2.3. Моделирование данных с помощью нелинейных многообразий

Подпространство, натянутое на *m'* главных компонент обладает свойством «минимума остаточной дисперсии» – средний квадрат расстояния от точек данных до этого подпространства минимален среди всех других линейных подпространств размерности *m'*.

Кажется перспективной идея построения «главных поверхностей» – нелинейных многообразий, обладающих тем же оптимальным свойством. В литературе можно встретить работы [58,66,67], в которых строятся такие поверхности с использованием, например, метода максимального правдоподобия.

Если вид параметрической зависимости координат точек многообразия неизвестен (нет априорных соображений о структуре данных), то задача построения неограниченного нелинейного многообразия, то есть вычисление координат каждой из его точек, является весьма трудоемкой с точки зрения вычислений.

Мы с самого начала будем следовать пути, на котором моделирующее многообразие будет предполагаться ограниченным и задается в конечном числе своих точек. Другими словами, мы будем строить *точечную аппроксимацию многообразия*.

Допустим, что мы имеем размещенную в пространстве данных *сетку узлов*, на которую и будет натягиваться искомое многообразие. Для этого вводится определенная процедура *интерполяции* между узлами.

Сразу заметим, что задания одних только положений узлов в пространстве данных недостаточно для восстановления многообразия. Мы должны, кроме того, обладать информацией о том, какие узлы являются на нем *соседними*. То есть на конечном множестве узлов должны быть введены *отношения соседства*. Для излагаемого ниже алгоритма SOM (Self-Organizing Maps – самоорганизующиеся карты Кохонена) необходимо знать не только какие узлы являются соседними, но также и какие являются *вторыми* соседями, *третьими* соседями и т.д.

Рассмотрим, например двумерное многообразие. Будем считать, что узлы образуют прямоугольную или гексагональную сетку (см. рис.17). Тогда отношения соседства определяются естественным образом – можно считать, что каждый узел (кроме крайних) на прямоугольной сетке имеет четыре первых, восемь вторых и т.д. соседей, на гексагональной – шесть первых, двенадцать вторых и т.д. соседей. Если многообразие является трехмерным, а сетка прямоугольной, то каждый узел имеет, соответственно, шесть первых соседей, и т.д.

Рассмотрим два алгоритма построения сетки – алгоритм *самоорганизующихся карт Кохонена* (SOM) и его модификации, а также алгоритм построения *упругих сеток*.

2.4. Алгоритм SOM и его модификации

Самоорганизующиеся карты Кохонена (SOM) – это модифицированный алгоритм *линейного векторного квантования данных*, то есть представления N точек данных с помощью меньшего числа точек-образцов (*samples*). Каждый из образов представляет и заменяет собой локальное сгущение данных. В результате такой замены данные представляются с определенной ошибкой аппроксимации – среднеквадратичного расстояния от точки данных до ближайшего к ней образца:

$$MSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_i - y_{BMU}(X_i))^2},$$

где $y_{BMU}(X_i)$ – ближайший к точке данных X_i образец.

«Изюминкой» метода SOM оказалось то, что получаемая при его применении система узлов (образцов) оказывается определенным образом упорядочена. Узлы могут быть представлены в виде прямоугольной или гексагональной сетки. Соседние на этой сетке узлы в результате действия алгоритма SOM оказываются соседними в пространстве данных, что дает после размещения точек данных по ближайшим узлам в случае двумерной или трехмерной сетки возможность визуализировать данные.

Опишем исходный вариант алгоритма SOM.

1. Сетка узлов инициализируется – размещается в пространстве данных. Простейшим вариантом является случайное расположение узлов, другой вариант – размещение сетки в пространстве, натянутом на главные компоненты. Существуют и другие, более или менее эффективные схемы инициализации (см., например, [70]).

2. Выбирается случайным образом или по порядку точка данных X_i.

3. Среди всех узлов сетки выбирается ближайший к точке X_i. Обозначим его радиус вектор через y_{BMU} (BMU – Best Matching Unit).

4. *Все* узлы сетки двигаются по направлению к X_i по правилу:

$$(y_j)' = y_j + h\big(r(y_j, y_{BMU}), t\big)(X_i - y_j), j = 1\ldots p,$$

где p – количество узлов, $h(x,t)$ – так называемая *функция соседства* (neighborhood function), $r(y_1,y_2)$ – расстояние между узлами y_1 и y_2, но не в пространстве данных, а согласно введенным на сетке отношениям соседства (то есть, если y_1 и y_2 являются ближайшими соседями, то $r(y_1,y_2)=1$, если вторыми, то $r(y_1,y_2)=2$, и т.д.), t – номер итерации.

5. Шаги 2-4 алгоритма повторяются до тех пор, пока либо не будет достигнута определенная точность, или так, чтобы каждая точка данных несколько раз поучаствовала в процессе адаптации узлов (то есть kN итераций, где k – число порядка нескольких единиц).

Функция соседства $h(x,t)$ выбирается таким образом, чтобы достигать максимума при $x = 0$ и монотонно спадать с ростом x. Наиболее популярными являются гауссов вид функции соседства и так называемая bubble-function.

Гауссов вид функции:

$$h(x,t) = \alpha(t)\exp\left(-\frac{1}{2}\frac{x^2}{\sigma^2(t)}\right),$$

Bubble-function:

$$h(x,t) = \begin{cases} \alpha(t), x \le \sigma(t) \\ 0, \quad x > \sigma(t) \end{cases}.$$

Здесь $\alpha(t)$ – *темп* обучения, $\sigma(t)$ – *радиус захвата соседей* (neighborhood width). В изначальном варианте SOM эти функции не зависят от условий обучения и просто монотонно уменьшаются от некоторого начального значения до нуля, например, по линейному закону:

$$\alpha(t) = \alpha_0(1 - t/n_{iter}),$$

$$\sigma(t) = \sigma_0(1 - t/n_{iter}),$$

где n_{iter} – число итераций, α_0 – число порядка десятых или сотых, σ_0 – число порядка нескольких единиц.

Обсудим, что происходит в результате каждой итерации алгоритма. Больше всего испытывает смещение узел y_{BMU}. Он сдвигается в направлении выбранной точки данных на величину $\alpha(t)$. Остальные узлы испытывают тем меньшие смещения, чем они дальше от y_{BMU} в указанном выше «сеточном» смысле. Заметные смещения испытывают узлы, которые являются для y_{BMU} соседями порядка $\sigma(t)$. Так как функции $\alpha(t)$ и $\sigma(t)$ убывают со временем, то и темп обучения и число узлов, участвующих в коллективном движении, уменьшается. В результате сетка модифицируется все меньше и меньше и в конечном счете «застывает».

Обычно настройку сетки производят в два этапа:

Этап 1. Ordering. На этом этапе обычно выбирается $\alpha_0 \approx 0.1$, $n_{iter} \approx N$, σ_0 – выбирается так, чтобы в движении участвовало более половины узлов.

Этап 2. Fine-tuning. На этом этапе обычно выбирается $\alpha_0 \approx 0.01$, $n_{iter} \approx 10N$, σ_0 – выбирается так, чтобы в движении участвовало 2-3 узла.

Алгоритм SOM обеспечивает случайное движение узлов таким образом, что среднее расстояние от точки данных до ближайшего к ней узла постоянно уменьшается. При этом узлы упорядочиваются согласно введенным на системе узлов отношениям соседства. Полученная сетка узлов в

результате становится более или менее гладкой, причем тем более гладкой, чем большие значения $\sigma(t)$ участвовали в настройке.

Со времени своего создания для алгоритма SOM было предложено множество модификаций, преследующих те или иные цели. Модификации алгоритма осуществлялись двумя основными техническими приемами:

а) изменение направления движения узлов – когда узел помимо того, что двигается по направлению к точке данных, смещается еще и в другом, выбираемом из тех или иных соображений, направлении;

б) настройка радиуса захвата соседей – когда величина $\sigma(t)$ становится разной для разных условий, в которых находится выбранный узел y_{BMU}.

Целью модификаций являлось ускорение работы алгоритма, улучшение точности аппроксимации при заданном числе узлов, динамическое (по ходу настройки) изменение числа узлов, улучшение гладкости или регулярности сетки.

Остановимся лишь на нескольких характерных модификациях.

Алгоритм Batch SOM [65]

Идея этой модификации – осуществлять движение узлов не поодиночке, а разом, за один такт. Последовательность действий – следующая:

1. Сетка узлов инициализируется.

2. Все множество данных разбивается на подмножества K_i, $i = 1\ldots p$, p – число узлов. Для точек из подмножества K_i ближайшим узлом сетки является узел y_i. Назовем такое подмножество *таксоном* узла y_i. В результате все точки данных распределяются «по ближайшим узлам».

3. Вычисляются центры таксонов $\omega_i = \frac{1}{n_i} \sum_{X_i \in K_i} X_i$.

4. Положение каждого узла модифицируется по правилу

$y_i = \omega_i + \varepsilon\alpha_i$,

где α_i – среднее центров таксонов узлов-первых соседей, ε – некоторый параметр порядка десятых единицы. Таким образом соседние узлы «тянут» настраиваемый узел к себе. В результате сетка становится более регулярной.

5. Шаги 2-4 повторяются определенное количество раз.

Алгоритм регуляризации SOM [60]

Идея этой модификации – сделать сетку более гладкой, локально спрямить слишком большие ее изгибы. Для этого вводится понятие «идеальное положение узла» – для одномерной сетки это точка $\tilde{y}_j^1$ ортогональной проекции узла на прямую, соединяющую два соседних узла (см. рис.32а)

Действие алгоритма вполне аналогично стандартному SOM, за исключением того, что правило настройки узла в случае прямоугольной сетки теперь имеет вид

$$(y_j)' = y_j + h(r,t)(X_i - y_j) + \tilde{h}(t)(\tilde{y}_j^1 - y_j) + \tilde{h}(t)(\tilde{y}_j^2 - y_j),$$

где $\tilde{y}_j^1, \tilde{y}_j^2$ – ортогональные проекции на прямые, соединяющие верхнего и нижнего, левого и правого соседей соответственно, $\tilde{h}(t)$ – функция, монотонно убывающая с номером итерации. Таким образом, кроме того, что узел смещается в направлении точки данных, он также испытывает смещение в сторону точек $\tilde{y}_j^1$ и $\tilde{y}_j^2$, что приводит к частичному спрямлению линии, соединяющей три соседних узла и к более гладкой сетке.

Алгоритм Density Tracking SOM [62]

Идея этой модификации (похожей на Batch SOM) – сделать так, чтобы в областях скопления данных оказалось больше узлов, чем в более «разреженных» областях:

1. Сетка узлов инициализируется.
2. Как и в алгоритме Batch SOM, все множество данных разбивается на подмножества K_i, $i = 1…p$, p – число узлов. Для точек из подмножества K_i ближайшим узлом сетки является узел y_i.
3. Вычисляются центры таксонов $\omega_i = \frac{1}{n_i} \sum_{X_i \in K_i} X_i$;
4. Рассчитывается количество точек данных в каждом таксоне.
5. Положение каждого узла модифицируется по правилу

$$y_i = \omega_i + \varepsilon\omega_j,$$

где ω_j – центр одного из соседних таксонов, в котором содержится максимальное (среди всех соседей) количество точек, ε – некоторый параметр порядка десятых единицы. Узел смещается в сторону более «весомого» соседа, в окрестности которого содержится большее количество точек данных.

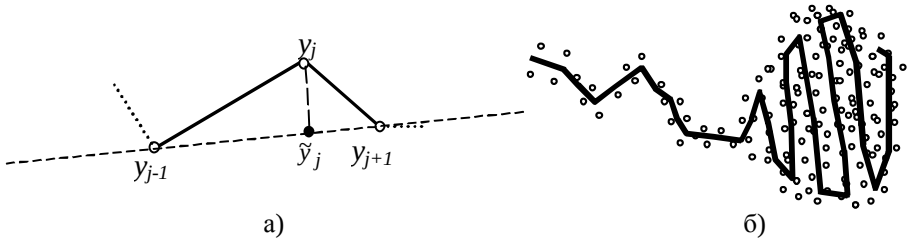
6. Шаги 2-5 повторяются определенное количество раз.

Алгоритм Adaptive SOM (AdSOM)

В работе [64] авторы указывают на общую проблему, возникающую при моделировании многомерного множества точек с помощью многообразий меньшей размерности. Для того, чтобы воспроизвести особенности многомерного множества, многообразие стремится «свернуться», образовать большое количество складок. При этом часть изгибов многообразия обусловлена самой структурой точек данных, часть – тем обстоятельством,

что размерность многообразия не соответствует размерности множества (см. рис.32б).

В случае SOM это означает, что для некоторых точек данных ближайший узел сетки и второй по близости не являются соседями на сетке. Такие точки называются *неустойчивыми*. Отношение числа неустойчивых точек к общему количеству точек называется *топографической ошибкой* картирования.



а) б)

Рис. 32. а) Иллюстрация к алгоритму регуляризации SOM. Точка $\tilde{y}_j$ – проекция на прямую,
соединяющую два соседних узла.
б) Иллюстрация к алгоритму Adaptive SOM. Часть изгибов SOM многообразия обусловлена самой структурой точек данных, часть – тем обстоятельством, кривая стремится аппроксимировать множество большей размерности. Большая часть точек в шаровом скоплении справа окажутся *неустойчивыми*.

Модификация AdSOM сводит величину топографической ошибки практически к нулю за счет локальной настройки радиуса захвата соседей. В результате карта оказывается менее «изогнутой», что приводит к несколько худшей точности аппроксимации (такая жертва неизбежна – это и составляет суть дилеммы «регулярность-точность»).

Настройка радиуса захвата соседей производится следующим образом:

1. В алгоритме AdSOM для каждого из узлов y_k назначается свой собственный радиус завата соседей σ_k. Далее, как и в стандартном SOM выбирается точка данных X_i. Для нее определяются *два* ближайших узла y_i и y_j. Если эти узлы являются первыми соседями, то точка – устойчива, и для нее производится обычная процедура движения узлов. Если нет, то производится настройка всех величин σ_k следующим образом:

$$\sigma_k = \begin{cases} r(y_i, y_j), & \max\{r(y_k, y_j), r(y_i, y_k)\} \le r(y_i, y_j) \\ r(y_i, y_j) - s, & s < r(y_i, y_j), \quad s = \min\{r(y_k, y_j), r(y_i, y_k)\} \\ 1, & \text{иначе} \end{cases}$$

где $r(y_i,y_j)$ – упомянутое выше «сеточное» расстояние. Иными словами, радиус захвата соседей равен сеточному расстоянию между узлами y_i и y_j для всех узлов, находящихся на сетке между ними и линейно спадает до 1 вне области их «влияния».

2. Время от времени (каждые n_{rec} итераций) значения всех σ_k пересчитываются:

$$(\sigma_k)' = (\sigma_k)^{\beta},$$

где $\beta < 1$ – еще один параметр метода.

Иерархические алгоритмы SOM

В некоторых работах [52,63,71] сетка предполагается «растущей», то есть количество узлов время от времени меняется.

Самый простой алгоритм – количество узлов удваивается после окончания процесса настройки, если в результате не достигнута необходимая точность. Новые узлы помещаются в промежутки между уже настроенными узлами и сетка донастраивается.

Более гибкий метод – вычисление «растяжений» ребер сетки вдоль отдельных строк и столбцов сетки. В том ряду (вертикальном или горизонтальном), где суммарное растяжение оказалось наибольшим, ребра делятся – посередине вставляются новые узлы. Сетка вновь донастраивается. Исследователь наблюдает за графиком изменения ошибки аппроксимации, и останавливает процесс деления ребер когда график ошибки выйдет на «плато» – свою пологую часть. Количество узлов сетки, которое отвечает началу пологой части графика ошибки считается «правильным» (для данного уровня точности).

Сделаем некоторые выводы. Почти во всех модификациях SOM вводятся параметры, с помощью которых сетка делается более регулярной – более гладкой или более равномерной, или лучше соответствующей локальной структуре данных. Вместе с тем ни в одной модификации нет указания на вид меры оптимальности построенной сетки. В следующем разделе мы явно введем такую меру, в результате оптимизации которой получится *алгоритм построения упругих сеток*. В алгоритме появятся два параметра – один будет явно «регулировать» гладкость построенной сетки, другой – ее равномерность.

Сначала мы рассмотрим случай двумерной прямоугольной сетки. Затем приведем алгоритм построения произвольной упругой сетки и рассмотрим различные способы ее настройки.

2.5. Алгоритм построения упругих сеток

2.5.1. Прямоугольная сетка

Рассмотрим двумерную прямоугольную сетку узлов, в которой p узлов по горизонтали, q узлов по вертикали. Перенумеруем узлы этой сетки с помощью двух индексов – y^{ij}, $i = 1\ldots p$, $j = 1\ldots q$. Сетка должна обладать следующими свойствами:

1) *Свойство близости к точкам данных*. Сетка должна быть в каком-то смысле аналогична плоскости первых двух главных компонент – оптимальной в смысле минимума среднего квадрата расстояния от точек данных до ближайшего узла при определенных ограничениях на свойства сетки.

2) *Свойство упругости по отношению к растяжению*. Это свойство до некоторой степени обеспечит *равномерность* сетки.

3) *Свойство упругости по отношению к изгибу*. Это свойство до некоторой степени обеспечит *гладкость* результирующего многообразия.

Как и в алгоритме Batch SOM разобьем все множество данных X на $p \times q$ подмножеств K_{ij} ($i = 1\ldots p$, $j = 1\ldots q$) (таксонов), в пределах каждого из которых точки подмножества оказываются ближе к узлу сетки y^{ij}, чем к какому-нибудь другому узлу. Обозначим это обстоятельство следующим образом

$$K_{ij} = \left\{ x \in X \left\| y^{ij} - x \right\|^2 \to \min_{i,j} \right\}$$

В качестве меры близости сетки к точкам данных выберем величину среднего (на одну точку данных) квадрата расстояния от точки до ближайшего узла сетки.

Каждый узел сетки (кроме граничных) имеет четырех соседей, с которыми он соединяется «ребром». Чем больше средняя (на один узел) длина ребра, тем сильнее сетка «растянута», поэтому мы должны по возможности минимизировать эту величину. Таким образом, в минимизируемый функционал должны войти разности между положениями соседних узлов. Степень изогнутости определим с помощью точечной оценки величины второй производной (с помощью так называемых *вторых разностей*). В результате получим следующий функционал «качества» построенной сетки:

$$D = \frac{D_1}{|X|} + \lambda \frac{D_2}{pq} + \mu \frac{D_3}{pq} \to \min$$

где $|X|$ - число точек в X; λ, μ - коэффициенты упругости, отвечающие за растяжение и изогнутость стеки соответственно; D_1, D_2, D_3 – слагаемые, отвечающие за свойства сетки, именно:

$D_1 = \sum_{ij} \sum_{X_k \in K_{ij}} \left\| X_k - y^{ij} \right\|^2$ – является мерой близости расположения узлов

сетки к данным. Здесь K_{ij} –подмножества точек из X, для которых узел сетки y^{ij} является ближайшим (*таксоны*);

$$D_2 = \sum_{i=1}^{p}\sum_{j=1}^{q-1}\left\|y^{ij} - y^{i,j+1}\right\|^2 + \sum_{i=1}^{p-1}\sum_{j=1}^{q}\left\|y^{ij} - y^{i+1,j}\right\|^2$$ – мера растянутости сетки;

$$D_3 = \sum_{i=1}^{p}\sum_{j=2}^{q-1}\left\|2y^{ij} - y^{i,j-1} - y^{i,j+1}\right\|^2 + \sum_{i=2}^{p-1}\sum_{j=1}^{q}\left\|2y^{ij} - y^{i-1,j} - y^{i+1,j}\right\|^2$$ – мера изогнутости (кривизны) сетки.

Отметим, что границы суммирования выбраны так, чтобы в функционале D_2 ребро не входило в сумму дважды.

Пусть метрика является евклидовой. В этом случае функционал D является квадратичным по положениям узлов y^{ij}, это значит, что при заданном разбиении множества точек данных на таксоны для его минимизации потребуется решить систему линейных уравнений размерами $pq \times pq$. Следовательно, эффективным методом минимизации функционала D окажется такой алгоритм:

Шаг 0. Узлы сетки так или иначе располагаются в пространстве данных.

Шаг 1. При заданных положениях узлов сетки производится разбиение множества данных на таксоны – подмножества K_{ij} .

Шаг 2. При заданном разбиении множества точек данных на таксоны производится минимизация функционала D.

Шаги 1 и 2 повторяются до тех пор пока функционал D не перестанет изменяться (в пределах заданной точности). Процесс сходится, поскольку на каждом этапе минимизации величина D, очевидно, будет уменьшаться, вместе с тем она ограничена снизу нулем (величина D неотрицательна). Более того, он сходится за конечное число шагов, поскольку число вариантов разбиения точек данных на таксоны конечно (хотя и может быть весьма велико).

Выпишем явно коэффициенты матрицы системы линейных уравнений, которую необходимо решать на каждой итерации алгоритма минимизации. Приводимые ниже выкладки весьма просты, хотя и громоздки.

Непосредственное дифференцирование дает:

$$\frac{1}{2}\frac{\partial D}{\partial y^{kl}} = a_{kl}^{(-2)} y^{k-2,l} + a_{kl}^{(-1)} y^{k-1,l} + a_{kl} y^{k,l} + a_{kl}^{(+1)} y^{k+1,l} + a_{kl}^{(+2)} y^{k+2,l} +$$

$$+ b_{kl}^{(-2)} y^{k,l-2} + b_{kl}^{(-1)} y^{k,l-1} + b_{kl}^{(+1)} y^{k,l+1} + b_{kl}^{(+2)} y^{k,l+2} - \sum_{x \in K_{kl}} x$$

где

$$a_{kl}^{(-2)} = \frac{\mu}{pq}(1-\delta_{l,2})(1-\delta_{l,1}),$$

$$a_{kl}^{(-1)} = \frac{\lambda}{pq}(\delta_{l,1} - 1) + \frac{2\mu}{pq}(\delta_{l,2} - 1)(1 - \delta_{l,1}),$$

$$a_{kl} = \frac{n_{kl}}{|X|} + \frac{\lambda}{pq}\left[4 - \delta_{k,1} - \delta_{k,p} - \delta_{l,1} - \delta_{l,q}\right] +$$

$$+ \frac{\mu}{pq}\left[\begin{array}{l}(1-\delta_{k,2})(1-\delta_{k,1}) + (1-\delta_{k,p-1})(1-\delta_{k,p}) - 2(1-\delta_{k,p})(1-\delta_{k,1}) + \\ + (1-\delta_{l,2})(1-\delta_{l,1}) + (1-\delta_{l,q-1})(1-\delta_{l,q}) - 2(1-\delta_{l,q})(1-\delta_{l,1})\end{array}\right]$$

$$a_{kl}^{(+1)} = \frac{\lambda}{pq}(\delta_{l,q} - 1) + \frac{2\mu}{pq}(\delta_{l,q-1} - 1)(1 - \delta_{l,q}),$$

$$a_{kl}^{(+2)} = \frac{\mu}{pq}(1 - \delta_{l,q-1})(1 - \delta_{l,q}),$$

$$b_{kl}^{(-2)} = \frac{\mu}{pq}(1 - \delta_{k,2})(1 - \delta_{k,1}),$$

$$b_{kl}^{(-1)} = \frac{\lambda}{pq}(\delta_{k,1} - 1) + \frac{2\mu}{pq}(\delta_{k,2} - 1)(1 - \delta_{k,1}),$$

$$b_{kl}^{(+1)} = \frac{\lambda}{pq}(\delta_{k,p} - 1) + \frac{2\mu}{pq}(\delta_{k,p-1} - 1)(1 - \delta_{k,p}),$$

$$b_{kl}^{(+2)} = \frac{\mu}{pq}(1 - \delta_{k,p-1})(1 - \delta_{k,p}),$$

где n_{kl} – число элементов в таксоне K_{kl}, δ_{ij} – символ Кронекера, множители вида $(1-\delta_{ij})$ введены для того, чтобы учесть «краевые эффекты». Если индексы k,l при y^{kl} не соответствуют никакому узлу сетки, то этот множитель автоматически обратит это слагаемое в ноль.

Уравнения $\frac{\partial D}{\partial y^{kl}} = 0$, k=1…p, l=1…q дают m систем линейных уравнений (по одной на каждую из m компонент векторов y^{kl}).

«Вытянем» набор y^{kl} в один столбец. В результате вектор неизвестных будет

$$x = (y_{11},..,y_{1q},y_{2,1},..,y_{2,q},.....y_{(p-1),1},...y_{(p-1),q},y_{p,1}...,y_{p,q})$$

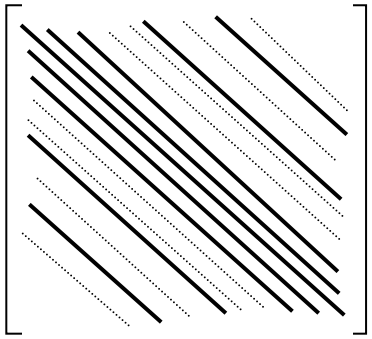
Система имеет вид $Ax=b$, где s-ая компонента вектора свободных членов равна

$b_s = \sum_{x \in K_{ij}} x$, $i = \left[\frac{s-1}{q}\right] + 1$, $j = s - \left[\frac{s-1}{q}\right] q$, […] – операция взятия целой части числа.

$$A_{st} = \begin{cases} a_{kl}, i = k, j = l \\ a_{kl}^{(-1)}, i = k-1, j = l \\ a_{kl}^{(+1)}, i = k+1, j = l \\ a_{kl}^{(-2)}, i = k-2, j = l \\ a_{kl}^{(+2)}, i = k+2, j = l \\ b_{kl}^{(-2)}, i = k, j = l-2 \\ b_{kl}^{(-1)}, i = k, j = l-1 \\ b_{kl}^{(+1)}, i = k, j = l+1 \\ b_{kl}^{(+2)}, i = k, j = l+2 \\ 0, else \end{cases}$$, где $$\begin{cases} i = \left[\frac{s-1}{q}\right] + 1 \\ j = s - \left[\frac{s-1}{q}\right] q \\ k = \left[\frac{t-1}{q}\right] + 1 \\ l = t - \left[\frac{t-1}{q}\right] q \end{cases}$$

Таким образом, матрица системы имеет вид

A =



то есть в ней есть пять «центральных» и четыре «побочные» диагонали. На протяжении всей работы алгоритма значения элементов матрицы остаются неизменными, изменяются лишь компоненты вектора *b* (которые зависят от способа разбиения множества точек данных на таксоны).

2.5.2. Непрямоугольные сетки

Упругая сетка вовсе не обязательно должна быть прямоугольной. В некоторых случаях, например, может оказаться разумным применение гексагональной сетки (см. рис.17). Для гексагональной сетки каждая точка имеет не два, а три «правых нижних соседа» и три «ребра жесткости». Вид функционалов D_2, D_3 в этом случае принимает вид:

$$D_2 = \sum_{i=1}^{p}\sum_{j=1}^{q-1}\left\|y^{ij} - y^{i,j+1}\right\|^2 + \sum_{i=1}^{p-1}\sum_{j=1}^{q}\left\|y^{ij} - y^{i+1,j}\right\|^2 +$$

$$+ \sum_{\substack{i=2 \\ j-нечет.}}^{p}\sum_{j=1}^{q-1}\left\|y^{ij} - y^{i-1,j+1}\right\|^2 + \sum_{\substack{i=1 \\ j-четн.}}^{p-1}\sum_{j=1}^{q}\left\|y^{ij} - y^{i+1,j+1}\right\|^2 ;$$

$$D_3 = \sum_{i=2}^{p-1}\sum_{j=1}^{q}\left\|2y^{ij} - y^{i-1,j} - y^{i+1,j}\right\|^2 +$$

$$\sum_{\substack{i=2 \\ j-нечет.}}^{p}\sum_{j=2}^{q-1}\left(\left\|2y^{ij} - y^{i-1,j-1} - y^{i,j+1}\right\|^2 + \left\|2y^{ij} - y^{i,j-1} - y^{i+1,j+1}\right\|^2\right) +$$

$$+ \sum_{\substack{i=2 \\ j-четн.}}^{p}\sum_{j=2}^{q-1}\left(\left\|2y^{ij} - y^{i,j-1} - y^{i+1,j+1}\right\|^2 + \left\|2y^{ij} - y^{i,j+1} - y^{i+1,j-1}\right\|^2\right)$$

Вычисление производных в этом случае приводит к громоздким выражениям, поэтому рассмотрим обобщение метода на случай произвольной двумерной сетки.

Теперь будем описывать сетку, явно указывая на способ соединения узлов и правило образования «ребер жесткости».

Положим, что сетка состоит из p узлов, каждому соответствует радиус-вектор y^i, $i = 1\ldots p$. Некоторые узлы соединяются между собой ребрами упругости E^i – их количество s штук. Три узла могут образовать ребро жесткости R^j – пусть их будет r штук. Каждое из s ребер упругости может иметь вес w_i, а каждое из ребер жесткости – вес v_j. Тогда общий вид функционала имеет вид

$$D = \frac{1}{|X|}D_1 + \frac{1}{p}(\lambda D_2 + \mu D_3),$$

$$D_1 = \sum_{i=1}^{p}\sum_{X_i \in K_i}\left(y^i - X_i\right)^2,$$

$$D_2 = \sum_{i=1}^{s} w_i\left(E^i(1) - E^i(2)\right)^2,$$

$$D_3 = \sum_{i=1}^{r} v_i\left(R^i(3) + R^i(2) - 2R^i(1)\right)^2,$$

где через $E^i(1)$ обозначен радиус вектор начала i-го ребра упругости, а через $E^i(2)$ – конец, далее $R^i(2)$, $R^i(3)$ – крайние точки ребра жесткости, $R^i(1)$ – центральный узел i-го ребра жесткости.

Введем обозначение

$$\Delta(x, y) = \begin{cases} 1, x = y \\ 0, x \neq y \end{cases}$$

Тогда дифференцирование дает

$$\frac{1}{2}\frac{\partial D_1}{\partial y^j} = n_j y_j - \sum_{X_i \in K_j} X_i ,$$

$$\frac{1}{2}\frac{\partial D_2}{\partial y^j} = \sum_{i=1}^{s} w_i \left(E^i(1) - E^i(2)\right)\left[\Delta(E^i(1), y^j) - \Delta(E^i(2), y^j)\right] ,$$

$$\frac{1}{2}\frac{\partial D_3}{\partial y^j} = \sum_{i=1}^{r} v_i \left(R^i(3) + R^i(2) - 2R^i(1)\right)\begin{bmatrix}\Delta(R^i(3), y^j) + \Delta(R^i(2), y^j) - \\ - 2\Delta(R^i(1), y^j)\end{bmatrix}$$

Обозначим

$$\Delta E^{ij} \equiv \Delta(E^i(1), y^j) - \Delta(E^i(2), y^j) ,$$

$$\Delta R^{ij} \equiv \Delta(R^i(3), y^j) + \Delta(R^i(2), y^j) - 2\Delta(R^i(1), y^j) ,$$

тогда

$$\frac{1}{2}\frac{\partial D_2}{\partial y^j} = \sum_{i=1}^{s} w_i \left(E^i(1) - E^i(2)\right)\Delta E^{ij} =$$

$$- \sum_{i=1}^{s} w_i \sum_{k=1}^{p} y_k \Delta(E^i(1), y^k)\Delta E^{ij} - \sum_{i=1}^{s} w_i \sum_{k=1}^{p} y_k \Delta(E^i(2), y^k)\Delta E^{ij} =$$

$$= \sum_{k=1}^{p} y_k \sum_{i=1}^{s} w_i \Delta E^{ij}\Delta E^{ik} = \sum_{k=1}^{p} y_k e_{jk};$$

$$\frac{1}{2}\frac{\partial D_3}{\partial y^j} = \sum_{i=1}^{r} v_i \left(R^i(3) + R^i(2) - 2R^i(1)\right)\Delta R^{ij} =$$

$$= \sum_{i=1}^{r} v_i \sum_{k=1}^{p} y^k \left(\Delta(R^i(3), y^k) + \Delta(R^i(2), y^k) - 2\Delta(R^i(1), y^k)\right)\Delta R^{ij} =$$

$$= \sum_{k=1}^{p} y^k \sum_{i=1}^{r} v_i \Delta R^{ij}\Delta R^{ik} = \sum_{k=1}^{p} y^k r_{jk},$$

где $e_{jk} = \sum_{i=1}^{s} w_i \Delta E^{ij}\Delta E^{ik}$, $r_{jk} = \sum_{i=1}^{r} v_i \Delta R^{ij}\Delta R^{ik}$. В результате получаем систему уравнений

$$\frac{1}{2}\frac{\partial D}{\partial y^j} = \sum_{k=1}^{p} y^k \left(\frac{n_j \delta_{jk}}{|X|} + \frac{\lambda}{p} e_{jk} + \frac{\mu}{p} r_{jk}\right) - \frac{1}{|X|}\sum_{X_i \in K_j} X_i = 0 , \quad j = 1 \ldots p .$$

Отсюда получаем систему линейных уравнений для нахождения одного из компонентов векторов положений узлов $\{y^i\}$:

$$\sum_{k=1}^{p} a_{jk} y^k = \frac{1}{|X|}\sum_{X_i \in K_j} X_i ,$$

$$a_{jk} = \frac{n_j \delta_{jk}}{|X|} + \frac{\lambda}{p} e_{jk} + \frac{\mu}{p} r_{jk}, \quad j = 1 \ldots p \qquad (*)$$

Поясним еще раз – вид уравнений одинаков для каждого из компонентов векторов y^i. Более того, элементы матрицы a_{ij} одинаковы для всех компонентов – необходимо лишь суммировать соответствующие компоненты векторов X_i в правой части уравнения.

Кроме того, если значения коэффициентов λ, μ не меняются, и способ соединения узлов в сетке неизменен, то в выражении для элементов матрицы a_{ij} меняется лишь первое слагаемое, связанное с разбиением множества точек данных на таксоны.

2.5.3. Применение сложных сеток

Отметим, что с помощью предложенного выше алгоритма можно строить одномерные, двумерные, вообще n-мерные сетки, необходимо лишь правильно разбить сетку на узлы, ребра упругости и ребра жесткости. Для целей визуализации более подходят двумерные упругие сетки, поскольку их можно «развернуть» на плоскости.

Предложенный способ построения непрямоугольных сеток позволяет строить замкнутые сетки (например, сферические или тороидальные). Для этих сеток существует некоторая сложность, связанная с отображением или «разворачиванием» их на плоскость. Как результат, такие сетки дают развертки, в которых границы «склеены» друг с другом.

Более полезным может оказаться применение способов настройки сетки, которые основаны на регулировании локальной гладкости и равномерности. В алгоритме упругих карт параметры метода λ, μ регулируют равномерность и гладкость сетки в целом, а с помощью весов отдельных ребер ω_i, ν_i можно регулировать локальные свойства сетки, «подгоняя» ее под локальные «детали»распределения данных.

Приведем некоторые соображения, которые могут быть здесь использованы. Все алгоритмы излагаются для случая прямоугольной сетки.

1) Адаптивный рост сетки

Идеи, лежащие в основе иерархических алгоритмов SOM могут применяться и для настройки упругой сетки.

Для того, чтобы рассчитать суммарные «натяжения» в вертикальном или горизонтальном ряду ребер упругости, нужно рассчитать величины

$$\varepsilon_k = \sum_{E^i \in G_k} w_i \left(E^i(2) - E^i(1) \right)^2,$$

где G_k – подмножество ребер упругости, которые образуют вертикальный или горизонтальный ряд. Тот ряд, для которого величина натяжения оказалась наибольшей, делится новыми узлами пополам, веса w_i при этом присваиваются новым образованным ребрам те же, что были у исходного делящегося ребра (см. рис.32).

2) Адаптивная настройка структуры сетки

После настройки сетки можно увеличивать точность аппроксимации следующим образом (рассмотрим случай прямоугольной сетки):

1. Рассчитывается число точек в каждом таксоне;

2. Для каждого из квадратов сетки вычисляется суммарное количество точек в таксонах его вершин;

3. Тот квадрат, для которого число точек оказалось наибольшим, делится на 4 части (см. рис.33); при этом веса вновь образовавшихся ребер упругости увеличиваются вдвое (поскольку длина ребра уменьшается вдвое, то вдвое должна увеличиться и его жесткость).

3) Адаптивная настройка весов

Существует способ перестроить сетку таким образом, чтобы ее структура оставалась прежней, а плотность узлов оказалась приблизительно пропорциональна плотности данных в окрестности этого узла. Этого можно достигнуть, делая сетку менее равномерной.

1. Рассчитывается число точек в каждом таксоне;

2. Для каждого из ребер упругости сетки вычисляется суммарное количество точек в таксонах его вершин;

3. Далее все веса ребер упругости пересчитываются, например, следующим образом:

$(w_i)' = \alpha w_i$, $\alpha = n_i / n_{aver}$,

где n_i – число точек в таксоне i-го узла, n_{aver} – среднее число точек в таксоне.

В результате те ребра, в окрестности которых плотность данных ниже среднего, становятся «мягче» и удлиняются, где выше – становятся «жестче» и укорачиваются. В конце концов данные должны быть распределены по таксонам более или менее равномерно.

4) Уменьшение топографической ошибки

Можно поставить целью уменьшение топографической ошибки и настраивать гладкость сетки таким образом, чтобы исчезли неустойчивые точки (см. выше описание алгоритма AdSOM):

1. Рассчитывается число точек в каждом таксоне, при этом подсчитывается величина «неустойчивости» узла – относительная доля неустойчивых точек в таксоне;

2. Выбирается узел y_1 с самой большой величиной неустойчивости;

3. По очереди выбираются неустойчивые точки таксона узла, выбранного на предыдущем шаге, для каждой выполняется процедура пересчета весов ребер жесткости R^i следующим образом:

3.1. Для каждой из выбранных точек рассчитывается второй по близости узел сетки $-y_2$; в силу определения неустойчивых точек он не является соседним;

3.2. От узла y_1 до y_2 «прокладывается» кратчайший маршрут из ребер жесткости – то есть ищутся ребра жесткости, с помощью которых можно соединить два узла, и для каждого из вошедших в маршрут ребер вес увеличивается:

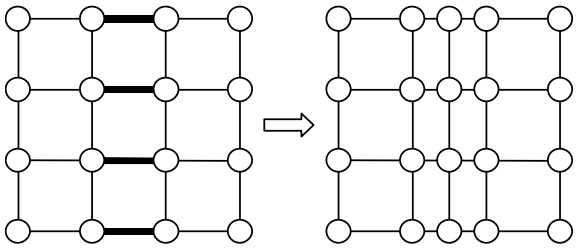
$$(v_i)' = v_i + \alpha n,$$

где α – параметр, n – длина маршрута (количество ребер жесткости, вошедших в маршрут);

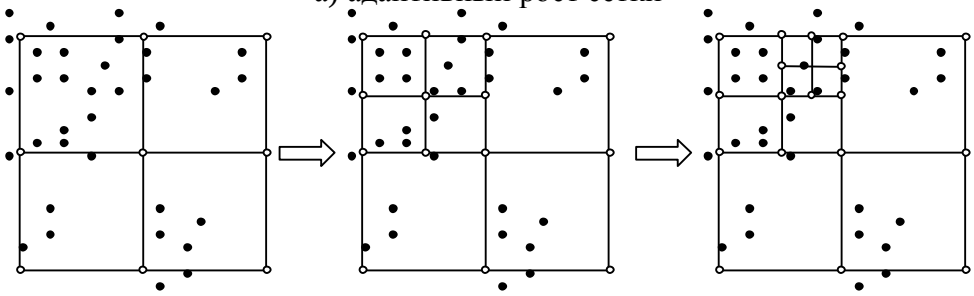
4. Для всех ребер жесткости, центральные узлы которых обладают *нулевой* величиной неустойчивости вес уменьшается:

$$(v_i)' = v_i - \alpha;$$

5. Сетка донастраивается;



а) адаптивный рост сетки



б) Адаптивная настройка структуры сетки

Рис. 33. Иллюстрации к работе алгоритмов адаптивного изменения структуры сетки

а) жирным выделен тот ряд ребер, которые оказались наиболее растянуты в пространстве данных;

б) квадрат, в окрестности вершин которого плотность узлов наибольшая, делится на четыре части, после донастройки сетки процесс продолжается.

6. Процесс повторяется до тех пор, пока либо величина топографической ошибки не достигнет допустимого уровня, либо число итераций не сатнет больше некоторого заданного числа.

2.5.4. Настройка сетки “online”

После того, как сетка настроена на определенном «базовом» наборе данных, может потребоваться «дообучить» сетку на новых данных, которые поступают динамически. Рассмотрим, как можно донастроить сетку на одном примере, который не входил в исходное обучающее множество.

В формуле (*) раздела 2.5.2 можно заметить, что от самих данных в матрице системы линейных уравнений зависит лишь первое слагаемое – $n_i/|X|\delta_{ij}$, где n_i – число точек данных в таксоне узла y_i. В столбце свободных членов стоят величины $\sum_{X_i \in K_i} X_i$ – сумма векторов данных, принадлежащих таксону K_i.

Это значит, что для однозначного построения сетки можно указать не весь набор данных, а только его частотный «словарь» – набор векторов-«образцов» по числу узлов в сетке и количество точек данных в каждом таксоне. С другой стороны, любое расположение сетки в пространстве данных задает определенный словарь.

Рассмотрим два варианта донастройки сетки:

1. *Вся информация о данных недоступна, доступен лишь словарь данных*, тогда вновь поступившая точка данных *x’* слегка корректирует словарь – увеличивает частоту n_k того «образца», который оказался к ней ближайшим (обозначим его x_k) и сдвигает сам этот образец на вектор

$$\Delta x_k = \frac{x' - x_k}{n_k + 1}.$$

Тогда изменяются и матрица системы, и вектор свободных членов. Вид новой системы $(A + \Delta A)(y + \Delta y) = (b + \Delta b)$, тогда для нахождения поправки Δy имеем $(A + \Delta A)\Delta y = \Delta b - \Delta A y$.

Поправка ΔA матрицы системы имеет единственный ненулевой элемент $1/N$ на пересечении k-го столбца и k-ой строки (N – число точек данных), поправка Δb столбца свободных членов – вектор с единственным ненулевым k-ым элементом, равным $\frac{\Delta x_k (n_k + 1) + x_k}{N}$. Таким образом, для нахождения новых положений узлов нужно найти малую поправку Δy, которая является решением системы

$$\sum_{i=1}^{p} (a_{ij} + \frac{1}{N}\delta_{ij}\delta_{jk})\Delta y_i = \frac{\Delta x_k (n_k + 1) + x_k - y_k}{N}\delta_{jk}, \; j = 1..p,$$

где p – число узлов.

Можно решить эту систему для каждой из компонент векторов и получить новые положения всех узлов $(y_i)' = y_i + \Delta y_i$. Однако дешевле с точки зрения вычислений найти приближенное решение и считать, что смещается только один, ближайший к *x’* узел, а остальные остаются на месте, то

есть $\Delta y_i = \Delta y_i \delta_{ik}$. Тогда получаем схему пересчета сетки для вновь поступившей точки данных *x'*:

а) Находим ближайший узел y_k.

б) Пересчитываем словарь $(x_k)' = x_k + \Delta x_k = \frac{x' + x_k n_k}{n_k + 1}$, $(n_k)' = n_k + 1$.

в) Рассчитываем новое положение узла $(y_k)' = y_k + \Delta y_k = \frac{y_k a_{kk} N + x'}{a_{kk} N + 1}$.

г) Пересчитываем коэффициенты матрицы $(a_{kk})' = a_{kk} + 1/N$.

д) Увеличиваем N: $N = N + 1$.

2. *Вся информация о данных доступна*. Тогда схема пересчета положений узлов сетки остается такой же как и в предыдущем случае, но в расчет можно включить шаг, учитывающий перерасчет словаря с учетом изменившегося положения узла (так как положение узла изменилось, то окружающие его данные могут «перескочить» с одного на другой таксон). Добавляем шаг

е) Учет «перескоков». Перебираем все точки в таксоне K_k и сравниваем расстояние до ближайшего узла с расстоянием до узла-ближайшего соседа (в пространстве данных). Если узел-сосед оказался теперь для точки ближайшим, то она «перескакивает» в его таксон. Также перебираем точки в таксоне узла-соседа. Если теперь для точки оказался ближайшим узел y_k, то она «перескакивает» в таксон K_k.

2.5.5. Доопределение сетки до многообразия

Используя сетку узлов, можно построить на ее основе непрерывное многообразие. Для этого подходит любой метод, в котором восстанавливается многообразие по конечному числу заданных точек.

Формально задача ставится следующим образом. Требуется восстановить вектор функцию $r = r(u,v)$ по значениям в конечном числе ее точек $\{y_i = r_i(u_i,v_i),\ i = 1\ldots p\}$. Пара чисел u_i, v_i – приписываемые заранее каждому узлу *внутренние координаты* на двумерной карте.

Рассмотрим самый простой случай построения кусочно-линейного многообразия. Для этого двумерная сетка предварительно *триангулируется* – для всего множества узлов указывается правило объединения их в *треугольники*.

Зададим внутренние координаты точки карты u, v в области определения вектор функции $r(u,v)$. Тогда, используя заданное правило триангуляции, можно определить тот треугольник, которому принадлежит выбранная точка (он будет ближайшим). Допустим, что этот треугольник об-

разован узлами с номерами i_1, i_2, i_3. Можно определить относительные координаты α, β точки относительно этих узлов. Например, определим их так:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u_{i1} \\ v_{i1} \end{pmatrix} + \alpha \begin{pmatrix} u_{i2} - u_{i1} \\ v_{i2} - v_{i1} \end{pmatrix} + \beta \begin{pmatrix} u_{i3} - u_{i1} \\ v_{i3} - v_{i1} \end{pmatrix}.$$

Тогда имеем систему линейных уравнений для определения α, β:

$$\alpha \begin{pmatrix} u_{i2} - u_{i1} \\ v_{i2} - v_{i1} \end{pmatrix} + \beta \begin{pmatrix} u_{i3} - u_{i1} \\ v_{i3} - v_{i1} \end{pmatrix} = \begin{pmatrix} u - u_{i1} \\ v - v_{i1} \end{pmatrix},$$

решив которую, найдем искомое значение вектор-функции:

$$r(u,v) = y_{i1} + \alpha(y_{i2} - y_{i1}) + \beta(y_{i3} - y_{i1}).$$

Построение такого кусочно-линейного многообразия наименее трудоемко с точки зрения вычислений. Однако, можно использовать и более изощренные техники – например, многомерную формулу Карлемана [5].

В результате мы получаем гладкую поверхность, проходящую через все узлы построенной сетки.

2.5.6. Проецирование данных на построенную карту

Как мы уже указывали, алгоритм построения карты не предполагает ее двумерности. С помощью одного и того же алгоритма можно строить сетки различной размерности – отличие будет лишь в способе соединения узлов в ребра жесткости и упругости. Вместе с тем, сетки разной размерности имеют различную «специализацию»:

а) *одномерные сетки* – наиболее подходящие кандидаты для решения задач *нелинейного факторного анализа*;

б) *двумерные сетки* удобнее всего использовать для *визуализации данных*;

в) *трехмерные сетки* задают некоторое эффективное трехмерное пространство, пригодное для решения задач визуализации данных, когда двух измерений недостаточно. Например, такая ситуация складывается при визуализации *вложений временных рядов* (см. раздел 2.6).

г) *m*-мерные сетки могут использоваться для эффективного *квантования данных*, то есть сжатия информации.

Наиболее простой способ переноса точки данных из пространства на построенную карту – *кусочно-постоянное проецирование*, когда каждой точке данных сопоставляется *ближайший узел карты*. Для такого способа проецирования даже не обязательно доопределять сетку узлов до многообразия.

Более интересны кусочно-непостоянные способы проецирования. Например, если по сетке узлов строится кусочно-линейное многообразие,

то можно предложить естественный способ *проецирования в ближайшую точку карты*.

Начнем с одномерных сеток. Введем понятие *расстояния от точки до отрезка*. Будем определять его следующим образом.

1. Выполним ортогональное проецирование на прямую, содержащую отрезок. Если проекция принадлежит отрезку, то искомое расстояние – это расстояние до проекции.

2. Иначе искомое расстояние – это расстояние до ближайшего конца отрезка.

Расстояние до треугольника найдем так:

1. Выполним ортогональное проецирование на плоскость, содержащую треугольник. Если точка проекции принадлежит треугольнику, то искомое расстояние – это расстояние до проекции.

2. Иначе искомое расстояние – это расстояние до ближайшей стороны треугольника (каждая из которых представляет собой отрезок).

Расстояние до тетраэдра найдем так:

1. Выполним ортогональное проецирование в трехмерное линейное многообразие, содержащее тетраэдр. Если точка проекции принадлежит тетраэдру, то искомое расстояние – это расстояние до проекции.

2. Иначе искомое расстояние – это расстояние до ближайшей стороны тетраэдра (каждая из которых представляет собой треугольник).

Продолжая аналогично, можно найти расстояние до любого k-мерного симплекса.

Одномерная кусочно-линейная карта состоит из отрезков. Поэтому ближайшая точка такой карты – это ближайшая точка ближайшего отрезка ломаной. Соответственно, ближайшая точка двумерной карты – это ближайшая точка ближайшего треугольника, и т.д.

Проектор в ближайшую точку построенного многообразия понятен и идея его неизменна для карт разной размерности. Однако, для некоторых целей он может оказаться слишком грубым (существуют целые области пространства, из которых точки проецируются в один узел). В случае одномерных сеток возможно применение алгоритма центрального проецирования [41]. В этом алгоритме центр проецирования выбирается в пересечении двух перпендикуляров к ребрам, которые прилегают к узлу, ближайшему к точке данных.

2.6. Моделирование вложений временных рядов

Геометрическую метафору облака точек в многомерном пространстве можно сопоставить не только данным, изначально представленным в виде таблицы «объект-признак», но и временному ряду. По ряду значений z_t, t=1,2…, M меняющейся величины мы можем сделать его *d-мерное вложение*, где каждая точка соответствует куску ряда из d последовательных элементов: $X_k = (z_k, z_{k+1}, \ldots, z_{k+d-1})$. Таким образом, временному ряду сопоставляется таблица, в которой первая строка – это первые d значений ряда, вторая – d значений ряда, начиная со второго, в третьей – d-окно сдвигается еще на одну позицию и т.д. (будем называть d-окном «рамку» ширины d, в которой мы рассматриваем кусок ряда). Такой способ представления временного ряда называется *вложением по Таккенсу* [20].

Соответствующая последовательность точек в d-мерном пространстве опишет определенную траекторию. Интересно, что эта траектория может целиком лежать в некотором k-мерном подпространстве ($k < d$), или не слишком сильно выходить за его пределы (находиться в нем в пределах заданной точности).

Будем называть *пространством вложения* d-мерное пространство, в которое вложена траектория. Каждая точка этого пространства соответствует вырезанной окном последовательности значений ряда длиной d. Назовем такую последовательность *паттерном*. Совокупность паттернов – это все возможные варианты поведения временного ряда на отрезках времени в d элементов.

На рис.34 приведено 4 примера наборов паттернов для простых функциональных временных рядов, и соответствующие им траектории в многомерном пространстве. Из рисунков видно, что при любой длине нарезки d (однако большей вводимой ниже *размерности ряда по Таккенсу*) и шаге дискретизации Δt траектория оказывается вложенной в подпространство меньшей размерности. Так, постоянный временной ряд оказался нульмерным (в пространстве вложения ему соответствует единственная точка $X_0 = (C,C\ldots C)$, $C = const$), линейный ряд – одномерный (в пространстве вложения он представлен прямой, не проходящей через начало координат и параллельной диагонали $x_1 = x_2 = \ldots = x_d$), функция синуса порождает «двумерный» временной ряд, а синус со второй гармоникой оказался четырехмерным.

Легко понять почему это так. Рассмотрим линейный ряд. Если нам известна первая точка паттерна (или, вообще, одна любая точка), то этот паттерн однозначно идентифицируется и восстанавливается. Иными словами, для того, чтобы определить поведение линейного ряда на ближайшие d шагов, достаточно знать одно-единственное начальное значение паттерна. Для синуса это не так. Здесь в наборе паттернов есть нисходящие, восходящие и «экстремальные» паттерны (которые также вначале либо возрастают, либо убывают), поэтому для однозначного восстановления паттерна необходимы две его точки. В последнем примере есть два вида восходящих и два вида нисходящих паттернов, это и приводит к тому, что

траектория целиком лежит в четырехмерном линейном подпространстве (заметим, что часть траектории оказалась, тем не менее, плоской).

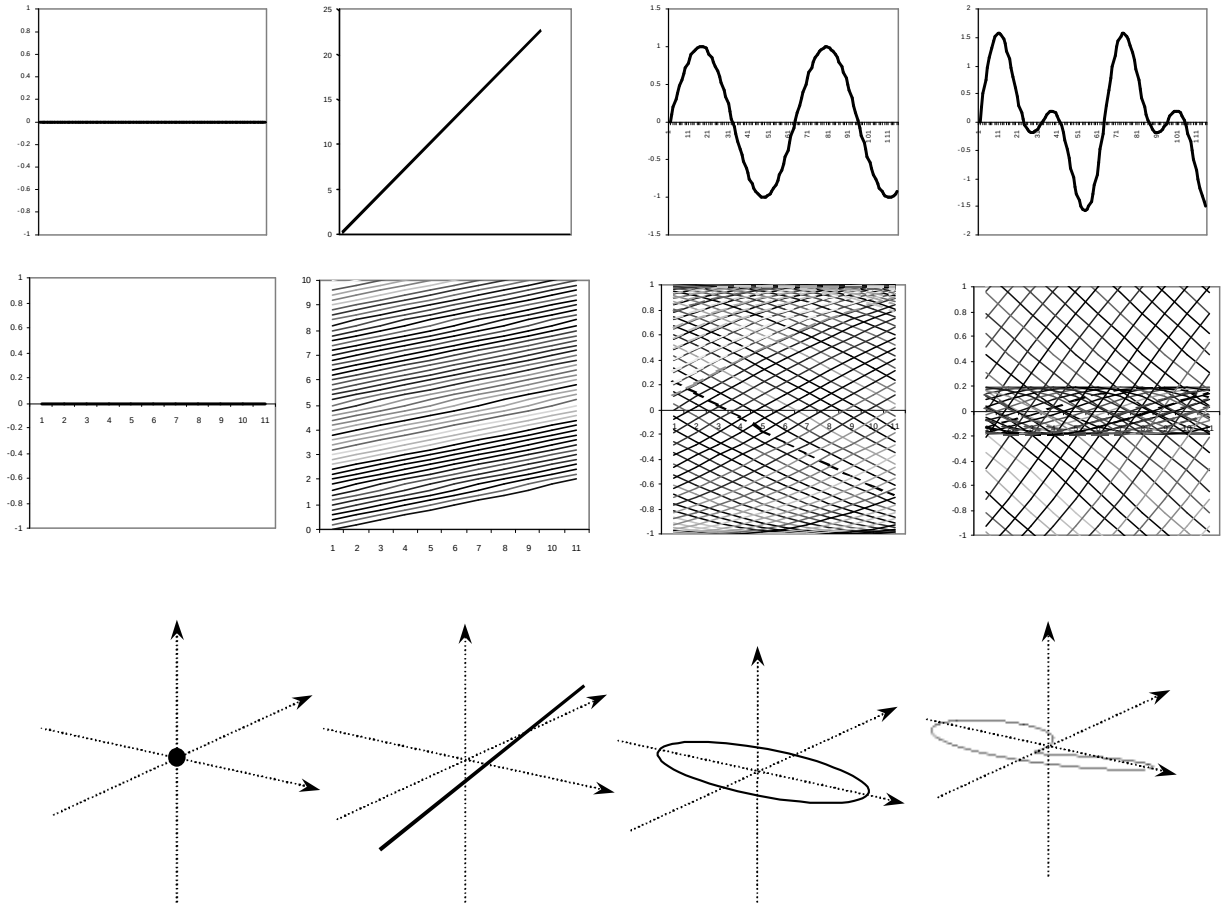


Рис.34. Иллюстрации к вложению временных рядов по Таккенсу
1 ряд графиков. Простые функциональные ряды, построенные на основе функций:
1. *f*(t)=0 2. *f*(t)=t. 3. *f*(t)=sin(t). 4. *f*(t)=sin(t)+0.8sin(2t).
2 ряд графиков. Семейство паттернов, построенных по соответствующим рядам.
1. Паттерн единственен; 2. Семейство патернов «одномерно»;
3. Семейство паттернов «двумерно»; 4. Семейство паттернов «одномерно».
3 ряд графиков. Траектории вложений рядов в 10-мерное пространство в пространстве, натянутом на 3 первые главные компоненты
1. Траектория – точка; 2. Траектория – прямая;
3. Траектория – эллипс; 4. Траектория – «четырехмерная» фигура Лиссажу.

Добавление белого шума к функциональному ряду синуса приводит к тому, что он остается двумерным лишь приблизительно. Добавление линейного тренда может привести к тому, что ряд станет одномерным (если функция в результате станет монотонной), или увеличит размерность ряда. Вообще, любая монотонная функция порождает одномерный временной ряд.

Задачу прогнозирования временного ряда можно поставить следующим образом: по m точкам ($m < d$) идентифицировать паттерн ряда и дать прогноз на d-m значений вперед. Если нужен более глубокий прогноз, то для выбора следующего паттерна можно воспользоваться m последними точками предыдущего. Таким образом, ряд будет «конструироваться» из заданного набора паттернов. Оправданность такого подхода основывается на предположении о том, что найденное семейство паттернов поведения является характерным для данного временного ряда. С другой стороны, «внезапное» появление принципиально новых паттернов свидетельствует о разладках в поведении ряда, смене режима, что тоже может оказаться полезной информацией.

Сколько же точек нужно взять для того, чтобы при выполнении указанного предположения однозначно восстановить паттерн? Ответ на этот вопрос зависит от того, какова эффективная размерность семейства паттернов, т.е. сколько паттернов в среднем проходит через ε-окрестность выбранной точки в окне. Это число и определит размерность временного ряда по Таккенсу.

Один из способов найти эффективную размерность линейного пространства, в котором находится траектория ряда в пространстве вложений – использование метода главных компонент. Действительно, первая главная компонента в пространстве вложений дает оптимальное в смысле среднеквадратичного расстояния между паттернами, приближенное описание заданного временного ряда с помощью «одномерного» ряда. Другими словами, набор паттернов реального ряда заменяется на однопараметрическое семейство *моделирующих паттернов*. В d-окне паттерны этого семейства не пересекаются. В случае использования плоскости первых двух главных компонент получаем двухпараметрическое моделирующее семейство и т.д.

Рассмотрим подробнее случай «одномерного» моделирования. Допустим, уравнение первой главной компоненты в пространстве вложений имеет вид

$$X = X_0 + Y_1 t,\ \|Y_1\|^2 = 1,\ t - \text{параметр}.$$

Здесь X_0 – точка пространства вложений, соответствующая геометрическому центру траектории ряда. В d-окне ей соответствует «усредненный» паттерн $\hat{X}_0$. Вектору главной компоненты Y_1 в d-окне соответствует «несмещенный» паттерн-образец $\hat{Y}_1$. В результате реальный паттерн $\hat{X}_k$ заменяется на модельный

$$X'_k = X_0 + Y_1(X_k - X_0, Y_1),$$

скобками обозначено стандартное скалярное произведение.

Совершенно аналогично происходит моделирование с помощью двумерного ряда по формуле

$$X'_k = X_0 + Y_1(X_k - X_0, Y_1) + Y_2(X_k - X_0, Y_2),$$

где Y_2 – вектор второй главной компоненты в пространстве вложений.

Будем теперь рассматривать среднюю точку паттерна как «текущую» точку ряда. Тогда $\frac{d-1}{2}-1$ первых значений в окне (предположим, что d – нечетное) соответствует «прошлому» точки, а $\frac{d-1}{2}-1$ последних значений «будущему». Метод сглаживания временных рядов с помощью «скользящего среднего» заключается в замене текущего значения ряда на среднее по всем точкам из d-окна. С точки зрения изложенного выше это соответствует проецированию в пространстве вложений на единичный вектор с совпадающими компонентами $Y_{cp}=\left(\frac{1}{\sqrt{d}},\frac{1}{\sqrt{d}},...,\frac{1}{\sqrt{d}}\right)$, то есть соответствует «одномерному моделированию» ряда. У вектора Y_1 компоненты не равны и это задает другой способ фильтрации ряда. Впрочем, если временной ряд достаточно гладкий и шаг дискретизации мал, то $Y_1 \approx Y_{cp}$.

2.7. Мультикартирование

Может оказаться так, что точность моделирования данных с помощью двумерного многообразия, вложенного в пространство большей размерности окажется недостаточно хорошей. Точность моделирования можно повышать с помощью таких приемов:

а) увеличение числа узлов;

б) уменьшение упругой энергии карты (карта начинает более плотно прилегать к данным).

И в том, и в другом случае карта становится менее гладкой, что приводит к ухудшению ее обобщающих способностей (см. раздел 1.2).

Идея мультикартирования или *итерационного моделирования* состоит в том, чтобы описывать данные последовательностью карт для разных наборов данных.

Первая карта описывает само облако точек в исходном пространстве данных.

Вторая карта описывает ошибки описания данных первой картой. Набор данных, на котором она строится – вектора первых остатков, полученные вычитанием из радиус-векторов исходных данных радиус-векторов их проекций на карте.

Третья карта описывает вторые остатки и т.д.

Моделирование данных с помощью набора карт можно назвать нелинейным факторным анализом данных. Карту данных можно назвать *нелинейным фактором* (одномерным, двумерным, трехмерным и т.д.).

Главная отличительная особенность подхода от традиционного факторного анализа – его нелинейность, то есть линейная модель данных $X_i = Qy_i + U$ заменяется на нелинейную:

$$X_i = F(u_i, v_i) + \Delta_i,$$

где u_i, v_i – внутренние координаты точки X_i на карте (рассматриваем двумерный случай), F – вектор-функция. Остатки Δ_i, в свою очередь, описываются следующей нелинейной моделью: $\Delta_i = F_1(u_i^1, v_i^1) + \Delta_i^{(1)}$ и так далее.

Сколько карт понадобится, чтобы описать данные с нулевой ошибкой? Рассмотрим случай полных данных и пусть в качестве факторов выступают направления главных компонент. Гарантируется, что ошибка описания станет нулевой, если использовать m линейных факторов, где m – размерность пространства. Это очевидно, поскольку тогда моделирование сводится к замене координатной системы. Вектора первых остатков лежат в линейном многообразии размерности m-1, ортогональном первому фактору, вектора вторых остатков образуют облако эффективной размерности m-2 и т.д.

Ситуация меняется, если используются нелинейные факторы. В этом случае вектора первых, вторых, третьих остатков уже не лежат в линейных многообразиях меньшей размерности – они, вообще говоря, по прежнему образуют m-мерное образование. В этом смысле никакое количество факторов не гарантирует нулевой ошибки моделирования, можно лишь гарантировать монотонное убывание дисперсии облака остатков с ростом числа факторов. Нелинейные факторы имеет смысл использовать только если они обеспечивают меньшую остаточную дисперсию, чем линейные для числа факторов, меньшего m. Соответственно, не имеет смысла использовать больше m нелинейных факторов.

2.8. Информационное моделирование с помощью упругих карт

Допустим, исследователь построил по данным карту самих данных, карту остатков, карту вторых остатков и т.д. – всего s карт. Назовем последовательность таких карт *информационной моделью данных*.

Подчеркнем отличие построенной модели от традиционных нейросетевых информационных моделей, описанных, например, в [44], где образом модели является «черный ящик» с p входами и m-p выходами (см. рис. 35а). Заметим, что на вход нейросети не могут подаваться «пробелы» – они должны быть уже определенным образом заполнены.

При моделировании данных с помощью многообразий на вход модели подается вектор пространства $\boldsymbol{x}$, а на выходе снимается «смоделированный» вектор того же пространства $\tilde{\boldsymbol{x}}$ (например, точка проекции на карте). Схема напоминает нейросетевую архитектуру «узкое горло», однако существенное отличие заключается в том, что среди компонент $x^1, x^2, x^3 \ldots, x^m$ вектора $\boldsymbol{x}$ могут быть «пустые» значения, а на выходе эти значения окажутся заполненными (восстановленными).

Таким образом, исследователь может в рамках одной и той же модели произвольно разделить множество признаков на «входные» и «выходные». Подавая на вход вектор, в котором заполнены входные компоненты вектора, в выходные оставлены «пустыми», на выходе можно снять восстановленные значения выходных признаков.

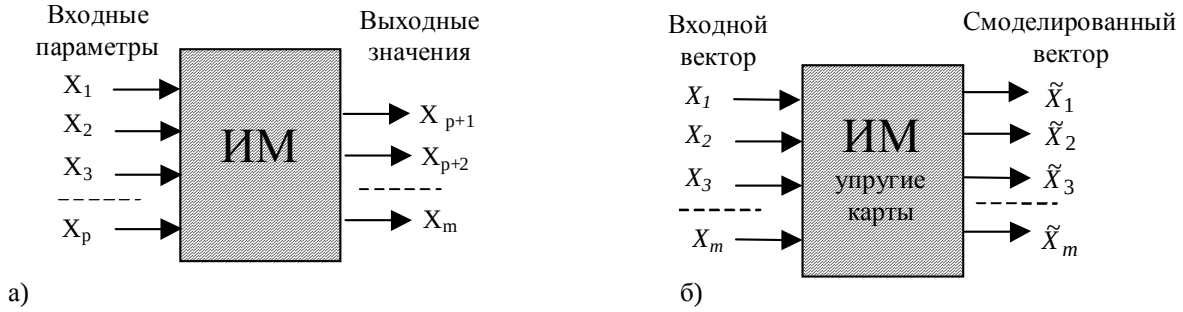


Рис. 35. Два типа информационных моделей.
а) «стандартная» нейросетевая схема моделирования;
б) моделирование данных с помощью многообразий.

Это открывает возможности для выявления взаимосвязей признаков, решения задач прогнозирования и построения регрессионных зависимостей между признаками.

С помощью информационной модели набор данных описывается с определенной *ошибкой обучения*:

$$Err_t = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_i - \tilde{X}_i)^2},$$

где X_i – вектор значений признаков i-го объекта в исходном наборе данных, $\tilde{X}_i$ – смоделированный вектор, N – число точек в наборе данных. В случае, когда исследователь ограничивается одной картой, величина Err_t совпадает введенной ранее величиной MSPE. Естественно ожидать, что с увеличением числа карт, ошибка обучения будет уменьшаться. Удобно пользоваться безразмерной величиной

$$err_t = \frac{Err_t}{\sigma},$$

где σ – корень из среднего квадрата расстояния данных до их среднего значения (среднеквадратичное отклонение).

Число карт выбирается таким образом, чтобы выполнялось условие $err_t < \varepsilon$, где ε – *допустимая относительная ошибка* обучения. Величина $(1-\varepsilon)\cdot 100\%$ называется *точностью моделирования*.

Увеличивая число карт, можно добиться сколь угодно малой ошибки описания обучающей выборки. Это обстоятельство можно использовать

для задачи сжатия информации. Однако качество модели в большей степени характеризуется величиной *ошибки обобщения*:

$$Err_g = \sqrt{\frac{1}{N_g}\sum_{i=1}^{N_g}(X_i^{(g)} - \tilde{X}_i^{(g)})^2},$$

где $X^{(g)}_i$ – вектор значений признаков i-го объекта *тестирующей выборки*, $\tilde{X}_i^{(g)}$ – соответствующий смоделированный вектор, N_g – число точек в тестирующей выборке. Тестирующая выборка обычно формируется с помощью случайного удаления из процесса настройки модели определенного процента примеров, которые в дальнейшем и формируют N_g тестирующих примеров. Понятно, что чем больше этот процент, тем точнее оценка величины ошибки обобщения, меньше примеров остается для настройки самой модели. Введем безразмерную величину ошибки обобщения:

$$err_g = \frac{Err_g}{\sigma}.$$

Если значение $(1\text{-}err_g)\cdot 100\%$ достигает 80-90%, то моделирование можно считать успешным, поскольку такова точность прогноза хорошего эксперта.

Рассмотрим возможные задачи, которые можно решать с помощью информационных моделей, построенных с помощью метода упругих карт:

1. Визуализация данных.

Основной особенностью и преимуществом построения *двумерных* информационных моделей является возможность наглядного представления данных и ошибок описания данных моделью. Любая из s построенных карт позволяет визуально анализировать распределение самих данных или погрешностей описания.

2. Группирование объектов.

С помощью карты можно производить разбиение объектов на группы. Это можно делать визуально, оценивая компактность и форму имеющихся в наборе сгущений данных. Если деление на группы нельзя произвести четко или не имеет смысла, то можно использовать следующий прием (ср. с примером из). На карту накладывается двумерный непрерывный цветной спектр. В результате каждая точка получает определенный цвет, причем точки, соседние на карте, получают близкие цвета. Полученные цвета можно использовать, например, если точки данных можно расположить на географической карте. Тогда сходные по значению признаков точки на карте будут иметь близкие цвета.

3. Оценка значимости признаков.

В наборе данных могут оказаться дублирующие друг друга (сильно скоррелированные) признаки или шумящие признаки, не несущие в себе никакой существенной информации для целей моделирования. Введем понятие значимости *i*-ого признака на *j*-ом объекте

$$\chi(i, X_j) = \left| err_t - err_t^{(i)} \right|,$$

где $err_t^{(i)}$ - ошибка обучения полученная при замене *i*-го признака у *j*-го объекта на некоторое предопределенное значение (в качестве такого значения может использоваться среднее значение признака, «пустое» значение или ноль). Назовем такую замену «фиксацией» признака.

Если в результате «фиксации» ошибка обучения изменилась не слишком сильно, то такой признак для данного объекта можно считать малозначимым.

Значимость *i*-го признака определим как

$$\chi_i = \frac{1}{N} \sum_{j=1}^{N} \chi(i, X_j)$$

Введем также понятие значимости набора из *k* признаков на *j*-ом примере

$$\chi(i_1 i_2 \ldots i_k, X_j) = \left| err_t - err_t^{(i_1 i_2 \ldots i_k)} \right|,$$

где $err_t^{(i_1 i_2 \ldots i_k)}$ - величина ошибки обучения, полученная при замене у j-го объекта признаков с номерами $i_1, i_2 \ldots i_k$ на предопределенное значение. Тогда значимость набора из *k* признаков определим как

$$\chi(i_1 i_2 \ldots i_k) = \frac{1}{N} \sum_{j=1}^{N} \chi(i_1 i_2 \ldots i_k, X_j).$$

Значимость признака может быть мала в двух случаях. Во-первых, признак действительно может оказаться малоинформативным. Во-вторых, он может быть зависим от остальных признаков. Малоинформативный признак не имеет большого смысла использовать в задачах информационного моделирования, тогда как с зависимыми признаками ситуация более сложна. Пусть несколько признаков образуют группу, в которой значения каждого могут быть восстановлены с определенной точностью при использовании значений других признаков из группы. При анализе признаков этой группы окажется, что значимость каждого отдельного признака мала. Однако, если один или несколько признаков в группе будут «зафиксированы», то значимости остальных могут резко возрасти.

Введем понятие значимости признака i на j-ом объекте при условии что признаки $i_1,i_2\ldots i_k$ фиксированы $\chi_f(i, X_j|i_1 i_2 \ldots i_k)$ и соответствующую условную значимость i-го признака $\chi_f(i|i_1 i_2 \ldots i_k) = \frac{1}{N}\sum_{j=1}^{N}\chi_f(i, X_j|i_1 i_2 \ldots i_k)$.

4. Отбор признаков

В рамках построенной информационной модели можно решать задачу выделения среди всех признаков группы наиболее информативных независимых признаков. Решение этой задачи имеет прикладное значение, но может потребовать большого количества вычислений. В этом направлении существуют следующие подходы [4]:

а) *полный перебор сочетаний* применим лишь в случае небольших размерностей, так как требует C_m^k оценок значимости;

б) *методы последовательного формирования группы*, когда из группы последовательно удаляются или добавляются один или несколько признаков по определенному критерию;

в) *стохастические методы*, когда группа формируется случайно и в зависимости от оценки значимости вероятность последующего выбора признаков, входящих в группу увеличивается или уменьшается;

г) *методы целенаправленного поиска*, позволяющие отбросить заведомо неприемлемые сочетания признаков.

Предложим алгоритм последовательного уменьшения группы признаков, основанный на использовании понятия условной значимости χ_f.

1. Из всех признаков фиксируется тот, чья значимость $\chi(\underline{i})$ минимальна. Пусть это будет признак i_1.

2. Рассчитываются значения $\chi_f(i|i_1)$.

3. Фиксируется признак i_2, для которого величина $\chi_f(i_2|i_1)$ наименьшая.

4. Рассчитываются значения $\chi_f(i|i_1 i_2)$ и т.д.

5. Процесс повторяется до тех пор, пока в группе не останется заданное число n признаков.

Другой класс задач связан с разбиением всех признаков на группы $S_1\ldots S_l$ в пределах каждой из которой признаки оказываются коррелированными, а признаки, принадлежащие разным группам относительно некоррелированы. Для линейных моделей соответствующий метод носит название *метода экстремальных группировок* [4].

Для выявления групп взаимосвязанных признаков может оказаться полезной визуализация *транспонированной задачи*, когда таблица данных транспонируется и признаки начинают играть роль номеров объектов в новом пространстве, а номера бывших объектов – названия новых признаков. Каждая точка при такой визуализации соответствует определенному при-

знаку, близкие точки соответствуют скоррелированным признакам. При таком представлении признаков исследователь может произвести разбиение на группы с помощью визуального анализа распределения точек на карте.

5. Восстановление пропущенных значений в данных.

Любой точке пространства признаков в информационной модели сопоставляется точка моделирующего многообразия в исходном пространстве, пространстве первых остатков, вторых остатков и т.д. В конечном итоге при наличии s карт точке X исходного пространства сопоставляется точка

$$\tilde{X} = P_{M0}(X) + P_{M_1}(X_1) + P_{M_2}(X_2) + ... + P_{M_{s-1}}(X_{s-1}),$$

где $P_{Mi}(X)$ – оператор проецирования в пространстве i-ых остатков, M_0 – исходная карта данных, M_1 – карта остатков, M_2 – карта вторых остатков и т.д., X_i – остатки: $X_1 = X - P_{M0}(X)$, $X_2 = X_1 - P_{M1}(X_1)$ и т.д.

Важно то, что если вектор X содержит пропущенные значения признаков, то в результирующем векторе $\tilde{X}$ все равно будут известны все компоненты. Значения соответствующих компонент $\tilde{X}$ можно использовать для восстановления пробелов в X или прогноза намеренно пропущенных значений.

6. Прогнозирование значений отдельных признаков.

Тот факт, что информационная модель позволяет правдоподобным образом восстановить отсутствующие компоненты вектора пространства, дает возможность использовать ее для прогнозирования значений отдельных признаков по информации (возможно неполной), содержащейся в других признаках.

Разделим всю совокупность признаков на *прогнозируемые* и *информационные*. Задавая значения части информационных признаков в векторе X, а остальные считая неизвестными, значения прогнозируемых признаков можно взять из вектора $\tilde{X}$. Чем большее число информационных признаков будет известно, тем более правдоподобным будет прогноз.

Рассмотрим в качестве примера таблицу данных экологических измерений. Допустим, что значения вредных выбросов различных химических веществ измерялись в разных точках, каждая из которых имеет две координаты на географической карте. Каждому измерению соответствуют окружающая температура воздуха, давление, время суток, направление ветра и другие характеристики. Таким образом, эти характеристики, а также географические координаты могут играть роль информационных признаков, а измеренные значения выбросов – роль прогнозируемых величин. Можно получить информационную раскраску географической карты, восстанавливая значения выбросов только по географическим координатам, считая

значения других информационных признаков неизвестными. Эти значения будут восстанавливаться некоторым правдоподобным образом, причем для каждой точки они будут разными, соответствуя тем условиям, в которых были проведены реальные измерения. Исследователь может «задать» температуру, и карта выбросов несколько видоизменится. Далее можно уточнять давление, время суток и другие значения информационных признаков, получая при этом более правдоподобные прогнозы и соответствующие им раскраски карты. Задавая временной ряд изменения значений информационных признаков исследователь получит возможность моделировать динамическую картину изменения значений прогнозируемых признаков. Таким образом, исследователь получает инструмент для моделирования составляющих экологической обстановки в зависимости от окружающих условий, причем тем более реалистичного моделирования, чем больший объем информации об экологических измерениях имеется в его распоряжении.

Работу такой прогнозирующей системы можно сравнить с экспертом, который по неполной информации, имеющейся в его распоряжении, «вспоминает» наиболее похожие случаи из своего опыта и корректирует свои выводы сообразно новым условиям. При этом чем больше информации попадает в руки такого эксперта, тем точнее и правдоподобнее оказывается его прогноз.

С помощью информационной модели исследователь может также решать обратную задачу прогнозирования – так, например, моделировать условия, при которых значение выбросов может превышать определенный порог. Об особенностях прямой и обратной задачи информационного моделирования можно подробнее прочитать в [44].

7. Построение регрессионных зависимостей

Задача построения регрессионных зависимостей одних признаков от других во многом подобна задаче прогнозирования значений. Различие состоит лишь в том, что значения всех информационных признаков считаются известными. Вся совокупность признаков делится на входные и выходные. Считается, что значения выходных признаков можно вычислить с заданной точностью ε_0 по значениям входных признаков. Отдельной задачей является выяснение вопроса о том, насколько правомерно предположение о существование зависимости между входными и выходными признаками. Известен метод (ритуал) анализа таблицы данных, позволяющий оценить оправданность такой гипотезы [35].

Разобъем таблицу на две части (см. рис.36) – слева сгруппируем входные признаки (их набор можно сформировать упомянутыми методами отбора наиболее значимых признаков), а справа – выходные. Зададимся определенной точностью ε. Теперь создадим две информационные модели: одна будет описывать всю таблицу данных с заданной точностью и содер-

жать s_1 карт, вторая будет построена только по набору входных признаков и описывать с точностью ε левую часть таблицы и содержать s_2 карт. Если число карт в первой и во второй модели будет приблизительно равным, т.е. $s_1 \approx s_2$, то можно считать, что в выходных признаках не содержится дополнительной информации, кроме той, что содержится во входных.

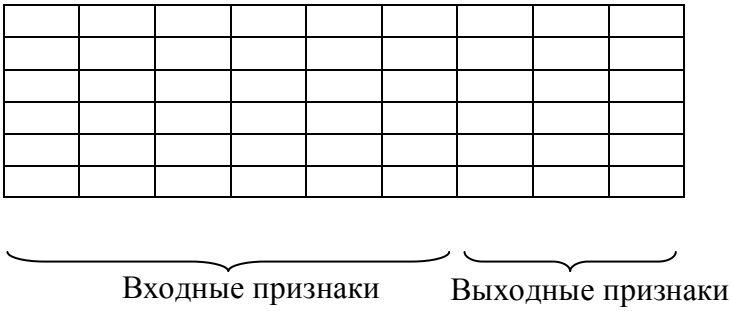


Рис.36. Деление таблицы на входные и выходные признаки. Можно сравнить число факторов, необходимых для описания всей таблицы, и только ее части из входных признаков. Если количество факторов оказалось одинаково – зависимость действительно существует.