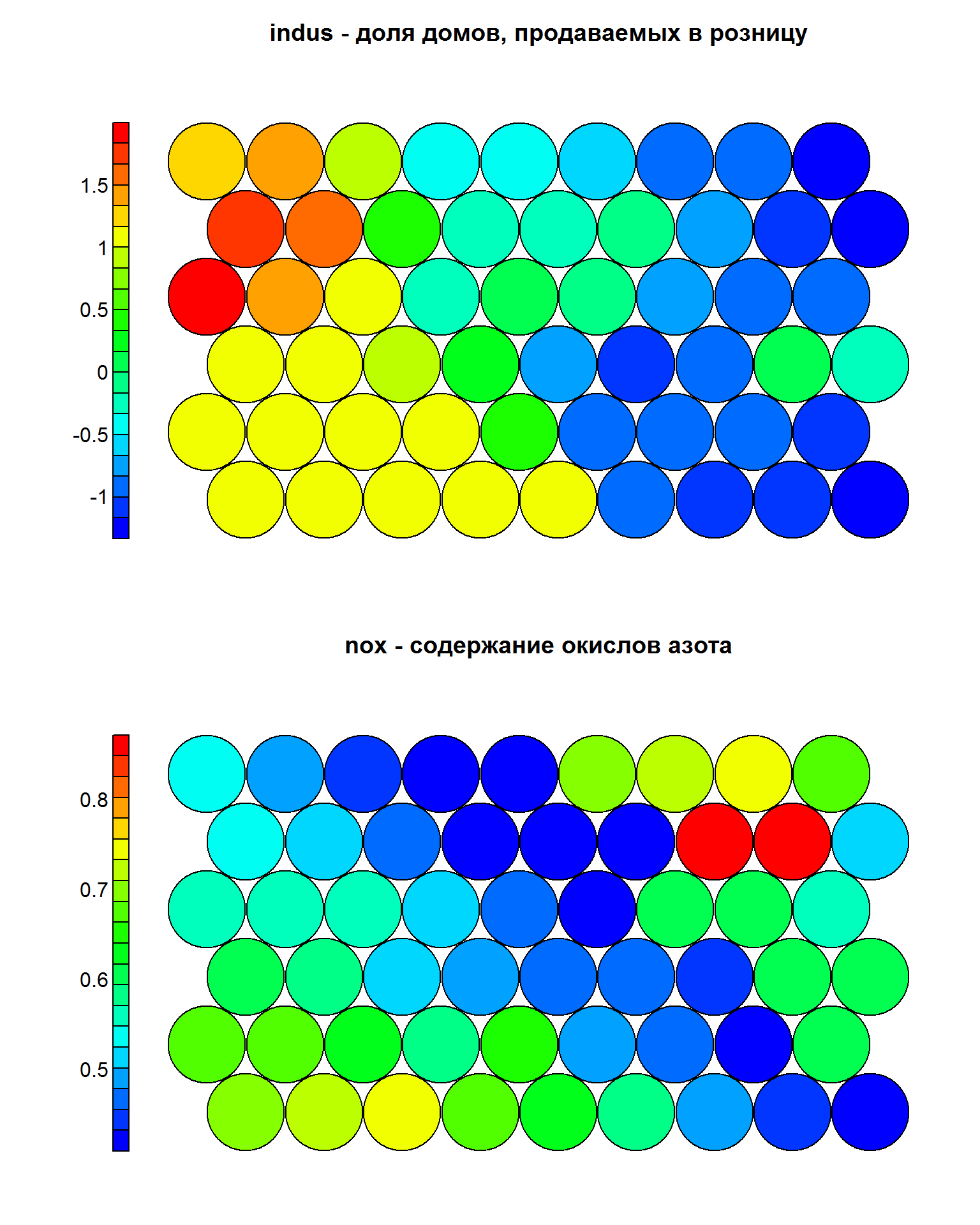


Рисунок 10.23: Карты SOM для стандартизованного и исходного показателя

Естественно, что значения активации каждого нейрона по каждому предиктору можно использовать для группировки узлов. Зададимся числом кластеров *k*=5

и выполним иерархическую кластеризацию (по умолчанию используются method = "complete" и distance = "euclidean"). Можно построить карты типа "mapping" (с метками объектов) или "codes" (с распределением доли вклада переменных). Остановимся на втором типе:

## Формируем матрицу "узлы × переменные"

mydata <- as.matrix(som\_model$codes[[1]])

# Используем иерархическую кластеризацию с порогом при k=5

som\_cluster <- cutree(hclust(dist(mydata)), 5)

# Определяем палитру цветов

pretty\_palette <- c("#1f77b4", '#ff7f0e', '#2ca02c',

'#d62728', '#9467bd', '#8c564b', '#e377c2')

# Показываем разными цветами кластеры узлов и переменные

plot(som\_model, type = "codes",

bgcol = pretty\_palette[som\_cluster])

add.cluster.boundaries(som\_model, som\_cluster)

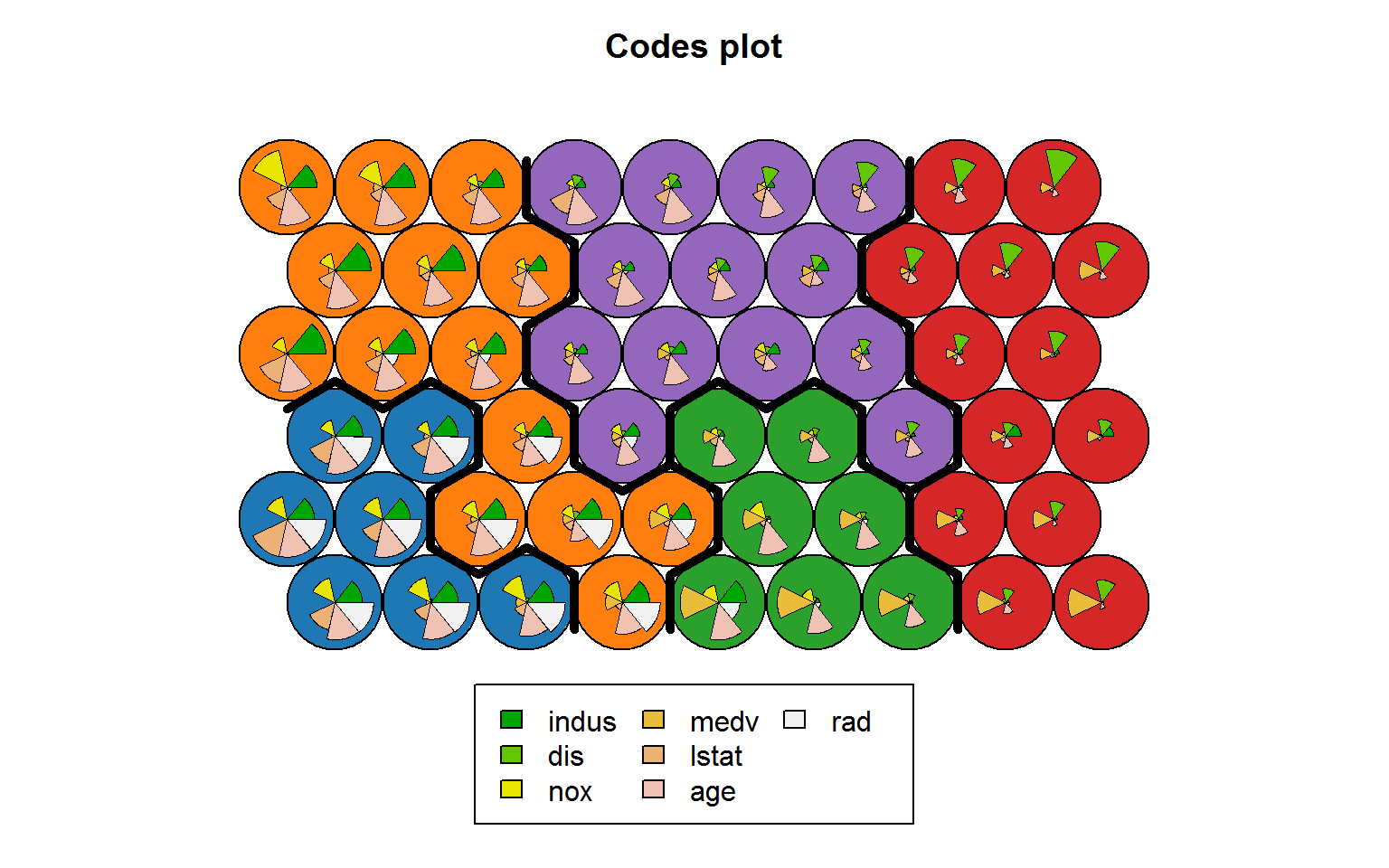
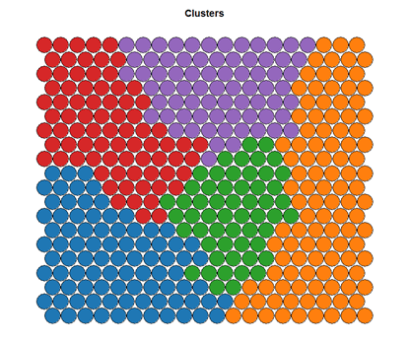


Рисунок 10.24: Кластеризация узлов карты SOM

Традиционно метод Кохонена рассматривается как эмпирический алгоритм, а выводы (в первую очередь, качественные) о структуре данных делаются на основе визуального анализа представленных карт. Основная трудность применения SOM, как и в случае анализа главных компонент, заключается в смысловой интерпретации топологии сети и связывании ее отдельных участков с некоторыми конкретными обобщениями из предметной области.

Однако алгоритм SOM нашел широкое применение в ГИС-технологиях, поскольку легко реализовать информационную цепочку от исходной таблицы к узлам решетки, а от них - к конкретным координатам на географической карте. Например, участники коллектива “Dublin R Users Group” использовали результаты переписи населения 2011 г., разбили территорию Дублина на 18500 маленьких площадок и описали проживающее на них население с использованием 767 переменных из 15 разделов. На ресурсах ([https://www.r-bloggers.com](https://www.r-bloggers.com/self-organising-maps-for-customer-segmentation-using-r/), [http://www.slideshare.net](http://www.slideshare.net/shanelynn/2014-0117-dublin-r-selforganising-maps-for-customer-segmentation-shane-lynn)) доступны полный комплект исходных данных, подробное описание и скрипты R. Основные результаты представлены на рис. [10.25](https://ranalytics.github.io/data-mining/105-Cohonen-Maps.html#fig:fig-10-25):



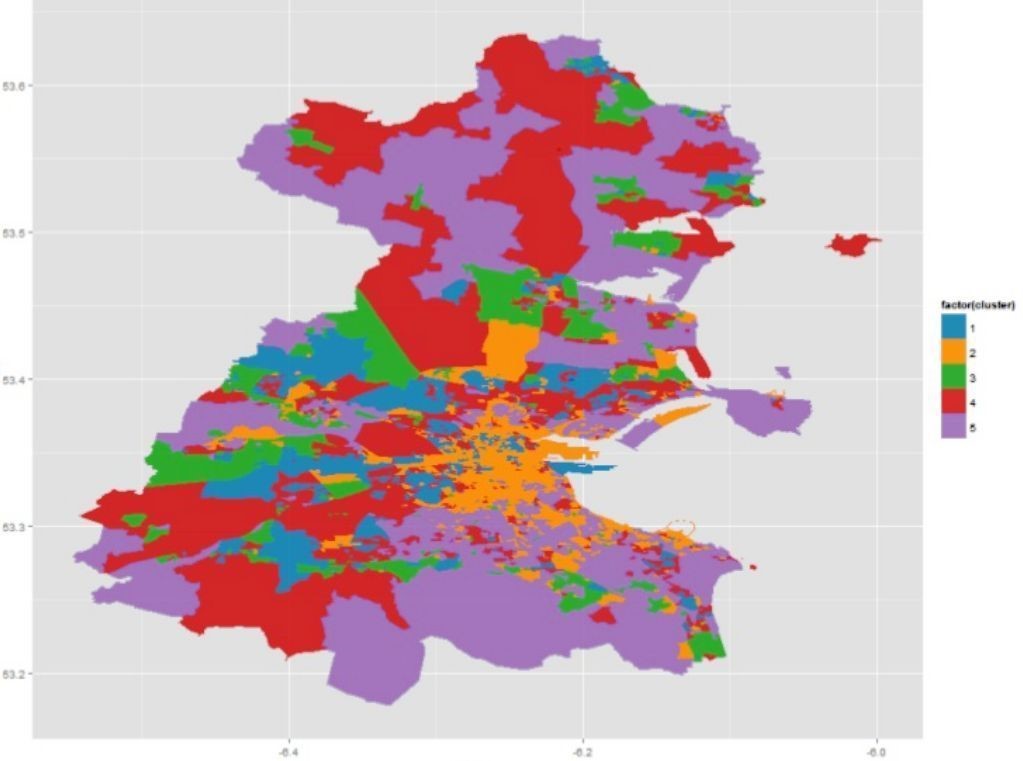


Рисунок 10.25: Кластерная форма карты SOM и ее отображение на карте Дублина (категории от 1 до 6 связываются с градациями жилой застройки и обеспеченности населения - от трущоб до благополучных элитных районов)

Похожий подход использован в программе ScanEx IMAGE Processor v3.6 (<http://www.scanex.ru>), руководство к которой содержит также достаточно подробное русскоязычное описание техники расчетов и формул, применяемых при обучении SOM.

[**https://loginom.ru/blog/som**](https://loginom.ru/blog/som)

[Платформа](https://loginom.ru/platform)[Цены](https://loginom.ru/platform/pricing)[Обучение](https://loginom.ru/skills)[Блог](https://loginom.ru/blog)[Вход](https://loginom.ru/u/hi)

* [Платформа](https://loginom.ru/platform)
* [Скачать бесплатную редакцию](https://loginom.ru/download)
* [Купить настольную редакцию](https://loginom.ru/platform/personal)
* [Запросить trial сервера](https://loginom.ru/trial-loginom-server)
* [Демостенды](https://demo.loginom.ru)
* [Документация](https://help.loginom.ru)
* [Демопримеры](https://samples.loginom.ru/)
* [Системные требования](https://loginom.ru/platform/system-requirements)
* [Цены](https://loginom.ru/platform/pricing)
* [Преимущества](https://loginom.ru/for)
* [Для аналитиков](https://loginom.ru/for/analyst)
* [Для IT-специалистов](https://loginom.ru/for/it)
* [Вопросы и ответы](https://qa.loginom.ru/)
* [Маркетплейс](https://marketplace.loginom.ru/)
* [Готовые решения](https://marketplace.loginom.ru/solutions/)
* [Интеграции](https://marketplace.loginom.ru/integrations/)
* [Библиотеки компонентов](https://marketplace.loginom.ru/libraries/)
* [Обучение](https://loginom.ru/skills)
* [Быстрый старт](https://loginom.ru/platform/quick-start)
* [Каталог курсов](https://skills.loginom.ru/)
* [Мастерская Loginom Skills](https://mastering.loginom.ru/)
* [Практикум по подготовке данных](https://data-preprocessing.loginom.ru/)
* [Клиенты](https://loginom.ru/clients)
* [Проекты](https://loginom.ru/blog/category/projects)
* [Отзывы](https://loginom.ru/reviews)
* [Блог](https://loginom.ru/blog)
* [Вики](https://wiki.loginom.ru/)
* [Loginom GPT](https://gpt.loginom.ru/)
* [Партнеры](https://loginom.ru/partners)
* [Партнерская программа](https://loginom.ru/partnership-program)
* [Партнерский портал](https://partners.loginom.ru/)
* [Академическая программа](https://loginom.ru/university-program)
* [Новости](https://loginom.ru/blog/category/education)
* [Вузы-участники](https://loginom.ru/university-program/participants)
* [Мероприятия](https://loginom.ru/university-program/events)
* [О компании](https://loginom.ru/company/about)
* [Контакты](https://loginom.ru/company/contacts)
* [Поддержка](https://loginom.ru/company/support)
* [Обратная связь](https://loginom.ru/company/feedback)

# Самоорганизующиеся карты Кохонена — математический аппарат

20 июля 2020

[2 комментария](https://loginom.ru/blog/som#comments)

https://loginom.ru/sites/default/files/styles/blur_full_header/public/header_images/som_1256x400.jpg?itok=qKOs0i2w

Самоорганизующиеся карты Кохонена – это одна из разновидностей нейросетевых алгоритмов. Этот алгоритм решает задачи кластеризации и проецирования многомерного пространства в пространство с более низкой размерностью. Он часто применяются для решения самых различных задач, от восстановления пропусков в данных до анализа и поиска закономерностей.

[Самоорганизующиеся карты](https://wiki.loginom.ru/articles/self-organizing-map.html) — это одна из разновидностей [нейросетевых алгоритмов](https://wiki.loginom.ru/articles/neural-network.html" \t "_blank). Основным отличием данной технологии от нейросетей, обучаемых по [алгоритму обратного распространения](https://wiki.loginom.ru/articles/back-propagation-algorithm.html), является то, что при обучении используется метод [обучения без учителя](https://wiki.loginom.ru/articles/unsupervised-learning.html), то есть результат обучения зависит только от структуры входных данных.

Нейронные сети данного типа часто применяются для решения самых различных задач, от восстановления пропусков в данных до анализа данных и поиска закономерностей, например, в финансовой задаче. В данной статье мы рассмотрим принципы функционирования и некоторые аспекты использования самоорганизующихся карт.

## Основы

Алгоритм функционирования самообучающихся карт (Self Organizing Maps — SOM) представляет собой один из вариантов [кластеризации](https://wiki.loginom.ru/articles/clustering.html) многомерных векторов. Примером таких алгоритмов может служить алгоритм [k-ближайших средних](https://wiki.loginom.ru/articles/k-means.html) (k-means). Важным отличием алгоритма SOM является то, что в нем все [нейроны](https://wiki.loginom.ru/articles/node.html) (узлы, центры классов…) упорядочены в некоторую структуру (обычно двумерную сетку).

При этом в ходе обучения модифицируется не только нейрон-победитель, но и его соседи, но в меньшей степени. За счет этого SOM можно считать одним из методов проецирования многомерного пространства в пространство с более низкой размерностью. При использовании этого алгоритма вектора, схожие в исходном пространстве, оказываются рядом и на полученной карте.

## Структура

SOM подразумевает использование упорядоченной структуры нейронов. Обычно используются одно и двумерные сетки. При этом каждый нейрон представляет собой n-мерный вектор-столбец w=[w1,w2,…,wn]Tw=[w1​,w2​,…,wn​]T, где nn определяется размерностью исходного пространства (размерностью входных векторов). Применение одно и двумерных сеток связано с тем, что возникают проблемы при отображении пространственных структур большей размерности (при этом опять возникают проблемы с понижением размерности до двумерной, представимой на мониторе).

Обычно нейроны располагаются в узлах двумерной сетки с прямоугольными или шестиугольными ячейками. При этом, как было сказано выше, нейроны также взаимодействуют друг с другом. Величина этого взаимодействия определяется расстоянием между нейронами на карте. На рисунке 1 дан пример расстояния для шестиугольной и четырехугольной сеток.

Рисунок 1. Расстояние между нейронами на карте для шестиугольной (а) и четырехугольной (б) сеток. При этом легко заметить, что для шестиугольной сетки расстояние между нейронами больше совпадает с евклидовым расстоянием, чем для четырехугольной сетки.

Количество нейронов в сетке определяет степень детализации результата работы алгоритма, и в конечном счете от этого зависит точность [обобщающей способности](https://wiki.loginom.ru/articles/generalization-ability.html) карты.

## Начальная инициализация карты

При реализации алгоритма SOM заранее задается конфигурация сетки (прямоугольная или шестиугольная), а также количество нейронов в сети. Некоторые источники рекомендуют использовать максимально возможное количество нейронов в карте. При этом начальный радиус обучения (neighborhood в англоязычной литературе) в значительной степени влияет на способность обобщения при помощи полученной карты.

В случае, когда количество узлов карты превышает количество примеров в обучающей выборке, то успех использования алгоритма в большой степени зависит от подходящего выбора начального радиуса обучения. Однако, в случае, когда размер карты составляет десятки тысяч нейронов, то время, требуемое на обучение карты, обычно бывает слишком велико для решения практических задач. Таким образом необходимо достигать допустимого компромисса при выборе количества узлов.

Перед началом обучения карты необходимо проинициализировать весовые коэффициенты нейронов. Удачно выбранный способ инициализации может существенно ускорить обучение, и привести к получению более качественных результатов. Существуют три способа инициирования начальных весов.

* Инициализация случайными значениями, когда всем весам даются малые случайные величины.
* Инициализация примерами, когда в качестве начальных значений задаются значения случайно выбранных примеров из обучающей выборки
* Линейная инициализация. В этом случае веса инициируются значениями векторов, линейно упорядоченных вдоль линейного подпространства, проходящего между двумя главными собственными векторами исходного набора данных. Собственные вектора могут быть найдены например при помощи процедуры Грама-Шмидта.

## Обучение

Обучение состоит из последовательности коррекций векторов, представляющих собой нейроны. На каждом шаге обучения из исходного набора данных случайно выбирается один из векторов, а затем производится поиск наиболее похожего на него вектора коэффициентов нейронов. При этом выбирается нейрон-победитель, который наиболее похож на вектор входов. Под похожестью в данной задаче понимается расстояние между векторами, обычно вычисляемое в евклидовом пространстве. Таким образом, если обозначим нейрон-победитель как c, то получим

∣∣x−wc∣∣=min⁡i{∣∣x−wi∣∣}∣∣x−wc​∣∣=mini​{∣∣x−wi​∣∣}

После того, как найден нейрон-победитель, производится корректировка весов нейросети. При этом вектор, описывающий нейрон-победитель и вектора, описывающие его соседей в сетке перемещаются в направлении входного вектора. Это проиллюстрировано на рисунке 2 для двумерного вектора.

Рисунок 2. Подстройка весов нейрона победителя и его соседей. Координаты входного вектора отмечены крестом, координаты узлов карты после модификации отображены серым цветом. Вид сетки после модификации отображен штриховыми линиями.

При этом для модификации весовых коэффициентов используется формула:

wi(t+1)=wi(t)+hci(t)∗[x(t)−w(t)],wi​(t+1)=wi​(t)+hci​(t)∗[x(t)−w(t)],

где tt обозначает номер эпохи (дискретное время). При этом вектор x(t)x(t) выбирается случайно из обучающей выборки на итерации tt. Функция h(t)h(t) называется функцией соседства нейронов и представляет собой невозрастающую функцию от времени и расстояния между нейроном-победителем и соседними нейронами в сетке. Эта функция разбивается на две части: собственно функцию расстояния и функции скорости обучения от времени, где ;r;r определяет положение нейрона в сетке.

Обычно применяется одна из двух функций от расстояния:

* простая константа — h(d,t)={const,d≤σ(t)0,d>σ(t)h(d,t)={const,d≤σ(t)0,d>σ(t)​
* гаусова функция — h(d,t)=e−d2σ(t))h(d,t)=e−2σ(t))d​

При этом лучший результат получается при использовании Гауссовой функции расстояния.

Часто эту величину называют радиусом обучения, который выбирается достаточно большим на начальном этапе обучения и постепенно уменьшается так, что в конечном итоге обучается один нейрон-победитель. Наиболее часто используется функция, линейно убывающая от времени.

Рассмотрим теперь функцию скорости обучения a(t)a(t). Эта функция также представляет собой функцию, убывающую от времени. Наиболее часто используются два варианта этой функции: линейная и обратно пропорциональная времени вида a(t)= A t+Ba(t)= t+B A​, где AA и BB — это константы.

Применение этой функции приводит к тому, что все вектора из обучающей выборки вносят примерно равный вклад в результат обучения.

Обучение состоит из двух основных фаз: на первоначальном этапе выбирается достаточно большое значение скорости обучения и радиуса обучения, что позволяет расположить вектора нейронов в соответствии с распределением примеров в выборке, а затем производится точная подстройка весов, когда значения параметров скорости обучения много меньше начальных. В случае использования линейной инициализации первоначальный этап грубой подстройки может быть пропущен.

## Применение алгоритма

Так как алгоритм SOM сочетает в себе два основных направления – векторное квантование и проецирование, то можно найти и основные применения этого алгоритма. Данную методику можно использовать для поиска и анализа закономерностей в исходных данных. После того, как нейроны размещены на карте, полученная карта может быть отображена. Рассмотрим различные способы отображения полученной карты.

### Раскраска, порожденная отдельными компонентами

При данном методе отрисовки полученную карту можно представить в виде слоеного пирога. Каждый слой которого представляет собой раскраску, порожденную одной из компонент исходных данных. Полученный набор раскрасок может использоваться для анализа закономерностей, имеющихся между компонентами набора данных.

После формирования карты мы получаем набор узлов, который можно отобразить в виде двумерной картинки. При этом каждому узлу карты можно поставить в соответствие участок на рисунке, четырех или шестиугольный, координаты которого определяются координатами соответствующего узла в решетке. Теперь для визуализации осталось только определить цвет ячеек этой картинки. Для этого и используются значения компонент.

Самый простой вариант — использование градаций серого. В этом случае ячейки, соответствующие узлам карты, в которые попали элементы с минимальными значениями компонента или не попало вообще ни одной записи, будут изображены черным цветом, а ячейки, в которые попали записи с максимальными значениями такого компонента, будут соответствовать ячейки белого цвета. В принципе, можно использовать любую градиентную палитру для раскраски.

Полученные раскраски в совокупности образуют атлас, отображающий расположение компонент, связи между ними, а также относительное расположение различных значений компонент.

### Отображение кластеров

Кластером будет являться группа векторов, расстояние между которыми внутри этой группы меньше, чем расстояние до соседних групп. Структура кластеров при использовании алгоритма SOM может быть отображена путем визуализации расстояния между опорными векторами (весовыми коэффициентами нейронов).

При использовании этого метода чаще всего используется унифицированная матрица расстояний (u-matrix). Для этого вычисляется расстояние между вектором весов нейрона в сетке и его ближайшими соседями. Затем эти значения используются для определения цвета, которым этот узел будет отрисован.

Обычно используют градации серого, причем чем больше расстояние, тем темнее отрисовывается узел. При таком использовании узлам с наибольшим расстоянием между ними и соседями соответствует черный цвет, а близлежащим узлам — белый.

[Литература](https://loginom.ru/blog/som)

<http://www/csee/umbc/edu/nicolas/clustering/p264-jain.pdf>

***Сети векторного квантования, обучающиеся с***

***учителем (LVQ-сети)***

*LVQ*-сети основаны на обучающемся векторном квантовании (*LVQ* —

*Learning Vector Quantization*) [3] и представляют собой слой Кохонена,

*обучающийся с учителем*. Для построения *LVQ*-сети задается количество

кластеров (нейронов) *n* , количество классов *m* ( *n* ³ *m* ) и принадлежность

каждого кластера определенному классу. Например, анализы пациентов

разделяются на 5 кластеров, 2 из которых соответствуют здоровым людям, а

3 — больным. Разделить кластеры по классам можно в тех же пропорциях,

что и распределение примеров соответствующих классов в обучающей

выборке. Соответствие между номерами кластеров и номерами классов

может быть произвольным. Для простоты интерпретации результатов работы

сети целесообразно пронумеровать кластеры последовательно. В нашем

примере первые два кластера соответствуют классу здоровых людей, а

последующие 3 — классу больных. В процессе обучения *LVQ*-сети веса

нейронов настраиваются с учетом принадлежности обучающих примеров и

кластеров одному классу. Обученная *LVQ*-сеть производит кластеризацию

входных векторов с учетом классов. Например, мы \_\_\_\_\_\_\_\_\_\_узнаем, что анализы

конкретного пациента принадлежат одному из кластеров, соответствующих