

Областное коммунальное высшее учебное заведение

«Институт предпринимательства «Стратегия»»

А.А.Шумейко, С.Л.Сотник

Интеллектуальный анализ данных

(Введение в Data Mining)

Учебное пособие

Днепропетровск

Издатель Белая Е.А.

2012

УДК 004.451.53

ББК 32.973.26.-018.2

Ш 96

*Рекомендовано к печати Ученым советом
областного коммунального высшего учебного заведения
«Институт предпринимательства «Стратегия»
(протокол № 10 от 10.11.2011 г.)*

Рецензенты:

П.И. Когут, доктор физико-математических наук, профессор Днепропетровского национального университета имени Олеся Гончара;

А.И. Михалев, доктор технических наук, профессор, зав. каф. информационных технологий и систем национальной металлургической академии Украины;

Т.И. Олешко, доктор технических наук, профессор, зав. каф. экономической кибернетики Национального авиационного университета.

Авторский коллектив:

Шумейко А.А., доктор технических наук, профессор, ректор областного коммунального высшего учебного заведения «Институт предпринимательства «Стратегия»

Сотник С.Л., технический директор ООО «Iveonik Systems»

Шумейко А.А.

Ш 96 Интеллектуальный анализ данных (Введение в Data Mining)/ А.А. Шумейко,
С.Л. Сотник. – Днепропетровск: Белая Е.А., 2012. – 212 с.

ISBN 978-617-645-025-2

Предлагаемое пособие посвящено одному из направлений обработки данных - Data Mining или интеллектуальному анализу данных. В последние годы к этой тематике проявляется большой интерес, как у пользователей сети, так и у разного рода разработчиков. Термин Data Mining используется для обозначения множества методов выделения информационных характеристик из большого количества плохо структурированных исходных данных. Исходники программных кодов, прилагаемых в пособии, можно найти по адресу www.sotnyk.com/code/DataminingBookSrc.rar

Для научно-технических и педагогических работников, аспирантов, студентов и всех желающих познакомиться с рассматриваемой тематикой

УДК 004.451.53

ББК 32.973.26.-018.2

ISBN 978-617-645-025-2

© Шумейко А.А., Сотник С.Л., 2012

Предисловие к электронному изданию

Несмотря на то, что данная книга издана в бумажном виде, основной формой её распространения должна стать электронная. Эта форма позволяет оперативно исправлять замеченные неточности и опечатки, а также расширять отдельные разделы. Поэтому, перед прочтением книги, рекомендуем проверить сайт www.sotnyk.com на предмет наличия свежих электронных версий.

Версия	Изменения
27-май-2012	Исправлены замеченные опечатки, переформатированы листинги, изменена нумерация глав, добавлено предисловие к электронной версии, все изображения переведены в цвет.

Введение

Obscurum per obscurius

Обработка данных, в том числе и результатов эксперимента, является важнейшим средством получения новых знаний не только в области естественных и технических наук, но и в экономике, социологии, политике, психологии, литературоведении и в других отраслях. Эти исследования дают критерии оценки обоснованности и приемлемости на практике любых теорий и теоретических предположений. Обработка данных направлена, как правило, на построение математической модели исследуемого объекта или явления, а также на получение ответа на вопрос: «Достоверны ли имеющиеся данные в пределах требуемой точности или допусков?».

Сама же математическая модель в зависимости от целей (исследование, управление, контроль) может быть использована для разных целей: для предметно-смыслового анализа объекта или явления, прогнозирования их состояния в разных условиях функционирования, управления ими в конкретных ситуациях, оптимизации отдельных параметров, а также для решения каких-то других специфичных задач.

Конечной целью любой обработки данных является выдвижение гипотез о классе и структуре математической модели исследуемого явления, определение состава и объема дополнительных измерений, выбор возможных методов последующей статистической обработки и анализ выполнения основных предпосылок, лежащих в их основе. Для ее достижения необходимо решить некоторые частные задачи, среди которых можно выделить следующие (см. [1], [2], [29]):

1. Анализ, выбраковка и восстановление аномальных (сбитых) или пропущенных измерений. Эта задача связана с тем, что исходная информация обычно неоднородна по качеству. В основной массе результатов прямых измерений, получаемых с возможно малыми погрешностями, в имеющихся данных часто имеются грубые ошибки или просчеты, вызванные разными причинами. К ним могут быть отнесены особенности информационной системы (например, при использовании рекомендующих систем), а также, сбои вычислительной техники, аномалии в работе измерительных приборов и т. д. Без глубокого анализа ка-

чества данных, устранения или, хотя бы, существенного уменьшения влияния аномальных данных на результаты последующей обработки можно сделать ложные выводы об изучаемом объекте или явлении.

2. Оценка параметров и числовых характеристик наблюдаемых случайных величин или процессов. Выбор методов последующей обработки, направленной на построение и проверку адекватности математической модели исследуемому явлению, существенно зависит от закона распределения наблюдаемых величин. Получаемые при решении этой задачи выводы о природе обрабатываемых данных могут быть как общими (независимость измерений, характер погрешностей и др.), так и содержать детальную информацию о свойствах данных (в том числе и статистических, таких как вид закона распределения, его параметры). Решение задачи предварительной обработки не является чисто математическим, а требует также и содержательного анализа изучаемого процесса, а при возможности, схемы и методики проведения эксперимента.
3. Группировка исходной информации при большом объеме обрабатываемых данных. При этом должны быть учтены особенности их законов распределения, которые выявлены на предыдущем этапе обработки.
4. Объединение нескольких групп измерений, полученных, возможно, в различное время или в различных условиях, для совместной обработки.
5. Выявление скрытых связей и взаимовлияния различных измеряемых факторов и результирующих переменных, последовательных измерений одних и тех же величин. Решение этой задачи позволяет отобрать те переменные, которые оказывают наиболее сильное влияние на результирующий признак. Выделенные факторы используются для дальнейшей обработки, в частности, методами регрессионного анализа. Анализ корреляционных связей делает возможным выдвижение гипотез о структуре взаимосвязи переменных и, в конечном итоге, о структуре модели объекта исследований.

В ходе предварительной обработки, кроме указанных выше задач, часто решают и другие, имеющие частный характер: отображение, преобразование и унификацию типа наблюдений, визуализацию многомерных данных и др.

Следует отметить, что в зависимости от конечных целей исследования, сложности изучаемого явления и уровня априорной информации о нем, объем задач, выполняемых в ходе предварительной обработки, может существенно изменяться. То же самое можно сказать и о соотношении целей и задач, которые решаются при предварительной обработке и на последующих этапах анализа, направленных на построение модели явления.

Предлагаемое пособие посвящено одному из направлений обработки данных, так называемому Data Mining или интеллектуальному анализу данных. В последние годы к этой тематике проявляется большой интерес, как у пользователей сети, так и у разного рода разработчиков. Термин Data Mining был введен Григорием Пиатецким-Шapiro в 1989 году и используется для обозначения множества методов выделения информационных характеристик из большого количества плохо структурированных исходных данных (см., например, [33], [56]).

В основе Data Mining лежат методы классификации и кластеризации, моделирования и прогнозирования, генетические и эволюционные алгоритмы, методы «мягких вычислений» ([1], [6], [12], [15], [23], [31] и др.) и пр. Нельзя оставлять в стороне и методы прикладной статистики, которые составляют фундамент Data Mining, прежде всего это корреляционный и регрессионный анализ, факторный и дискриминантный анализ ([2], [22], [34], [42] и др.) и многое другое.

Большое внимание в данном пособии удалено обсуждению программной реализации методов Data Mining. В качестве инструмента программирования используется C#.

Исходники программных кодов, прилагаемых в пособии, можно найти по адресу www.sotnyk.com/code/DataminingBookSrc.rar.

Данное пособие не претендует на полное изложение методов интеллектуального анализа данных, но может быть полезно для студентов, преподавателей и всех, кому интересна эта тематика.

1. Базовая информация

Кое-что необходимое из линейной алгебры

В дальнейшем нам понадобятся базовые знания из курса линейной алгебры и теории вероятностей. Не углубляясь подробно, приведем в этом параграфе информацию, необходимую при изложении материала последующих разделов.

Напомним, что для матрицы A размером $n \times m$ под транспонированием понимаем замену строк столбцами с теми же номерами, в частности, если

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ x_{4,1} & x_{4,2} \end{bmatrix},$$

то

$$X^T = \begin{bmatrix} x_{1,1} & x_{2,1} & x_{3,1} & x_{4,1} \\ x_{1,2} & x_{2,2} & x_{3,2} & x_{4,2} \end{bmatrix}.$$

Важной характеристикой, используемой в дальнейшем, является скалярное произведение двух векторов

$$\langle X, Y \rangle = X^T Y = x_1 y_1 + x_2 y_2 + \dots + x_n y_n = \sum_{i=1}^n x_i y_i,$$

где

$$X^T = [x_1 \ x_2 \ \dots \ x_n] \text{ и } Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

С другой стороны, замечая, что $\langle X, Y \rangle = |X| |Y| \cos \theta$, где θ угол между векторами X и Y , имеем

$$\cos \theta = \frac{X^T Y}{|X| |Y|}.$$

Таким образом, скалярное произведение характеризует отклонение вектора X от вектора Y . Евклидовой метрикой или длиной вектора называется число

$$|X| = \sqrt{\langle X, X \rangle} = \sqrt{\sum_{i=1}^n x_i^2},$$

а евклидовым расстоянием между двумя векторами назовем число

$$|X - Y| = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}.$$

Векторы X_1, X_2, \dots, X_m называются линейно независимыми, если равенство

$$\alpha_1 X_1 + \alpha_2 X_2 + \dots + \alpha_m X_m = 0$$

выполняется тогда и только тогда, когда все $\alpha_i = 0, i = 1, 2, \dots, m$.

Если выполнение этого условия возможно хотя бы при одном $\alpha_i \neq 0$, то эта система векторов является линейно зависимой.

Множество всех векторов размерности n называется векторным пространством V той же размерности.

Множество векторов $\{U_1, U_2, \dots, U_m\}$ называется базисом векторного пространства V , если для $\forall v \in V$ найдется такое множество $\{\alpha_i\}_{i=1}^m$, что

$$v = \alpha_1 U_1 + \alpha_2 U_2 + \dots + \alpha_m U_m.$$

Базис $\{U_1, U_2, \dots, U_m\}$ называется ортогональным, если $U_i \perp U_j$ для $\forall i \neq j$ (т.е. $\langle U_i, U_j \rangle = 0$), если же при этом $|U_i| = 1, i = 1, \dots, m$, то базис называется ортонормированным.

Если для квадратной матрицы A размером $m \times m$ найдется ненулевой вектор X такой, что выполняется условие $AX = \lambda X$, то X называется собственным вектором матрицы A , а число λ называется собственным значением матрицы. Таким образом, линейное преобразование, реализованное матрицей A , переводит собственный вектор X в коллинеарный, направленный в ту же сторону, если $\lambda > 0$, и в обратную, если $\lambda < 0$. Отметим несколько важных свойств собственных чисел.

- Если матрица A действительна и симметрична (то есть $A^T = A$), то все собственные значения действительны.
- Если матрица не сингулярная (то есть ее ранг равен числу строк), то ее собственные числа не нулевые.

- Если матрица A положительно определена (то есть $X^TAX > 0$), то все ее собственные числа положительны.

Кое-что необходимое из теории вероятностей

Пусть Ω - множество всех возможных исходов некоторых событий, и S – алгебра событий, то есть S совокупность подмножеств множества Ω , для которого выполнены следующие условия:

1. S содержит невозможное и достоверное события.
2. Если события A_1, A_2, \dots (конечное или счетное множество) принадлежит S , то S принадлежит сумма, произведение и дополнение этих событий.

Вероятностью называется функция $P(A)$, определенная на S , принимающая действительные значения и удовлетворяющая аксиомам:

- Аксиома неотрицательности: $\forall A \in S : P(A) \geq 0$.
- Аксиома нормированности: вероятность достоверного события равна единице: $P(\Omega) = 1$.
- Аксиома аддитивности: вероятность суммы несовместных событий равна сумме вероятностей этих событий: если $A_i \cap A_j = \emptyset (i \neq j)$, то

$$P\left(\bigcup_k A_k\right) = \sum_k P(A_k).$$

Приведем свойства вероятности.

1. $P(\emptyset) = 0$;
2. $P(A) \leq 1$;
3. $A \subset B \Rightarrow P(A) < P(B)$;
4. $P(A \cup B) = P(A) + P(B) - P(A \cap B)$.

Вероятность того, что произойдет событие A при условии, что произошло событие B , называется условной вероятностью $P(A|B)$ и вычисляется следующим образом

$$P(A | B) = \frac{P(A \cap B)}{P(B)}.$$

В случае, если события A и B независимы, то $P(A \cap B) = P(A)P(B)$, и для условной вероятности можно записать

$$P(A | B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A)P(B)}{P(B)} = P(A).$$

Пусть

$$A = (A \cap B_1) \cup (A \cap B_2) \cup (A \cap B_3) \cup \dots \cup (A \cap B_n),$$

тогда из формулы условной вероятности получаем

$$P(A) = P(A|B_1)P(B_1) + P(A|B_2)P(B_2) + \dots + P(A|B_n)P(B_n),$$

то есть

$$P(A) = \sum_{k=1}^n P(A | B_k)P(B_k).$$

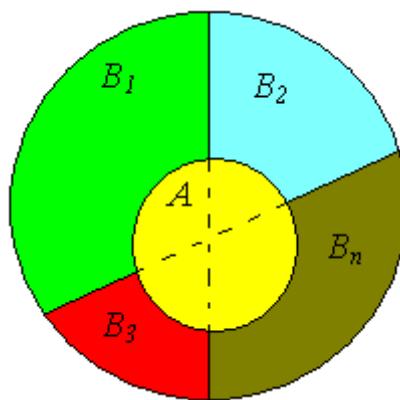


Рис. 1.1. Иллюстрация формулы полной вероятности.

Важную роль в дальнейших рассуждениях играет формула Байеса или теорема гипотез. Это утверждение позволяет переоценить вероятность гипотез B_i , принятых до опыта (события) и называемых априорными (a priori – до опыта) по результатам уже проведенного опыта, то есть используя апостериорные (a posteriori – после опыта) вероятности. Пусть B_1, B_2, \dots, B_n составляющие множества S, тогда вероятность того, что событие A приведет к событию B_i будет равно

$$P(B_i | A) = \frac{P(B_i \cap A)}{P(A)} = \frac{P(A | B_i)P(B_i)}{\sum_{k=1}^n P(A | B_k)P(B_k)}.$$

Случайной величиной X называется функция, определенная на множестве событий Ω , которая каждому элементарному событию ω ставит в соответствие число $X(\omega)$. Случайная величина может быть дискретная и непрерывная.

Любое правило, позволяющее находить вероятности произвольных событий $A \subseteq S$, называется законом распределения случайной величины, и при этом говорят, что случайная величина подчиняется данному закону распределения.

Функцией распределения случайной величины X называется функция $F(x)$, которая для любого $x \in R$ равна вероятности события $\{X \leq x\}$

$$F(x) = P\{X \leq x\}.$$

Функция распределения обладает следующими свойствами

1. $0 \leq F(x) \leq 1$.
2. $F(x)$ неубывающая функция, то есть, если $x_2 > x_1$, то $F(x_2) \geq F(x_1)$.
3. $F(-\infty) = 0, F(+\infty) = 1$.
4. $P\{a \leq X < b\} = F(b) - F(a)$.

Для дискретной случайной величины X положим

$$p(x) = P(X = x),$$

тогда

$$F(x) = P(X \leq x) = \sum_{a \leq x} P(X = a) = \sum_{a \leq x} p(a).$$

Для непрерывной случайной величины функцию $f(x) \geq 0$ назовем функцией плотности распределения вероятностей, если

$$F(a) = P(X \leq a) = \int_{-\infty}^a f(x)dx.$$

Тогда

$$P(a \leq X \leq b) = \int_a^b f(x)dx.$$

Заметим, что

$$\frac{d}{dx} F(x) = f(x),$$

кроме того,

$$P(x = a) = \int_a^a f(x)dx = 0, \text{ и } P(-\infty \leq x \leq \infty) = \int_{-\infty}^{\infty} f(x)dx = 1.$$

Рассмотрим некоторые важные характеристики случайной величины. Первый момент случайной величины называется математическим ожиданием. Для дискретного случая математическое ожидание будет иметь вид

$$\mu = E(X) = \sum_x xp(x),$$

для непрерывного случая

$$\mu = E(X) = \int_{-\infty}^{\infty} xf(x)dx.$$

Величина

$$\sigma^2 = \text{var}(X) = E((X - E(X))^2)$$

называется вариацией или дисперсией. Это число характеризует разброс случайной величины, а σ называется среднеквадратичным отклонением случайной величины от математического ожидания.

Особую роль в теории вероятностей играет нормальный закон (закон Гаусса), что обусловлено, прежде всего, тем фактом, что он является предельным законом, к которому приближаются (при определенных условиях) другие законы распределения.

Будем говорить, что непрерывная случайная величина X распределена по нормальному закону $N(\mu, \sigma)$, если ее плотность распределения имеет вид (функция Гаусса)

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), x \in R.$$

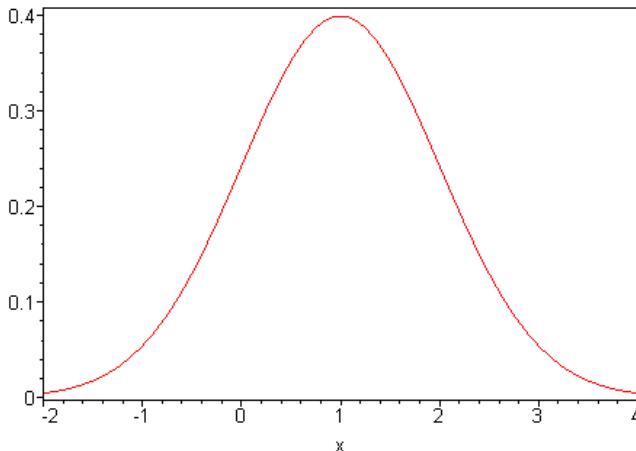


Рис. 1.2. График функции плотности нормального распределения (функция Гаусса) для $\mu = 1, \sigma = 1$.

Заметим, что, в соответствии с нормальным законом распределяются самые различные величины, например, ошибки измерений, износ деталей в механизмах, вес плодов и животных, рост человека, колебания курса акций и многое другое.

Несмотря на важность одномерного случая, для нас более актуальным является рассмотрение случая многих переменных.

Упорядоченный набор (X_1, X_2, \dots, X_n) случайных величин X_i ($i = 1, 2, \dots, n$), заданных на одном и том же множестве Ω называется n -мерной случайной величиной. Одномерные случайные величины X_1, X_2, \dots, X_n называются компонентами n -мерной случайной величины. Компоненты удобно рассматривать как координаты случайного вектора $X = (X_1, X_2, \dots, X_n)$ в пространстве n измерений.

Упорядоченная пара (X, Y) двух случайных величин называется двумерной случайной величиной. Полной характеристикой системы (X, Y) является ее закон распределения вероятностей, указывающий область возможных значений системы случайных величин и вероятности этих значений.

Функцией распределения двумерной случайной величины (X, Y) называется функция $F(x, y)$, которая для любых двух действительных чисел x и y равна вероятности совместного выполнения двух событий $\{X \leq x\}$ и $\{Y \leq y\}$, то есть

$$F(x, y) = P\{X \leq x, Y \leq y\} = P(\omega \in \Omega | X(\omega) \leq x, Y(\omega) \leq y).$$

Для дискретной случайной пары (X, Y) в качестве функции плотности положим

$$p(x, y) = P(X = x, Y = y),$$

в непрерывном случае $f(x, y) \geq 0$ назовем функцией плотности распределения вероятностей, если

$$F(a, b) = P(X \leq a, Y \leq b) = \int_{-\infty}^a \int_{-\infty}^b f(x, y) dx dy.$$

Тогда

$$P(a \leq x \leq b, c \leq y \leq d) = \int_a^b \int_c^d f(x, y) dx dy$$

Заметим, что

$$\frac{\partial^2}{\partial x \partial y} F(x, y) = f(x, y).$$

Важную роль в дальнейших исследованиях играет смешанный момент второго порядка, называемый ковариацией

$$\text{cov}(X, Y) = E((X - E(X))(Y - E(Y))) = E(XY) - E(X)E(Y).$$

В координатной форме ковариацию можно записать в виде

$$\text{cov}(X, Y) = \sum_{i=1}^n \sum_{j=1}^m (x_i - \mu_X)(y_j - \mu_Y)p_{i,j},$$

где $p_{i,j} = p(x_i, y_j)$.

Если для двух случайных величин X и Y при возрастании одной случайной величины есть тенденция к возрастанию другой, то $\text{cov}(X, Y) > 0$, если при возрастании одной случайной величины есть тенденция к убыванию другой, то $\text{cov}(X, Y) < 0$.

Если же поведение не предсказуемо, то $\text{cov}(X, Y) = 0$. В этом случае говорят, что случайные величины некоррелируемы, но это не значит, что они независимы, правда, для независимых случайных величин $\text{cov}(X, Y) = 0$. Нормализованную ковариацию называют корреляцией

$$-1 \leq \text{cor}(X, Y) = \frac{\text{cov}(X, Y)}{\sqrt{\text{var}(X)\text{var}(Y)}} \leq 1.$$

Пусть $X = (X_1, X_2, \dots, X_n)$ вектор, координаты которого являются случайными величинами, тогда

$$\text{cov}(X) = \text{cov}(X_1, X_2, \dots, X_n) = \Sigma = E((X - u)(X - u)^T) =$$

$$= \begin{pmatrix} E((X_1 - \mu_1)(X_1 - \mu_1)) & \cdots & E((X_n - \mu_n)(X_1 - \mu_1)) \\ \vdots & \ddots & \vdots \\ E((X_1 - \mu_1)(X_n - \mu_n)) & \cdots & E((X_n - \mu_n)(X_n - \mu_n)) \end{pmatrix}.$$

Матрица Σ называется ковариационной.

Для случая многих переменных непрерывная n -мерная случайная величина \mathbf{X} распределена по нормальному закону $N(\mu, \Sigma)$, если ее плотность распределения имеет вид

$$\begin{aligned} f(x) &= \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} ((x - \mu)^T \Sigma^{-1} (x - \mu))\right) = \\ &= \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} \left((x_1 - \mu_1, \dots, x_n - \mu_n) \Sigma^{-1} \begin{pmatrix} x_1 - \mu_1 \\ \vdots \\ x_n - \mu_n \end{pmatrix} \right)\right), \end{aligned}$$

здесь Σ ковариационная матрица, $|\Sigma|$ - ее определитель и Σ^{-1} матрица, обратная к ковариационной.

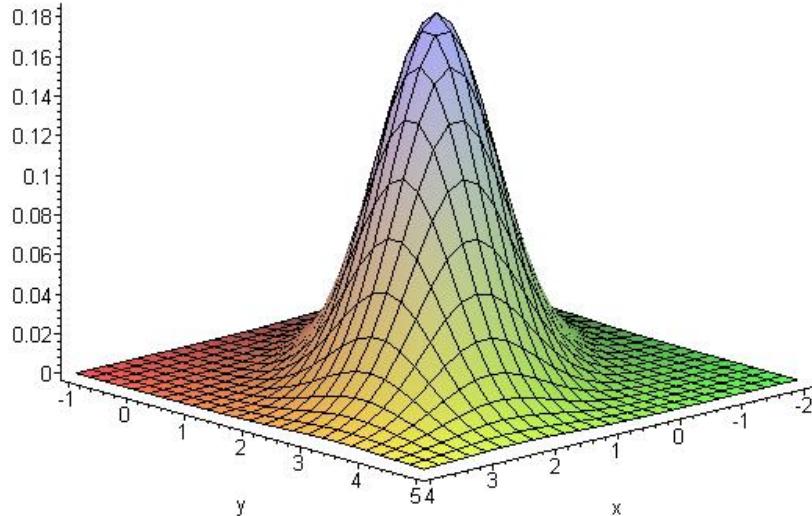


Рис. 1.3. График функции плотности $N\left(\begin{pmatrix} 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}\right)$.

Если все величины (X_1, X_2, \dots, X_n) независимы, то функция плотности будет иметь вид

$$f(x) = \prod_{i=1}^n \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left(-\frac{(x_i - \mu_i)^2}{2\sigma_i^2}\right).$$

Пусть Φ матрица, столбцы которой являются нормированными собственными векторами матрицы Σ , тогда (в силу ортонормированности)

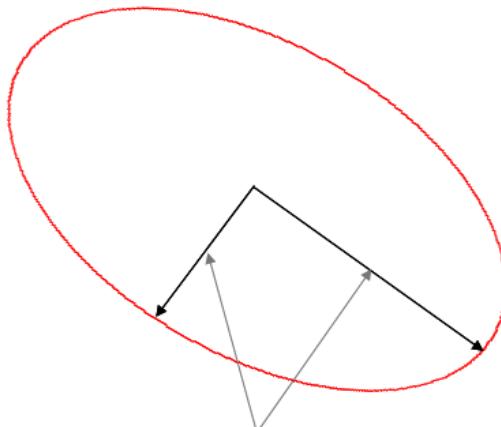
$$\Phi^{-1} = \Phi^T.$$

Если $\Sigma\Phi = \Phi\Lambda$, то Λ диагональная матрица с соответствующими собственными значениями матрицы Σ на диагонали, тогда $\Sigma = \Phi\Lambda\Phi^{-1}$ и, следовательно, $\Sigma^{-1} = \Phi\Lambda^{-1}\Phi^{-1}$. Через $\Lambda^{-1/2}$ обозначим матрицу, такую, что $\Lambda^{-1/2}\Lambda^{-1/2} = \Lambda^{-1}$, тогда $\Sigma^{-1} = (\Phi\Lambda^{-1/2})(\Phi\Lambda^{-1/2})^T = \Xi\Xi^T$.

Таким образом,

$$((x - \mu)^T \Sigma^{-1} (x - \mu)) = ((x - \mu)^T \Xi \Xi^T (x - \mu)) = (\Xi^T (x - \mu))^T (\Xi^T (x - \mu)) = |\Xi^T (x - \mu)|^2.$$

Замечая, что матрица Ξ представляет собой матрицу преобразований (поворотов и растяжений), получаем, что точки x , удовлетворяющие условию $|\Xi^T (x - \mu)|^2 \equiv const$ лежат на эллипсе.

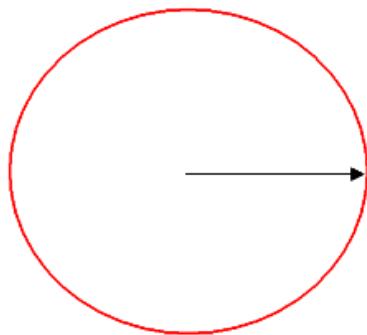


Собственные векторы матрицы Σ

Рис. 1.4. Множество точек равноудаленных от центра

в смысле расстояния Махalanобиса.

Число $\sqrt{((x - \mu)^T \Sigma^{-1} (x - \mu))}$ называется расстоянием Махalanобиса (Mahalanobis) между x и μ . В частности, если все величины (X_1, X_2, \dots, X_n) независимы, то в этом случае расстояние Махalanобиса вырождается в Евклидово расстояние $\sqrt{((x - \mu)^T (x - \mu))}$.



*Рис. 1.5. Множество точек равноудаленных от центра
в смысле расстояния Евклида.*

Приведем пример двумерной функции Гаусса.

Пусть $\mu = (1, 2)$, $\Sigma = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$, тогда линии уровня соответствующей функции

Гаусса будут иметь вид

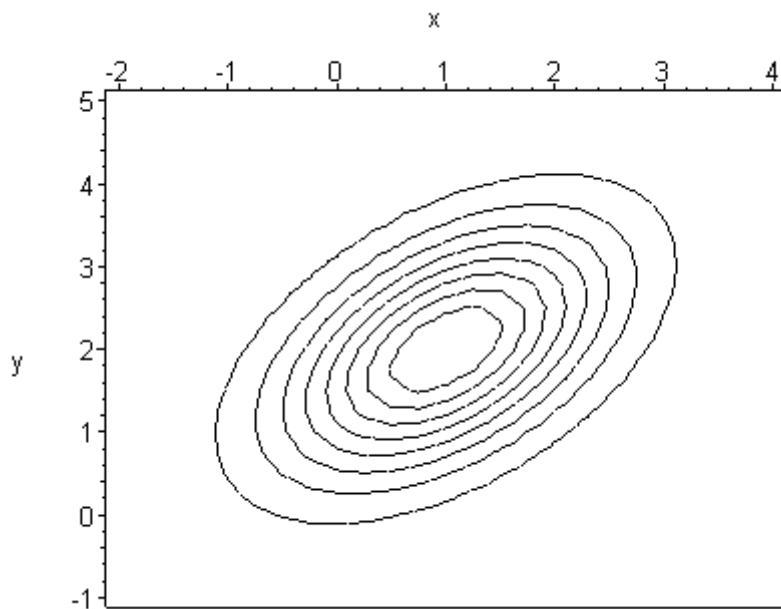


Рис. 1.6. Изолинии функции Гаусса при $\mu = (1, 2)$, $\Sigma = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$.

Если $\mu = (1, 2)$, $\Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, то

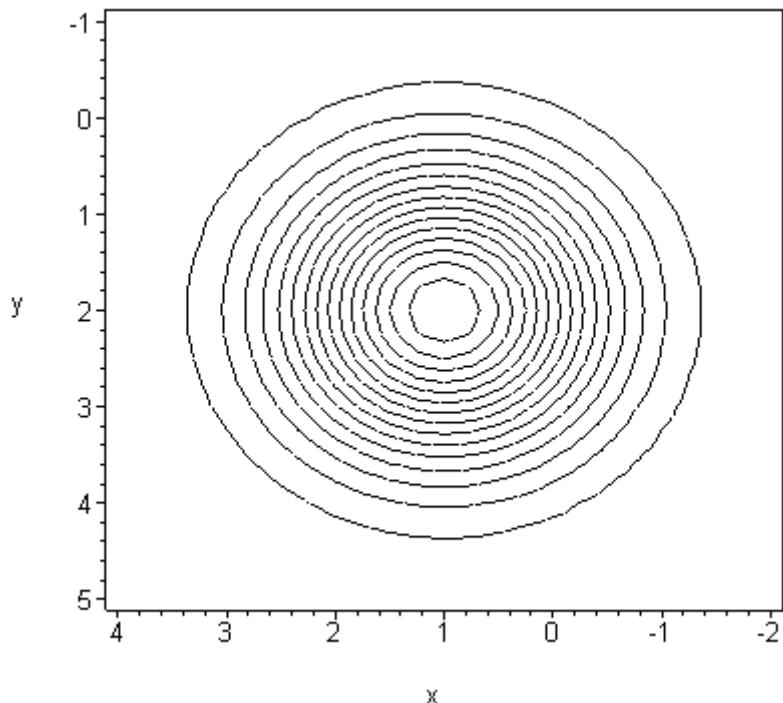


Рис.1.7. Изолинии сферической функции Гаусса при $\mu = (1,2)$, $\Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.

Если n -мерная случайная величина X имеет плотность $N(\mu, \Sigma)$, то величина AX имеет плотность $N(A^T\mu, A^T\Sigma A)$, таким образом для любой случайной величины \mathbf{X} можно подобрать преобразование, переводящее в случайную величину со сферической плотностью распределения.

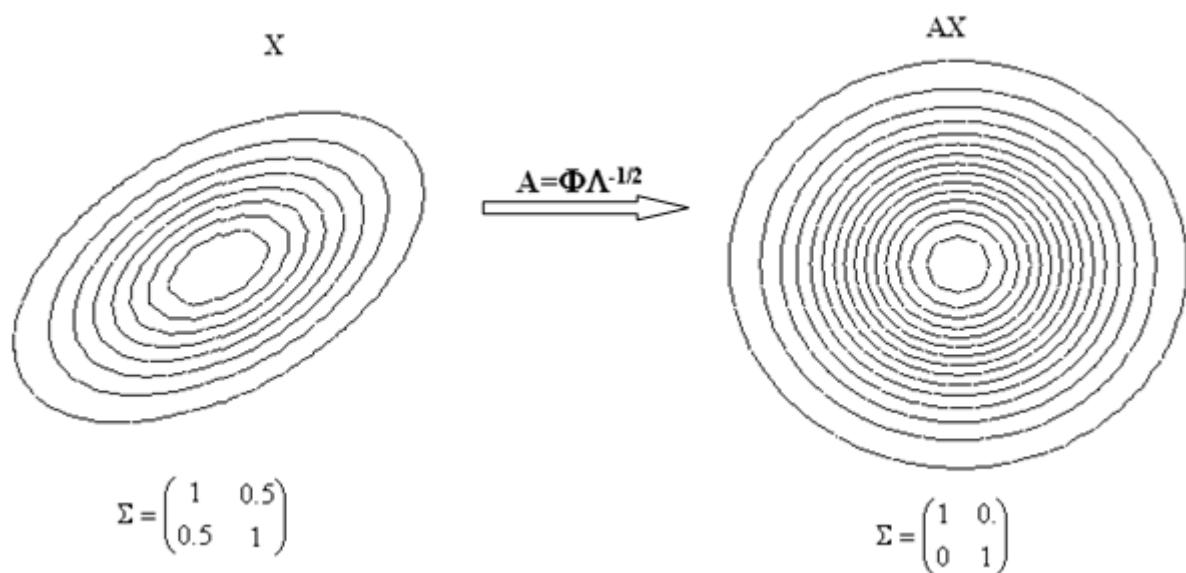


Рис. 1.8. Преобразование случайной величины с плотностью распределения при $\Sigma = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$ случайную величину со сферической функцией плотности.

2. Метод наименьших квадратов

Пусть дана система точек

x_i	x_0	x_1	...	x_N
t_i	t_0	t_1	...	t_N

Табл. 2.1. Экспериментальные данные.

где число точек N велико и данные получены с ошибкой. Такая ситуация является традиционной при обработке экспериментальных данных. В этом случае использование интерполяционных методов нецелесообразно. Кроме того, возможна ситуация, когда известна априорная информация об исследуемом процессе, тогда вид приближающего аппарата определяется технологическими условиями или природой явления. Выбор коэффициентов приближающей функции обусловлен, прежде всего адекватностью используемой модели, то есть, ошибка описания исходных данных должна быть достаточно мала. Понятно, что выбор критерия близости является принципиальным. Как правило, при решении такого рода задач, используют среднеквадратическое расстояние. Это обусловлено, прежде всего, тем, что в этом случае, при использовании линейных методов приближения, функция цели представляет собой квадратичную функцию – параболоид. В силу выпуклости, параболоид имеет единственный экстремум, поэтому необходимое условие экстремума совпадает с достаточным, что существенно упрощает задачу поиска минимального значения функции цели. Соответственно, метод нахождения экстремума этой функции цели называется методом наименьших квадратов ([27], [29], [30], [32], [110]).

Перейдем к изложению метода наименьших квадратов.

Поставим в соответствие исходным данным, заданным в виде табл. 2.1., функцию вида

$$F\left(\{a_i\}_{i=0}^n, t\right) = \sum_{i=0}^n a_i \varphi_i(t) = a_0 \varphi_0(t) + a_1 \varphi_1(t) + \dots + a_n \varphi_n(t), \quad (2.1)$$

где $\varphi_i(t)$, $i = 0, \dots, n$ - базисные функции, a_i – неизвестные коэффициенты, подлежащие определению. В частности, если в качестве базисных функций использовать степенные маномы $\varphi_i(t) = t^i$, задача сводится к поиску полинома

$$F(\{a_i\}_{i=0}^n, t) = \sum_{i=0}^n a_i t^i = a_0 + a_1 t + \dots + a_n t^n$$

степени n , приближающего исходную таблицу.

Для определения коэффициентов a_i будем искать функцию $F(\{a_i\}_{i=0}^n, t)$, отклонение значений которой от заданных таблицей значений x_i минимально в некотором среднем интегральном смысле.

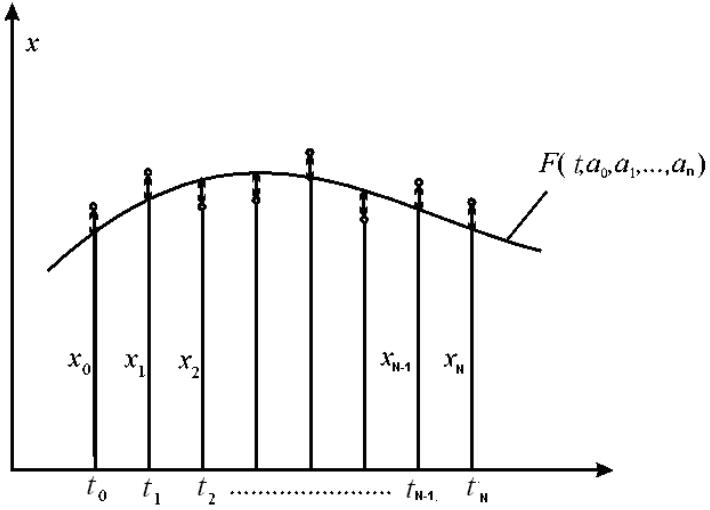


Рис. 2.1. Иллюстрация МНК.

В частности, в дискретном методе наименьших квадратов строится функционал цели

$$\begin{aligned} S(a_0, a_1, \dots, a_n) &= \sum_{i=0}^N (F(a_0, a_1, \dots, a_n, t_i) - x_i)^2 \rho_i^2 = \\ &= \sum_{i=0}^N (a_0 \varphi_0(t_i) + a_1 \varphi_1(t_i) + \dots + a_n \varphi_n(t_i) - x_i)^2 \rho_i^2 \end{aligned} \quad (2.2)$$

где ρ_i – некоторые неотрицательные числа (весовые коэффициенты). Если все значения равноправны, то весовые коэффициенты берутся равными единице.

Геометрически функционал (2.2) представляет собой сумму квадратов отклонений с весом ρ_i экспериментальных данных x_i от значений аппроксимирующей функции $F(a_0, a_1, \dots, a_n, t)$ в точках t_i ($i=0, \dots, N$).

Необходимым (а в данном случае, в силу выпуклости функционала цели, и достаточным) условием минимума функции многих переменных

$$S(a_0, a_1, \dots, a_n) \rightarrow \min_{a_0, a_1, \dots, a_n}$$

является равенство нулю ее частных производных первого порядка по независимым переменным.

$$\begin{cases} \frac{\partial S}{\partial a_0} = 2 \sum_{i=0}^N (F(a_0, a_1, \dots, a_n, t_i) - x_i) \varphi_0(t_i) \rho_i^2 = 0, \\ \frac{\partial S}{\partial a_1} = 2 \sum_{i=0}^N (F(a_0, a_1, \dots, a_n, t_i) - x_i) \varphi_1(t_i) \rho_i^2 = 0, \\ \dots \\ \frac{\partial S}{\partial a_n} = 2 \sum_{i=0}^N (F(a_0, a_1, \dots, a_n, t_i) - x_i) \varphi_n(t_i) \rho_i^2 = 0. \end{cases} \quad (2.3)$$

Полученная система представляет собой систему линейных алгебраических уравнений порядка $n + 1$ относительно неизвестных a_0, a_1, \dots, a_n . Система разрешима при условии $n \leq N$. Ее матрица является симметрической и положительно определенной. Решения a_0, a_1, \dots, a_n доставляют *минимум* функционалу (2.2).

Решение данной системы может быть осуществлено любым из известных методов (например, методом Гаусса, Крамера и др.). Подставляя найденные в результате решения системы (1.3) значения a_0, a_1, \dots, a_n в (1.1), получаем функцию $F(t)$, наилучшим образом приближающую исходные данные 1.1 в среднеквадратическом смысле. Качество такого приближения может быть оценено величиной среднеквадратичного отклонения

$$\sigma = \sqrt{\frac{1}{N+1} \sum_{i=0}^N (x_i - F(t_i))^2 \rho_i^2}.$$

Достаточно часто естественным условием является требование описания исходных данных прямой или параболой. В этом случае говорят, что используется линейная или квадратичная регрессия.

Рассмотрим случай описания априорных данных прямой, то есть используем метод линейной регрессии. Опишем исходные данные $(t_i, x_i), i = 0, 1, \dots, N$ прямой $x = at + b$. В этом случае функция цели (1.2) примет вид

$$S(a, b) = \sum_{i=0}^N (at_i + b - x_i)^2 \rightarrow \min_{a, b}.$$

Необходимое (и, в данном случае, достаточное) условие экстремума имеет вид

$$\begin{cases} \frac{\partial}{\partial a} S(a,b) = 2 \sum_{i=0}^N t_i (at_i + b - x_i) = 0, \\ \frac{\partial}{\partial b} S(a,b) = 2 \sum_{i=0}^N (at_i + b - x_i) = 0, \end{cases}$$

или, что то же,

$$\begin{cases} a \sum_{i=0}^N t_i^2 + b \sum_{i=0}^N t_i = \sum_{i=0}^N x_i t_i, \\ a \sum_{i=0}^N t_i + b(N+1) = \sum_{i=0}^N x_i. \end{cases}$$

Отсюда, применяя метод Крамера решения систем линейных уравнений, сразу получаем коэффициенты прямой (линейной регрессии)

$$a = \frac{(N+1) \sum_{i=0}^N x_i t_i - \sum_{i=0}^N t_i \sum_{i=0}^N x_i}{(N+1) \sum_{i=0}^N t_i^2 - \left(\sum_{i=0}^N t_i \right)^2}, b = \frac{\sum_{i=0}^N t_i^2 \sum_{i=0}^N x_i - \sum_{i=0}^N t_i \sum_{i=0}^N x_i t_i}{(N+1) \sum_{i=0}^N t_i^2 - \left(\sum_{i=0}^N t_i \right)^2}.$$

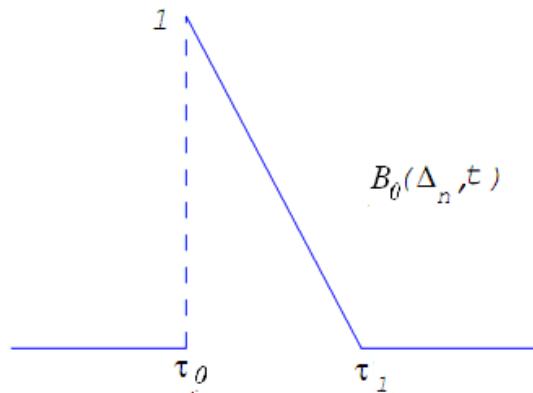
Достаточно распространенным аппаратом приближения являются кусочно-полиномиальные функции или сплайны. Наиболее распространенным видом сплайнов являются ломаные или полигональные функции. Рассмотрим в качестве регрессионной модели ломаную с весовой функцией равной единице. Пусть Δ_n фиксированное разбиение отрезка $[t_0, T]$ точками $t_i (i = 0, 1, 2, \dots, n)$, и $\mathfrak{R}(\Delta_n)$ множество ломаных $P(\Delta_n, t) = P(\{a_i\}_{i=0}^n, \Delta_n, t)$ с узлами в точках разбиения Δ_n . Тогда задача нахождения кусочно-линейной регрессионной модели с фиксированными узлами имеет вид

$$\inf \left\{ \sum_{i=0}^N (x_i - P(\Delta_n, t_i))^2 \mid P(\Delta_n) \in \mathfrak{R}(\Delta_n) \right\}.$$

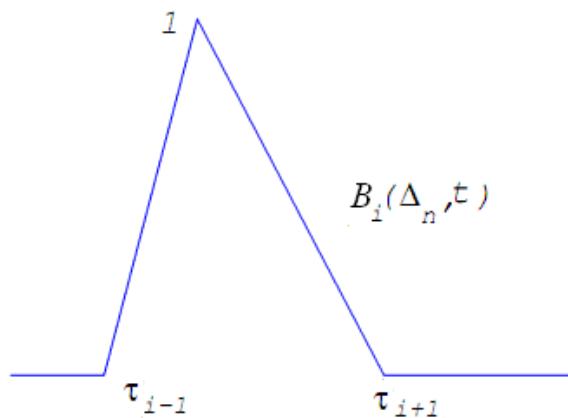
Нетрудно видеть, что

$$P(\Delta_n, t) = P(\{a_i\}_{i=0}^n, \Delta_n, t) = \sum_{i=0}^n a_i B_i(\Delta_n, t),$$

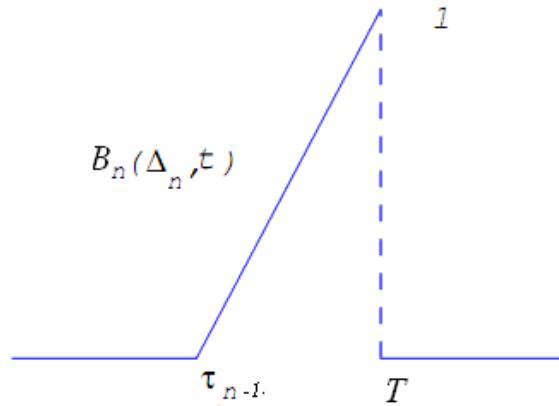
где $B_i(\Delta_n, t) (i = 0, \dots, n)$ базисные функции, которые можно записать в следующем виде:



$$B_0(\Delta_n, t) = \begin{cases} (\tau_1 - t)(\tau_1 - \tau_0)^{-1}, & t \in [\tau_0, \tau_1], \\ 0, & \text{иначе,} \end{cases}$$



$$B_i(\Delta_n, t) = \begin{cases} (\tau_{i-1} - t)(\tau_{i-1} - \tau_i)^{-1}, & t \in [\tau_{i-1}, \tau_i], \\ (\tau_{i+1} - t)(\tau_{i+1} - \tau_i)^{-1}, & t \in [\tau_i, \tau_{i+1}], \\ 0, & \text{иначе,} \end{cases} \quad (i = 2, \dots, n-1).$$



$$B_n(\Delta_n, t) = \begin{cases} (\tau_{n-1} - t)(\tau_{n-1} - T)^{-1}, & t \in [\tau_{n-1}, T], \\ 0, & \text{иначе,} \end{cases}$$

Выпишем функцию цели

$$S(a_0, a_1, \dots, a_n) = \sum_{i=0}^N \left(\sum_{j=0}^n a_j B_j(\Delta_n, t_i) - x_i \right)^2,$$

и найдем решение задачи $S(a_0, a_1, \dots, a_n) \rightarrow \min_{a_0, a_1, \dots, a_n}$. Необходимое и достаточное условие экстремума будет иметь вид

$$\frac{\partial}{\partial a_i} S(a_0, a_1, \dots, a_n) = 0, i = 0, 1, \dots, n.$$

Нахождение экстремума сводится к решению системы уравнений

$$\begin{pmatrix} \langle B_0, B_0 \rangle & \langle B_0, B_1 \rangle & \cdots & \langle B_0, B_n \rangle \\ \langle B_1, B_0 \rangle & \langle B_1, B_1 \rangle & \cdots & \langle B_1, B_n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle B_n, B_0 \rangle & \langle B_n, B_1 \rangle & \cdots & \langle B_n, B_n \rangle \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \langle x, B_0 \rangle \\ \langle x, B_1 \rangle \\ \vdots \\ \langle x, B_n \rangle \end{pmatrix},$$

где

$$\langle x, B_j \rangle = \sum_{i=0}^N x_i B_j(\Delta_n, t_i), j = 0, 1, \dots, n,$$

а замечая, что $\langle B_i, B_j \rangle = 0, \forall i, j : |i - j| \geq 2$, получаем систему уравнений с трехдиагональной матрицей

$$A = \begin{pmatrix} \langle B_0, B_0 \rangle & \langle B_0, B_1 \rangle & 0 & \dots & 0 \\ \langle B_1, B_0 \rangle & \langle B_1, B_1 \rangle & \langle B_1, B_2 \rangle & \dots & 0 \\ 0 & \langle B_2, B_1 \rangle & \langle B_2, B_2 \rangle & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & \langle B_n, B_n \rangle \end{pmatrix}.$$

Применяя метод прогонки, получаем эффективный алгоритм нахождения уравнения кусочно-линейной регрессии с фиксированными узлами.

Приведем пример построения ломаной методом наименьших квадратов.

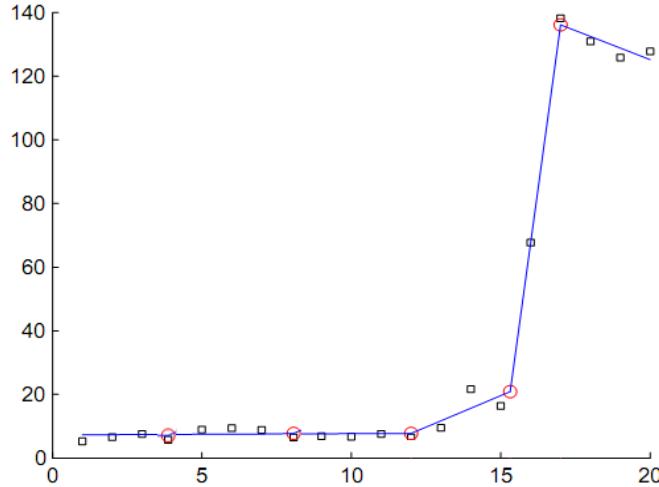


Рис. 2.1. Приближение дискретных данных ломаной по МНК.

Линеаризация при методе наименьших квадратов

Приведенная методология определения аппроксимирующих функций методом наименьших квадратов годится лишь для функций, у которых неопределенные коэффициенты заданы линейно. Если же это условие не выполняется, то прямое использование метода наименьших квадратов невозможно.

Как же быть в этом случае? Возможно ли вообще использовать в таких случаях метод наименьших квадратов? Да. Возможно. Но тогда нужно проводить некоторые дополнительные построения, линеаризующие (по коэффициентам) приближающую функцию (см. [27]).

Проиллюстрируем это на нескольких примерах.

- Пусть приближающая функция имеет вид $x = \frac{1}{\alpha t + \beta}$.

Тогда для x_i ошибка будет равна

$$\delta_i = x_i - \frac{1}{\alpha t_i + \beta}. \quad (2.4)$$

Прямое использование метода наименьших квадратов приводит к минимизации величины

$$\sum_{i=1}^n \delta_i^2 = \sum_{i=1}^n \left(x_i - \frac{1}{\alpha t_i + \beta} \right)^2. \quad (2.5)$$

Взяв частные производные по α и β , и приравняв их нулю, получим систему из двух нелинейных уравнений, которая не подлежит точному решению. В связи с этим, проведем некоторые построения.

Рассмотрим величины

$$\Delta_i = x_i(\alpha t_i + \beta) - 1, (i=1,2,\dots,n).$$

Установим зависимость между Δ_i и δ_i . Из (2.4) получаем

$$\alpha t_i + \beta = \frac{1}{x_i - \delta_i}.$$

Тогда

$$\Delta_i = \frac{x_i}{x_i - \delta_i} - 1 = \frac{\delta_i}{x_i - \delta_i}, (i=1,2,\dots,n),$$

и, следовательно, при малых Δ_i

$$\delta_i = \frac{x_i \Delta_i}{\Delta_i + 1} \approx x_i \Delta_i.$$

Тогда задача (2.5) сводится к задаче определения коэффициентов α и β так, чтобы величина

$$\sum_{i=1}^n (\Delta_i)^2 = \sum_{i=1}^n (1 - \alpha t_i x_i - \beta x_i)^2 x_i^2.$$

была минимальной.

Таким образом, мы пришли к задаче (1.2) при условии, что приближаемая функция тождественно равна единице и $\varphi_0(t) = tx(t)$, $\varphi_1(t) = x(t)$ с весом $\rho_i = x_i$.

Погрешность приближения имеет вид

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n \left(x_i - \frac{1}{\alpha t_i + \beta} \right)^2}.$$

2. Рассмотрим другой пример. Пусть приближающая функция имеет вид

$$x = \frac{t}{\alpha t + \beta}.$$

Для x_i ошибка будет равна

$$\delta_i = x_i - \frac{t_i}{\alpha t_i + \beta} \quad (2.6)$$

и

$$\Delta_i = x_i(\alpha t_i + \beta) - t_i, \quad (i = 1, 2, \dots, n).$$

Установим связь между Δ_i и δ_i . Из (1.6) имеем

$$\alpha t_i + \beta = \frac{t_i}{x_i - \delta_i}.$$

Тогда

$$\Delta_i = \frac{t_i x_i}{x_i - \delta_i} - t_i = \frac{t_i \delta_i}{x_i - \delta_i}, \quad (i = 1, 2, \dots, n),$$

Отсюда при малых Δ_i

$$\delta_i = \frac{x_i \Delta_i}{\Delta_i + t_i} \approx \frac{x_i}{t_i} \Delta_i.$$

Для малых δ_i

$$\sum_{i=1}^n \delta_i^2 \approx \sum_{i=1}^n \left(\frac{x_i}{t_i} \Delta_i \right)^2$$

и

$$\sum_{i=1}^n \left(\frac{x_i}{t_i} \Delta_i \right)^2 = \sum_{i=1}^n (t_i - \alpha t_i x_i - \beta x_i)^2 \left(\frac{x_i}{t_i} \right)^2.$$

Таким образом, мы пришли к задаче (2.2) при условии, что приближаемая функция тождественно равна t и $\varphi_0(t) = tx(t)$, $\varphi_1(t) = x(t)$ и $\rho_i = \frac{x_i}{t_i}$.

Погрешность приближения имеет вид

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n \left(x_i - \frac{t_i}{\alpha t_i + \beta} \right)^2}.$$

3. Наконец, пусть приближающая функция имеет вид

$$x = \frac{\alpha t + \beta}{\gamma t + 1}.$$

Задача состоит в нахождении коэффициентов α, β, γ , при которых величина

$$\sum_{i=1}^n \delta_i^2 = \sum_{i=1}^n \left(x_i - \frac{\alpha t_i + \beta}{\gamma t_i + 1} \right)^2$$

будет минимальной. Линеаризуем эту задачу.

Пусть

$$\Delta_i = \gamma t_i x_i + x_i - \alpha t_i - \beta, (i = 1, 2, \dots, n).$$

Установим связь между Δ_i и δ_i . Из предыдущего получаем

$$\frac{\Delta_i}{\gamma t_i + 1} = x_i - \frac{\alpha t_i + \beta}{\gamma t_i + 1}.$$

Таким образом, имеем

$$\frac{\Delta_i}{\gamma t_i + 1} = \delta_i.$$

Для малых δ_i получаем задачу, эквивалентную искомой

$$\sum_{i=1}^n \left(\frac{\Delta_i}{\gamma t_i + 1} \right)^2 = \sum_{i=1}^n (x_i - \alpha t_i - \beta + \gamma t_i x_i)^2 \left(\frac{1}{\gamma t_i + 1} \right)^2 \rightarrow \min.$$

Использовать метод наименьших квадратов не представляется возможным, так как в "вес" входит неизвестный параметр γ . Поэтому рассмотрим итерационный метод пошагового уточнения весовых коэффициентов.

Для задачи (1.2) положим $\varphi_0(t) = t, \varphi_1(t) = 1, \varphi_2(t) = -tx(t)$ и $\rho_i = 1$.

Решая эту задачу, получаем первое приближение $\alpha_1, \beta_1, \gamma_1$.

Полагая теперь $\varphi_0(t) = t, \varphi_1(t) = 1, \varphi_2(t) = -tx(t), \rho_i = \frac{1}{\gamma_1 t_i + 1}$. и снова решая эту за-

дачу, получаем $\alpha_2, \beta_2, \gamma_2$. Продолжая этот процесс при $\varphi_0(t) = t, \varphi_1(t) = 1, \varphi_2(t) = -tx(t), \rho_i = \frac{1}{\gamma_2 t_i + 1}$, получим следующее приближение значений α, β, γ .

Итерацию будем продолжать до тех пор, пока не будут выполняться соотношения

$$\begin{cases} |\alpha_k - \alpha_{k-1}| < \varepsilon, \\ |\beta_k - \beta_{k-1}| < \varepsilon, \\ |\gamma_k - \gamma_{k-1}| < \varepsilon, \end{cases}$$

где ε - заданная погрешность.

Естественно, все множество используемых регрессионных моделей не исчерпывается дробно-линейными функциями, достаточно часто используются степенные и показательные функции.

4. Будем искать приближающую функцию в виде $x = \alpha t^\beta$. Для всех $i = 1, 2, \dots, n$ положим $\delta_i = y_i - \alpha t_i^\beta$ и

$$\Delta_i = \ln x_i - \ln \alpha - \beta \ln t_i = \ln \frac{x_i}{\alpha t_i^\beta}.$$

Как и ранее, установим связь между этими величинами. Из первого равенства найдем $\alpha t_i^\beta = x_i - \delta_i$ и подставим во второе

$$\Delta_i = \ln \frac{x_i}{x_i - \delta_i}.$$

Отсюда $x_i - \delta_i = x_i \exp(-\Delta_i)$ при малых Δ_i можем записать

$\delta_i = x_i(1 - \exp(-\Delta_i)) \approx x_i \Delta_i$. Таким образом, задачу минимизации величины $\sum_{i=1}^n \delta_i^2$ можно

заменить задачей минимизации величины

$$\sum_{i=1}^n (x_i \Delta_i)^2 = \sum_{i=1}^n (\ln x_i - \ln \alpha - \beta \ln t_i)^2 x_i^2,$$

которая является задачей (2.2).

Погрешность приближения имеет вид

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \alpha t_i^\beta)^2}.$$

5. Пусть приближающая функция задана в виде $x = \alpha \beta^t$. Для всех $i = 1, 2, \dots, n$ положим $\delta_i = x_i - \alpha \beta^{t_i}$ и $\Delta_i = \ln x_i - \ln \alpha - t_i \ln \beta = \ln \frac{x_i}{\alpha \beta^{t_i}}$.

Найдем связь между этими величинами.

Выражая из первого равенства $\alpha \beta^{t_i} = x_i - \delta_i$ и подставляя во второе, получаем

$$\Delta_i = \ln \frac{x_i}{x_i - \delta_i}.$$

Следовательно, $\delta_i = x_i(1 - \exp(-\Delta_i)) \approx x_i \Delta_i$. Таким образом, по аналогии, задачу минимизации величины $\sum_{i=1}^n \delta_i^2$ можно заменить задачей минимизации величины

$$\sum_{i=1}^n (x_i \Delta_i)^2 = \sum_{i=1}^n (\ln x_i - \ln \alpha - t_i \ln \beta)^2 x_i^2,$$

которая является задачей (2.2).

Погрешность приближения будет иметь вид

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \alpha \beta^{t_i})^2}.$$

3. Метод главных компонент

(Principle Component Analysis)

В рассмотренных ранее случаях нам был известен вид регрессионной модели и требовалось определить те или иные количественные характеристики, определяющие эту модель. А что, если нет информации ни о качественных, ни о количественных характеристиках регрессии? Как быть в этом случае? Эффективным методом решения такого рода задач является метод главных компонент (Principle Component Analysis - PCA).

Метод главных компонент (МНК) — один из основных способов уменьшения размерности данных с потерей минимального количества информации, разработан Карлом Пирсоном (Karl Pearson) в 1901 г. Применяется во многих областях, таких как распознавание образов, компьютерное зрение, сжатие данных и т. п. Нахождение главных компонент сводится к вычислению собственных векторов и собственных значений ковариационной матрицы исходных данных. Иногда метод главных компонент называют преобразованием Кархунена-Лоэва (Karhunen-Loeve) или преобразованием Хотеллинга (Hotelling transform) (см., например, [1],[2]).

Перейдем к рассмотрению метода МГК (PCA). Вначале найдем константу μ , которая наилучшим образом описывает исходные данные

$$\varepsilon(\mu) = \sum_{i=1}^n (x_i - \mu)^2 \rightarrow \min_{\mu} .$$

Для нахождения минимума приравняем производную нулю и найдем значение μ доставляющее минимум

$$\frac{d}{d\mu} \varepsilon(\mu) = -2 \sum_{i=1}^n (x_i - \mu) = 0 \Rightarrow \sum_{i=1}^n \mu = \sum_{i=1}^n x_i \Rightarrow \mu n = \sum_{i=1}^n x_i \Rightarrow \mu = \frac{1}{n} \sum_{i=1}^n x_i .$$

Далее проведем центрирование данных, то есть, переопределим исходные значения $x_{new} = x_{old} - \mu$, вычтя из каждого полученное значение μ . Ясно, что новые данные имеют математическое ожидание равное нулю

$$E(X - E(X)) = E(X) - E(X) = 0.$$

По сути дела, мы сделали параллельный перенос в существующей системе координат.

В дальнейшем будем считать, что исходные данные центрированы и мы хотим найти самое точное представление данных $D = \{x_1, \dots, x_n\}$ в некотором подпространстве W , которое имеет размерность $k < n$.

Пусть $\{e_1, \dots, e_k\}$ ортонормированный базис W . Любой вектор из W может быть написан в виде линейной комбинации векторов базиса, следовательно, x_1 можно поставить в соответствие некоторый вектор $\sum_{i=1}^k \alpha_{1,i} e_i$ из W . Ошибка между ними вычисляется следующим образом

$$\varepsilon_1 = \left\| x_1 - \sum_{i=1}^k \alpha_{1,i} e_i \right\|_2^2 = \left\langle x_1 - \sum_{i=1}^k \alpha_{1,i} e_i, x_1 - \sum_{i=1}^k \alpha_{1,i} e_i \right\rangle.$$

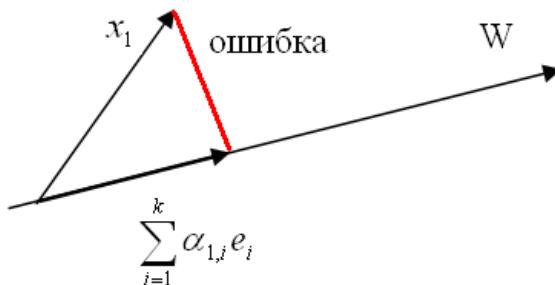


Рис. 3.1. Иллюстрация ошибки восстановления вектора.

Чтобы найти полную ошибку, нам нужно просуммировать величины ошибок по всем x_j , поэтому полная ошибка равна

$$\underbrace{\varepsilon(e_1, \dots, e_k, \alpha_{1,1}, \dots, \alpha_{n,k})}_{\text{unknowns}} = \sum_{j=1}^n \varepsilon_j^2 = \sum_{j=1}^n \left\| x_j - \sum_{i=1}^k \alpha_{j,i} e_i \right\|_2^2. \quad (3.1)$$

Чтобы минимизировать ошибку, нужно взять частные производные и учесть ограничения на ортогональность $\{e_1, \dots, e_k\}$. Вначале упростим соотношение (3.1)

$$\begin{aligned} \varepsilon(e_1, \dots, e_k, \alpha_{1,1}, \dots, \alpha_{n,k}) &= \sum_{j=1}^n \left\| x_j - \sum_{i=1}^k \alpha_{j,i} e_i \right\|_2^2 = \sum_{j=1}^n \|x_j\|_2^2 - 2 \sum_{j=1}^n x_j^T \sum_{i=1}^k \alpha_{j,i} e_i + \sum_{j=1}^n \sum_{i=1}^k \alpha_{j,i}^2 = \\ &= \sum_{j=1}^n \|x_j\|_2^2 - 2 \sum_{j=1}^n \sum_{i=1}^k \alpha_{j,i} x_j^T e_i + \sum_{j=1}^n \sum_{i=1}^k \alpha_{j,i}^2. \end{aligned}$$

Тогда

$$\frac{\partial}{\partial \alpha_{m,l}} \varepsilon(e_1, \dots, e_k, \alpha_{1,1}, \dots, \alpha_{n,k}) = -2x_m^T e_l + 2\alpha_{m,l}.$$

Необходимое и достаточное условие экстремума будет иметь вид

$$-2x_m^T e_l + 2\alpha_{m,l} = 0 \Rightarrow \alpha_{m,l} = x_m^T e_l.$$

Таким образом, ошибка (3.1) примет вид

$$\varepsilon(e_1, \dots, e_k) = \sum_{j=1}^n \|x_j\|_2^2 - 2 \sum_{j=1}^n \sum_{i=1}^k (x_j^T e_i) x_j^T e_i + \sum_{j=1}^n \sum_{i=1}^k (x_j^T e_i)^2.$$

Упрощая это соотношение, получаем

$$\varepsilon(e_1, \dots, e_k) = \sum_{j=1}^n \|x_j\|_2^2 - \sum_{j=1}^n \sum_{i=1}^k (x_j^T e_i)^2. \quad (3.2)$$

Учитывая, что $\langle a, b \rangle = a^T b$ и $\langle b, a \rangle = \langle a, b \rangle$, получаем

$$(a^T b)^2 = (a^T b)(a^T b) = (b^T a)(a^T b) = b^T (aa^T) b,$$

следовательно,

$$\varepsilon(e_1, \dots, e_k) = \sum_{j=1}^n \|x_j\|_2^2 - \sum_{i=1}^k e_i^T \left(\sum_{j=1}^n (x_j x_j^T) \right) e_i = \sum_{j=1}^n \|x_j\|_2^2 - \sum_{i=1}^k e_i^T S e_i,$$

где $S = \sum_{j=1}^n (x_j x_j^T)$ является ковариационной матрицей.

Следующим шагом будет минимизация $\varepsilon(e_1, \dots, e_k) = \sum_{j=1}^n \|x_j\|_2^2 - \sum_{i=1}^k e_i^T S e_i$ при условии $e_i^T e_i = 1$ для всех i . Используя метод неопределенных множителей Лагранжа (Lagrange), введем множители $\lambda_1, \dots, \lambda_k$ и, замечая, что $\sum_{j=1}^n \|x_j\|_2^2 \equiv Const$, выпишем функцию цели

$$\ell(e_1, \dots, e_k) = \sum_{i=1}^k e_i^T S e_i - \sum_{i=1}^k \lambda_i (e_i^T e_i - 1)$$

Замечая, что $\frac{d}{dX} (X^T X) = \frac{d}{dX} \langle X, X \rangle = 2X$ и если А симметрическая матрица, то

$$\frac{d}{dX} (X^T A X) = 2AX, \text{ получаем,}$$

$$\frac{\partial}{\partial e_m} \ell(e_1, \dots, e_k) = 2S e_m - 2\lambda_m e_m = 0,$$

то есть, $Se_m = \lambda_m e_m$. Таким образом, необходимо найти решение уравнения $(S - \lambda I)e = 0$ (здесь I - единичная матрица), что эквивалентно тому, что λ_m и e_m есть собственные значения и собственные векторы ковариационной матрицы S .

При этом ошибка приобретает вид

$$\varepsilon(e_1, \dots, e_k) = \sum_{j=1}^n \|x_j\|_2^2 - \sum_{i=1}^k \lambda_i \|e_i\|_2^2 = \sum_{j=1}^n \|x_j\|_2^2 - \sum_{i=1}^k \lambda_i. \quad (3.3)$$

Минимизация (2.3) состоит в выборе базиса W из k собственных векторов матрицы S , которые соответствуют k наибольшим собственным значениям. Большее собственное значение S дает большую вариацию в направлении соответственного собственного вектора. Этот результат можно переформулировать следующим образом – проекция X на подпространство размерности k , которая обеспечивает наибольшую вариацию. Таким образом МГК может трактоваться следующим образом - берем ортогональный базис и вращаем его пока на одном из направлений не получим максимальную вариацию. Фиксируем это направление и вращаем остальные, пока не найдем второе направление и так далее.

Пусть $\{e_1, \dots, e_n\}$ все собственные векторы матрицы S , сортированные в порядке уменьшения соответственного собственного значения, тогда для любого

$$x_i = \sum_{j=1}^n \alpha_{i,j} e_j = \underbrace{\alpha_{i,1} e_1 + \dots + \alpha_{i,k} e_k}_{approximation} + \overbrace{\alpha_{i,k+1} e_{k+1} + \dots + \alpha_{i,n} e_n}^{error}$$

коэффициенты $\alpha_{m,l} = x_m^T e_l$ являются координатами главных компонент, и чем больше значение k , тем получаем лучшую аппроксимацию. При этом главные компоненты располагаются в порядке значимости, более важные имеют меньший номер.

Приведем алгоритмизацию МГК .

Пусть $D = \{x_1^0, \dots, x_n^0\}$ исходные (оригинальные) данные, каждый из этих векторов x_i^0 имеет размерность N

1. Найдем среднее $\mu = \frac{1}{n} \sum_{i=1}^n x_i^0$.
2. Вычтем среднее из каждого вектора $x_i = x_i^0 - \mu$.
3. Найдем ковариационную матрицу $S = \sum_{j=1}^n x_j x_j^T$.

4. Вычислим собственные векторы $\{e_1, \dots, e_k\}$, соответствующие к наибольшим собственным значениям S .

5. Пусть $\{e_1, \dots, e_k\}$ составляют матрицу $E = [e_1 \dots e_k]$.

6. Тогда самой близкой аппроксимацией к x является $z = E^T x$.

Рассмотрим пример.

Пусть дано множество данных $D^0((x_1^0, y_1^0), \dots, (x_8^0, y_8^0))$, определенных таблицей:

x	1	2	3	4	5	6	7	8
y	2	3	2	4	4	7	6	7

Табл. 3.1. Исходные данные.

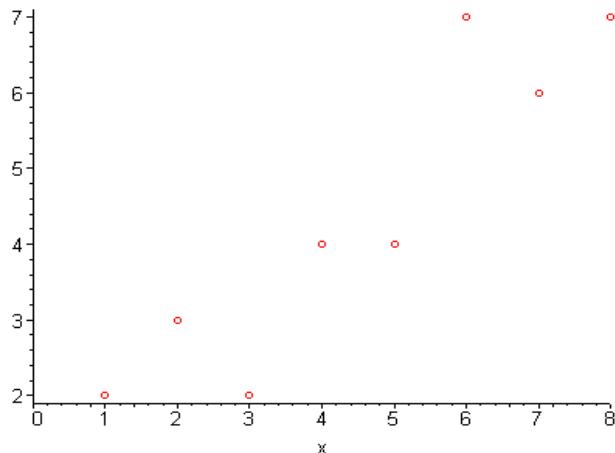


Рис. 3.3. Исходные данные.

Найдем среднее значение $\mu = (4.5, 4.375)$, тогда после центрирования данные D примут вид

x	-3,5	-2,5	-1,5	-0,5	0,5	1,5	2,5	3,5
y	-2,375	-1,375	-2,375	-0,375	-0,375	2,625	1,625	2,625

Табл. 3.2. Центрированные данные.

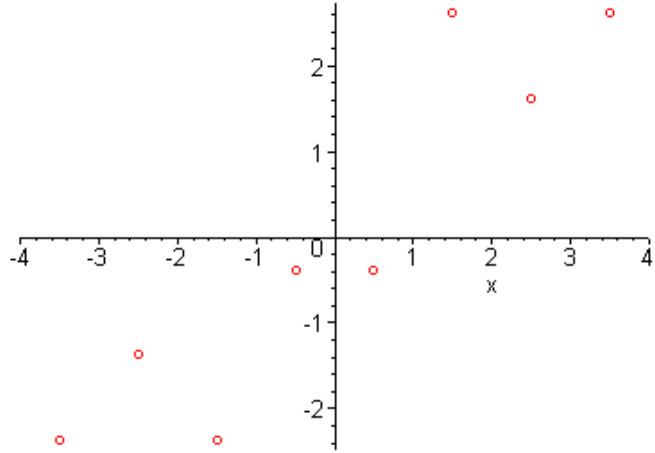


Рис. 3.4. Параллельный сдвиг, совмещающий начало координат с математическим ожиданием исходных данных.

Тогда

$$s_{1,1} = \langle x, x \rangle = \sum_{i=1}^8 x_i x_i = 42, \quad s_{2,1} = s_{1,2} = \langle x, y \rangle = \sum_{i=1}^8 x_i y_i = 32,5, \quad s_{2,2} = \langle y, y \rangle = \sum_{i=1}^8 y_i y_i = 29,875,$$

и ковариационная матрица будет иметь вид

$$S = \begin{pmatrix} s_{1,1} & s_{1,2} \\ s_{2,1} & s_{2,2} \end{pmatrix} = \begin{pmatrix} 42 & 32,5 \\ 32,5 & 29,875 \end{pmatrix}.$$

Решая уравнение

$$\begin{vmatrix} 42 - \lambda & 32,5 \\ 32,5 & 29,875 - \lambda \end{vmatrix} = 0 \Leftrightarrow (42 - \lambda)(29,875 - \lambda) - (32,5)^2 = 0$$

получаем собственные значения $\lambda_1 = 68,99810959, \lambda_2 = 2,876890413$.

Для определения собственных векторов $e_1 = (e_{1,1}, e_{1,2})$ и $e_2 = (e_{2,1}, e_{2,2})$ найдем любое нетривиальное решение системы

$$\begin{cases} (s_{1,1} - \lambda_1)e_{1,1} + s_{1,2}e_{1,2} = 0, \\ s_{1,2}e_{1,1} + (s_{2,2} - \lambda_1)e_{1,2} = 0, \end{cases}$$

и, соответственно, системы

$$\begin{cases} (s_{1,1} - \lambda_2)e_{2,1} + s_{1,2}e_{2,2} = 0, \\ s_{1,2}e_{2,1} + (s_{2,2} - \lambda_2)e_{2,2} = 0. \end{cases}$$

Так как значения λ выбирались из условия равенства нулю главного определителя данной системы уравнений, то уравнения линейно зависимы и для нахождения

любого нетривиального решения в качестве одной неизвестной взять любое ненулевое значение и, подставив его, найти из любого уравнения значение второй неизвестной, например, $e_{1,1} = 1, e_{1,2} = 0,8307110643$ и $e_{2,1} = 1, e_{2,2} = -1,203787987$.

Таким образом, вектор $e_1 = (1, 0.8307110643)^T$, соответствует собственному значению $\lambda_1 = 68.99810959$, а значению $\lambda_2 = 2.876890413$ соответствует вектор $e_2 = (1, -1.203787987)^T$. Большему собственному значению соответствует более главное направление. Нормируя собственные векторы единицей, получаем $e_1 = (0.7692123649, 0.6389932223)^T$ и $e_2 = (0.6389932223, -0.7692123648)^T$.

Остается выписать главную компоненту $z_1 = e_1^T D$:

z_1	-4,20985	-2,80164	-2,67142	-0,62428	0,14498	2,83117	2,96139	4,3696
-------	----------	----------	----------	----------	---------	---------	---------	--------

Табл. 3.3. Первая главная компонента.

Соответственно, вторая компонента $z_2 = e_2^T D$ будет иметь вид

z_2	-0,4096	-0,53982	0,86839	-0,03104	0,60795	-1,0607	0,34751	0,21729
-------	---------	----------	---------	----------	---------	---------	---------	---------

Табл. 3.4. Вторая главная компонента.

Заметим, что для получения приближения исходных данных (нецентрированных) нужно прибавить соответствующее среднее значение.

Восстановление данных одной главной компонентой (то есть проекциями исходных данных на главное направление) будет иметь вид $x_i = e_{1,1} z_{1,i} + \mu_1$, $y_i = e_{1,2} z_{1,i} + \mu_2$:

$0,769 \times z_1 + 4,5$	1,26174	2,3449	2,4451	4,0198	4,61158	6,6778	6,778	7,8611
$0,639 \times z_1 + 4,375$	1,6849	2,5848	2,668	3,9761	4,4676	6,1841	6,2673	7,41671

Табл. 3.5. Исходные данные, восстановленные по первой главной компоненте.

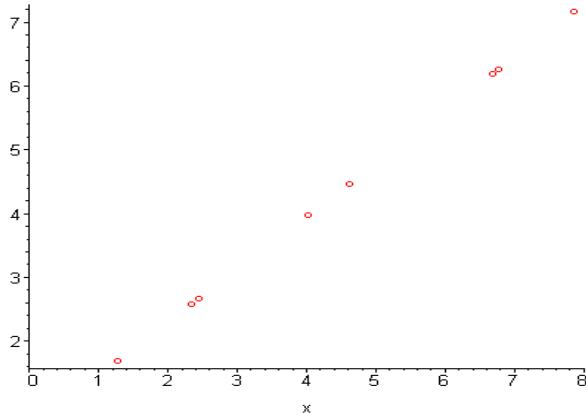


Рис. 3.6. Представление данных одной главной компонентой.

Итерационный алгоритм вычисления главных компонент

Описанный метод определения главных компонент является достаточно ресурсоемким и неустойчивым, особенно в случае, если собственные значения матрицы близки к нулю.

Часть более эффективным является использование итерационного метода определения главных компонент. Для этой цели рассмотрим задачу (3.1) с другой точки зрения.

Для случая $i=1$ задача (3.1) сводится к определению одной компоненты e_1 , которая наилучшим образом восстанавливает все исходные данные $\{x_1, \dots, x_n\}$

$$\varepsilon(e_1, \alpha_{1,1}, \dots, \alpha_{n,1}) = \sum_{j=1}^n \|x_j - \alpha_{j,1}e_1\|_2^2 \rightarrow \min \quad (3.4)$$

по всем e_1 и $\{\alpha_{i,1}\}_{i=1}^n$ при условии $\sum_{i=1}^n \alpha_{i,1}^2 = 1$.

Если $\{\tilde{\alpha}_{i,1}\}_{i=1}^n$ и \tilde{e}_1 есть решение этой задачи и $\Delta x_j = x_j - \tilde{\alpha}_{j,1}\tilde{e}_1$ - ошибка восстановления данных одной первой главной компонентой, то решая задачу

$$\sum_{j=1}^n \|\Delta x_j - \alpha_{j,2}e_2\|_2^2 \rightarrow \min$$

по всем e_2 и $\{\alpha_{i,2}\}_{i=1}^n$ при условии $\sum_{i=1}^n \alpha_{i,2}^2 = 1$, получаем вторую главную компоненту \tilde{e}_2

и соответствующий вектор $\{\tilde{\alpha}_{i,2}\}_{i=1}^n$ и т.д.

При фиксированных $\{\alpha_{i,1}\}_{i=1}^n$ задача (3.4) решается методом наименьших квадратов. В силу того, что функция цели представляет собой квадратичный функционал,

необходимое и достаточное условия экстремума совпадают. Таким образом, решение задачи сводится к поиску решения уравнения

$$\frac{\partial}{\partial e_1} \varepsilon(e_1, \alpha_{1,1}, \dots, \alpha_{n,1}) = -2 \sum_{j=1}^n (x_j - \alpha_{j,1} e_1) \alpha_{j,1} = -2 \left(\sum_{j=1}^n x_j \alpha_{j,1} - \sum_{j=1}^n \alpha_{j,1}^2 e_1 \right).$$

Отсюда получаем

$$e_1 = \frac{\sum_{j=1}^n x_j \alpha_{j,1}}{\sum_{j=1}^n \alpha_{j,1}^2},$$

учитывая условие нормирования единицей, то есть $\sum_{i=1}^n \alpha_{i,1}^2 = 1$, имеем

$$e_1 = \sum_{j=1}^n x_j \alpha_{j,1}.$$

Следующий шаг будем делать исходя из предположения, что в задаче (3.4) нам известна компонента e_1 и требуется найти экстремум по $\{\alpha_{i,1}\}_{i=1}^n$

$$\frac{\partial}{\partial \alpha_{\nu,1}} \varepsilon(e_1, \alpha_{1,1}, \dots, \alpha_{n,1}) = -2(x_\nu - \alpha_{\nu,1} e_1) e_1 = -2(\langle x_\nu, e_1 \rangle - \alpha_{\nu,1} \langle e_1, e_1 \rangle) = 0,$$

то есть

$$\alpha_{\nu,1} = \frac{\langle x_\nu, e_1 \rangle}{\langle e_1, e_1 \rangle},$$

где, как обычно, $\langle x, y \rangle$ - скалярное произведение векторов x и y .

Далее, считая найденные $\{\alpha_{i,1}\}_{i=1}^n$ известными, повторяем весь процесс, пока не произойдет стабилизация ошибки. Полученные e_1 будем считать первой главной компонентой \tilde{e}_1 . Тогда $\Delta x_j = x_j - \tilde{\alpha}_{j,1} \tilde{e}_1$ - ошибка восстановления данных одной первой главной компонентой.

Применяя этот алгоритм к ошибке восстановления Δx_j , находим вторую главную компоненту e_2 вместе с коэффициентами $\alpha_{j,2}$, и т.д.

Приведем алгоритмизацию этого алгоритма.

Вначале центрируем данные, вычитая из исходных данных среднее значение и в дальнейшем считаем, что данные в среднем равны нулю.

1. Положим номер итерации $\nu = 1$.

2. Выбираем стартовые значения $\{\alpha_{i,1}^v\}_{i=1}^n$, например, пусть все они между собой равны, то есть $\alpha_{i,1}^v = \frac{1}{\sqrt{n}}, i = 1, 2, \dots, n$.

3. Вычисляем $e_1^v = \sum_{j=1}^n x_j \alpha_{j,1}^v$.

4. Далее находим $\beta_i = \frac{\langle x_i, e_1^v \rangle}{\langle e_1^v, e_1^v \rangle}$, и, нормируя единицей, получаем

$$\alpha_{i,1}^{v+1} = \frac{\beta_i}{\sqrt{\sum_{j=1}^n \beta_j^2}}.$$

5. Полагаем $v = v + 1$.

6. Проводим проверку критерия остановки, в качестве этого может быть либо стабилизация коэффициентов $\{\alpha_{i,1}^v\}_{i=1}^n$, либо стабилизация главной компоненты e_1^v , либо проверка на заранее заданное фиксированное число итераций. Если условие окончания итерационного процесса не выполнено, то переходим к пункту 3.

Проиллюстрируем итерационный алгоритм поиска главных компонент на том же примере, который приведен выше.

Для уже отцентрированных данных (см. таблицу 3.2) приведем несколько итераций. Итак, пусть вначале $v = 1$ и $\alpha_{i,1}^1 = \frac{1}{\sqrt{2}}, i = 1, 2$. Вычисляя $e_{1,j}^1 = \alpha_{1,1}^1 x_j + \alpha_{2,1}^1 y_j$, получаем:

e_1^1	-4.1542	-2,740	-2,740	-0,619	0,088	2,917	2,917	4,33
---------	---------	--------	--------	--------	-------	-------	-------	------

Табл. 3.6. Первое приближение главной компоненты.

Далее вычислим $\beta_i = \frac{\langle x_i, e_1^1 \rangle}{\langle e_1^1, e_1^1 \rangle} = (0.7697, 0.64447)$ и после нормировки получаем

$$\alpha_{i,1}^2 = \frac{\beta_i}{\sqrt{\beta_1^2 + \beta_2^2}} = (0.7667, 0.64343).$$

Таким образом, после первой итерации приближенные значения исходных данных будут равны $\tilde{x}_i = \alpha_{1,1}^2 e_{1,i}^1 + \mu_1$, $\tilde{y}_i = \alpha_{2,1}^2 e_{1,i}^1 + \mu_2$ (результаты сравните с таблицей 3.5):

\tilde{x}	1,3148	2,399	2,399	4,0256	4,5678	6,736	6,736	7,82
\tilde{y}	1,702	2,612	2,612	3,977	4,4319	6,2517	6,2517	7,172

Табл. 3.7. Исходные данные, восстановленные по первому приближению главной компоненты.

После десяти итераций получаем $\tilde{x}_i = \alpha_{1,1}^{11} e_{1,i}^{10} + \mu_1$, $\tilde{y}_i = \alpha_{2,1}^{11} e_{1,i}^{10} + \mu_2$ (результаты сравните с таблицей 3.5):

\tilde{x}	1,2617	2,3449	2,445	4,02	4,6115	6,6778	6,7778	7,861
\tilde{y}	1,685	2,585	2,668	3,976	4,468	6,1841	6,2673	7,1671

Табл. 3.8. Исходные данные, восстановленные по десятой итерации приближения главной компоненты.

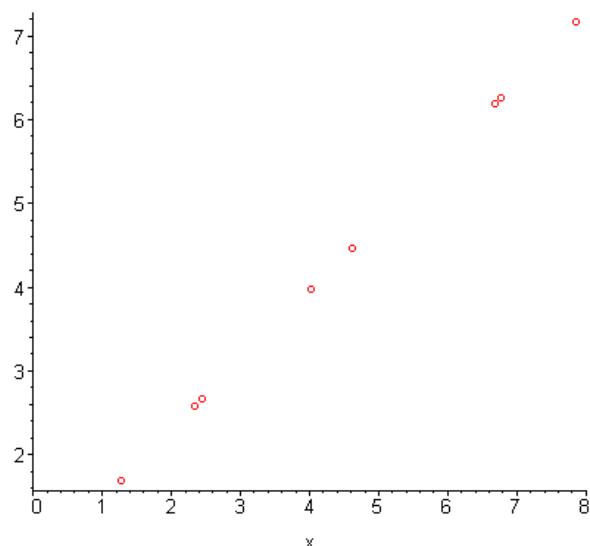


Рис. 3.7. Представление данных восстановленных по десятой итерации приближения главной компоненты.

Оптимальный переход от модели RGB к оптимальной трехкомпонентной модели

Приведем один пример использования метода главных компонент в такой области, как компьютерная графика. Тут важную роль играет перевод изображения из пространства равноправных цветовых характеристик в пространство неравноправных. Любое изображение визуализируется с использованием смешения равноправных цветовых компонент - красной, зеленой и синей составляющих - модель RGB. В качестве неравноправных цветовых компонент, как правило, используют, соответственно, тоже три компоненты – значение освещенности (люминесцентную составляющую), характеристику теплых тонов и характеристику холодных тонов. Использование неравноправных цветовых компонент используют для сжатия изображений и видео-потоков, применяя к каждой из неравноправных компонент свой метод сжатия (см., например, [28]).

Можно подходить к проблеме построения неравноправного цветового пространства с другой точки зрения, исходя из максимальной информативности каждой компоненты. Применим метод главных компонент для получения неравноправной трехкомпонентной модели оптимальной с точки зрения минимизации среднеквадратичной ошибки восстановления исходного изображения. То есть, первая из полученных цветовых компонент будет нести наибольшую информацию об изображении среди всех полученных цветовых компонент, а вторая будет содержать наибольшую информацию среди оставшихся.

Таким образом, в нашей терминологии, задача (3.1) примет вид

$$\left\| R - \sum_{i=1}^3 \alpha_{r,i} e_i \right\|_2^2 + \left\| G - \sum_{i=1}^3 \alpha_{g,i} e_i \right\|_2^2 + \left\| B - \sum_{i=1}^3 \alpha_{b,i} e_i \right\|_2^2 \rightarrow \min,$$

где минимум берется по всем $\alpha_{r,i}, \alpha_{g,i}, \alpha_{b,i}$ и e_i $i=1,2,3$.

В качестве данных рассмотрим тестовое изображение Lena.



Рис.3.8. Тестовое изображение Lena.

Применяя метод главных компонент, получаем

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 0.767785 & 0.45439 & 0.4517034 \\ -0.6164395 & 0.716085 & 0.3274513 \\ -0.174667 & -0.5298553 & 0.8299064 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix}.$$

Здесь в первом столбце матрицы стоят коэффициенты $\alpha_{r,1}, \alpha_{r,2}, \alpha_{r,3}$, во втором - $\alpha_{g,1}, \alpha_{g,2}, \alpha_{g,3}$ и в третьем - $\alpha_{b,1}, \alpha_{b,2}, \alpha_{b,3}$. Тогда восстановление тестового изображения по одной компоненте можно записать в виде

$$R_{i,j} = 0.767785 Y_{i,j},$$

$$G_{i,j} = 0.45439 Y_{i,j},$$

$$B_{i,j} = 0.4517034 Y_{i,j},$$

где $Y_{i,j}$ – значения первой главной компоненты e_1 , соответствующие пикселям с координатами (i,j) .



Рис. 3.9. Изображение Lena, восстановленное по одной первой главной компоненте.

4. Мягкие вычисления в обработке данных

Введение в мягкие вычисления

Термин «мягкие вычисления» ввел в 1994 году основоположник направления нечеткой логики – Лотфи Заде. Под этим термином подразумевалась совокупность эмпирических, нечетких, приближенных методов решения задач, не имеющих точных алгоритмов решения за полиномиальное время. В настоящее время, к мягким вычислениям принято относить следующие методы:

- Нейронные сети
- Эволюционные стратегии
- Нечеткая логика
- Многоагентные системы (интеллект стаи)
- Различные эвристические алгоритмы поиска решений
- Теория хаоса и др.

Многие алгоритмы, используемые в данных направлениях, относят к разряду интеллектуальных. Но термин интеллект (и искусственный интеллект) сейчас настолько размылись, что хотелось бы его остановиться на нем подробнее.

Под термином «интеллект» мы будем понимать свойство чего-либо решать интеллектуальные задачи. Интеллектуальной задачей назовем такую задачу, для решения которой еще не придумано чёткого алгоритма. Другими словами, интеллект – это свойство человека, компьютера или чего-то еще, позволяющее создавать новые алгоритмы.

Такое определение позволяет нам сразу же отсеять множество маркетинговых (в плохом смысле этого слова) определений интеллекта, привязанных к стиральным порошкам, лекарствам и т.п. Интересно также применить данное определение к человеческим профессиям.

- Программист. Продуктом деятельности программиста является программа – алгоритм в чистом виде. Не думаю, что найдется кто-то, кто сможет утверждать, что данная профессия неинтеллектуальна, хотя... есть и здесь подвод-

ные камни. К примеру, в программе есть кусок кода, решающий уже готовую проблему. Программист берет его, меняет несколько символов (констант) и получает дублированный код, адаптированный к новой предметной области. Я говорю о типичном грехе copy/paste. Не буду здесь рассуждать о недостатках такого подхода – взглянем на это с другой стороны. Новый алгоритм не был рожден. Значит, интеллектуальность такого результата близка к нулю.

- Мошенник. Есть типичные схемы, когда преступления совершаются по накатанной технологии. Здесь не нужно большого интеллекта. Такие мошенники обычно попадаются рано или поздно. Но есть и «звезды», которые каждый раз изобретают необычные ходы, не повторяются. Фактически, каждый раз пишется новый алгоритм. Интеллект налицо.
- Сыщик. Думаю, что профессия сыщика и мошенника составляет синергетическую пару. Чем более интеллектуален мошенник, тем больше интеллекта нужно сыщику, чтобы разоблачить преступника, изобретая ответные нестандартные методы поиска.
- Кассир (банк, магазин и т.п.). В этой профессии изобретение нового алгоритма – зло. Все нужно делать тщательно, аккуратно и одинаково. Ни в коем случае не хочу сказать, что кассиры – неумные люди. Для того чтобы научиться данной работе, интеллект необходим. Зачастую немалый. Но во время самой работы «блок» алгоритмизации необходимо отключать. Поэтому не могу назвать эту профессию интеллектуальной согласно принятому определению.
- Водитель. Сочетание заранее алгоритмизированной деятельности (соблюдение правил дорожного движения, стандартные приёмы управления автомобилем) и постоянный поиск алгоритмов решения возникающих проблем. Это распознавание дорожной ситуации при большом количестве участвующих объектов, движущихся в разных направлениях с разными скоростями, поиск (суб-)оптимального маршрута при движении в незнакомом месте, реагирование на изменившиеся условия движения в уже знакомых местах (ямы, ремонт дороги, новые дорожные знаки). Неудивительно, что эту деятельность еще не автоматизировали до конца, есть только отдельные компоненты, помогающие в решении отдельных задач, например, GPS-навигаторы.

Думаю, на этом перечисление интеллектуальных особенностей профессий можно завершить – читатель без труда может сам проанализировать интересующие его направления. Также, у нас появился дополнительный критерий оценки интеллектуальности деятельности – чем тяжелее это направление автоматизируется, тем интеллектуальнее оно (это временный критерий, пока еще человек является самым ин-

теллектуальным «устройством» на Земле). Кстати, это и подсказка тем, кто пытается автоматизировать сложную деятельность – можно не только повышать интеллект машины, но и снижать степень интеллектуальности задачи. Например, полностью автоматизировать выращивание и сбор урожая на произвольном поле на открытой местности достаточно сложно – слишком много неизвестных. Но если предварительно подготовить поле «по линейке», поместить все растения в теплицу, то сложность задачи уменьшится. Решить её станет проще. А ваш интеллект проявится в подготовке алгоритма такого упрощения.

Возвратимся к методам, называемыми «мягкими вычислениями». То, что они ассоциируются с интеллектуальными задачами, связано, скорее всего, с тем, что они обладают высокой универсальностью и могут быть применены до того, как будет разработан хороший специализированный метод решения задачи.

Другим важным общим свойством мягких вычислений является свойство адаптивности – подстройки под задачу. Это вносит в решение задачи этап обучения (также часто связываемый с интеллектуальной деятельностью) и уменьшает для исследователя сложность задачи. Теперь он может работать с задачей как с черным или серым ящиком, не вникая до конца в тонкости работы управляемой системы в надежде, что эти тонкости будут скомпенсированы «интеллектом» метода.

Именно поэтому владение перечисленными выше методами является большим плюсом для любого исследователя, программиста, ученого, инженера, аналитика.

Следствием универсальности методов, является также их хорошие комбинаторные способности. Часто можно встретить системы, где нейронная сеть обучается при помощи генетического алгоритма, нечеткие классификаторы являются основой индивидуумов в многоагентных системах и т.п.

Эволюционные вычисления

Краткая история

Эволюционные методы придуманы и используются очень давно. Уже больше миллиарда лет только на Земле. Разумеется, речь идет о естественной эволюции. Именно этот процесс, по мнению большинства ученых, дал нам *Homo Sapience*, используемый как эталон интеллектуальности. Разумеется, есть люди, которые оспаривают

вают эту точку зрения. Что ж, не будем с ними спорить. Даже, если правы они, алгоритмы, которые получены в результате моделирования этих «неправильных» процессов показывают свою эффективность во многих задачах. И только, исходя из этого, они уже имеют право на использование.

Первым ученым, предложившим стройную теорию происхождения новых видов, явился Чарльз Дарвин. Именно он в своей работе «Происхождение видов», вышедшей в 1859 году, показал, что основой эволюции являются следующие процессы:

- Изменчивость – новые особи популяции практически всегда немного (а иногда сильно) отличаются от своих родителей.
- Отбор – естественный или искусственный отбор отсеивают неудачные варианты изменений. Жить остаются только удачные (более приспособленные).
- Наследственность – изменения, произошедшие в одном из поколений, наследуются потомками.

Совокупность этих движущих сил и позволяет видам приспосабливаться к изменяющейся окружающей среде, совершенствоваться и выживать. Но во времена Дарвина ясен был только механизм отбора. То, как появлялись новые признаки и как они передавались потомкам, стало ясно гораздо позже. В 1944 году, О. Эйвери, К. Маклеод и М. Маккарти опубликовали результаты своих исследований. В них доказывалось, что за наследственные процессы в организмах отвечает «кислота дезоксирибозного типа». Что это за кислота, мир узнал еще позже – 27 апреля 1953 года в журнале «Nature» была опубликована знаменитая статья Уотсона и Крика, а мир впервые услышал о двухцепочечной спирали ДНК.

С тех пор прошло уже более полувека, но до сих пор нельзя сказать, что о механизме функционирования ДНК мы знаем все. Мы постоянно узнаем все новые и новые детали. Что разные участки ДНК мутируют с разной частотой. Что многие гены могут присутствовать в ДНК в разном количестве экземпляров (copy number variation) и от этого может зависеть продукция того или иного белка, чувствительность к различным заболеваниям [131].

Несмотря на продолжающийся процесс познания естественных процессов, они с самого начала привлекали исследователей, которые хотели повторить аналогичный процесс с использованием вычислительной техники. К сегодняшнему дню накопи-

лось огромное количество различных алгоритмов. Перечисление основных направлений, конечно же хочется начать с генетических алгоритмов и классификационных систем Голланда. Впервые они были опубликованы в начале 60-х годов, но основное свое распространение получили после выхода книги, ставшей классикой – «Адаптация в естественных и искусственных системах» [93]. На территории бывшего СССР также работали в данном направлении, несмотря на то, что сразу после войны по непонятным соображениям был введен лозунг «генетика и кибернетика – продажные девки империализма». А ведь эволюционные вычисления объединяли эти два направления. Тем не менее, Л. А. Расстригиным в 70-е годы в рамках теории стохастического поиска был предложен ряд алгоритмов, моделировавших различные стороны поведения живых организмов. Эти идеи получили дальнейшее развитие в посвященных эволюционному моделированию работах И. Л. Букатовой. Ю. И. Неймарком было предложено осуществлять поиск глобального экстремума на основе множества независимых автоматов, при этом моделировались процессы рождения, развития и смерти особей. Также большой вклад в развитие эволюционных вычислений внесли Уолш и Фогел.

Каждая из этих школ взяла из известных на то время принципов эволюции что-то свое. После этого оно было упрощено до такой степени, что процесс можно было провести (промоделировать) на компьютере.

Как и многие эмпирические алгоритмы, эволюционные алгоритмы не гарантируют нахождения хорошего результата. Также они бывают сложны в настройке. При неправильных параметрах может проявляться склонность к вырождению, плохая поисковая способность. В этом случае не стоит сразу забрасывать эволюционные алгоритмы – стоит попытаться помочь им своим, естественным интеллектом («поиграться» с настройками, например). Здесь можно вспомнить, что если какого-то вида осталось очень мало – меньше десятка особей, то в природе он обречен на вымирание из-за вырождения. Также, природой не было изобретено колесо. Но еще лучше вспоминать чудные по конструкции тела птиц и их крылья, обтекаемых мант, эхолокаторы дельфинов и летучих мышей, черную плесень, использующую в качестве энергии излучение разрушенного реактора на чернобыльской АЭС [69].

Генетический алгоритм

Генетический алгоритм (далее ГА) является одним из самых известных эволюционных алгоритмов. Он прост в принципах работы, но имеет большой потенциал для развития, что и попытаемся показать в данном разделе.

По своей сути, ГА является алгоритмом для поиска глобального оптимума многоэкстремальной функции. Для этого он использует, с определенной степенью приближения, модель размножения живых организмов. Для того, чтобы решить проблему, нам нужно представить её в виде так называемой фитнесс-функции от многих переменных (также называемой оценочной):

$$f(x_1, x_2, x_3, \dots, x_N)$$

Для решения задачи нам необходимо найти глобальный максимум или минимум (это не принципиально, поскольку поиск максимума легко заменяется поиском минимума этой же функции, взятой со знаком минус и наоборот). При этом на значения входных переменных обычно налагаются определенные ограничения, как минимум по диапазону их изменения.

Перед тем, как мы рассмотрим работу ГА, нам необходимо представить все входные переменные в виде хромосом. Под хромосомами в ГА подразумеваются цепочки символов, с которыми и производятся дальнейшие операции. Для кодирования параметров чаще всего применяют следующие два метода:

- двоичный формат;
- формат с плавающей запятой.

При использовании двоичного формата, под параметр выделяется N бит (для каждого параметра это N может быть различным). Поскольку для каждого из этих параметров имеются ограничения MIN и MAX, то взаимный переход между значениями параметров в формате с плавающей запятой и их бинарным представлением можно записать в следующем виде:

$$g = (r - \text{MIN}) / (\text{MAX} - \text{MIN}) \cdot (2^N - 1);$$

$$r = g \cdot (\text{MAX} - \text{MIN}) / (2^N - 1) + \text{MIN};$$

здесь g – бинарное представление параметра, помещенное в N бит; r – значение параметра в формате с плавающей запятой. Часто также используют последующее преоб-

разование полученного бинарного представления в код Грея – это позволяет уменьшить разрушительную силу мутаций (тут мы немного забегаем вперед).

Полученные бинарные представления каждого параметра выкладывают в цепочку (строку) бит, которая дальше называется хромосомой.

При работе с параметрами в формате с плавающей запятой, их значения также выкладываются в цепочку бит, но без указанного выше преобразования, прямо в том представлении, с которым работает процессор компьютера.

После того, как мы закодировали все необходимые параметры в виде хромосомы, можем приступать к основному циклу генетического алгоритма:

1. Генерация первоначальной случайной популяции.
2. Генерация следующего поколения.
3. Отбраковка худших решений во вновь сгенерированном поколении.
4. Если не достигнут критерий окончания, переходим на шаг 2.
5. Окончание работы. Экземпляр, у которого самое лучшее значение фитнес-функции, является искомым решением.

Ключевым здесь конечно же является этап 2. Его можно детализировать следующим образом:

1. Сортируем родительское поколение в соответствии со значением фитнес-функции для каждого экземпляра.
2. Пока не сгенерировано достаточное количество экземпляров новых поколений:
 - 2.1. Отбираем двух родителей.
 - 2.2. Объединяем их хромосомы (крессовер).
 - 2.3. Применяем другие генетические операторы.

Опишем немного подробнее каждый из шагов генерации.

2.1. Для отбора используются различные стратегии. Например, просто случайнym образом выбираем из N самых лучших экземпляров. Другим распространенным подходом является турнирный. Он заключается в том, что для каждого из родителей выбирается случайная пара (или больше) претендентов. Из них используется тот, у которого значение фитнес-функции лучше. Таким образом, более приспособленные особи чаще будут родителями, но менее приспособленные также имеют шанс пройти.

Для ускорения сходимости также часто используется стратегия элитизма – в следующее поколение решений проходят без изменений самые лучшие из имеющихся решений предыдущего поколения (элита). Но с этим подходом нужно быть крайне

осторожным – при недостаточном размере популяции, она очень быстро становится похожей на элиту и поиск новых решений практически прекращается.

2.2. и 2.3. – применение генетических операторов. Основой ГА являются два оператора: кроссовер, который объединяет решения родителей и мутация, которая обеспечивает поисковые способности. Кроме этих основных операторов могут также применяться дополнительные операторы, например, инверсия.

Оператор кроссовера работает с битовыми строками двух родительских хромосом. Наиболее простым вариантом является одноточечный кроссовер. В этом случае каждая из родительских хромосом перерезается в одной, случайно выбранной точке. Хромосома потомка формируется из «головы» хромосомы одного предка и «хвоста» второго:

Предок 1:	1001101110101 100110	→	1001101110101 <u>010101</u>
Предок 2:	0010110010110 010101		

Оператор кроссовера может быть и более сложным – двухточечным или даже совершенно другим (к этому мы еще вернемся). Главное, чтобы он объединял решения предков, и потомку не было необходимости находить удачные решения заново.

Оператор мутации – это просто случайное изменение хромосомы в одном или большем количестве бит:

10011011 <u>1</u> 0101100110	→	10011011 <u>0</u> 0101100110
------------------------------	---	------------------------------

Мутация – разрушительный оператор. В большинстве случаев, он нарушает решение или даже приводит особь к неработоспособному состоянию. Поэтому вероятность его применения не должна быть чрезмерно высокой. Но отсутствие мутаций сводит на нет способность ГА к поиску глобального оптимума во всем пространстве решений.

Приведенный в качестве примера дополнительных операторов, оператор инверсии заключается в циклической перестановке бит в хромосоме случайное количество раз:

1001101110101100110 → 0110100110111010110

Теперь у нас есть все составные части генетического алгоритма, и мы можем закрепить их на практике.

Простой пример реализации ГА

Попробуем найти глобальный экстремум одной из тестовых функций. Она известна под названием De Jong 2:

$$f(x, y) = \frac{100}{100 \cdot (x^2 - y)^2 + (x - 1)^2 + 1}$$

Данная функция является овражной с достаточно малым наклоном в районе максимума. Максимум функции равен 100 при значении $x=y=1$. Конечно же, сделаем вид, что нам это неизвестно, только известно, что максимум находится при значениях параметров (и “ x ” и “ y ”) где-то между -1.28 и +1.28.

Программа является консольной, что позволяет не концентрироваться на интерфейсной части, а сразу же переходить к алгоритмике. Из-за этого, и в связи с учебной направленностью примеров, основные настройки задаются в тексте программ в виде констант. На сленге программистов, параметры данной программы являются «захардкожеными». Однако минусом это будет в том случае, если пользователю программы не нужно иметь доступа к коду программы. Мы же надеемся, что такой доступ будет происходить достаточно активно, поэтому правку параметров в тексте программы вполне можно считать своеобразным интерфейсом для программиста. Т.е., в качестве интерфейса пользователя к изменению параметров, мы будем использовать IDE.

Определимся, как мы будем представлять геном. В первом примере применим бинарный способ кодирования генома, без использования кода Грэя. Цепочку бит будем хранить в типе `int`, причем, старшие 16 бит будут отвечать за параметр x , а младшие – за параметр y . Поскольку к каждому геному также привязано значение и фитнесс-функции, по которому придется производить отбор наиболее приспособленных экземпляров (сортировкой), то удобно геном и это значение объединить в одной записи. Используем для этого уже готовый класс из .NET Framework –

`KeyValuePair<int, double>`. При этом в поле Key у нас будет находиться геном, а в Value – $f(x, y)$. В новых версиях для этого удобно использовать шаблон `Tuple<>`, но данный код написан так, чтобы не требовать установки чего-то более нового, чем .NET Framework 3.5.

Определим поля и константы, используемые далее:

```
private static Random _rnd = new Random();
const int GenerationSize = 10000;
const int GenerationNumbers = 200;
const double MutationProbability = 0.2;
```

Основной цикл работы программы сосредоточен в методе Main:

```
public static void Main(string[] args)
{
    // Инициализируем нулевое поколение.
    List<KeyValuePair<int, double>> generation = GenerateRandom();
    SortGeneration(generation);

    // Основной цикл генерации
    for (int genNum = 1; genNum < GenerationNumbers; ++genNum)
    {
        generation = GenerateNewGeneration(generation, true);
        SortGeneration(generation);
        Console.WriteLine("Лучшая особь = " + Weight(generation[0].Key));
        Console.WriteLine("Generation " + genNum);

        Console.WriteLine("x = " + GetX(generation[0].Key));
        Console.WriteLine("y = " + GetY(generation[0].Key));
        Console.WriteLine("Genome = {0:X}", generation[0].Key);

        Console.Write("Press any key to continue . . . ");
        Console.ReadKey(true);
    }
}
```

Здесь мы видим вызов случайной инициализации стартового поколения и генерация последовательности дочерних поколений. Критерием остановки генерации поколений у нас является номер поколения. В конце цикла мы выводим параметры лучшего экземпляра в консольном выводе. Поскольку коллекция экземпляров у нас

отсортирована по убыванию приспособленности, то самым лучшим будет экземпляр с номером 0.

Пройдемся теперь по вызываемым методам. Генерация первого, случайного поколения:

```
private static List<KeyValuePair<int, double>> GenerateRandom()
{
    List<KeyValuePair<int, double>> result = new List<KeyValuePair<int,
        double>>();
    for (int i = 0; i < GenerationSize * 2; ++i)
    {
        int genome = _rnd.Next();
        result.Add(new KeyValuePair<int, double>(genome, Weight(genome)));
    }
    return result;
}
```

Здесь и в процедуре генерации нового поколения, мы генерируем в 2 раза больше особей, чем будет потом участвовать в турнире. Таким образом, мы отбраковываем самые неприспособленные особи («уродцев»), которые совершенно «незижнеспособны». Этой начальной отбраковкой занимается следующий метод:

```
private static void SortGeneration(List<KeyValuePair<int, double>> generation)
{
    generation.Sort((x, y) => y.Value.CompareTo(x.Value));
    if (generation.Count > GenerationSize)
        generation.RemoveRange(GenerationSize,
                               generation.Count - GenerationSize);
}
```

Здесь происходит сортировка коллекции с геномами (используются достаточно краткие в записи лямбда-выражения). После этого оставляется уже необходимое количество наиболее приспособленных особей.

Немного остановимся на процедуре «взвешивания» – вычисления фитнес-функции. Она выполняется методом `Weight`, который в качестве параметра принимает цепочку бит генома:

```
private static double GetY(int genome)
{
    int y = genome & 0xffff;
    return y * (1.28 + 1.28) / (0x10000 - 1.0) - 1.28;
```

```
}

private static double GetX(int genome)
{
    int x = (genome >> 16) & 0xffff;
    return x * (1.28 + 1.28) / (0x10000 - 1.0) - 1.28;
}

private static double Sqr(double x)
{
    return x * x;
}

private static double Weight(int genome)
{
    double x = GetX(genome);
    double y = GetY(genome);
    return 100 / (100 * Sqr(Sqr(x) - y) + Sqr(1 - x) + 1);
}
```

Немного остановимся на методах `GetX` и `GetY`. Они извлекают из генома (32 бита) часть, ответственную за хранение параметра x и y соответственно, после чего преобразуют её в формат с плавающей запятой. При этом используется формула преобразования представления из предыдущего раздела, где $\text{MIN}=-1.28$, $\text{MAX}=1.28$, $N=16$ (бит на параметр), $2^N=0x10000$.

И, наконец, процедура генерации нового поколения:

```
private static List<KeyValuePair<int, double>> GenerateNewGeneration(
    List<KeyValuePair<int, double>> parents, bool useElitism)
{
    List<KeyValuePair<int, double>> result = new List<KeyValuePair<int,
        double>>();
    // Реализация элитизма
    if (useElitism)
        result.Add(parents[0]);

    while (result.Count < GenerationSize * 2)
    {
        // Турнирный выбор предков
        int parent1a = _rnd.Next(GenerationSize);
```

```
int parent1b = _rnd.Next(GenerationSize);
int parent2a = _rnd.Next(GenerationSize);
int parent2b = _rnd.Next(GenerationSize);
int parent1 = Math.Min(parent1a, parent1b);
int parent2 = Math.Min(parent2a, parent2b);

// Генерация с кроссовером
int mask = (~0 << _rnd.Next(32));
int child = parents[parent1].Key & mask |
            parents[parent2].Key & ~mask;

// Мутация
if (_rnd.NextDouble() > MutationProbability)
    child ^= 1 << _rnd.Next(32);

result.Add(new KeyValuePair<int, double>(child, Weight(child)));
}

return result;
}
```

В данном методе сосредоточена демонстрация отбора с элитизмом, турнирный выбор предков, кроссовер (одноточечный) и мутация. Как и при генерации нулевого поколения, данный метод генерирует в два раза больше потомков, чем будет использовано при генерации нового поколения дальше. Этим излишком опять займется наш метод (уже рассмотренный выше) SortGeneration. При тех настройках, которые указаны в программе, будет использован вариант, называемый ГА с элитизмом. В нашем случае, он заключается в том, что один наиболее приспособленный индивидуум гарантированно попадает в новое поколение. Такой подход гарантирует неухудшение показателей самого лучшего решения из поколения в поколение, часто обеспечивает более высокую стартовую скорость поиска решения. В то же время, может мешать более полному исследованию пространства решений и ускорению вырождения популяции. Читатель может самостоятельно поэкспериментировать с разными настройками и понаблюдать, как они влияют на качество и скорость поиска решений.

Запустив программу с приведенными настройками, довольно часто можно получить вот такую картину:

...

Поколение 197

```
Лучшая особь = 99,999998156313
Поколение 198
Лучшая особь = 99,999998156313
Поколение 199
x = 0,999995727473869
y = 0,999995727473869
Геном = E3FFE3FF
Нажмите "Ввод" для выхода . . .
```

Указанные параметры x и y – это наиболее близкий к числу 1.0 узел из множества узлов, которые получаются при разбиении промежутка $[-1.28\dots+1.28]$ на 2^{16} отрезка. Этот результат можно наблюдать не всегда – мы ведь работаем с вероятностным (ну хорошо, псевдовероятностным) процессом, поскольку используем генератор случайных чисел (класс Random). А пространство решений, которые нужно исследовать достаточно велико.

Конечно, пример не самый сложный. Его вполне можно решить различными аналитическими и градиентными методами. Но... Если для вас не составляет труда привести параметры в цепочку бит и закодировать фитнесс-функцию, а также устроит субоптимальное значение, то вы вполне можете обойтись без анализа функции на экстремум или выбора того или иного градиентного метода. Если у вас этих знаний нет, то такой подход сэкономит затраты, связанные с вызовом специалиста или с самостоятельным более глубоким изучением предметной области. Конечно, не бесплатно – за счет вас как специалиста по программированию и крайне нерациональному использованию вычислительной техники. Что лучше, в каждом случае решается индивидуально. Но наличие в вашем арсенале такого интеллектуального помощника, как ГА, не будет мешать – это точно.

Ближе к реальности, или пространственный кроссовер

Задачи, похожие на решаемую в предыдущем разделе проблему поиска экстремума тестовой функции De Jong 2, любят приводить в учебных пособиях по ГА. Понятно почему – проблема обозрима, ответ известен, легко проверить. В то же время, такие примеры оставляют двойственное впечатление. Ведь их часто нетрудно решить и другими, тоже простыми методами – от случайного перебора (метод Монте-Карло)

до градиентных методов типа покоординатной оптимизации. Либо, наоборот, трудно любыми методами.

Мощь ГА раскрывается в примерах, в которых моделируемая система разделяется на подсистемы, решения в которых могут быть найдены параллельно. Часто для эффективности работы в конкретной задаче, приходится адаптировать классические генетические операторы, изобретать новые.

Другим важным моментом является то, что в реальных задачах зачастую наиболее трудоемкой частью процесса просчета является не генерация нового поколения, а оценка приспособленности каждой особи. Поэтому некоторое усложнение схемы генерации новых поколений не сильно увеличивает интегральную трудоемкость алгоритма, а вот увеличение «выхода» хороших потомков – очень даже уменьшает её.

Попробуем показать это все на примере решения задачи, приближенной к реальной. Итак, задача:

Воюем. У нас есть войска противника, расположенные неравномерно на определенной территории и оружие – 10 боеголовок ядерного оружия. Известно, что при взрыве боеголовки в радиусе поражения *KillingRadius*, не остается ничего живого. Упрощенно будем считать, что за пределами данного радиуса, боевые единицы противника остаются живыми. Нашей задачей будет найти такие координаты для каждой боеголовки, чтобы у противника после удара осталось как можно меньше войск.

Конечно, если у вас более пацифистские настроения, эту же задачу можно представить и иначе – есть средства на постройку 10 магазинов в определенном районе города. Люди готовы ходить в магазин, расположенный не далее, чем за *M* (аналог *KillingRadius*) метров от дома. Люди проживают неравномерно по выбранному району. Необходимо расположить магазины таким образом, чтобы покрыть сервисом наибольшее количество жителей.

Далее, в тексте программы названия идентификаторов выбраны исходя из первой формулировки задачи, но это не будет означать специализации только на первом варианте постановки проблемы.

В качестве входных данных для тестового приложения будут поступать изображения в формате PNG. Данный формат используется, поскольку он сжимает без потери информации, а для определения яркости точек это важно. Боевая единица про-

тивника кодируется черной точкой (яркости цветовых компонент в RGB – 0, 0, 0). Территории, накрытые взрывами, будем показывать «томатным» цветом. Такой способ подачи входных данных дает возможность наглядно увидеть полученный результат, а также задействовать при решении графическую карту компьютера, поскольку для оценки полученного результата можно его просто нарисовать: поверх исходного изображения нарисовать залитые окружности с радиусом поражения.

В этот раз программа будет представлять собой WinForms-приложение с единственным окном, поскольку довольно любопытно наблюдать за динамикой работы генетического алгоритма. А оценить её только по числам, бегущим в консоли, в данном случае не очень удобно. Различные же параметры приложения, как и в прошлом примере, можно изменить прямо в тексте программы, используя в качестве редактора IDE.

Прежде всего, решим, какой вариант кодирования пространственной информации в хромосомы мы будем использовать. Самый простой вариант – последовательно закодировать координаты в битовую строку так, как это было показано в предыдущем разделе. После этого, применить обычные операторы кроссовера и мутации. Но, практика показывает, что в таких случаях очень часто возникают различные проблемы – проблема конкурирующих решений, слабое качество синтезируемых решений. Более подробно данные проблемы описаны в работе [38]. Там же предложен и один из возможных вариантов решения этих проблем – кроссовер, основанный на пространственном положении точек решения. Далее, будем сокращенно его называть пространственным кроссовером.

Итак, будем хранить данные, относящиеся к одной точке в виде неразделяемого набора данных. В биологии такие признаки называются сцепленными из-за того, что передаются потомкам, как правило, вместе. Т.е., в нашем случае это будут координаты на плоскости по осям X и Y.

В программе одному «взрыву», соответствует следующий класс:

```
public class Explosion
{
    public float X { set; get; }
    public float Y { set; get; }

    public Explosion Clone()
```

```
{  
    Explosion copy = new Explosion()  
    {  
        X = this.X,  
        Y = this.Y,  
    };  
    return copy;  
}  
  
static Random rnd = new Random();  
  
public static Explosion GenerateRandom(TaskSpecification spec)  
{  
    return new Explosion()  
    {  
        X = (float)(rnd.NextDouble() * spec.Bound.Width  
            + spec.Bound.Left),  
        Y = (float)(rnd.NextDouble() * spec.Bound.Height  
            + spec.Bound.Top),  
    };  
}  
}
```

В этом классе хранятся координаты взрыва (X, Y), а также представлено несколько вспомогательных методов. Упомянутый в одном из методов класс TaskSpecification, хранит некоторые ограничения и начальные условия задачи:

```
public class TaskSpecification  
{  
    public RectangleF Bound { get; set; }  
    public int ExplosionsNumber { get; set; }  
    public float KillingRadius { get; set; }  
    public Bitmap OriginalField { get; set; }  
}
```

Здесь представлены (по порядку) координаты границ, за которые не следует выходить при расположении взрывов (свойство Bound), количество боеголовок (фактически, это длина генома), радиус поражения каждой боеголовки и оригинальное поле (карта) сражения, на котором отмечены войска противника.

Перед тем, как привести код класса, ответственного за хранение генетической информации особи и работы с ней, посмотрим, что собой представляет пространственный кроссовер.

Как уже было сказано выше, если к списку пространственных координат двух хороших, но разных родителей, применить обычный одно- или двухточечный кроссовер, мы, скорее всего, получим нежизнеспособного потомка. Основных причин тут две. Прежде всего, у разных особей, одна и та же точка может храниться в разных местах генома, и при обмене генетической информацией, мы можем поместить почти рядом геометрически две точки, находящиеся в разных местах генома родителей. При этом оголится какой-то другой участок. Поначалу, генетический алгоритм будет тратить всю свою мощь на то, чтобы отобрать удачный способ кодирования координат и только потом, если к этому моменту еще не произойдет вырождения популяции, начнет собственно поиск удачного решения. Это и есть проблема конкурирующих решений – когда одному и тому же решению могут соответствовать множество способов кодирования этого решения, что очень увеличивает разрушающее свойство кроссовера.

Второй проблемой является то, что если обращать внимание только на место, где расположена точка в геноме, то кроссовер будет случайным образом брать точки из одного родительского решения и из другого. Эти точки будут разбросаны по всему пространству поиска. Однако понятно, что часто точки, расположенные вблизи друг от друга, образуют своеобразные подкомплексы решений, которые поддерживают друг друга. И случайное изъятие или добавление точек в такой ансамбль точек, существенно нарушает качество решений.

Обе эти проблемы призван решить пространственный кроссовер. При своей работе он ориентируется на положение точек в пространстве решаемой задачи. На следующем эскизе показана схема объединения двух родительских геномов.

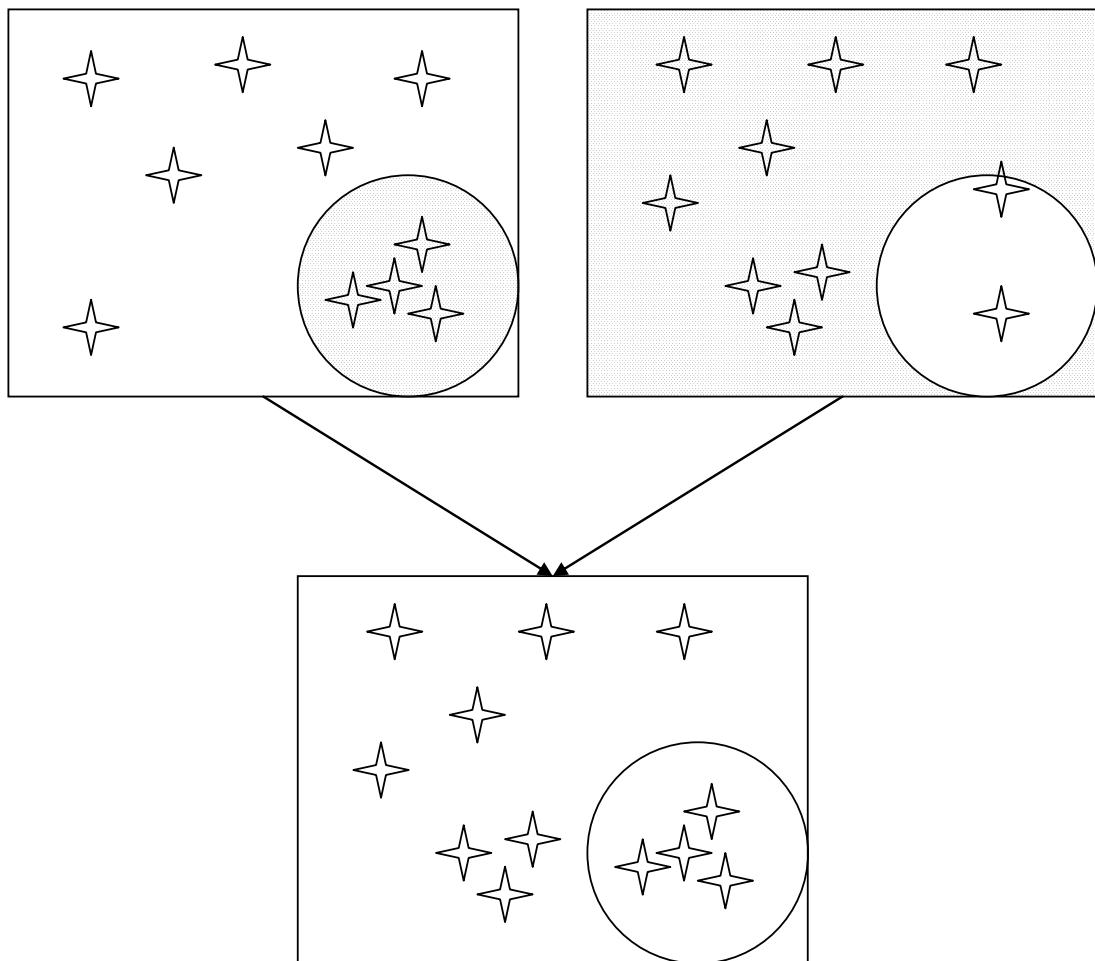


Рис. 4.1. Схема пространственного кроссовера

Пространственный кроссовер состоит из следующих шагов:

1. Выбираем в пространстве решаемой задачи произвольную (случайную) точку.
2. У обоих родителей вырезаем одинаковую (по координатам центра и радиусу) окружность (сферу или гиперсферу для пространств с размерностью более 2-х).
3. У одного родителя берем точки, которые лежат внутри окружности, у другого – снаружи.
4. Объединяем взятые точки в одно решение.
5. Корректируем количество точек в потомке – удаляем (случайным образом), если есть лишние, либо добавляем еще не использованные точки родителей.

Такой подход позволяет сохранить уже сложившиеся локальные ансамбли точек и с большей вероятностью (по сравнению с обычным кроссовером и способом кодирования).

Теперь, после краткого рассмотрения схемы пространственного кроссовера, мы можем рассмотреть выбранные места класса Individual (особь). При этом мы про-

пустим, не связанные непосредственно с ГА места прорисовки изображений, вспомогательные небольшие методы.

Прежде всего, генотип хранится в виде списка точек взрывов. Также есть поле, в которое заносится степень приспособленности особи. У нас это количество оставшихся после взрыва черных точек.

```
public List<Explosion> Explosions = new List<Explosion>();  
public double Fitness = Double.MaxValue;
```

Первое поколение генерируется случайным образом при помощи статического метода GenerateRandom:

```
public static Individual GenerateRandom(TaskSpecification spec)  
{  
    var result = new Individual();  
    for (int i = 0; i < spec.ExplosionsNumber; ++i)  
        result.Explosions.Add(Explosion.GenerateRandom(spec));  
    return result;  
}
```

Теперь рассмотрим реализацию генетических операторов. Мутация проста. Произвольная точка решения заменяется случайным образом. Для нашего случая этого хватает, но для большого количества точек (сто, тысячи взрывов) может понадобиться расширить реализацию, чтобы единомоментно могло изменяться большее количество точек.

```
public void Mutate(TaskSpecification spec)  
{  
    Explosions[rnd.Next(Explosions.Count)] = Explosion.GenerateRandom(spec);  
}
```

Более длинным является код пространственного кроссовера.

```
public void SpatialCrossoverWith(Individual parent2, TaskSpecification spec)  
{  
    float x0 = (float)(rnd.NextDouble() * spec.Bound.Width  
        + spec.Bound.Width);  
    float y0 = (float)(rnd.NextDouble() * spec.Bound.Height  
        + spec.Bound.Height);  
    float radius = (float)(rnd.NextDouble()  
        * Math.Max(spec.Bound.Width, spec.Bound.Height));
```

```
List<Explosion> newGenome = new List<Explosion>();
bool[] used1 = new bool[spec.ExplositionsNumber];
bool[] used2 = new bool[spec.ExplositionsNumber];
bool[] deleted = new bool[spec.ExplositionsNumber * 2];
// Добавляем точки из первого решения
for (int i = 0; i < spec.ExplositionsNumber; ++i)
{
    Explosion sp = this.Explosions[i];
    if (radius > Math.Abs(sp.X - x0) + Math.Abs(sp.Y - y0))
    {
        newGenome.Add(sp.Clone());
        used1[i] = true;
    }
}
// Добавляем точки из второго решения
for (int i = 0; i < spec.ExplositionsNumber; ++i)
{
    Explosion sp = parent2.Explosions[i];
    if (radius <= Math.Abs(sp.X - x0) + Math.Abs(sp.Y - y0))
    {
        newGenome.Add(sp.Clone());
        used2[i] = true;
    }
}

// Приводим количество точек нового решения к необходимому
if (newGenome.Count > spec.ExplositionsNumber)
{
    // Удаляем лишние точки
    int num = newGenome.Count - spec.ExplositionsNumber;
    for (int i = 0; i < num; ++i)
    {
        int delNum;
        do
        {
            delNum = rnd.Next(newGenome.Count);
        } while (deleted[delNum]);
        deleted[delNum] = true;
    }
    List<Explosion> tmpGenome = new
List<Explosion>(spec.ExplositionsNumber);
```

```
for (int i = 0; i < newGenome.Count; ++i)
    if (!deleted[i])
        tmpGenome.Add(newGenome[i]);
    newGenome = tmpGenome;
}

else if (newGenome.Count < spec.ExplosionsNumber)
{
    // Добавляем недостающие из первого решения
    while (newGenome.Count < spec.ExplosionsNumber)
    {
        int addNum;
        do
        {
            addNum = rnd.Next(spec.ExplosionsNumber);
        } while (used1[addNum]);
        used1[addNum] = true;
        newGenome.Add(this.Explosions[addNum]);
    }
}

// Копируем полученное решение в геном первого родителя
Explosions = newGenome;
}
```

Здесь мы видим все те шаги, которые были описаны для пространственного кроссовера. Но есть и особенность. При добавлении точек из предков, расстояние до точки мы измеряем в пространстве L_1 (метрика улиц). Это можно заметить, проанализировав фрагмент «`radius > Math.Abs(sp.X - x0) + Math.Abs(sp.Y - y0)`». Принципиально это ничего не меняет, просто из пространства родителей будут вырезаться квадраты (поворнутые на 45 градусов), а не окружности.

Также, при первоначальном тестировании примера, оказалось, что после первоначального прогресса и нахождения наиболее перспективных конфигураций решений, далее идет крайне медленный процесс адаптации полученных решений в основном за счет мутации. Но мутация в том виде, в котором она была приведена выше, является достаточно грубым средством – ведь вместо одного взрыва получаем случайным образом другой. И вероятность того, что он лишь слегка подкорректирует предыдущий, очень мала. Поэтому в пример был добавлен еще один оператор, на-

званный стекпингом (Stepping). Фактически это тоже мутация, но мягкая, которая лишь немногого смешает взрыв, к которому она применяется:

```
public void Stepping(TaskSpecification spec)
{
    int indexOfChange = rnd.Next(Explosions.Count);
    float shiftX = (float)((rnd.NextDouble() - 0.5) * spec.KillingRadius *
        0.25);
    float shiftY = (float)((rnd.NextDouble() - 0.5) * spec.KillingRadius *
        0.25);
    Explosion explosion = Explosions[indexOfChange];
    explosion.X = Limit(explosion.X + shiftX, spec.Bound.Left,
        spec.Bound.Right);
    explosion.Y = Limit(explosion.Y + shiftY, spec.Bound.Top,
        spec.Bound.Bottom);
}

private float Limit(float x, float min, float max)
{
    if (x < min)
        return min;
    if (x > max)
        return max;
    return x;
}
```

Один шаг ГА выглядит следующим образом (файл MainForm.cs, там же можно поменять вероятности применения генетических операторов и другие настройки алгоритма):

```
private List<Individual> NewGeneration(List<Individual> generation)
{
    var result = new List<Individual>(GenerationSize);

    if (UseElitism)
        result.Add(generation[0].Clone());

    while (result.Count < GenerationSize)
    {
        Individual parent1 = GetTournamentResult(generation);
        Individual newIndividual = parent1.Clone();
        if (rnd.NextDouble() < CrossoverProb)
```

```
{  
    Individual parent2 = GetTournamentResult(generation);  
    newIndividual.SpatialCrossoverWith(parent2, spec);  
}  
if (rnd.NextDouble() < MutationProb)  
    newIndividual.Mutate(spec);  
if (rnd.NextDouble() < SteppingProb)  
    newIndividual.Stepping(spec);  
  
result.Add(newIndividual);  
}  
  
return result;  
}
```

Посмотрим, что у нас получилось. Исходное изображение (можно найти в списке исходных файлов примера под именем 128x128_2.png) имеет размер 128x128 пикселов. Изначально на изображении 5910 черных пикселов, которые мы и должны максимально «накрыть взрывами».

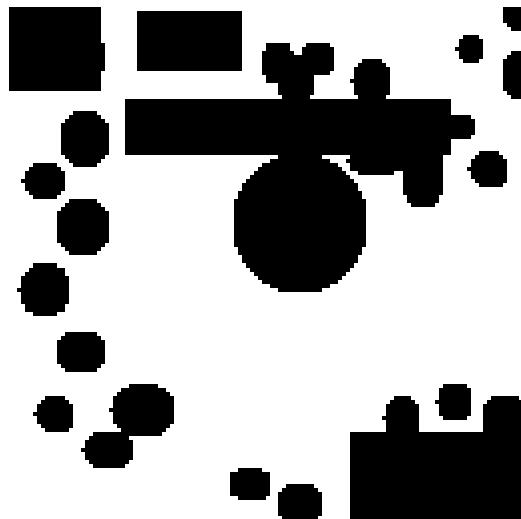


Рис. 4.2. Исходная область для поражения

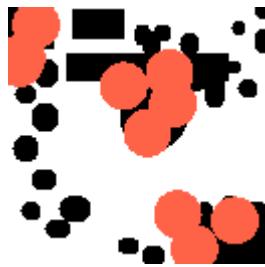
Радиус области поражения – 12 пикселов, количество взрывов – 10. Размер популяции – 1000 особей. Использовался элитизм, турнирный отбор из 500 лучших особей. Вероятность применения пространственного кроссовера – 0.4, мутации – 0.1,

степпинга – 0.3. Конечно же, при каждом запуске результаты будут несколько отличаться – ведь мы имеем дело с вероятностным процессом.

Работа с программой происходит следующим образом – запускаем приложение, загружаем нужную картинку (кнопка “Load sample”), запускаем эволюцию (нажимаем кнопку “Start”). Текущий результат эволюции отражается в главном окне программы, а промежуточные результаты записываются в том же каталоге, из которого была загружена картинка (файл с таким же именем и расширением TXT, а также png-картинки).



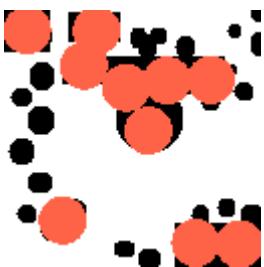
лучший представитель нулевого (случайного) поколения. Качество 3435.



лучший представитель 5-го поколения. Качество 3170.



лучший представитель 20-го поколения. Качество 2688.



лучший представитель 100-го поколения. Качество 2214.



лучший представитель 500-го поколения. Качество 2153.

Рис. 4.3. Результаты применения эволюционного алгоритма

В приведенном примере мы показали лишь одну из возможных адаптаций ГА к практике. Природа является только первоначальным образцом для подражания, но мы не обязаны копировать все детали. Достаточно соблюдать только базовые принципы и руководствоваться здравым смыслом. Перечислим некоторые принципы, позволяющие добиться более быстрой сходимости процесса эволюции, либо улучшить качество исследования предметной области:

Ускорение сходимости решения	Улучшают качество работы (поисковые способности) ГА
<ul style="list-style-type: none">✓ Увеличение прессинга естественного отбора✓ Уменьшение количества особей, допускаемых к размножению✓ Использование элитизма✓ Уменьшение общего количества генерируемого потомства✓ Выполнение алгоритма параллельно на нескольких компьютерах (процессорах)	<ul style="list-style-type: none">✓ Уменьшение прессинга естественного отбора✓ Увеличение объема генетического материала✓ Диплоидия (также увеличивает количество генетического материала)✓ Разбиение популяции на части. Это также является и способом легко распараллелить алгоритм

Генетическое программирование

Рассмотренный выше генетический алгоритм, работал с закодированной в генах информацией. Что это за информация, в общем-то, не важно. Это вполне могут быть и команды какой-то вычислительной машины – программа, а качество получаемой особи будет зависеть от того, насколько хорошо эта программа выполняет поставленную задачу. Основным минусом будет то, что длина программы будет фиксирована. Да и классические генетические операторы не способствуют высокой вероятности появления работоспособных потомков.

Поэтому для эволюционного составления программ были разработаны иные (по сравнению с ГА) методы хранения генома и других реализаций генетических операторов. Хронологически первой формой, была древообразная форма хранения генома (Рис. 4.4.), предложенная Н.Крамером [90] и Дж.Коза[107].

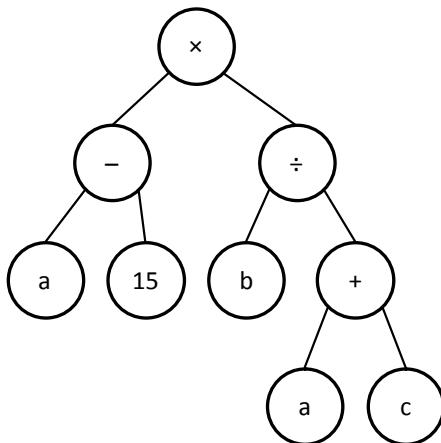


Рис. 4.4. Алгоритм вычисления значения функции $(a-15)(b/(a+c))$, представленный в виде дерева.

В качестве других форм, которые также часто используются, можно назвать линейную (Рис.4.5) и сетевую (или графовую) формы (Рис.4.6).

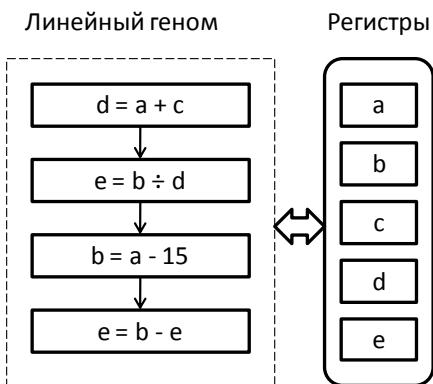


Рис. 4.5. Алгоритм значения функции $(a-15)(b/(a+c))$, представленный в линейном виде (код какого-то виртуального процессора).

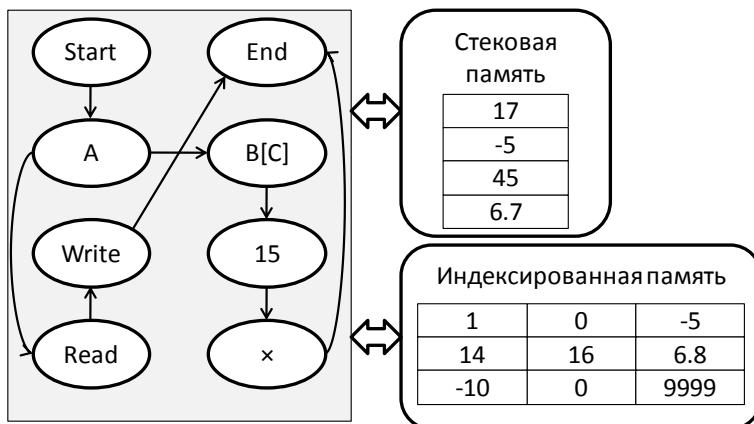


Рис. 4.6. Геном небольшой программы для виртуального процессора, представленный в сетевом (графовом) виде.

Операция мутации в случае генетического программирования (ГП), похожа на аналогичную операцию у ГА, но добавляется разнообразие изменений, которые она

может внести в геном. Для линейного генома, например, к случайному изменению произвольной команды, добавляются операции вставки случайной команды или удаления в случайному месте.

Несколько сложнее также выглядит и операция обмена генетическим материалом у предков – кроссовер, хотя основной принцип остается тем же – берется часть генома одного родителя и часть второго. Приведем примеры кроссовера для линейного (Рис.4.7) и древовидного представления генома (Рис.4.8.).

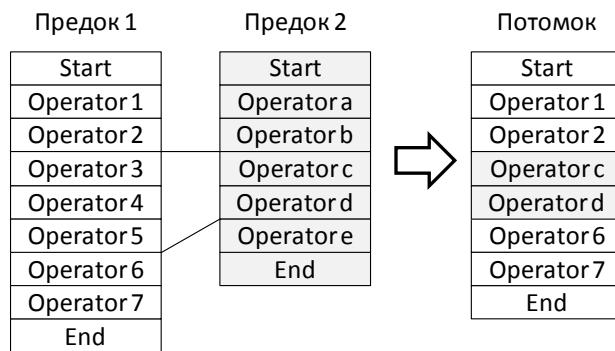


Рис. 4.7. Кроссовер для линейного представления генома.

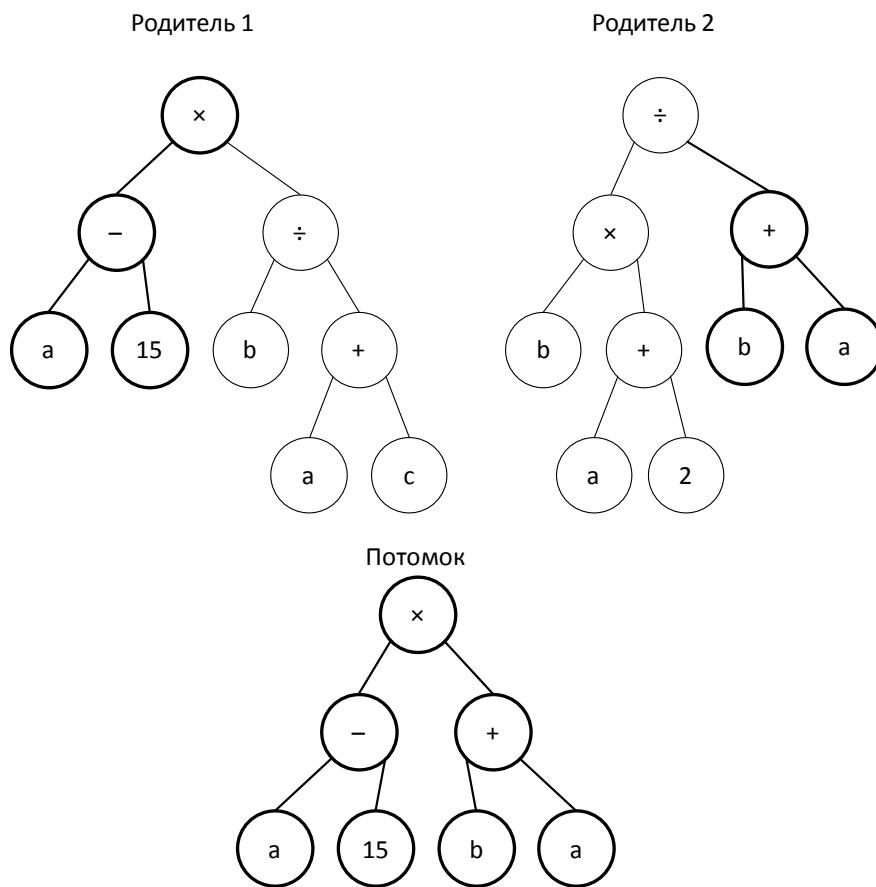


Рис. 4.8. Кроссовер при древовидном представлении генома.

Операция кроссовера чрезвычайно разрушительна для обычных программ и подавляющее количество потомков после такой операции становятся нежизнеспособными. Поэтому в систему команд часто вводят необычные операции, например, перехода по комплементарным меткам [37], автоопределляемые функции (ADF) [84] и много других усовершенствований.

Сами программы также «борются» с тем, что их разрушает. Например, путем увеличения количества инtronов – кусков кода, которые содержат ничего не делающие операции [84]:

- ✓ (NOT (NOT X))
- ✓ (AND ... (OR X X))
- ✓ (+ ... (- X X))
- ✓ (+ X 0)
- ✓ (* X 1)
- ✓ (* ... (DIV X X))
- ✓ (MOVE-LEFT MOVE-RIGHT)
- ✓ (IF (2=1) ... X)
- ✓ (A := A)

Такой способ защиты (как и дублирование самых важных участков генома) не является изобретением ГП. Похожие механизмы имеются и в геномах обычных клеток. Рассмотрение всех тонкостей генетического программирования может послужить темой отдельной книги (и не одной). Поэтому предложим заинтересованному читателю далее изучать данное направление по специализированной литературе. Хорошим введением является [84]. А ниже мы рассмотрим относительно простой пример поиска решения для генома переменной длины.

To be, or not to be...

Как говорилось в известном советском фильме – «друзья, а не замахнуться ли нам на Вильяма нашего Шекспира?» Попробуем и мы погреться в лучах славы бессмертных. Причем способом, который наверняка бы оценил Джонатан Свифт¹.

Рассмотрим следующую задачу. Нам необходимо отгадать некоторую фразу. Для того чтобы узнать, отгадали мы фразу или нет, мы показываем любую строку, а

¹ Имеется в виду произведение «Путешествия Гулливера», в котором описывается изобретатель, построивший машину, выдававшую случайные сочетания всех существующих слов. Осмысленные же предложения записывались, чтобы составить полную энциклопедию всех наук и искусств. Конечно же, Свифт в данном случае критиковал данное направление – но он тогда ничего не знал о генетическом программировании.

нам говорят число (фитнесс-функцию), которая представляет собой разность количества букв, которые стоят на своих местах, с теми, что занимают не свою позицию:

$$f(a, b) = \sum_{i=0}^{\max(\text{length}(a), \text{length}(b))-1} c(a, b, i)$$
$$c(a, b, i) = \begin{cases} 1, & \text{при } i < \text{length}(a) \cap i < \text{length}(b) \cap a[i] = b[i] \\ -1, & \text{иначе} \end{cases}$$

Здесь a – строка-образец. Она известна только вычислителю фитнесс-функции. b – строка, которую мы предъявили для оценки. $\text{length}()$ – функция, вычисляющая длину переданной строки. Индексация символов начинается с нулевой позиции.

Имея только такие скучные данные и приблизительный размер текста, попробуем отгадать фразу из бессмертного произведения. А именно, монолог Гамлета, начало которого вынесено в заголовок данной главы. Текст монолога взят со страницы http://ru.wikipedia.org/wiki/To_be_or_not, со всеми знаками препинания.

Приложение SimpleGP является консольным, все данные и настройки забиты в код, поэтому для того, чтобы их поменять, нужно использовать IDE и перекомпилировать программу. Начнем с точки входа в программу – файл Program.cs:

```
class Program
{
    static void Main(string[] args)
    {
        GPWorld world = new GPWorld();
        for (int g = 0;
            world.Generation[0].Genome != GPWorld.SecretPattern; ++g)
        {
            WriteWorldState(g, world);
            world.Next(true);
        }
        WriteWorldState("last", world);
        Console.WriteLine("The solution found!");
        Console.ReadLine();
    }

    private static void WriteWorldState(object generation, GPWorld world)
    {
        Console.WriteLine("Best individual {0}:", generation);
        Console.WriteLine("  Genome=" + world.Generation[0].Genome);
    }
}
```

```
        Console.WriteLine(" Length=" + world.Generation[0].Genome.Length);
        Console.WriteLine(" Fitness=" + world.Generation[0].Fitness);
        Console.WriteLine("-----");
    }
}
```

Метод Main включает в себя генерацию нулевого поколения и основной цикл – генерацию новых поколений. Цикл прерывается тогда, когда будет найдена точная фраза – она хранится в константе SecretPattern класса GPWorld. Но это место хранения выбрано только для упрощения примера – на самом деле, исходный текст может храниться где угодно, хоть на другом компьютере. Главное, чтобы была возможность выполнить с ним две операции – сравнение (как условие окончания цикла) и оценку похожести (вычисление fitness-функции).

Метод WriteWorldState выводит на консоль лучшего представителя текущей популяции.

В классе Program используются еще два класса – Individual (инкапсулирует особь) и GPWorld (инкапсулирует популяцию и операции над ней и отдельными особями). Рассмотрим их подробнее. Individual.cs:

```
public class Individual
{
    public string Genome = "";
    public double Fitness = 0;
}
```

Класс прост – в своих полях он хранит только геном (строку) и кэшированное значение фитнесс функции (в основном для сортировки особей в популяции). Основная логика генетических операций сосредоточена в файле GPWorld.cs:

```
public class GPWorld
{
    private static char[] _possibleLetters = "abcdefghijklmnopqrstuvwxyz'!
    ABCDEFGHIJKLMNOPQRSTUVWXYZ,;.-? \r\n".ToCharArray();
    private Random _rnd = new Random();
    private StringBuilder _sb = new StringBuilder();
```

Имеет смысл прокомментировать перечисленные выше поля. Поле_possibleLetters задает перечень возможных символов, которые нам могут встретиться в искомой фразе. В данном случае мы ограничились английским алфавитом,

знаками препинания, пробелом и переводом строки. Поля `_rnd` и `_sb` хранят объекты, которые не хочется создавать каждый раз – на это тратятся ресурсы процессора (выделить память, освободить память). В данном случае это небольшая оптимизация, рассчитанная на то, что программа выполняется в однопоточном режиме.

```
public const string SecretPattern =  
@"To be, or not to be, that is the question;
```

...

Здесь мы сократим многострочный строковый литерал – полностью его можно увидеть в исходных файлах, прилагаемых к книге, либо можно просто запустить программу и дождаться её окончания.

...

```
And lose the name of action.";  
  
public List<Individual> Generation;  
public const int GenerationSize = 100000;  
public const double CrossoverProbability = 0.4;  
public const double MutationProbability = 0.2;
```

Именно этот блок констант служит для настройки параметров примера.

```
public GPWorld()  
{  
    Generation = GenerateRandomGeneration();  
    SortGeneration(Generation);  
}
```

Конструктор генерирует новую случайную популяцию, после чего она сортируется по убыванию фитнес-функции.

```
private void SortGeneration(List<Individual> generation)  
{  
    generation.Sort((y, x) => x.Fitness.CompareTo(y.Fitness));  
}  
  
private List<Individual> GenerateRandomGeneration()  
{  
    List<Individual> result = new List<Individual>();  
    for (int i = 0; i < GenerationSize; ++i)  
    {  
        int length = _rnd.Next(1, SecretPattern.Length * 2);  
        _sb.Length = 0;
```

```
    for (int l = 0; l < length; ++l)
    {
        _sb.Append(RandomPossibleChar());
    }
    result.Add(new Individual()
    {
        Genome = _sb.ToString(),
    });
    result[i].Fitness = CalcFitness(result[i].Genome);
}
return result;
}
```

Вспомогательный метод RandomPossibleChar() выполняет то, что и сказано в его названии – возвращает случайно выбранный символ из числа возможных.

```
private char RandomPossibleChar()
{
    return _possibleLetters[_rnd.Next(_possibleLetters.Length)];
}
```

Метод CalcFitness() вычисляет фитнесс-функцию в соответствии со словесным описанием, заданным нами в начале раздела.

```
public double CalcFitness(string genome)
{
    double result = 0;
    int maxLength = Math.Max(SecretPattern.Length, genome.Length);
    int minLength = Math.Min(SecretPattern.Length, genome.Length);

    for (int i = 0; i < maxLength; ++i)
    {
        if (i >= minLength) {
            result -= 1;
        } else if (genome[i] != SecretPattern[i]) {
            result -= 1;
        } else {
            result += 1;
        }
    }
    return result;
}
```

Метод Next() выполняет генерацию нового поколения. Принимаемый параметр булевого типа указывает, используется ли стратегия элитизма. Если используется – один самый лучший экземпляр предыдущего поколения гарантированно переходит в следующее без изменений. Отбор производится турнирным методом из представителей наиболее приспособленной половины популяции. Таким образом, несколько усиливается давление отбора.

```
public void Next(bool useElitism)
{
    List<Individual> newGeneration = new List<Individual>();
    if (useElitism)
        newGeneration.Add(Generation[0]);

    while (newGeneration.Count < GenerationSize)
    {
        // Турнирный выбор предков из более приспособленной половины
        int parent1a = _rnd.Next(GenerationSize / 2);
        int parent1b = _rnd.Next(GenerationSize / 2);
        int parent2a = _rnd.Next(GenerationSize / 2);
        int parent2b = _rnd.Next(GenerationSize / 2);
        int parent1 = Math.Min(parent1a, parent1b);
        int parent2 = Math.Min(parent2a, parent2b);

        string newGenome = (_rnd.NextDouble() < CrossoverProbability) ?
            Crossover(Generation[parent1].Genome,
                      Generation[parent2].Genome) :
            Generation[parent1].Genome;

        if (_rnd.NextDouble() < MutationProbability)
            newGenome = Mutate(newGenome);

        Individual individual = new Individual()
        {
            Genome = newGenome,
        };
        individual.Fitness = CalcFitness(individual.Genome);
        newGeneration.Add(individual);
    }

    SortGeneration(newGeneration);
}
```

```
Generation = newGeneration;
```

```
}
```

Метод **Mutate()** с равной вероятностью выполняет операции вставки случайного символа, удаления или замены.

```
private string Mutate(string genome)
{
    // Случайная вставка
    if (_rnd.NextDouble() < 0.33)
    {
        int pos = _rnd.Next(genome.Length + 1);
        _sb.Length = 0;
        for (int i = 0; i < pos; ++i)
            _sb.Append(genome[i]);
        _sb.Append(RandomPossibleChar());
        for (int i = pos; i < genome.Length; ++i)
            _sb.Append(genome[i]);
        genome = _sb.ToString();
    }

    // Случайное удаление
    if (genome.Length > 0 && _rnd.NextDouble() < 0.33)
    {
        int pos = _rnd.Next(genome.Length);
        genome = genome.Remove(pos, 1);
    }

    // Случайное изменение
    if (genome.Length > 0 && _rnd.NextDouble() < 0.33)
    {
        int pos = _rnd.Next(genome.Length);
        _sb.Length = 0;
        for (int i = 0; i < genome.Length; ++i)
            if (i != pos)
                _sb.Append(genome[i]);
            else
                _sb.Append(RandomPossibleChar());
        genome = _sb.ToString();
    }

    return genome;
}
```

}

Операция кроссовера похожа на одноточечный кроссовер у обычного ГА, но приспособлена для обработки геномов переменной и неравной длины. Результирующий геном имеет длину, равной genome1. Если второй геном оказывается короче, то к строке добавляются случайные символы из множества допустимых. Если длиннее – то используется только часть символов в пределах длины первого генома.

```
private string Crossover(string genome1, string genome2)
{
    int pos = _rnd.Next(genome1.Length);
    _sb.Length = 0;
    for (int i = 0; i < genome1.Length; ++i)
        if (i > pos)
            _sb.Append(genome1[i]);
        else
            if (i < genome2.Length)
                _sb.Append(genome2[i]);
            else
                _sb.Append(RandomPossibleChar());
    return _sb.ToString();
}
```

В данной реализации кроссовера отсутствуют операции, которые приводят к сдвигу фрагментов строк к началу или к концу. Это связано с тем, что оценочная функция оценивает только символы, точно попавшие в свою позицию. Поэтому любые сдвиги приведут практически в каждом случае к тому, что геном будет «испорчен». Подобные операции, можно реализовать самостоятельно вместе с модификацией фитнесс функции. В этом случае фитнесс функция должна добавлять очки геному, если в нем имеются общие с образцом подстроки, пусть даже они стоят не на своих местах.

Итак, у нас все готово для запуска. Запускаем программу и наблюдаем за работой алгоритма. Здесь приведем фрагменты работы:

Поколение 0:

```
Genome=EA-B;-gOcLaRGhjaQonoAFaHWz
S?Q1BBQs;UaoUIpGBUJYLoUeU
pd--.eEh?g.XqfYWpoPa Lm
```

```
...
;u,sW.rAFbkYs
Length=1409
Fitness=-1350
```

Поколение 500:

```
Genome=TE be, or notGto b?, tpan .' Ghe eBeJtiln;
WhetRer Vpis irblgrBgT tfekqi dxto iuWdIrw
The SliJ?sLaIn Ao
...
ADd losG theonCme of a-tikn.
Length=1442
Fitness=182
```

Поколение 1000:

```
Genome=To be, or not to be, that is the qBestion;
Whether 'tis nobler in the mind to sufder
The Slings ann Arrmws of outrageous
...
ADd lose the name of action.
Length=1442
Fitness=1066
```

Поколение 1426:

```
Genome=To be, or not to be, that is the question;
Whether 'tis nobler in the mind to suffer
The Slings and Arrows of outrageous Fortune
...
And lose the name of action.
Length=1442
Fitness=1442
```

Интеллект стаи

Многие методики, относящиеся к мягким вычислениям, эксплуатируют идею того, что множество относительно простых объектов, работающих по вполне понятным правилам, объединяясь, демонстрируют поведение, намного превышающее по

интеллектуальности поведение отдельного индивидуума. Сюда можно отнести и нейронные сети и эволюционные алгоритмы. Про нейронные сети, в данном пособии мы подробно рассказывать не будем – о них нужно говорить либо много (что выливается в отдельный курс), либо ничего. К счастью, теория нейронных сетей проработана и описана достаточно хорошо для читателя любого уровня, поэтому при желании несложно найти именно то учебное пособие, которое позволит вникнуть в данную тематику более глубоко. С эволюционными алгоритмами мы познакомились в предыдущем разделе. Но, видимо, наиболее ярко и явно идея суммирования индивидуальных интеллектов, воплощается в таком направлении исследований, как «Интеллект стаи» (swarm intelligence).

Не хотелось бы давать какое-то скучное определение направления, поэтому давайте, познакомимся с ним, используя фрагмент фантастического произведения Станислава Лема – «Непобедимый» [26].

Небольшая преамбула. Большой космический корабль землян «Непобедимый» садится на планету, которая кажется безжизненной. Обнаружены только очень простые «растения» и «насекомые» на основе полупроводников и металлов. Но постепенно начинают происходить странные вещи – люди теряют память, погибают, технике кто-то наносит сильнейшие удары. Начинается более подробное изучение «простейших»:

«"Пленники" занимали во время совещания почетное место в закрытом стеклянном сосуде, стоявшем посреди стола. Их осталось всего десятка полтора, остальные были уничтожены в процессе изучения. Все эти создания обладали тройственной симметрией и напоминали формой букву Y, с тремя остроконечными плечами, соединяющимися в центральном утолщении...

Каждый кристаллик соединялся с тремя; кроме того, он мог соединяться концом плеча с центральной частью любого другого, что давало возможность образования многослойных комплексов. Соединение не обязательно требовало соприкосновения, кристалликам достаточно было сблизиться, чтобы возникшее магнитное поле удерживало все образование в равновесии...

Кроме цепи, заведующей такими движениями, каждый черный кристаллик содержал в себе еще одну схему соединений, вернее ее фрагмент, так как она, казалось,

составляла часть какой-то большой структуры. Это высшее целое, вероятно возникающее только при объединении огромного количества элементов, и было истинным мотором, приводящим тучу в действие. Здесь, однако, сведения ученых обрывались. Они не ориентировались в возможностях роста этих сверхсистем, и уж совсем темным оставался вопрос об их "интеллекте". Кронотос допускал, что объединяется тем больше элементарных единиц, чем более трудную проблему им нужно решить. Это звучало довольно убедительно, но ни кибернетики, ни специалисты по теории информации не знали ничего соответствующего такой конструкции, то есть "произвольно разрастающемуся мозгу", который свои размеры примеряет к величине намерений.

Часть принесенных Роханом "насекомых" была испорчена. Остальные демонстрировали типовые реакции. Единичный кристаллик мог подпрыгивать, подниматься и висеть в воздухе почти неподвижно, опускаться, приближаться к источнику импульсов либо удаляться от него. При этом он не представлял абсолютно никакой опасности, не выделял, даже при угрозе уничтожения. ... Зато объединившись даже в сравнительно небольшую систему, "насекомые" начинали, при воздействии на них магнитным полем, создавать собственное поле, которое уничтожало внешнее, при нагревании стремились избавиться от излишка тепла инфракрасным излучением - а ведь ученые располагали лишь маленькой горсточкой кристалликов».

Чем закончилось противостояние людей и этой интеллектуальной стаи, описывать не будем – если вас заинтересовало само произведение, прочтите его. Оно того стоит. Нас же оно заинтересовало как описание практически идеальной системы «интеллекта стаи».

Почему это направление так привлекает исследователей? Наверное, потому, что в окружающем мире все мы видим примеры таких «интеллектов»:

- Животные (человек как высшее проявление) – уже понятно, что за все очень сложное поведение отвечает система относительно простых (но все еще не до конца изученных) элементов, нейронов.
- Группы ученых чаще всего могут решать более сложные в интеллектуальном отношении задачи, чем одиночки. Хотя, очень часто здесь происходит не параллельное объединение, а последовательное – научные школы, учителя-ученики.

- Производство компьютеров (и любой сложной современной техники) – риск-нем утверждать, что ни один из ныне живущих людей не знает полностью цикл производства компьютера во всех деталях. Начиная с добычи нефти для производства пластика и металла для производства проводов, заканчивая вопросами дизайна рекламных буклетов для магазинов. Тем не менее, множество людей, объединяясь, успешно производят и продают всю эту технику.
- Муравьи и пчелы – рой пчел или муравьиная стая для стороннего наблюдателя производят впечатление чего-то гораздо более интеллектуального, чем отдельные особи. Например, когда наблюдается процесс поиска пищи и оптимизация пути, по которому муравьи приносят найденное в муравейник.

Именно последний пример моделируется в «Муравьиных алгоритмах», на примере которых мы хотим раскрыть направление Интеллекта Стai.

Оригинальный алгоритм (Ant Colony Optimization) возник в процессе наблюдения за тем, как живые муравьи вида Argentine Ant обследуют территорию вокруг муравейника, находят пищу и несут её в муравейник, постоянно оптимизируя (сокращая) путь, который проходит каждый из муравьев. Эти исследования проводились в 1989-м году Госсом и в 1990-м Денеборгом. Первая математическая формализация алгоритма предложена в 1992-м году Марко Дориго [79].

Живые муравьи во время поисков пищи ходят вокруг муравейника случайным образом по тропам, которые не являются физическими дорожками, «протоптанными» поколениями насекомых. Основная ориентация у муравьев происходит за счет феромонов, к которым они очень чувствительны и которыми они помечают все вокруг. Более того, каждый муравейник имеет свой индивидуальный запах и муравей того же вида но из другого муравейника, будет воспринят как враг. Существуют слепые муравьи, которые в пространстве ориентируются только за счет запаха и осязания.

Найдя пищу, муравей-разведчик возвращается домой, используя одну из троп. При этом весь его путь помечается особым феромоном. По возвращении в муравейник, этот муравей «зовет» (назовем это действие таким образом, чтобы не усложнять описание) за собой других, рабочих муравьев и начинается процесс переноса пищи в муравейник.

Рабочие муравьи идут, ориентируясь на запах, оставленный муравьем-разведчиком. При этом, они усиливают запах, оставляя свои следы на дорожке в том случае, если находят пищу. Таким образом, тропинки становятся все более и более

заметными. Но они не всегда идут одним и тем же путем – иногда муравьи сбиваются с пути и тогда случайным образом ищут место, помеченное феромоном. Иногда это приводит к нахождению более короткого пути. Также, в это время происходит процесс испарения феромонов. Если бы его не было, то первоначальный путь всегда имел бы самый сильный запах и процесс поиска более короткого пути не происходил бы.

Представим этот процесс графически.

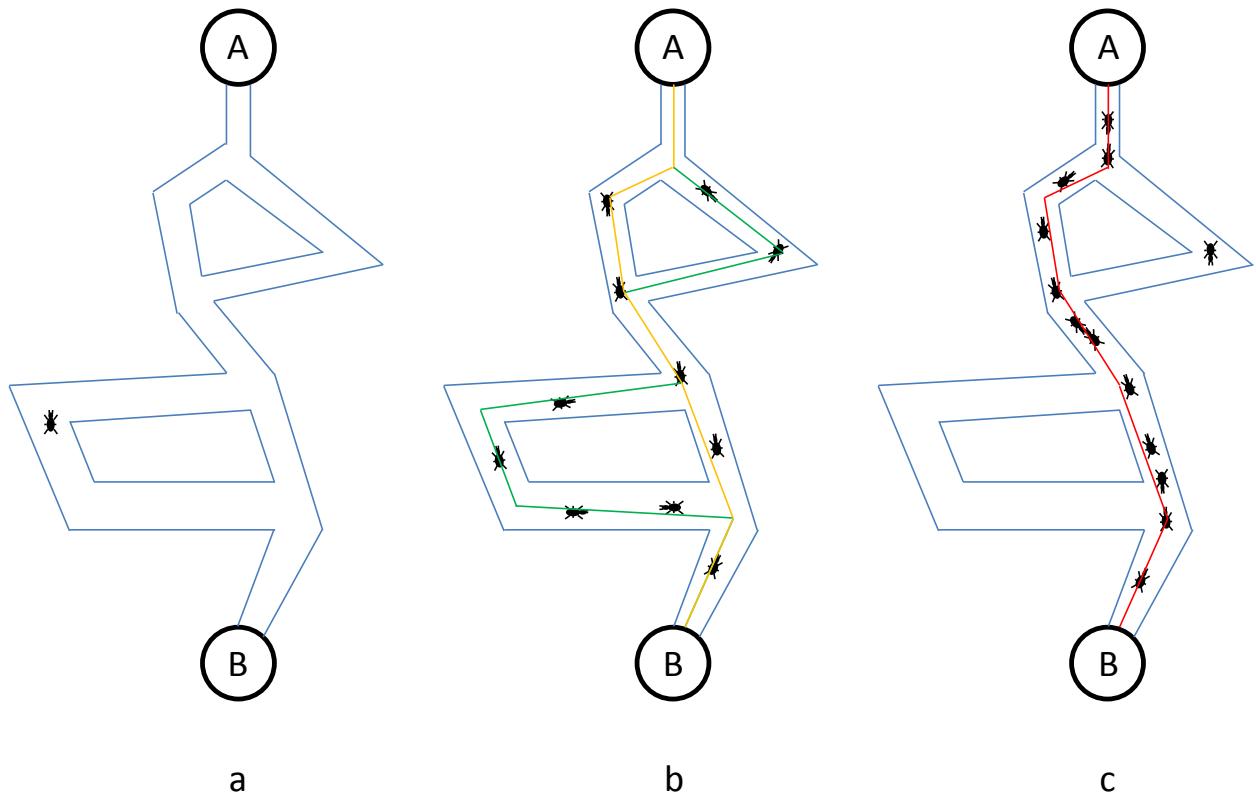


Рис. 4.9. Процесс оптимизации пути при переносе найденной пищи (A) в муравейник (B).

На иллюстрации а), муравей разведчик находит еду, после чего произвольным путем возвращается домой.

б) – рабочие муравьи переносят пищу в муравейник, прокладывая феромонные тропы. По более короткому пути муравьи успевают пройти в большем количестве, поэтому постепенно этот путь становится все более «пахнущим».

с) – большая часть муравьев движется по самому короткому пути и только отдельные особи используют другие пути.

Опишем простейший муравьиный алгоритм более формально. Лабиринт задан графом (вершины и ребра). Считаем, что муравьи ищут оптимальный путь (самый короткий) между двумя вершинами (муравейник и еда).

Пока (не выполнены условия выхода)

1. Создаем муравьев.
2. Муравьи ищут решения.
3. Обновление уровня феромона.

Рассмотрим каждый из шагов цикла более подробно.

1. Начальные точки, куда помещаются муравьи, зависят от ограничений задачи. В простейших случаях мы можем их всех поместить в одну точку, либо случайно распределить по площади лабиринта. На этом же этапе каждое ребро лабиринта помечается небольшим положительным числом, характеризующим запах феромона. Это нужно для того, чтобы на следующем шаге у нас не было нулевых вероятностей.
2. Определяем вероятность перехода из вершины i в вершину j по следующей формуле:

$$P_{ij}(t) = \frac{\tau_{ij}(t)^\alpha \left(\frac{1}{d_{ij}}\right)^\beta}{\sum_{j \in \text{connected nodes}} \tau_{ij}(t)^\alpha \left(\frac{1}{d_{ij}}\right)^\beta}$$

Здесь $\tau_{ij}(t)$ – уровень феромона, d_{ij} – эвристическое расстояние, α/β – константы.

Если $\alpha=0$, то наиболее вероятен выбор ближайшего соседа и алгоритм становится «жадным».

В случае $\beta=0$, наиболее вероятен выбор только на основе уровня феромона, что приводит к застреванию на уже «протоптанных» путях.

Как правило, используется некоторое компромиссное значение этих величин, подбираемое экспериментально для каждой задачи.

3. Обновление уровня феромона производится следующим образом:

$$\tau_{ij}(t+1) = (1 - \rho)\tau_{ij}(t) + \sum_{k \in \text{used edge}(ij)} \frac{Q}{L_k(t)}$$

Здесь ρ – параметр, задающий интенсивность испарения, $L_k(t)$ – цена текущего решения для k -го муравья, а Q характеризует порядок цены оптимального решения.

Таким образом, выражение $\frac{Q}{L_k(t)}$ определяет количество феромона, которым муравей k пометил ребро (ij) .

Задача коммивояжера и муравьиный алгоритм

Чтобы проверить способность муравьиных алгоритмов решать серьезные задачи, часто используют известную задачу коммивояжера. Не будем подробно описывать её – она достаточно известна. Есть набор городов, которые должен посетить коммивояжер, побывав в каждом по одному разу и вернувшись в тот город, из которого путешествие было начато. Задачей здесь является оптимизация пути таким образом, чтобы пройденное расстояние было наименьшим. Это одна из самых простых постановок задачи и именно её мы попробуем решить.

Задачи, связанные с графами, неудобно показывать в консоли, поэтому для этой задачи мы выберем обычное приложение WinForms. Читатель может самостоятельно изменить, откомпилировать, запустить пример, взяв проект из приложения к данному курсу. Но вся логика сосредоточена в файле AntOptimizer.cs, его и рассмотрим подробнее.

Прежде всего, в классе AntOptimizer определены два класса – обертки, не несущие особого функционала:

```
public class AntOptimizer
{
    class Town
    {
        public double X { set; get; }
        public double Y { set; get; }
    }

    class RouletteSector
    {
        public double Start = 0;
        public int Town = -1;
    }
}
```

Класс Town хранит координаты «городов», а RouletteSector используется при имитации рулетки с секторами разного размера.

Далее идут определения приватных полей. `_towns` – список городов. `_distances` – таблица расстояний между городами, чтобы не нужно было их вычислять каждый раз. Но в принципе, при недостатке памяти, вполне может быть заменена функцией, вычисляющей эти значения на лету.

```
Town[] _towns = null;
Random _rnd = new Random();
double[,] _distances = null;
double[,] _scents = null;
double GetScent(int town1, int town2)
{
    return _scents[Math.Min(town1, town2), Math.Max(town1, town2)];
}
void AddScent(int town1, int town2, double scent)
{
    _scents[Math.Min(town1, town2), Math.Max(town1, town2)] += scent;
}
```

Таблица, хранящая ферромонные пометки (запах) – `_scents`, используется не полностью, а только нижняя треугольная часть (если первый индекс принять за x, а второй за y координаты). Поэтому введены вспомогательные методы `GetScent` и `AddScent`, которые приводят индексы в нужный порядок.

Далее следуют описания публичных полей, которые используются для вывода числовых результатов в GUI. Функции полей соответствуют их названиям.

```
public Ant BestWay = null;
public double BestWayLength = Double.MaxValue;
public int StepCount = 0;
```

Конструктор. Имеет 3 основных функциональных блока (не считая проверки входящих параметров).

Прежде всего, инициализируется случайными значениями список координат городов. Координаты мы берем из промежутка 0...1. Выбор диапазона для алгоритма не важен – просто нам так удобнее.

Далее, инициализируются таблицы расстояния между городами и таблица запахов. Таблицу запахов мы инициализируем отличными от нуля значениями, чтобы начальная вероятность выбора той или иной дорожки была положительной.

```
public AntOptimizer(int townNumber)
```

```
{  
    // Проверяем параметры.  
    if (townNumber < 3)  
        throw new ArgumentOutOfRangeException("townNumber", townNumber,  
            "'townNumber' should be more than 2");  
    // Инициализируем города случайными координатами.  
    _towns = new Town[townNumber];  
    for (int t = 0; t < townNumber; ++t)  
    {  
        _towns[t] = new Town()  
        {  
            X = _rnd.NextDouble(),  
            Y = _rnd.NextDouble()  
        };  
    }  
    // Заполняем таблицу расстояний между городами.  
    _distances = new double[townNumber, townNumber];  
    for (int t1 = 0; t1 < townNumber; ++t1)  
    {  
        Town town1 = _towns[t1];  
        for (int t2 = 0; t2 < townNumber; ++t2)  
        {  
            Town town2 = _towns[t2];  
            _distances[t1, t2] = Math.Sqrt(Sqr(town1.X - town2.X)  
                + Sqr(town1.Y - town2.Y));  
        }  
    }  
    // Инициализируем таблицу запахов.  
    _scents = new double[townNumber, townNumber];  
    for (int i = 0; i < townNumber; ++i)  
        for (int j = i + 1; j < townNumber; ++j)  
            _scents[i, j] = 1.0;  
}
```

Метод Step несет в себе логику одного шага оптимизации при помощи муравьиного алгоритма. Из каждого города пускается по муравью, которые обходят все города, ориентируясь по ферромонным следам и модифицируя их.

```
public void Step(double alpha, double beta, double q)  
{  
    StepCount++;  
    int townNumber = _towns.Length;
```

```
// По очереди пускаем муравьев. Проверяем, какой маршрут наилучший.
for (int a = 0; a < townNumber; ++a)
{
    Ant ant = new Ant(this, a);
    double wayLength = ant.Move(alpha, beta, q);
    if (wayLength < BestWayLength)
    {
        BestWay = ant;
        BestWayLength = wayLength;
    }
}
// Испаряем ферромоны
for (int i = 0; i < townNumber; ++i)
    for (int j = i + 1; j < townNumber; ++j)
        _scents[i, j] = _scents[i, j] * 0.9;
}
```

Вспомогательные методы Sqr и DrawMap мы опустим – в них нет алгоритмически сложных моментов.

Последний оставшийся класс – Ant. Он инкапсулирует «поведение» отдельного муравья. Поле _pathHash, которое можно увидеть в списке приватных полей служит для быстрого определения, присутствует ли определенный город в списке проходимых. Сам список, упорядоченный по порядку посещения, находится в поле Path.

```
public class Ant
{
    private AntOptimizer _owner;
    private int _townNumber = -1;
    private int _startTown;
    private Dictionary<int, int> _pathHash = new Dictionary<int, int>();

    public List<int> Path = new List<int>();
```

Конструктор – принимает в качестве входных параметров экземпляр класса-владельца (из которого берутся координаты городов, расстояния между ними) и номер города из которого необходимо стартовать процесс оптимизации.

```
public Ant(AntOptimizer owner, int startTown)
{
    _owner = owner;
    _startTown = startTown;
```

```
    _townNumber = _owner._towns.Length;
}
```

Метод Move производит поиск пути, ориентируясь на запах и расстояния. В процессе движения производится маркировка использованных «дорожек» свежей порцией ферромонов. В качестве параметров принимает параметры алгоритма альфа, бета и q.

```
public double Move(double alpha, double beta, double q)
{
    double pathLength = 0;

    Path.Add(_startTown);
    _pathHash.Add(_startTown, _startTown);
    int currentTown = _startTown;

    for (int i = 1; i < _townNumber; ++i)
    {
        currentTown = FindNextTown(currentTown, alpha, beta);
        Path.Add(currentTown);
        _pathHash.Add(currentTown, currentTown);
    }

    Path.Add(_startTown);
    for (int i = 0; i < _townNumber; ++i)
        pathLength += _owner._distances[Path[i], Path[i + 1]];

    for (int i = 0; i < _townNumber; ++i)
        _owner.AddScent(Path[i], Path[i + 1], q / pathLength);

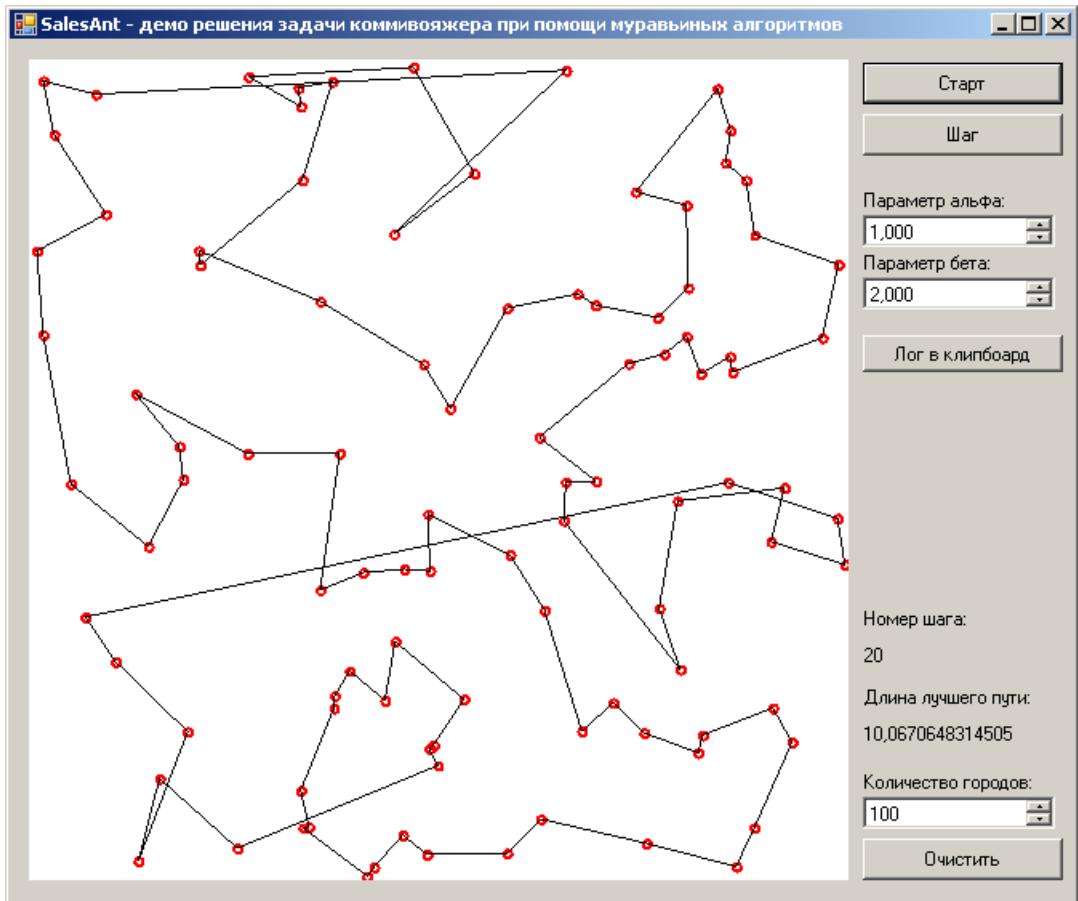
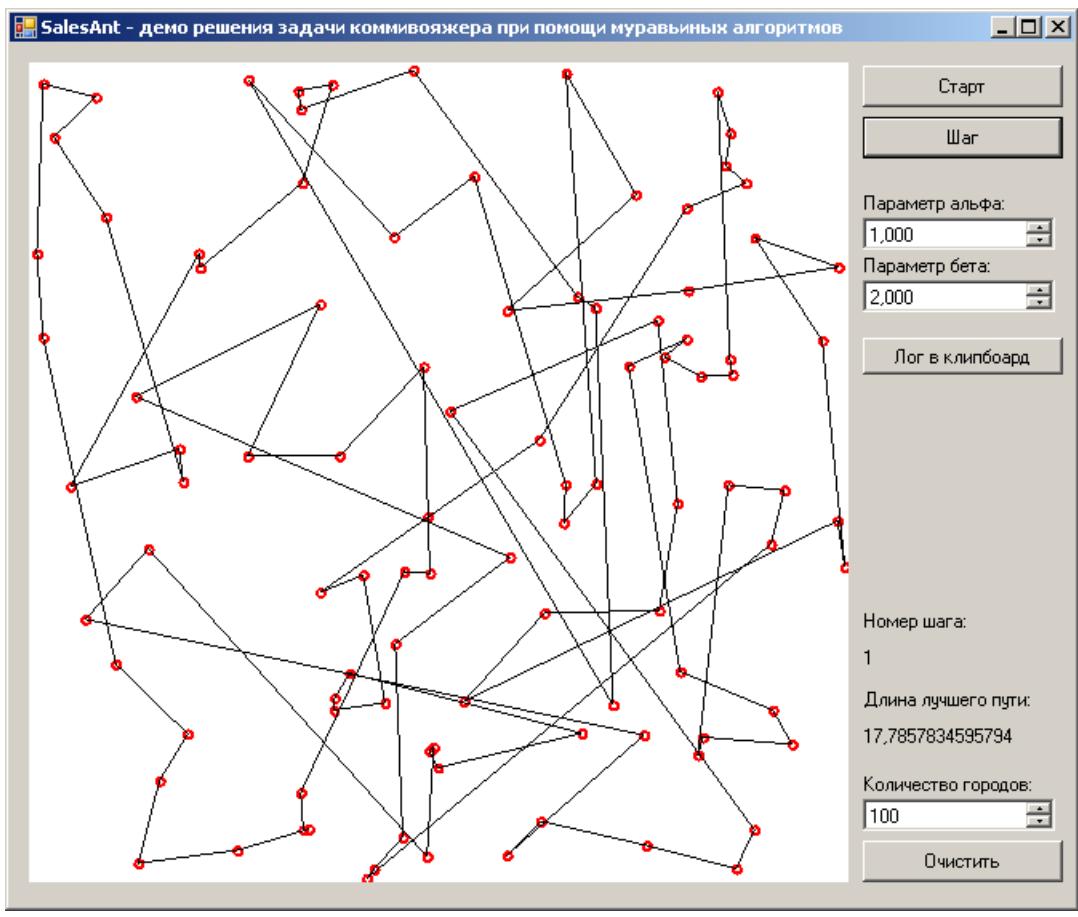
    return pathLength;
}
```

Метод FindNextTown имитирует ruletку с неравными секторами.

```
private int FindNextTown(int currentTown, double alpha, double beta)
{
    // Заполняем "ruleтку" с неравными секторами
    List<RouletteSector> rouletteSectors = new List<RouletteSector>();
    double sum = 0;
    for (int i = 0; i < _townNumber; ++i)
    {
        if (! _pathHash.ContainsKey(i))
```

```
{  
    rouletteSectors.Add(  
        new RouletteSector()  
        {  
            Start = sum,  
            Town = i,  
        } );  
    sum +=  
        Math.Pow(_owner.GetScent(currentTown, i), alpha) /  
        Math.Pow(_owner._distances[currentTown, i], beta);  
}  
}  
// Выбираем на "рулетке" случайный сектор бинарным поиском  
double val = _owner._rnd.NextDouble() * sum;  
int a = 0; int b = rouletteSectors.Count;  
while (b - a > 1)  
{  
    int c = (a + b) / 2;  
    if (rouletteSectors[c].Start < val)  
        a = c;  
    else  
        b = c;  
}  
return rouletteSectors[a].Town;  
}  
}
```

Итак, мы рассмотрели код, который реализует алгоритм муравьиного алгоритма. Посмотрим, как это выглядит графически. Запустим программу, установим количество городов равным 100, а параметр бета равным 2. Далее следуют скриншоты начала работы алгоритма, середина и установившийся процесс. Поскольку при каждом запуске города имеют случайные координаты, то скриншоты у каждого будут свои.



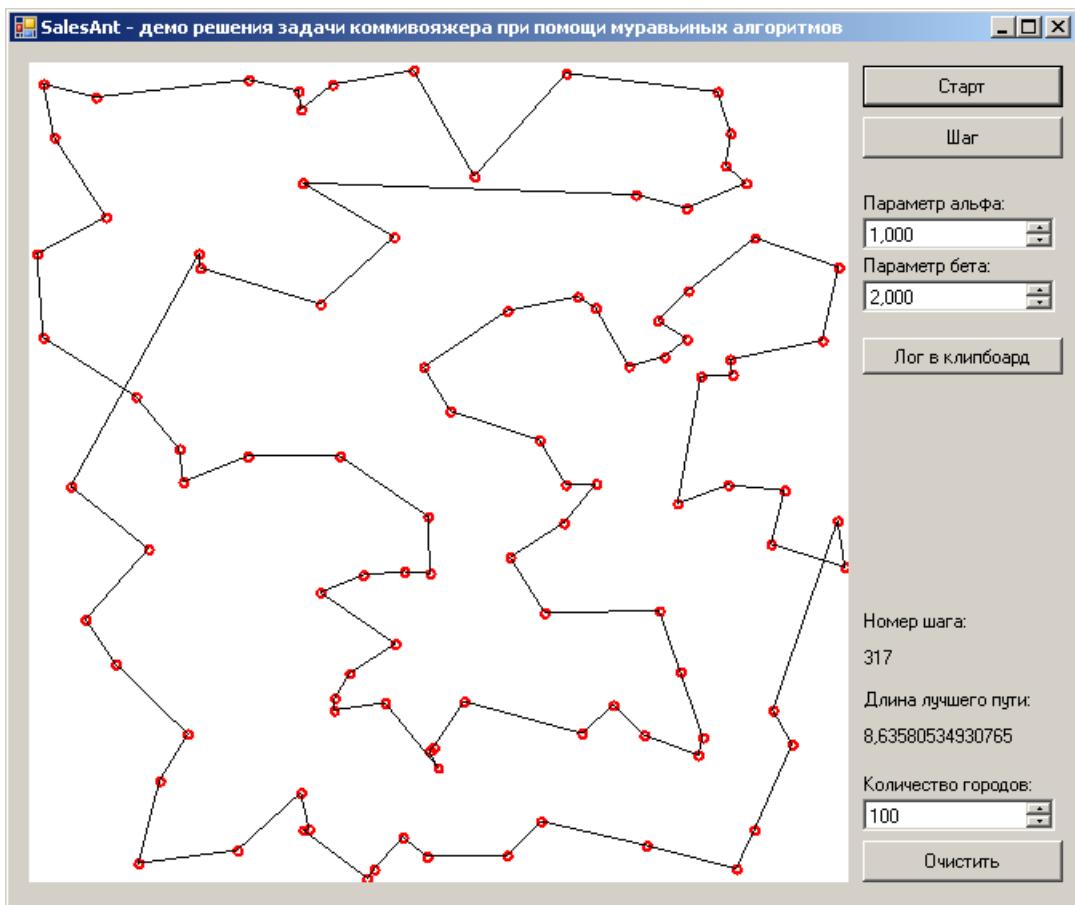


Рис. 4.10. Скриншоты решения задачи коммивояжера

Видно, что в установившемся решении есть еще петли в пути, что говорит о некоторой неоптимальности, но она не столь велика, как вначале.

Можно поиграться с настройками алгоритма. Например, если при данном количестве городов мы выберем параметр бета равным 10, то решение найдется быстрее. Но поиск замрет быстрее. Также, можно попытаться убрать мелкие петли, добавив элитных муравьев. Пробуйте! Наблюдать искусственную жизнь всегда занятно – чувствуешь себя чуть-чуть богом...

5. Методы кластеризации

С увеличением объемов информации, обрабатываемой, хранимой и получаемой в результате работы информационных систем и процессов, в ходе деятельности предприятий или научно-исследовательской деятельности, ее обработка и анализ становятся затруднительным. Таким образом, возникает необходимость первоначальной обработки информации для ее структурирования, выделения характерных признаков, обобщения, сортировки.

Для этой цели применяют процессы классификации и кластеризации, позволяющие проводить первичную обработку информации, для ее последующего анализа ([1],[15],[23],[31],[83]).

Практически все методы классификации и кластеризации ([35]) основаны на гипотезе компактности. Её можно сформулировать следующим образом – объекты, относящиеся к одному классу, должны быть расположены компактно хотя бы в одном из возможных пространств описания объекта.

Классификация

Классифицирование — процесс упорядочения или распределения объектов (наблюдений) по классам с целью отражения отношений между ними. Класс — это множество объектов, имеющих определенный общий признак, отличающий эту совокупность от других объектов. В качестве классификационного деления можно принять различные признаки, зависящие от цели классификации. В основу класса всегда кладут наиболее важный признак объекта, отвечающий цели классификации.

Классифицировать объект — значит указать класс, к которому относится данный объект. Результатом классификации объекта является наименование класса, что определяется алгоритмом классификации в результате его применения к данному конкретному объекту.

Обучение классификатора — процесс настройки алгоритма в случае, когда задано конечное множество объектов, для которых известно, к каким классам они отно-

сятся. Это множество называется выборкой. Принадлежность к тому или иному классу остальных объектов не известна.

Кластеризация

Кластеризация — процесс разбиения заданной выборки объектов (наблюдений) на подмножества (как правило, непересекающиеся), называемые кластерами, так, чтобы каждый кластер состоял из схожих объектов, а объекты разных кластеров существенно отличались.

Одной из целей кластеризации является выявление внутренних связей между данными путём определения кластерной структуры. Разбиение наблюдений на группы схожих объектов позволяет упростить дальнейшую обработку данных и принятие решений, применяя к каждому кластеру свой метод анализа - “divide et impera” (стратегия «разделяй и властвуй»).

Одним из приложений кластеризации является решение задачи сжатия данных. В случае, если исходная выборка избыточно большая, то можно сократить её, оставив несколько наиболее характерных представителей от каждого кластера.

Другой сферой использования кластеризации является обнаружение новизны в исследуемом множестве объектов. Выделяются нетипичные объекты, которые не удается присоединить ни к одному из кластеров. Для решения задач методами кластерного анализа, необходимо задавать количество кластеров заранее. В одном случае число кластеров стараются сделать поменьше. В другом случае важнее обеспечить высокую степень сходства объектов внутри каждого кластера, а кластеров может быть сколько угодно. В третьем случае наибольший интерес представляют отдельные объекты, не вписывающиеся ни в один из кластеров.

Кластеризация

Общие понятия

Пусть $\mathfrak{I} = \{x_i\}_{i=1}^n$ множество объектов, представленных набором атрибутов $x_i = \{t_1^i, t_2^i, \dots, t_m^i\}$, где t_v^i принимает значения из заданного множества T_v^i . Задача класте-

ризации состоит в построении множества $C = \{c_\nu\}_{\nu=1}^k$ и отображения $F: \mathfrak{I} \rightarrow C$ заданного множества объектов на множество кластеров.

Кластер содержит объекты из \mathfrak{I} похожие (по заданному критерию) друг на друга

$$x_i \in c_\nu, x_j \in c_\nu \Rightarrow d(x_i, x_j) < \varepsilon,$$

где $d(\bullet, \circ)$ – мера близости между объектами (расстояние), а ε – максимальное значение порога, формирующего один кластер.

Наиболее используемыми мерами близости является понятие расстояния между двумя точками $\|x_i - x_j\|$. Приведем некоторые наиболее часто используемые определения расстояния.

- Евклидово (среднеквадратическое или ℓ_2) расстояние

$$\|x_i - x_j\|_2 = \sqrt{\sum_{\nu=0}^m (t_\nu^i - t_\nu^j)^2},$$

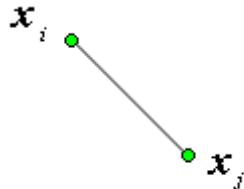


Рис.5.1. Иллюстрация Евклидова расстояния.

- расстояние ℓ_1 (в англоязычной литературе – манхэттенское или городское расстояние)

$$\|x_i - x_j\|_1 = \sum_{\nu=0}^m |t_\nu^i - t_\nu^j|$$

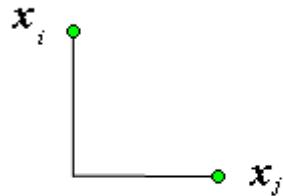


Рис. 5.2. Иллюстрация расстояния ℓ_1 .

используется в том случае, когда нужно уменьшить влияние отдельных выбросов;

- Чебышевское (равномерное или ℓ_∞) расстояние

$$\|x_i - x_j\|_\infty = \max_{v=0,\dots,m} |t_v^i - t_v^j|,$$

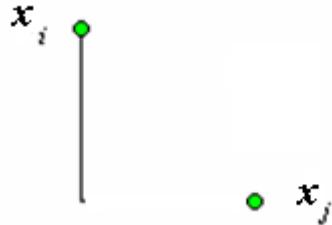


Рис. 5.3. Иллюстрация расстояния Чебышева.

используется если нужно увеличить влияние отдельных выбросов;

- расстояние Махалонобиса

$$\|x_i - x_j\|_M = \sqrt{(x_i - x_j) \Sigma^{-1} (x_i - x_j)^T},$$

используется если нужно учесть корреляцию на конкретном классе. Здесь Σ корреляционная матрица.

Заметим, что во многих случаях вместо расстояния в качестве критерия близости используется значение косинуса угла между двумя векторами

$$\cos \varphi_{i,j} = \frac{x_i^T x_j}{\|x_i\|_2 \|x_j\|_2}$$

или коэффициент корреляции

$$\sigma_{i,j} = \frac{\sum_{v=1}^n (t_i^v - \mu_i)(t_j^v - \mu_j)}{\sqrt{\sum_{v=1}^n (t_i^v - \mu_i)^2 \sum_{v=1}^n (t_j^v - \mu_j)^2}}.$$

Для задачи кластеризации очень важно, чтобы все данные были соизмеримы, т.е. чтобы не получилось, что одна часть данных измеряется в упаковках, а другая – в килограммах, другими словами, на первом этапе кластеризации всегда нужно проводить нормализацию данных. Кроме того, при проведении процедуры кластеризации нужно заранее задаваться либо числом кластеров, либо критерием, характеризующем близость элементов внутри кластера либо удаленность самих кластеров друг от друга.

Условно методы кластеризации разбиваются на два класса - иерархические и неиерархические. В неиерархических алгоритмах присутствует наличие условия остановки и количества кластеров. Основой этих алгоритмов является гипотеза о сравнительно небольшом числе скрытых факторов, которые определяют структуру связи между признаками. Иерархические алгоритмы не завязаны на количество кластеров. Эта характеристика определяется по динамике слияния и разделения кластеров во время построения дерева вложенных кластеров (дендограммы). В свою очередь, иерархические алгоритмы делятся на агломеративные, которые строятся путем объединения элементов, то есть уменьшением количества кластеров, и дивизимные, основанные на разделении (расщеплении) существующих групп (кластеров) (см., например, [95], [96], [112]).

Иерархические методы

Агломеративные алгоритмы

На первом шаге с каждым элементом \mathfrak{I} связывается свой кластер $c_i^0 = \{x_i\}$. На каждом последующем шаге два наиболее близких кластера c_i^v и c_i^μ объединяются в один, результатом чего будет множество из $n-1$ кластера $c_1^1 = \{c_v^0, c_\mu^0\}, c_i^1 = c_1^0 (i \neq \{v, \mu\})$.

Далее этот процесс повторяется. В пределе получим один кластер, совпадающий с \mathfrak{I} .

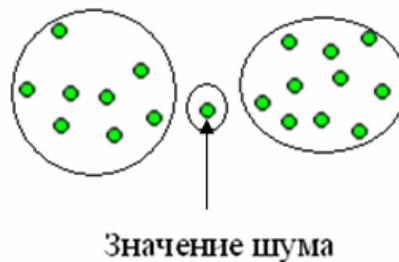


Рис. 5.5. Исходные данные.

При объединении v и μ -го кластеров в i -й кластер нужно вычислить расстояние от нового кластера до j -го кластера. Традиционно, для пересчета расстояния во время

слияния кластеров используются старые значения расстояний. Как правило, при этом используют следующие критерии:

- минимальное расстояние

$$d_{\min}(c_i, c_j) = \min \{ \|x - y\| \mid x \in c_i, y \in c_j \},$$

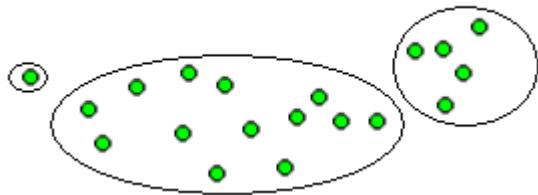


Рис. 5.6. Результат применения агломеративной кластеризации с минимальным расстоянием.

Данный алгоритм способствует росту вытянутых кластеров. К недостаткам алгоритма, прежде всего, следует отнести чувствительность к шуму.

- максимальное расстояние

$$d_{\max}(c_i, c_j) = \max \{ \|x - y\| \mid x \in c_i, y \in c_j \}$$

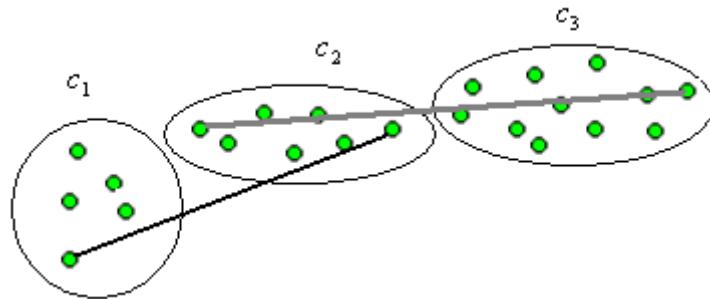


Рис. 5.7. Результат применения агломеративной кластеризации с максимальным расстоянием.

Алгоритм на основе максимального расстояния способствует формированию компактных кластеров. К недостаткам относится эффект разрушения вытянутых кластеров.

- среднее расстояние

$$\tilde{d}(c_i, c_j) = \frac{1}{n_i n_j} \sum_{x \in c_i} \sum_{y \in c_j} \|x - y\|;$$

- расстояние между центрами кластеров

$$d_\mu(c_i, c_j) = \|\mu_i - \mu_j\|.$$

На практике достаточно неплохо работают алгоритмы, основанные на среднем расстоянии, но, с точки зрения робастости и эффективности с вычислительной точки зрения, более эффективны алгоритмы, основанные на расстоянии между центрами кластеров.

Дивизимные алгоритмы

Идеология дивизимных алгоритмов двойственна агломеративным. На первом шаге все множество \mathfrak{I} представляется как один кластер, и на каждом шаге один из существующих кластеров разбивается на два.

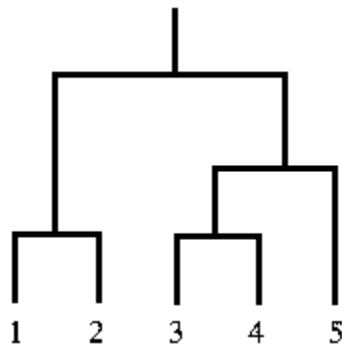


Рис. 5.8. Иллюстрация дивизимного алгоритма кластеризации.

Самый простой, из такого рода алгоритмов, описан Смитом Макнаотоном в 1965 году. Его суть в том, что вначале выбирается элемент кластера $c_0^1 = \mathfrak{I}$, который наиболее удален от центра кластера, и этот элемент формирует новый кластер c_2^1 , оставшиеся элементы, формируют $c_1^1 = c_0^1 \setminus c_2^1$. На каждом последующем шаге элемент из c_1^1 , для которого разница между расстоянием до центра кластера c_2^1 и расстоянием до центра кластера c_1^1 , наибольшая, переносится в c_2^1 . Этот процесс продолжается до тех пор, пока эта разность не станет отрицательной, то есть пока не будут выбраны все элементы из c_1^1 , которые ближе к центру кластера c_2^1 . Результатом будет расщепление одного кластера на два. Далее этот процесс итерационно продолжается. Выбор рас-

щепляемого кластера может проводиться исходя из разных соображений, например, выбирается кластер с наибольшим диаметром $\max_{i,j} \|x_i - x_j\|$, где элементы x_i , и x_j лежат в одном кластере. Как правило, дивизимные алгоритмы используются в случае малых выборок.

Неиерархические алгоритмы

Неиерархические алгоритмы приобрели большую популярность, ввиду того, что в их основе лежит та или иная задача оптимизации, то есть группирование исходного множества объектов в кластеры является решением некоторой экстремальной задачи. Рассмотрим несколько наиболее популярных методов.

Одним из самых популярных методов численного анализа является метод наименьших квадратов. Для задачи кластеризации он выглядит следующим образом

$$\sum_{j=1}^k \sum_{i=1}^{n_j} |x_i - s_j|^2 \rightarrow \min$$

по всем s_j и k .

Численная реализация этой задачи называется методом k-средних.

Метод k-средних

Идея метода состоит в следующем - вначале выбирается k произвольных исходных центров из множества \mathfrak{X} . Далее все объекты разбиваются на k групп, наиболее близких к соответствующему центру. На следующем шаге вычисляются центры найденных кластеров. Процедура повторяется итерационно до тех пор, пока центры кластеров не стабилизируются.

Алгоритм разбиения объектов x_i ($i=0, 1, \dots, n$) основан на минимизации межкластерного расстояния, в случае, если в качестве расстояния используется среднеквадратичная норма ℓ_2 , то есть целевой функцией является

$$S = \sum_{j=1}^k \sum_{i=1}^{n_j} \left\{ |x_i - \mu_j|^2 \mid x_i \in c_j \right\},$$

где x_i - i -й объект, а c_j представляет собой j -й кластер с центром μ_j .

Структура алгоритма состоит в следующем:

1. Для инициализации алгоритма выбираем k центров кластеров.
2. Каждому из n объектов ставим в соответствие кластер, исходя из минимизации ℓ_2 нормы между объектом и центром соответствующего кластера.
3. Пересчитываем центры вновь полученных кластеров.
4. Для каждого i , такого, что $x_i \in c_j$ вычислим

$$h = \arg \min \left\{ \frac{n_r \|x_i - \mu_r\|_2}{n_r - 1} \right\},$$

где n_r число объектов кластера C_r .

Для решения этой задачи среди всех элементов кластера $x \in c_i$ найдем элемент z , минимизирующий уклонение $\sum_{x \in c_i} \|x - z\|_2^2$, для чего найдем решение задачи

$$\frac{\partial}{\partial z} \sum_{x \in c_i} \|x - z\|_2^2 = \frac{\partial}{\partial z} \sum_{x \in c_i} (\|x\|_2^2 - 2x^T z + \|z\|_2^2) = \sum_{x \in c_i} (-x + z) = 0, \text{ то есть } z = \frac{1}{n_i} \sum_{x \in c_i} x.$$

5. Если выполняется условие

$$\frac{n_h \|x_i - \mu_h\|_2}{n_h - 1} < \frac{n_j \|x_i - \mu_j\|_2}{n_j - 1},$$

то следует переместить объект x_i из кластера c_j в кластер c_h , после чего пересчитать значения центров кластеров.

6. Если $i < n$, то переходим к шагу 4, иначе к шагу 3.

Критерием остановки алгоритма может служить либо достижение заданного числа итераций алгоритма, либо достижение функцией цели заданного значения порога.

Метод эффективен в случае, если данные делятся на компактные группы, которые можно описать сферой. Использование индикаторной функции позволяет упростить запись базового алгоритма и записать в следующем виде

Пусть $C = \{c_i\}_{i=1}^k$ множество кластеров с центрами

$$\mu_i = \frac{\sum_{j=1}^n \{x_j \mid x_j \in c_i\}}{\sum_{j=1}^n \{1 \mid x_j \in c_i\}} = \frac{\sum_{j=1}^n u_j^i x_j}{\sum_{j=1}^n u_j^i},$$

где u_j^i индикаторная функция, то есть

$$u_j^i = \begin{cases} 1, & \text{если } x_j \in c_i, \\ 0, & \text{в противном случае.} \end{cases}$$

Целевая функция

$$S(C, \mathfrak{I}) = \sum_{i=1}^k \sum_{j=1}^n u_j^i d(x_j, \mu_i),$$

и условия

$$\sum_{i=1}^n u_j^i = 1, 0 < \sum_{j=1}^k u_j^i \leq n,$$

то есть, каждый элемент может быть только в одном кластере, и, кластер не может быть пустым или содержать элементов больше, чем их исходное количество.

Условие остановки выполнения алгоритма после v -го шага будет иметь вид

$$|S^v(C, \mathfrak{I}) - S^{v-1}(C, \mathfrak{I})| < \varepsilon,$$

где ε выбранный порог.

Заметим, что при вычислении критерия принадлежности, можно учитывать размер кластера, что позволяет улучшить эффективность алгоритма. Критерий того, что j -й элемент принадлежит i -му, а не k -му кластеру будет иметь вид:

$$\frac{n_i}{n_i - 1} d(x_j, \mu_i) < \frac{n_k}{n_k - 1} d(x_j, \mu_k),$$

где n_i количество элементов, соотнесенных кластеру c_i . Скорость сходимости метода - $O(n)$.

Недостатки метода

К недостаткам этого метода, прежде всего, нужно отнести:

- Наличие априорной информации о количестве кластеров;
- Чувствительность к изолированным удаленным элементам;
- Существенная зависимость скорости сходимости метода от начального выбора центров кластеров.

Приведем иллюстрацию метода k-средних из http://en.wikipedia.org/wiki/K-means_algorithm

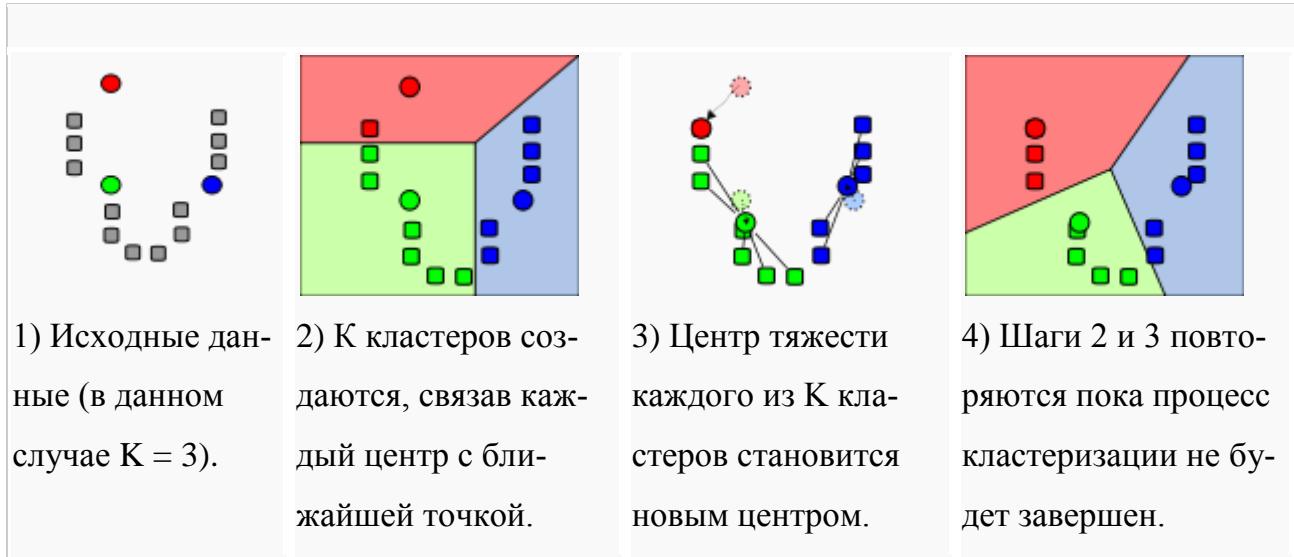


Рис. 5.9. Иллюстрация метода k -средних. Стартовые центры кластеров выбраны случайно и показаны красным, синим и зеленым цветом.

Fuzzy k-means

Этот алгоритм является обобщением предыдущего, в случае, если кластеры являются нечеткими множествами, и, элемент может принадлежать разным кластерам с разной степенью надежности.

Пусть $w \in (1, \infty)$ (обычно берется $w=2$) весовой коэффициент нечеткости. Пусть $C = \{c_i\}_{i=1}^k$ множество кластеров с центрами

$$s_i = \frac{\sum_{j=1}^n (u_j^i)^w x_j}{\sum_{j=1}^n (u_j^i)^w},$$

где

$$u_j^i = \begin{cases} 1, & \text{если } x_j \in c_i, \\ 0, & \text{в противном случае.} \end{cases}$$

Целевая функция

$$S(C, \mathfrak{I}) = \sum_{i=1}^k \sum_{j=1}^n (u_j^i)^w d(x_j, \mu_i),$$

и условия

$$\sum_{i=1}^n u_j^i = 1,0 < \sum_{j=1}^k u_j^i \leq n,$$

то есть, каждый элемент может быть только в одном кластере, и, кластер не может быть пустым или содержать элементов больше, чем их исходное количество.

Условие остановки выполнения алгоритма после v -го шага будет иметь вид

$$|S^v(C, \mathfrak{I}) - S^{v-1}(C, \mathfrak{I})| < \varepsilon,$$

где ε выбранный порог. Скорость сходимости метода - $O(n)$.

Кластеризация Гюстафсона-Кесселя

По сути, тот же алгоритм, но используем корреляционные зависимости кластера, благодаря чему, кластеры вместо сферической формы становятся эллипсоидами, что позволяет более качественно проводить разбиение, в случае, если элементы \mathfrak{I} вытянуты вдоль каких-либо направлений, другими словами, если существуют устойчивые словосочетания, определяющие кластер, то лучше рассматривать элементы на принадлежность к кластеру вдоль этих сочетаний.

Короче, все тот же метод k-средних, но используется расстояние Махalanобиса.

FOREL (ФОРмальный ЭЛемент)

Одна из модификаций k- средних. Отличие состоит в том, что под близостью элементов в кластере понимается покрытие их сферой заданного радиуса.

Схема работы алгоритма состоит в следующем - берется центр кластера (на первом шаге это произвольный элемент), и, к кластеру приписываются все элементы, которые удалены от центра не более чем на заданное расстояние R . Потом пересчитывается центр, в качестве которого берется средняя точка полученного кластера, и, заново пересчитываются элементы кластера. Так продолжается до тех пор, пока центр не стабилизируется.

Замечание

Метод k-средних, как и его модификации, опирается на априорную информацию о количестве классов. Это можно обойти. Применяем метод k-средних или его модификацию последовательно для каждого k и вычисляем максимальную ошибку. Как только значение этой ошибки стабилизируется, то есть количество кластеров устоялось, так сразу и останавливаемся. Метод долгий. Что есть, то есть.

Приведем иллюстрацию использования метода k-средних для кластеризации изображений. Итак, пусть у нас дано тестовое изображение «Апельсин»



Рис. 5.10. Тестовое изображение

Рассмотрим метод k-средних для массива $(i, j, r_{i,j}, g_{i,j}, b_{i,j})$. Количество кластеров выберем равным пяти.

После первого шага получаем следующие кластеры:

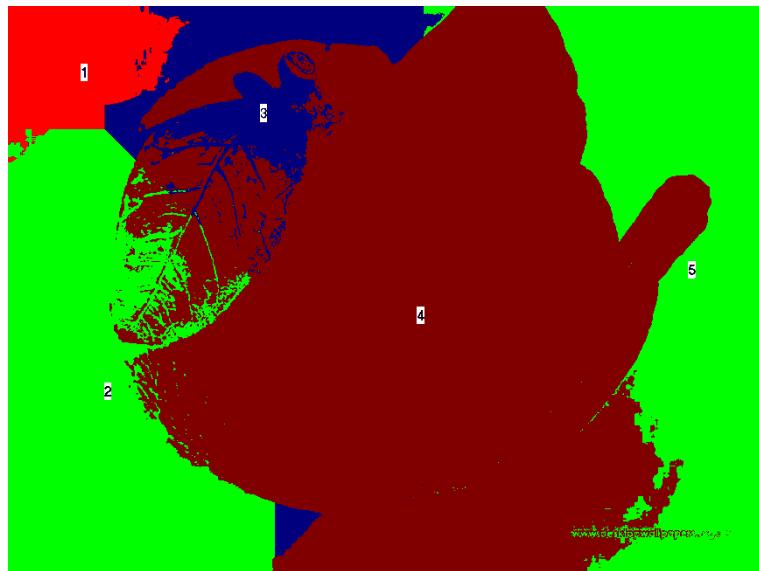


Рис. 5.11. Стартовое распределение кластеров

После шестого шага кластеры распределяются следующим образом:

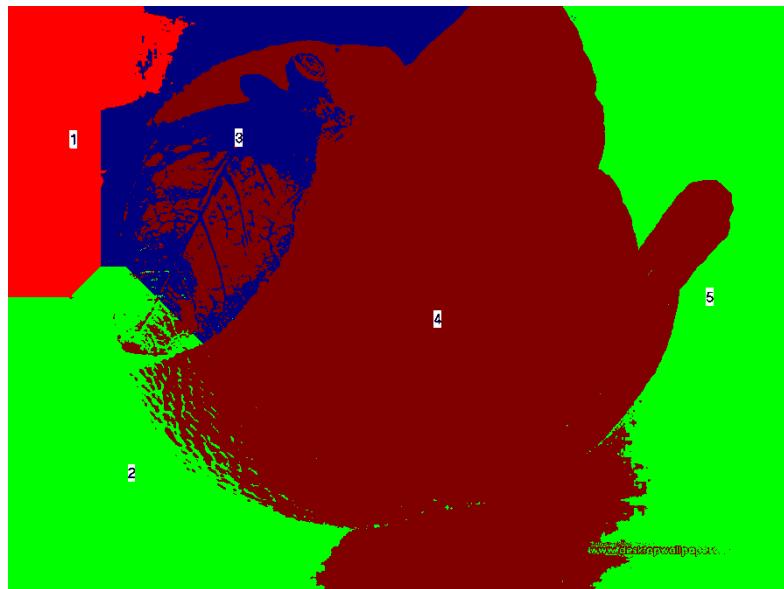


Рис. 5.12. Распределение кластеров после шостого шага

Картишка существенно не меняется даже после ста шагов:

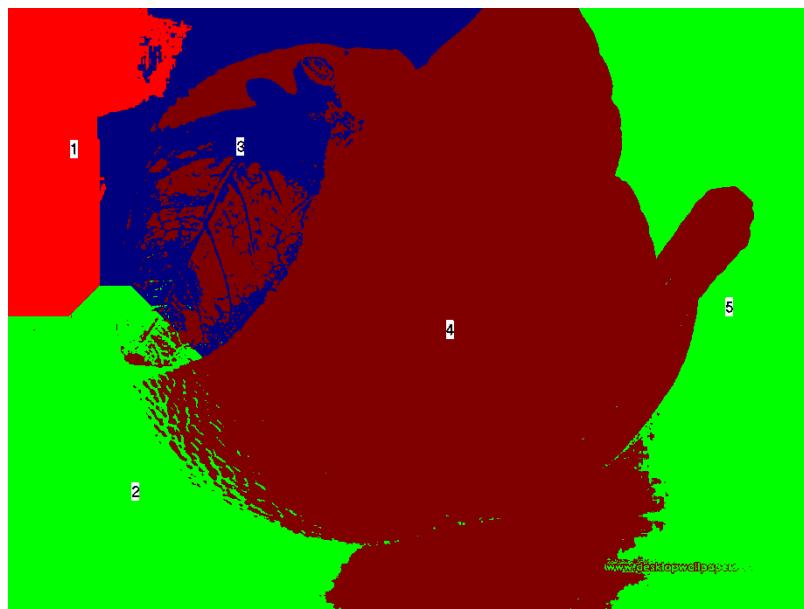


Рис. 5.13. Окончательное распределение кластеров.

Метод корреляционных плеяд

В основе этого метода лежит идея построения кластеров, исходя из максимизации корреляционной связи, т.е. сумма модулей коэффициентов корреляции между параметрами одной группы должна быть достаточно велика, а связь между параметрами из разных групп - мала. По корреляционной матрице объектов строится граф, который затем разбивается на подграфы. Элементы, соответствующие каждому из подграфов, и образуют кластер.

Рассмотрим корреляционную матрицу $C = \{c_{i,j}\}, i, j = 1, 2, \dots, n$ исходных объектов и проведем упорядочивание, которое производится на основании принципа максимального корреляционного пути: все n объектов связываются при помощи $n-1$ линий (ребер), так, чтобы сумма модулей коэффициентов корреляции была максимальной. Опишем построение графа. Вначале рассмотрим вершины графа, соответствующие объектам x_l и x_m . Ребру, соединяющему эти вершины поставим в соответствие вес $c^{(1)} = |c_{l,m}|$. Затем, исключив $c_{l,m}$, находим наибольший коэффициент в m -ом столбце матрицы (это соответствует нахождению элемента, который наиболее сильно после x_l "связан" с x_m , и наибольший коэффициент в ℓ -ой строке матрицы (это соответствует нахождению элемента, наиболее сильно после x_m "связанного" с x_l). Из най-

денных таким образом двух коэффициентов корреляции выбирается наибольший - пусть это будет $c^{(2)} = |c_{l,j}|$. Вершину x_j соединяем с x_l , и, соответствующему ребру проставляем значение $c^{(2)}$. Затем находим объекты, наиболее связанные с x_l , x_m и x_j , и, выбираем из найденных коэффициентов корреляции наибольший. Пусть это будет $c^{(3)} = |c_{j,q}|$. Потребуем, чтобы на каждом шаге появлялся новый элемент, поэтому уже используемые элементы, исключаются. Следовательно, $q \neq l, q \neq m, q \neq j$. Далее вершину графа, соответствующую x_q , соединяем с x_j , и т.д. На каждом шаге определяются элементы, наиболее сильно связанные с двумя последними рассмотренными элементами, а затем выбирается один из них, соответствующий большему коэффициенту корреляции. Процедура заканчивается после $n-1$ -го шага. Граф состоит из n вершин, соединенных $n-1$ -м ребром. После этого задается пороговое значение ε , и все ребра, соответствующие меньшим, чем ε , коэффициентам корреляции, исключаются из графа. Полученные подграфы формируют кластеры.

6. Классификаторы

Лавинообразное количество информации, вырабатываемое человечеством, привело к понятию автоматизации извлечения знаний – Data Mining. Это направление связано с широким спектром задач - от распознавания размытых образов до создания поисковых машин ([121]). Важной составляющей Data Mining является обработка текстовой информации. Такого рода задачи опираются на понятие классификации и кластеризации ([23], [88], [95]). Классификация заключается в определении принадлежности некоторого элемента одному из заранее созданных классов. Кластеризация подразумевает разбиение множества элементов на кластеры, количество которых определяется локализацией элементов заданного множества в окрестностях некоторых естественных центров этих кластеров. Реализация задачи классификации изначально должна опираться на заданные постулаты, основные из которых – априорная информация о первичном множестве объектов и мера близости элементов и классов.

Постановка задачи классификации

Будем использовать следующую модель задачи классификации.

Пусть –

Ω – множество объектов распознавания (пространство образов).

$\omega \in \Omega$ объект распознавания (образ).

$g(\omega): \Omega \rightarrow \mathbb{R}$, $\mathbb{R} = \{1, 2, \dots, n\}$ – индикаторная функция, разбивающая пространство образов Ω на n непересекающихся классов $\Omega^1, \Omega^2, \dots, \Omega^n$. Индикаторная функция неизвестна наблюдателю.

X – пространство наблюдений, воспринимаемых наблюдателем (пространство признаков).

$x(\omega): \Omega \rightarrow X$ – функция, ставящая в соответствие каждому объекту ω точку $x(\omega)$ в пространстве признаков. Вектор $x(\omega)$ – это образ объекта, воспринимаемый наблюдателем.

В пространстве признаков определены непересекающиеся множества точек $\Xi[i] \subset X$ $i=1, 2, \dots, n$, соответствующих образам одного класса.

$\varphi(x)$: $X \rightarrow \mathbb{R}$ решающее правило - оценка для $g(\omega)$ на основании $x(\omega)$, т.е. $\varphi(x) = \varphi(x(\omega))$.

Пусть $x_\nu = x(\omega_\nu), \nu = 1, 2, \dots, N$ доступная наблюдателю информация о функциях $g(\omega)$ и $x(\omega)$, но сами эти функции наблюдателю неизвестны. Тогда $(g_\nu, x_\nu), \nu = 1, 2, \dots, N$ — есть множество прецедентов.

Задача заключается в построении такого решающего правила $\varphi(x)$, чтобы распознавание проводилось с минимальным числом ошибок.

Основные направления исследования проблемы классификации

Обычный случай — считать пространство признаков евклидовым, а качество решающего правила измерять частотой появления правильных решений. Как правило, его оценивают, наделяя множество объектов Ω некоторой вероятностной мерой. Байесовский подход (см., например, [64]) исходит из статистической природы наблюдений. За основу берется предположение о существовании вероятностной меры на пространстве образов, которая либо известна, либо может быть оценена. Цель состоит в разработке такого классификатора, который будет правильно определять наиболее вероятный класс для пробного образа. Тогда задача состоит в определении “наиболее вероятного” класса. Байесовский подход основывается на предположении о существовании некоторого распределения вероятностей для каждого параметра. Недостатком этого метода является необходимость постулирования как существования априорного распределения для неизвестного параметра, так и знание его количественных характеристик.

Использованию поиска соответствия предшествует построение множества статистик, в которых содержится количество текстов в данном классе и список используемых термов вместе со своими счетчиками.

Для определения подходящего класса текстов для заданного текста строится структура из неповторяющихся термов и их счетчиков $-(w_i, n(w_i))$.

Через M обозначим размер множества статистик. Классы, на принадлежность к которым проверяется текст, обозначим через $c_j (j = 0, \dots, M - 1)$. Для каждого слова w_i из проверяемого текста, в каждой статистике находим это слово и соответствующий

счетчик $n(w_i, c_j)$ (здесь j ($j=0, 1, \dots, M-1$)-номер класса (элемент множества статистик)).

Через $n(c_j)$ обозначим число текстов в j -м классе. Минимизация риска и вероятности ошибки эквивалентны разделению пространства признаков на n областей. Если области смежные, то они разделены поверхностью решения в многомерном пространстве. Для случая построения разделяющей поверхности предпочтительней использовать методы классификации отличные от Байесовской. Использование вероятностных характеристик определяется на распределение Гаусса, которое очень широко используется по причине вычислительного удобства и адекватности во многих случаях.

Если известно, или с достаточным основанием можно считать, что плотность распределения функций правдоподобия $P(x|\Omega^i)$ является гауссовской, то применение классификатора Байеса приводит к тому, что образы, характеризующиеся нормальным распределением проявляют тенденцию к группированию вокруг среднего значения, а их рассеивание пропорционально среднеквадратическому отклонению σ . Вероятностные методы опираются на информацию о плотности распределения вероятностей каждого класса. К сожалению, в реальных задачах информация о плотности распределения отсутствует.

J.Rocchio [118] для решения задачи автоматической классификации объектов аэрокосмической съемки, предложил алгоритм TF-IDF (term frequency / inverse document frequency). Несколько слов о TF-IDF. TF-IDF - статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов или корпуса. Вес некоторого слова пропорционален количеству использований этого слова в документе и обратно пропорционален частоте использования слова в других документах коллекции.

Мера TF-IDF часто используется в задачах анализа текстов и информационного поиска, например, как один из критериев релевантности документа поисковому запросу, при расчёте меры близости документов при кластеризации.

TF (*term frequency* — частота слова) — отношение числа вхождения некоторого слова к общему количеству слов документа. Таким образом, оценивается важность слова t_i в пределах отдельного документа

$$TF = \frac{n_i}{\sum_k n_k},$$

где n_i есть число вхождений слова в документ, а в знаменателе — общее число слов в данном документе.

IDF (*inverse document frequency* — обратная частота документа) — инверсия частоты, с которой некоторое слово встречается в документах коллекции. Учёт IDF уменьшает вес широкоупотребительных слов

$$IDF = \log \frac{|D|}{|(d_i \supset t_i)|}.$$

Где $|D|$ — количество документов в корпусе; $|(d_i \supset t_i)|$ — количество документов, в которых встречается t_i (когда $n_i \neq 0$).

Таким образом, мера TF-IDF является произведением двух сомножителей: TF*IDF. Большой вес в TF-IDF получат слова с высокой частотой в пределах конкретного документа и с низкой частотой употреблений в других документах.

В 1974 г. вышла книга В.Н.Вапника и А.Я.Червоненкиса [7], положившая начало целой серии их работ в этой области. Предложенные авторами методы распознавания образов и статистическая теория обучения, лежащая в их основе, оказались весьма успешными. Алгоритмы классификации и регрессии под общим названием SVM во многих случаях успешно заменили нейронные сети и в данное время применяются очень широко.

Идея метода основывается на предположении о том, что наилучшим способом разделения точек в n -мерном пространстве является $n-1$ плоскость (заданная функцией $f(x)$), равноудаленная от точек, принадлежащих разным классам. Метод опорных векторов (Support Vector Machine - SVM) относится к группе граничных методов. Эта группа методов определяет классы при помощи границ областей. Опорными векторами называются объекты множества, лежащие на границах областей. Классификация считается хорошей, если область между границами пуста. Однако сложность построения SVM-модели заключается в том, что чем выше размерность пространства, тем сложнее с ним работать, что существенно ограничивает использование SVM.

Стochastic classifiers

Использование теоремы Байеса (Bayes) для принятия решения

Рассмотрим один из наиболее популярных методов классификации, основанный на стохастических принципах. Основой этого метода является априорная информация о распределении вероятностей существующих классов C_i .

Априорная (исходная) вероятность интерпретируется как описание информации в отсутствии свидетельства события. Апостериорная (последующая) вероятность принимает данное свидетельство во внимание. Априорная вероятность появления события у при наблюдении x вычисляется следующим образом

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}, \quad (6.1)$$

где $P(x)$ априорная вероятность наблюдения x, и, соответственно, $P(y)$ априорная вероятность появления события y.

Рассмотрим следующую задачу – по росту человека принять решение о том, кто это мужчина или женщина.

Длина тела	Мужчина	Женщина
Карликовая	ниже 129.9 см	ниже 121.9 см
Очень малая	130—149.9 см	121—139.9 см
Малая	150—159.9	140—148.9 см
Ниже средней	160—163.9 см	149—152.9 см
Средняя	164—166.9 см	153—155.9 см
Выше средней	167—169.9 см	156—158.9 см
Большая	170—179.9 см	159—167.9 см
Очень большая	180—199.9 см	168—186.9
Гигантская	от 200 см и выше	от 187 см и выше

На рост человека может влиять этническая принадлежность к той или иной группе народов. Так, к примеру, средний рост китайцев – 164,8 см (у мужчин) и 154,5 см (у женщин), а средний рост нидерландцев – 184,8 см и 168,7 см, соответственно.

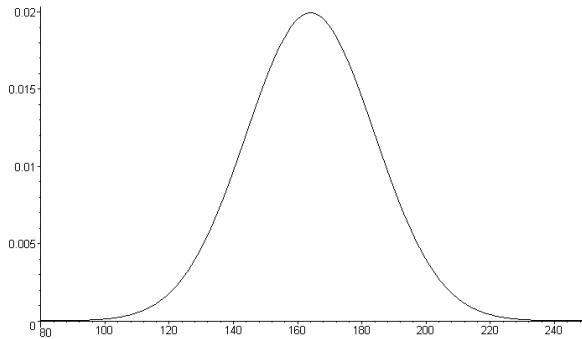
Но мы будем основываться на том факте, что средний рост мужчины на планете 164 см, а женщины — 154 см.

Будем считать, что рост людей имеет нормальное распределение $N(\mu, \sigma^2)$

$$p(l) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(l-\mu)^2}{2\sigma^2}\right),$$

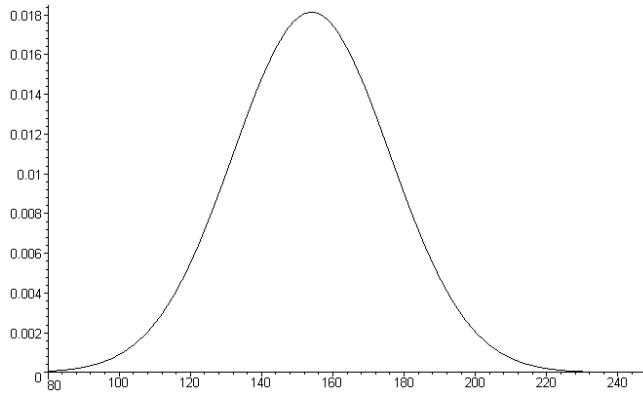
в частности, рост мужчин имеет распределение

$$p(l|\text{мужч.}) = \frac{1}{20\sqrt{2\pi}} \exp\left(-\frac{(l-164)^2}{2(20)^2}\right),$$

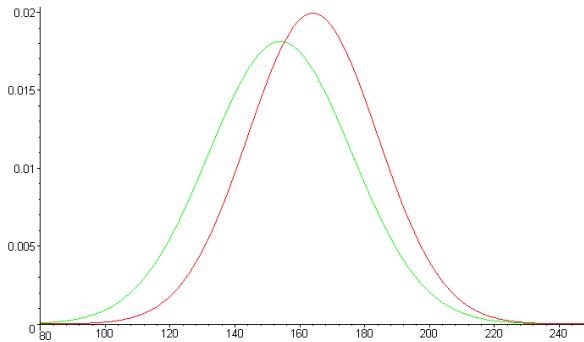


а для женщин

$$p(l|\text{женщ.}) = \frac{1}{22\sqrt{2\pi}} \exp\left(-\frac{(l-154)^2}{2(22)^2}\right).$$



Нужно оценить параметры этих распределений для их использования в теореме Байеса для принятия решения о том, что данный рост соответствует мужчине или женщине.



Построим классификатор максимального правдоподобия. Он основывается на том, что приоритет отдается событию, у которого при данном наблюдении большая вероятность, то есть, если при данном значении роста l выполняется неравенство

$$p(l|\text{мужч.}) > p(l|\text{женщ.}),$$

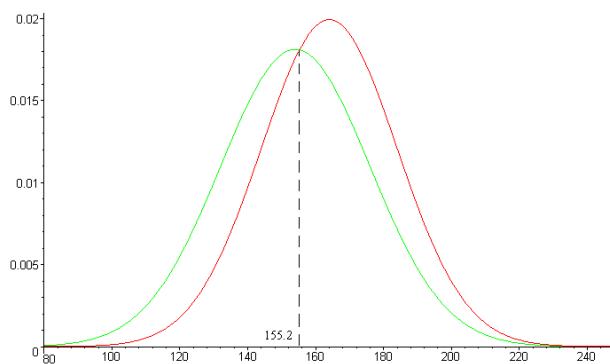
то есть

$$\frac{1}{20\sqrt{2\pi}} \exp\left(-\frac{(l-164)^2}{2(20)^2}\right) > \frac{1}{22\sqrt{2\pi}} \exp\left(-\frac{(l-154)^2}{2(22)^2}\right),$$

то принимаем решение, что наблюдаем мужчину, в противном случае

$$\frac{1}{20\sqrt{2\pi}} \exp\left(-\frac{(l-164)^2}{2(20)^2}\right) < \frac{1}{22\sqrt{2\pi}} \exp\left(-\frac{(l-154)^2}{2(22)^2}\right)$$

считаем, что наблюдаем женщину. В этом случае точка равновесия, в которой мы не можем определить пол человека соответствует росту в 155,2 см. Если рост меньше 155,2 см., то классификатор максимального правдоподобия считает, что данный человек женщина, если больше-мужчина.



Априорная вероятность наблюдаемости мужчины и женщины разная. Будем считать, что вероятность того, что встретится мужчина $P(\text{мужч.}) = \frac{2}{5}$, а вероятность того, что встретится женщина $P(\text{женщ.}) = \frac{3}{5}$.

В соответствии с правилом Байеса

$$P(\text{мужч.} | l) = \frac{p(l|\text{мужч.})P(\text{мужч.})}{p(l)} \text{ и } P(\text{женщ.} | l) = \frac{p(l|\text{женщ.})P(\text{женщ.})}{p(l)}.$$

В этом случае классификатор будет иметь вид - если выполняется неравенство

$$\frac{p(l|\text{мужч.})P(\text{мужч.})}{p(l)} > \frac{p(l|\text{женщ.})P(\text{женщ.})}{p(l)},$$

или, что то же,

$$p(l|\text{мужч.})P(\text{мужч.}) > p(l|\text{женщ.})P(\text{женщ.}),$$

то данный человек мужчина, в противном случае-женщина.

Таким образом, решая неравенство

$$\frac{2}{5} \cdot \frac{1}{20\sqrt{2\pi}} \exp\left(-\frac{(l-164)^2}{2(20)^2}\right) > \frac{3}{5} \cdot \frac{1}{22\sqrt{2\pi}} \exp\left(-\frac{(l-154)^2}{2(22)^2}\right),$$

получаем, что если $l > 173.4$, то принимаем решение, что данный человек мужчина, если $l < 173.4$, то женщина.

Наивный байесовский классификатор

Байесовский классификатор основывается на том, что известны априорные вероятности гипотез $P(c_i)$, то есть, вероятности выпадения классов c_i ($i=1,2,\dots,k$). Ответа на вопрос: как их найти, байесовский классификатор не дает ([108]). Наивный байесовский классификатор позволяет решить эту проблему, правда, плохо, даже очень плохо, но, все же, это лучше, чем ничего. Наиболее часто используемая область наивного байесовского классификатора относится к задаче классификации текстов, где этот метод позволяет получить достаточно неплохие результаты ([114]). Наивный байесовский классификатор предполагает условную независимость атрибутов, в частности, при обработке текстов, предположения наивного классификатора совершенно обескураживающие – вероятность появления слова не зависит от других слов в до-

кументе и, более того, не зависит от размера документа. Как ни удивительно, наивный байесовский классификатор для обработки текстов оказался довольно эффективным и получил широкое распространение, в частности, для фильтрации спама ([43]).

Перейдем к изложению и обсуждению наивного байесовского классификатора.

Предполагается, что алгоритм классификации работает на некотором множестве документов $D = \{b_i\}$. Все множество документов разбивается на непересекающиеся подмножества классов

$$C = \{c_i\}, \bigcup_i b_i = D, c_i \bigcap_j c_j = \emptyset (i \neq j).$$

Задачей классификации является определение класса, к которому относится данный документ. Каждому элементу b ставится в соответствие набор признаков $b = \{w_i\}$. Набор документов, определяющий класс, в дальнейшем мы будем называть обучающей выборкой.

Далее применяется алгоритм классификации для выделения документов наиболее соответствующих заданному классу.

Для того, чтобы применить теорему Байеса (6.1) для классификации документов, делаются следующие предположения:

$$P(c_j) = \frac{n(c_j)}{\sum_j n(c_j)},$$

где $n(c_j)$ – количество термов в классе c_j .

Предполагается, что все термы (слова, словосочетания) независимы, соответственно

$$P(w_i | c_j) = \frac{n(w_i, c_j)}{n(c_j)},$$

где $\{w_i\}$ – набор термов в документе b ; $n(w_i, c_j)$ – количество термов w_i в классе c_j .

Для определения подходящей категории документов для заданного документа, нужно получить соответствующее множество словоформ. По множеству словоформ строится структура из неповторяющихся слов и их счетчиков $-(w_i, n_i)$.

Определение подходящей категории начинается с корня дерева множеств статистики. Через M обозначим количество множеств статистики в данном узле дерева. Категории, на принадлежность к которым мы проверяем документ, обозначим через

$c_j (j=0, \dots, M-1)$. Для каждого слова w_i в каждом множестве статистики находим это слово и соответствующий счетчик $n(w_i, c_j)$ (здесь $j (j=0, 1, \dots, M-1)$ -номер категории (множества статистики)). Через $n(c_j)$ обозначим число документов в j -й категории. Кроме того, пусть

$$N_j(w_i) = \frac{n(w_i, c_j)}{n(c_j)}$$

нормированный счетчик слова w_i в j -й категории.

Тогда вероятность соответствия уникального (то есть каждое слово встречается только один раз) слова w_i j -й категории будет равна

$$P(c_j | w_i) = \frac{N_j(w_i)}{S(w_i)} n(w_i, c_j), \quad (6.2)$$

где

$$S(w_i) = \sum_{j=0}^{M-1} N_j(w_i).$$

Если $P(c_j | w_i) = 0$, то нужно взять это число равным малому значению, например, равным 0,0001.

Тогда вероятность того, что документ соответствует категории $c_j (j=0, \dots, M-1)$ будет равна

$$P(c_j | \{w_i\}) = P(c_j) \prod_i P(c_j | w_i),$$

где произведение берется по всем словам исследуемого множества словоформ и

$$P(c_j) = \frac{n(c_j)}{\sum_{j=0}^{M-1} n(c_j)}$$

- априорная вероятность встречи категории c_j . Заметим, что если документ содержит большое количество слов, которые не встречаются в категории $c_j (j=0, \dots, M-1)$, то значение $P(c_j | \{w_i\})$ может выйти за пределы определения переменной. Поэтому, значение $P(c_j | \{w_i\})$ нужно контролировать, и, если оно выходит за пределы значения переменной, то ограничивать заданным малым числом, например, просто брать равным 0.

В случае, если не отбрасываются стоп-слова (т.е. не несущие информацию о тематике текста, например, «тогда», «если», «но» и т.п.), то имеет смысл уменьшить влияние слов, которые встречаются в большом числе категорий. Для этого имеет смысл использовать конструкцию

$$P(c_j|w_i) = \frac{N_j(w_i)}{S(w_i)} n(w_i, c_j) \log \frac{M+1/2}{M(w_i)+1/2},$$

где $M(w_i)$ - количество категорий, в которых встречается слово w_i .

Заметим, что в этом случае, если все категории содержат все слова документа, то алгоритм покажет несоответствие. Таким образом, этот случай (когда все категории содержат все слова) должен обрабатываться с использованием формулы (6.2).

После первого шага определяем k -штук категорий с наибольшим значением $P(c_j|\{w_i\})$. Сохраняем их название и значения $P(c_j|\{w_i\})$. В соответствии с названием каждого из этих множеств, заходим в соответствующий пункт и проводим обработку. Если информация отсутствует, то этот пункт пропускается. После того, как будут обработаны все дочерние (по определенным на первом шаге) категории, из них и сохраненных на предыдущем шаге, выбирается k -штук категорий с наибольшим значением $P(c_j|\{w_i\})$. Сохраняем их название и значения $P(c_j|\{w_i\})$, после чего, переходим к следующему шагу, но только по тем категориям, которые не были сохранены на предыдущем шаге. Процесс продолжаем до тех пор, пока не будет категорий, по которым можно осуществлять проверку.

Результатом выполнения программы будет k -штук категорий, наиболее вероятных для исследуемого документа, с точки зрения критерия Байеса.

Пример реализации классификатора «Наивный Байес»

Рассмотрим пример программной реализации наивного Байесовского классификатора. В качестве модельной задачи² будем рассматривать следующую. На многих сайтах, посвященных фильмам, книгам, товарам, есть возможность оставлять отзывы-рецензии. Попробуем классификатор научить отличать отрицательную рецензию от положительной.

² Формулировка задачи и местонахождение исходных данных подсмотрено на странице <http://kseeker.narod.ru/index-i-013.html?r=1107115>

Для этого подготовим несколько примеров положительных отзывов и отрицательных. Т.е., у нас будет производиться так называемое обучение с учителем. В качестве параметров, по которым классификатор будет пытаться определять эмоциональную окраску текста, будут отдельные слова, присутствующие в текстах. Точнее, частоты их вхождения в тексты.

Для упрощения программы, сделаем её консольной, а все примеры (обучающие и текстовые) поместим прямо в исходный код. Начнем с общей схемы использования классификатора (файл program.cs). В теле основного класса программы объявлен экземпляр классификатора. Его мы рассмотрим чуть ниже.

Далее идут два примера отрицательных отзывов (класс «-») и два положительных (класс «+»). Эти примеры передаются классификатору для обучения (создания статистики). И затем классификатор тестируется на одном явно положительном и одном явно отрицательном примере.

```
class Program
{
    private static Classifier classifier = new Classifier();

    static void Main(string[] args)
    {
        // Источник мнений:
        // http://www.kinopoisk.ru/level/1/film/468194/

        // Учим классифицировать отрицательные мнения
        classifier.Train(@"

            Собственно о фильме сказать практически нечего,
            ведь фильма-то мы в конце концов даже и не увидели.

            Кривляния актеров чередовались с кадрами в стиле
            «как не надо снимать кино», в то время сценарист
            готовился к контрольной работе по природоведению,
            а многоуважаемый режиссер посещал сеансы
            по корректировке врожденной умственной неполноценности.

            ", "-");

        classifier.Train(@"

            Сказать, что я осталась довольна увиденным будет
            не совсем правильно, скорее вся эта история оставила
```

меня совершенно равнодушной, что довольно странно.
Я всегда очень эмоционально смотрю кино. Почти в каждом
фильме, даже самом трешевом, есть эпизоды,
которые мне запоминаются.
", "-");

```
// Учим классифицировать положительные мнения
classifier.Train(@"
Пошла на этот фильм в самый первый день проката.
Ожидала достаточно много просто по трейлеру
когда в первый раз увидела его в кино,
захотелось посмотреть и сам фильм из-за юмора,
в первую очередь.
```

Не разочаровалась. Пожалуй, единственным минусом
можно назвать сам сюжет — несколько банален, и жесток.
А вот всё остальное, если сравнивать с другими
русскими картинами, на высоком уровне. Итак, опишу.

", "+");

```
classifier.Train(@"
Наверное, только в посредственных и стереотипных
вещах можно увидеть самое главное.
Фильм Александра Черняева «Ирония любви» — яркий
пример извечного проходного сказочного сюжета о том,
как принцесса влюбляется в обычного «Ивана», сюжета,
перенесенного в наши реалии. Фильм можно ругать
сколько угодно, но в нем в очередной раз подмечено
много интересного о психологии современного человека
", "+");
```

```
// Это положительное мнение к фильму
string opinion = @"
В целом фильм стоит воспринимать как приятную
милую картину, которая заставляет умиляться, улыбаться,
сопереживать героям. Советую к просмотру девушкам,
мечтающим о принце на белом коне и верящим в любовь.
";
WriteHypothesis(opinion);
```

```
// А это — отрицательное
```

```
opinion = @"  
Хреновый, скучный фильм. Пока я его досмотрел, поспал  
три раза. Не стоит терять время на это недоразумение.  
";  
WriteHypotesis(opinion);  
  
Console.ReadLine();  
}  
  
private static void WriteHypotesis(string opinion)  
{  
    Console.WriteLine("Отзыв:");  
    Console.WriteLine(opinion);  
    foreach (var item in classifier.GetHypotesis(opinion))  
        Console.WriteLine(  
            string.Format("Гипотеза: {0}, вероятность {1}",  
            item.Key, item.Value));  
    Console.WriteLine();  
}  
}
```

Рассмотрим теперь собственно алгоритмическую часть классификатора – класс Classifier.

Основные методы, которые видны другим классам – метод, запускающий дообучение Train, и метод, выполняющий собственно классификацию – GetHypotesis. Прежде всего, вспомогательные поля класса. Их назначение раскрыто в комментариях:

```
public class Classifier  
{  
    /// <summary>  
    /// Обратный индекс – структура, позволяющая быстро найти, сколько раз  
    /// тот или иной токен встретился для указанного класса.  
    /// </summary>  
    private List<Dictionary<string, int>> indices = new List<Dictionary<string,  
    int>>();  
  
    /// <summary>  
    /// Строковые имена классов.  
    /// </summary>
```

```
/// </summary>
private List<string> classes = new List<string>();

/// <summary>
/// Количество документов в обучающей выборке.
/// </summary>
private int numOfDocs = 0;

/// <summary>
/// Количество документов в обучающей выборке каждого класса.
/// </summary>
private List<int> numOfDocsPerClass = new List<int>();

/// <summary>
/// Количество токенов в обучающей выборке.
/// </summary>
private int numOfToks = 0;

/// <summary>
/// Количество токенов в обучающей выборке каждого класса.
/// </summary>
private List<int> numOfToksPerClass = new List<int>();
```

Метод, запускающий обучение (дообучение):

```
/// <summary>
/// Метод позволяет добавить в обучающую выборку определенного класса
/// новый текст.
/// </summary>
/// <param name="sample">Текст документа.</param>
/// <param name="className">Название класса. Если такого класса еще не
/// было, он будет создан.</param>
public void Train(string sample, string className)
{
    int classIndex = classes.IndexOf(className);
    // Если такого класса еще нет, создаем его.
    if (classIndex < 0)
    {
        classIndex = classes.Count;
        classes.Add(className);
        numOfDocsPerClass.Add(0);
```

```
        numOfToksPerClass.Add(0);
        indices.Add(new Dictionary<string, int>());
    }

    // Модифицируем статистику и обратный индекс.
    numOfDocs++;
    numOfDocsPerClass[classIndex]++;
    foreach (string token in Tokenize(sample.ToLowerInvariant()))
    {
        numOfToks++;
        if (!indices[classIndex].ContainsKey(token))
            indices[classIndex].Add(token, 0);

        indices[classIndex][token]++;
        numOfToksPerClass[classIndex]++;
    }
}
```

Текущая реализация для поиска индекса класса по его названию использует метод `List.IndexOf()`. Для большого количества классов он будет выполняться достаточно медленно, но для двух-трех классов он будет работать даже быстрее хэш-таблиц. К тому же, для большого количества классов, не очень хорошо будет работать уже сам Байесовский классификатор. Поэтому, нет ничего страшного в такой простой реализации поиска даже не для демо-примера.

В данном методе и методе, который классифицирует текст, используется вспомогательный метод `Tokenize`. Его реализация сравнительно проста и базируется на регулярных выражениях:

```
/// <summary>
/// Регулярное выражение, позволяющее разбить текст на слова.
/// </summary>
Regex wordsRegex = new Regex(@"\w+", RegexOptions.Compiled |
    RegexOptions.IgnoreCase);

/// <summary>
/// Метод разбивает текст на слова, в нем содержащиеся.
/// </summary>
/// <param name="sample">Исходный текст.</param>
/// <returns>Коллекция строковых токенов.</returns>
private IEnumerable<string> Tokenize(string sample)
```

```
{  
    foreach (Match match in wordsRegex.Matches(sample))  
        yield return match.Value;  
}
```

Из хитростей здесь можно разве что заметить оператор `yield return`, появившийся в 3-й версии языка C#. Он позволяет сократить код методов, возвращающих перечисления.

И, наконец, метод, классифицирующий текст.

```
/// <summary>  
/// Метод возвращает гипотезы о принадлежности переданного текста  
/// определенному классу, отсортированные по убыванию вероятности.  
/// </summary>  
/// <param name="opinion">Текст, который необходимо классифицировать.</param>  
/// <returns>Отсортированный список гипотез.</returns>  
public List<KeyValuePair<string, double>> GetHypotesis(string opinion)  
{  
    List<KeyValuePair<string, double>> result = new List<KeyValuePair<string,  
        double>>();  
  
    for (int classIndex = 0; classIndex < classes.Count; ++classIndex)  
    {  
        // Подсчитываем априорную вероятность.  
        double prob = numOfDocsPerClass[classIndex] / (double)numOfDocs;  
        // Накапливаем (мультилинируем) статистику для данного класса.  
        foreach (string token in Tokenize(opinion.ToLowerInvariant()))  
        {  
            int counter;  
            if (!indices[classIndex].ContainsKey(token))  
                counter = 0;  
            else  
                counter = indices[classIndex][token];  
  
            prob *= (counter + 1) /  
                (double)(numOfToksPerClass[classIndex] + numOfToks);  
        }  
        result.Add(new KeyValuePair<string, double>  
            (classes[classIndex], prob));  
    }  
}
```

```
// Сортируем результат по убыванию вероятности.  
result.Sort((y, x) => x.Value.CompareTo(y.Value));  
  
return result;  
}
```

Здесь мы видим алгоритм, описанный в предыдущей главе. В качестве априорной вероятности берется количество документов в классе при обучении, деленное на общее количество документов. Если априорные вероятности одинаковы, можно в данной строке присваивать просто единицу (или любое другое, существенно отличающееся от нуля положительное значение) – ведь нам важны относительные вероятности гипотез, а не их абсолютные значения. Подсчет вероятностей из количеств вхождений происходит «на лету». Это позволяет легко использовать класс Classifier в режиме «дообучения», когда неверно распознанный текст отправляется методу Train. Также следует обратить внимание на выражение «(counter + 1)». Добавляемая единица позволяет избежать потери точности в случае противоречивых данных, когда в оцениваемом тексте есть слова, отсутствующие в обеих категориях. Без единицы, произведение давало бы нам ноль и остальные попадания (даже в большом количестве) уже не имели бы значения.

Теперь запускаем программу и смотрим на результат:

Отзыв:

В целом фильм стоит воспринимать как приятную милую картину, которая заставляет умиляться, улыбаться, сопереживать героям. Советую к просмотру девушкам, мечтающим о принце на белом коне и верящим в любовь.

Гипотеза: +, вероятность 4,84045320064746E-69

Гипотеза: -, вероятность 7,04909421309603E-70

Отзыв:

Хреновый, скучный фильм. Пока я его досмотрел, поспал три раза. Не стоит терять время на это недоразумение.

Гипотеза: -, вероятность 3,11910811752617E-41

Гипотеза: +, вероятность 8,62700828239524E-42

Как видим, результат вполне правдоподобен. Конечно, не стоит рассчитывать, что данный классификатор покажет большую точность во всех случаях. Для этого в него необходимо внести некоторые изменения. Например, слова перед их обработкой, необходимо лемматизировать – приводить к основной форме. Или, хотя бы выполнить родственную операцию стемминга. Это особенно актуально для богатых в плане морфологии русского и украинского языков.

Далее, желательно в качестве параметров рассматривать также цепочки из нескольких слов. В самом алгоритме, для избежания потери точности на длинных текстах, нужно использовать вместо перемножения вероятностей (частот), сложение их логарифмов.

Таких улучшений можно предложить множество, однако их реализацию оставим читателю.

EM-алгоритм

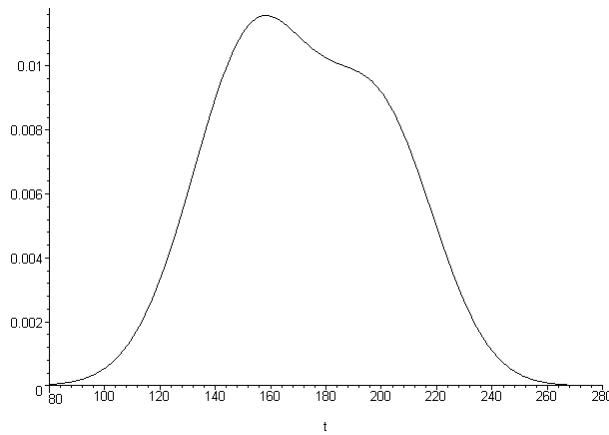
Результаты этого раздела имеют довольно сложную историю. По всей видимости, впервые ем-алгоритм был предложен М.И. Шлезингером [46], после чего неоднократно переоткрывался и на данный момент является одним из наиболее используемых методов разделения смесей (см., например, [75], [133], [137]).

Как уже отмечалось ранее, байесовский классификатор опирается на тот факт, что известны априорные вероятности $p(C_i)$ и условные вероятности $p(x|C_i)$. К сожалению, такую полную информацию имеем не всегда.

Рассмотрим один из методов решения этой задачи. В предположении, что известен только общий вид распределения вероятности, нужно оценить его параметры. Достаточно часто известно, что искомое распределение представляет собой линейную комбинацию, например, нормальных или биномиальных распределений. В этом случае говорят, что данное распределение представляет собой смесь нормальных, биномиальных (либо иных) распределений. В частности, если известно, что функция плотности вероятности каждого класса есть нормальное распределение $N(\mu_i, \sigma_i^2)$, то

нужно оценить параметры μ_i, σ_i . Для байесовского классификатора эти параметры известны.

Пусть $p(x, \theta_i)$ плотность вероятности того, что наблюдение получено из i -й компоненты смеси распределений.



Тогда

$$p(x, \theta_i) = p(x)P(\theta_i|x) = \omega_i p_i(x).$$

Апостериорную вероятность $P(\theta_i|x)$ того, что наблюдение x_j получено из i -й компоненты смеси, обозначим через $g_{i,j}$. Из формулы полной вероятности выпишем условие нормировки

$$\sum_{i=1}^k g_{i,j} = 1, j = 1, \dots, n.$$

Тогда при известных ω_i и $p_i(x_j)$ из теоремы Байеса легко получить

$$g_{i,j} = \frac{\omega_i p_i(x_j)}{\sum_{\nu=1}^k \omega_\nu p_\nu(x_j)}, i = 1, \dots, k, j = 1, \dots, n.$$

Функция

$$F(\Theta) = \prod_{j=1}^n p(x_j, \theta_i)$$

называется функцией правдоподобия от $\Theta = \{(\omega_i, \theta_i)\}_{i=1}^k$ по выборке $X = \{x_1, \dots, x_n\}$.

Традиционно через MLE (Maximum Likelihood Estimate) – метод максимального правдоподобия, называют метод, доставляющий

$$\hat{\Theta} = \arg \max_{\Theta} (F(\Theta)).$$

Заметим, что в силу монотонности алгоритма

$$\hat{\Theta} = \arg \max_{\Theta} (F(\Theta)) = \arg \max_{\Theta} (\ln(F(\Theta))).$$

Покажем, что при известных $g_{i,j}$, используя MLE, можно получить эффективный метод разделения параметров смеси. Запишем MLE в виде

$$\ln F(\Theta) = \ln \prod_{j=1}^n p(x_j, \theta_i) = \sum_{j=1}^n \ln \sum_{i=1}^k \omega_i p_i(x_j) \rightarrow \max_{\Theta} \text{при условии } \sum_{i=1}^k \omega_i = 1.$$

Используя метод множителей Лагранжа, выпишем лагранжиан этой задачи

$$L(\Theta, \Omega) = \sum_{j=1}^n \ln \sum_{i=1}^k \omega_i p_i(x_j) - \lambda \left(\sum_{i=1}^k \omega_i - 1 \right).$$

Приравнивая частные производные по неизвестным параметрам нулю, получаем

$$\frac{\partial}{\partial \omega_i} L(\Theta, \Omega) = \sum_{j=1}^n \frac{\omega_i p_i(x_j)}{\sum_{v=1}^k \omega_v p_v(x_j)} - \lambda = 0, i = 1, \dots, k..$$

Умножая обе части полученного соотношения на ω_i и суммируя их по всем i , получаем

$$\sum_{j=1}^n \sum_{i=1}^k \frac{\omega_i p_i(x_j)}{\sum_{v=1}^k \omega_v p_v(x_j)} = \lambda \cdot \sum_{i=1}^k \omega_i.$$

Замечая, что

$$\sum_{i=1}^k \frac{\omega_i p_i(x_j)}{\sum_{v=1}^k \omega_v p_v(x_j)} = 1 \text{ и } \sum_{i=1}^k \omega_i = 1,$$

получаем

$$\sum_{j=1}^n 1 = \lambda \text{ и } \lambda = n.$$

Таким образом имеем

$$\omega_i = \frac{1}{n} \sum_{j=1}^n \frac{\omega_i p_i(x_j)}{\sum_{v=1}^k \omega_v p_v(x_j)} = \frac{1}{n} \sum_{j=1}^n g_{i,j}, i = 1, \dots, k..$$

Теперь, замечая, что $p_i(x)$ зависит от θ_i , возьмем частную производную лагранжиана от θ_i и приравняем ее нулю

$$\frac{\partial}{\partial \theta_i} L(\Theta, \Omega) = \sum_{j=1}^n \frac{\omega_i}{\sum_{v=1}^k \omega_v p_v(x_j)} \frac{\partial}{\partial \theta_i} p_i(x_j) = 0, i = 1, \dots, k.$$

Каждое из слагаемых умножим и разделим на $p_i(x_j)$, соответственно

$$\frac{\partial}{\partial \theta_i} L(\Theta, \Omega) = \sum_{j=1}^n \frac{\omega_i p_i(x_j)}{\sum_{v=1}^k \omega_v p_v(x_j)} \frac{\partial}{\partial \theta_i} \ln p_i(x_j) = \sum_{j=1}^n g_{i,j} \frac{\partial}{\partial \theta_i} \ln p_i(x_j) = 0, i = 1, \dots, k.$$

Отсюда получаем

$$\frac{\partial}{\partial \theta_i} \sum_{j=1}^n g_{i,j} \ln p_i(x_j) = 0, i = 1, \dots, k,$$

что совпадает с необходимым условием максимума в задаче максимизации функции правдоподобия с весом

$$\sum_{j=1}^n g_{i,j} \ln p_i(x_j) \rightarrow \max_{\Theta}, i = 1, \dots, k.$$

Таким образом ЕМ-алгоритм с фиксированным числом компонентов смеси можно записать в следующем виде.

Пусть $X = \{x_1, \dots, x_n\}$ выборка наблюдений, k -число компонентов смеси, $\Theta = \{(\omega_i, \theta_i)\}_{i=1}^k$ - начальное приближение параметров смеси, и ε число, определяющее остановку алгоритма.

ЕМ-алгоритм состоит из последовательного применения двух шагов.

E-шаг (expectation)

$$g_{i,j}^0 = g_{i,j};$$

$$g_{i,j} = \frac{\omega_i p_i(x_j)}{\sum_{v=1}^k \omega_v p_v(x_j)}, i = 1, \dots, k, j = 1, \dots, n.$$

$$\delta = \max \left\{ |g_{i,j}^0 - g_{i,j}| \right\}$$

M-шаг (maximization)

$$\sum_{j=1}^n g_{i,j} \ln p_i(x_j) \rightarrow \max_{\Theta}, i = 1, \dots, k;$$

$$\omega_i = \frac{1}{n} \sum_{j=1}^n g_{i,j}, i = 1, \dots, k.$$

Если $\delta > \varepsilon$, то переходим к Е-шагу, если $\delta \leq \varepsilon$, то возвращаем найденные параметры смеси $\Theta = \{(\omega_i, \theta_i)\}_{i=1}^k$.

Заметим, что если известен вид функции плотности, то задачу МЛЕ можно выписать в явном виде. Рассмотрим случай, когда известно, что смесь состоит из нормальных распределений $N(\mu_i, \sigma_i^2)$, где $i = 1, \dots, k$. Тогда задача

$$\sum_{j=1}^n g_{i,j} \ln p_i(x_j) \rightarrow \max_{\Theta}, i = 1, \dots, k$$

запишется в виде

$$\sum_{j=1}^n g_{i,j} \ln \left(\frac{1}{\sigma_i \sqrt{2\pi}} \exp \left(-\frac{(x_j - \mu_i)^2}{2\sigma_i^2} \right) \right) \rightarrow \max_{\Theta}, i = 1, \dots, k,$$

или, что то же,

$$G(\{\mu_i, \sigma_i\}_{i=1}^k) = \sum_{j=1}^n g_{i,j} \left(-\ln(\sigma_i \sqrt{2\pi}) - \frac{(x_j - \mu_i)^2}{2\sigma_i^2} \right) \rightarrow \max_{\Theta}, i = 1, \dots, k.$$

Приравнивая частные производные нулю, получаем

$$\frac{\partial}{\partial \mu_i} G(\{\mu_\nu, \sigma_\nu\}_{\nu=1}^k) = -\sum_{j=1}^n g_{i,j} \frac{(x_j - \mu_i)}{\sigma_i^2} = 0,$$

отсюда

$$\mu_i = \frac{\sum_{j=1}^n g_{i,j} x_j}{\sum_{j=1}^n g_{i,j}}.$$

Аналогично,

$$\frac{\partial}{\partial \sigma_i} G(\{\mu_\nu, \sigma_\nu\}_{\nu=1}^k) = -\sum_{j=1}^n g_{i,j} \left(\frac{1}{\sigma_i} - \frac{(x_j - \mu_i)^2}{\sigma_i^3} \right) = -\sum_{j=1}^n g_{i,j} \left(\frac{\sigma_i^2 - (x_j - \mu_i)^2}{\sigma_i^3} \right) = 0,$$

таким образом, отсюда получаем

$$\sigma_i^2 = \frac{\sum_{j=1}^n g_{i,j} (x_j - \mu_i)^2}{\sum_{j=1}^n g_{i,j}},$$

что позволяет получить параметры распределения в явном виде.

Заметим, что для многомерного случая нормальное распределение описывается следующим соотношением

$$p(x|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{n/2} |\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i)\right),$$

где Σ - корреляционная матрица. Использование МЛЕ позволяет оценить параметры распределения следующим образом

$$\mu_i = \frac{\sum_{j=1}^n g_{i,j} x_j}{\sum_{j=1}^n g_{i,j}}, \quad \Sigma_i = \frac{\sum_{j=1}^n g_{i,j} (x_j - \mu_i)(x_j - \mu_i)^T}{\sum_{j=1}^n g_{i,j}}.$$

В качестве иллюстрации ЕМ-алгоритма приведем пример, приведенный О.Векслер ([133]).

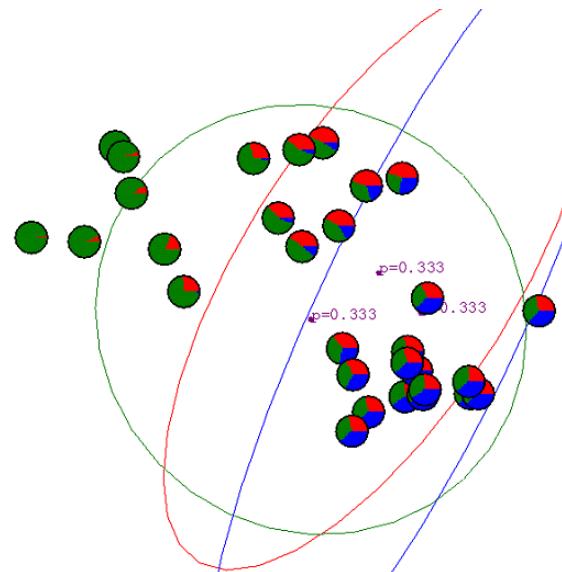


Рис. 6.1. Начальное приближение.

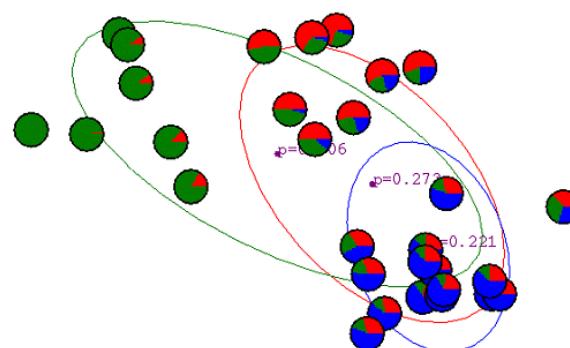


Рис. 6.2. Результат первой итерации.

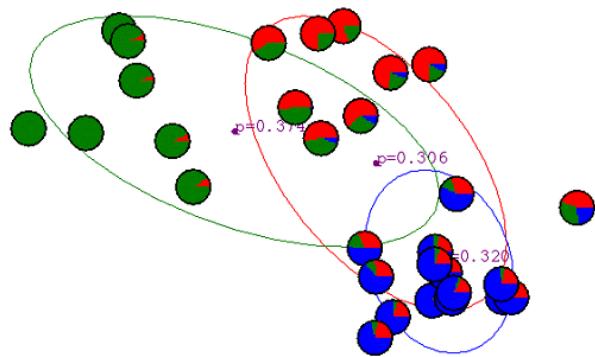


Рис. 6.3. Результат второй итерации.

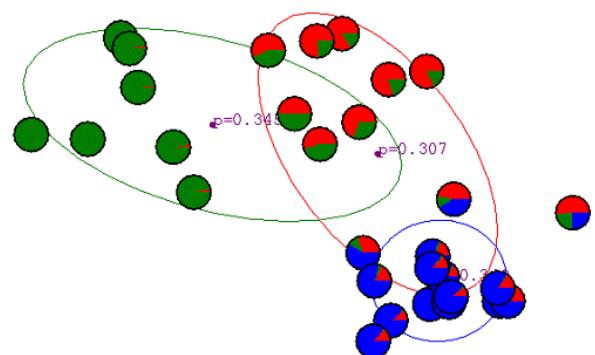


Рис. 6.4. Что получилось после третьей итерации.

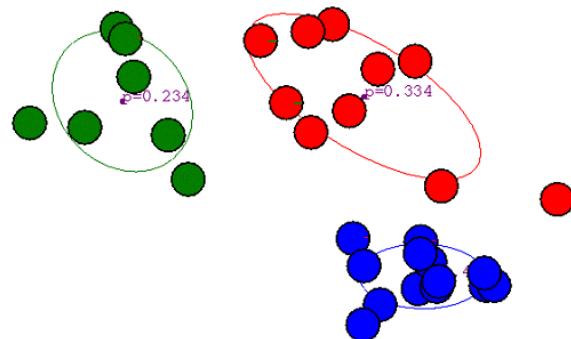


Рис. 6.5. Результат применения двадцати итераций.

Линейный дискриминантный анализ

Теперь, когда мы умеем оценивать параметры смеси нормальных распределений, рассмотрим задачу классификации.

Пусть даны классы $c_i (i = 1, 2, \dots, k)$, функция $g_i(x)$, такая, что если

$$g_i(x) > g_j(x) \quad \forall i \neq j, \text{ то } x \in c_i,$$

то такая функция называется дискриминантной или разделяющей классы.

В качестве дискриминантной функции совершенно естественно использовать априорную вероятность попадания в класс c_i при выпадении события x

$$g_i(x) = P(c_i | x).$$

Тогда в соответствии с теоремой Байеса

$$g_i(x) = \frac{P(x | c_i)P(c_i)}{P(x)}.$$

Так как $P(x)$ не зависит от классов, то в качестве дискриминантной функции можно взять

$$g_i(x) = P(x | c_i)P(c_i).$$

В силу монотонности логарифма можно в качестве дискриминантной функции использовать эквивалентную

$$g_i(x) = \ln P(x | c_i) + \ln P(c_i).$$

Тогда, если данные класса c_i распределены по нормальному закону $N(\mu_i, \Sigma_i)$

$$P(x | c_i) = \frac{1}{(2\pi)^{n/2} |\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i)\right),$$

то

$$g_i(x) = -\frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) - \frac{n}{2} \ln(2\pi) - \frac{1}{2} \ln |\Sigma_i| + \ln P(c_i).$$

Так как $\frac{n}{2} \ln(2\pi)$ постоянная, то дискриминантную функцию можно записать в

эквивалентном виде

$$g_i(x) = -\frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) - \frac{1}{2} \ln |\Sigma_i| + \ln P(c_i). \quad (6.3)$$

Отсюда, перемножая, получаем

$$\begin{aligned} g_i(x) &= -\frac{1}{2} \left(x^T \Sigma_i^{-1} x - 2\mu_i^T \Sigma_i^{-1} x + \mu_i^T \Sigma_i^{-1} \mu_i \right) - \frac{1}{2} \ln |\Sigma_i^{-1}| + \ln P(c_i) = \\ &= x^T \left(-\frac{1}{2} \Sigma_i^{-1} \right) x + \mu_i^T \Sigma_i^{-1} x + \left(-\frac{1}{2} \mu_i^T \Sigma_i^{-1} \mu_i - \frac{1}{2} \ln |\Sigma_i^{-1}| + \ln P(c_i) \right). \end{aligned} \quad (6.4)$$

Замечая, что

$$x^T W x = \sum_{i=1}^n \sum_{j=1}^n w_{i,j} x_i x_j,$$

получаем, что квадратичная функция

$$g_i(x) = x^T A_i x + B_i x + D_i,$$

где

$$A_i = -\frac{1}{2} \Sigma_i^{-1}, B_i = \mu_i^T \Sigma_i^{-1}, D_i = -\frac{1}{2} \mu_i^T \Sigma_i^{-1} \mu_i - \frac{1}{2} \ln |\Sigma_i^{-1}| + \ln P(c_i).$$

Таким образом, дискриминантная функция будет состоять из дуг кривых второго порядка (эллипсов, парабол и пр.).

Приведем несколько частных случаев построения дискриминантной функции.

Пусть

$$\Sigma_i = \sigma^2 I = \begin{pmatrix} \sigma^2 & 0 & \cdots & 0 \\ 0 & \sigma^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma^2 \end{pmatrix} \quad (i=1,\dots,k),$$

то есть случайные величины (X_1, X_2, \dots, X_n) независимы с разным математическим ожиданием, но с одной и той же дисперсией. В этом случае

$$\Sigma_i^{-1} = \frac{1}{\sigma^2} I = \begin{pmatrix} \frac{1}{\sigma^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma^2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sigma^2} \end{pmatrix} \text{ и } |\Sigma_i| = \sigma^{2n}.$$

Замечая, что при этом $\frac{1}{2} \ln |\Sigma_i^{-1}|$ является постоянной, то из (6.3) получаем следующую дискриминантную функцию

$$g_i(x) = -\frac{1}{2\sigma^2} (x - \mu_i)^T (x - \mu_i) + \ln P(c_i) = -\frac{1}{2\sigma^2} (x^T x - \mu_i^T x - x^T \mu_i + \mu_i^T \mu_i) + \ln P(c_i),$$

а так как $x^T x$ не зависит от классов, то получаем

$$g_i(x) = -\frac{1}{2\sigma^2}(-2\mu_i^T x + \mu_i^T \mu_i) + \ln P(c_i) = a_i x + b_i, \quad (6.5)$$

где

$$a_i = \frac{\mu_i^T}{\sigma^2} \text{ и } b_i = \ln P(c_i) - \frac{\mu_i^T \mu_i}{2\sigma^2}.$$

Таким образом, в этом случае дискриминантная функция является линейной, то есть, для двух переменных это прямая, для трех – плоскость, а в общем случае – гиперплоскость.

Заметим, что если при этом все $P(c_i) = \frac{1}{k}, i = 1, 2, \dots, k$, то линейные дискриминантные функции приводят к разбиению Вороного.

Пример 6.1. Пусть даны три класса, данные которых описываются нормальным распределением с параметрами

$$\mu_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \mu_2 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \mu_3 = \begin{pmatrix} 2 \\ -1 \end{pmatrix} \text{ и } \Sigma_i = \begin{pmatrix} 4 & 0 \\ 0 & 4 \end{pmatrix} (i = 1, 2, 3).$$

Кроме того, пусть даны априорные вероятности выпадения классов

$$P(c_1) = \frac{5}{12}, P(c_2) = \frac{1}{4}, P(c_3) = \frac{1}{3}.$$

В соответствии с (5.3), выпишем дискриминантные функции

$$g_1(x) = \frac{(0,0)}{4} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \left(\ln \frac{5}{12} - \frac{0}{8} \right) = -0.8754683,$$

$$g_2(x) = \frac{(2,2)}{4} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \left(\ln \frac{1}{4} - \frac{8}{8} \right) = -2.3862943 + \frac{1}{2}(x_1 + x_2),$$

$$g_3(x) = \frac{(2,-1)}{4} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \left(\ln \frac{1}{3} - \frac{5}{8} \right) = -1.7261229 + \frac{1}{4}(2x_1 - x_2).$$

Тогда функция, разделяющая классы c_1 и c_2 будет равна

$$g_1(x) = g_2(x) \Rightarrow -0.6931471806 = -2.098612289 + \frac{1}{2}(x_1 + x_2) \Rightarrow x_1 + x_2 - 3.0217 = 0.$$

Функция, разделяющая классы c_1 и c_3 будет иметь вид

$$g_1(x) = g_3(x) \Rightarrow -0.6931471806 = -2.416759469 + \frac{1}{4}(2x_1 - x_2) \Rightarrow 2x_1 - x_2 - 3.39257 = 0.$$

И, наконец,

$$g_2(x) = g_3(x) \Rightarrow -2.098612289 + \frac{1}{2}(x_1 + x_2) = -2.416759469 + \frac{1}{4}(2x_1 - x_2) \Rightarrow x_2 = 0.883576.$$

Заметим, что $g_1(x)=g_2(x)=g_3(x)$ имеет решение $x_1=2.138075$, $x_2=0.883576$.

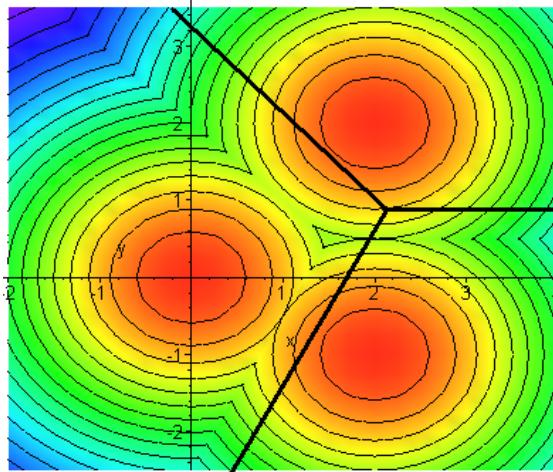


Рис. 6.6. Пример линейных дискриминантных функций.

Пусть $\Sigma_i = \Sigma$ ($i=1,\dots,k$), тогда величина $\frac{1}{2}\ln|\Sigma_i^{-1}|$ не зависит от класса, и, из (6.3)

получаем следующую дискриминантную функцию

$$g_i(x) = -\frac{1}{2\sigma^2}(x - \mu_i)^T(x - \mu_i) + \ln P(c_i) = -\frac{1}{2\sigma^2}(x^T x - \mu_i^T x - x^T \mu_i + \mu_i^T \mu_i) + \ln P(c_i),$$

а так как $x^T \Sigma^{-1} x$ не зависит от классов, то получаем

$$g_i(x) = -\frac{1}{2}(-2\mu_i^T \Sigma^{-1} x + \mu_i^T \Sigma^{-1} \mu_i) + \ln P(c_i) = a_i x + b_i, \quad (6.6)$$

где

$$a_i = \mu_i^T \Sigma^{-1} \text{ и } b_i = \ln P(c_i) - \frac{\mu_i^T \Sigma^{-1} \mu_i}{2},$$

Следовательно, и в этом случае дискриминантная функция является линейной.

Пример 6.2. Пусть даны три класса, данные которых описываются нормальным распределением с параметрами

$$\mu_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \mu_2 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \quad \mu_3 = \begin{pmatrix} 2 \\ -1 \end{pmatrix} \text{ и } \Sigma_i = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} (i=1,2,3),$$

и априорные вероятности выпадения классов

$$P(c_1) = \frac{5}{12}, \quad P(c_2) = \frac{1}{4}, \quad P(c_3) = \frac{1}{3}.$$

В соответствии с (6.4), выпишем дискриминантные функции

$$g_1(x) = -0.8754683,$$

$$g_2(x) = -5.3862943 + 2(x_1 + x_2),$$

$$g_3(x) = x_1 - 2.09861229.$$

Тогда функция, разделяющая классы c_1 и c_2 будет равна

$$g_1(x) = g_2(x) \Rightarrow x_1 + x_2 - 2.2554128 = 0.$$

Функция, разделяющая классы c_1 и c_3 будет иметь вид

$$g_1(x) = g_3(x) \Rightarrow x_1 - 1.2231435 = 0.$$

И, наконец,

$$g_2(x) = g_3(x) \Rightarrow x_1 + 2x_2 = 3.28768.$$

При этом $g_1(x)=g_2(x)=g_3(x)$ имеет решение $x_1=1.22314255$, $x_2=1.03226926$.

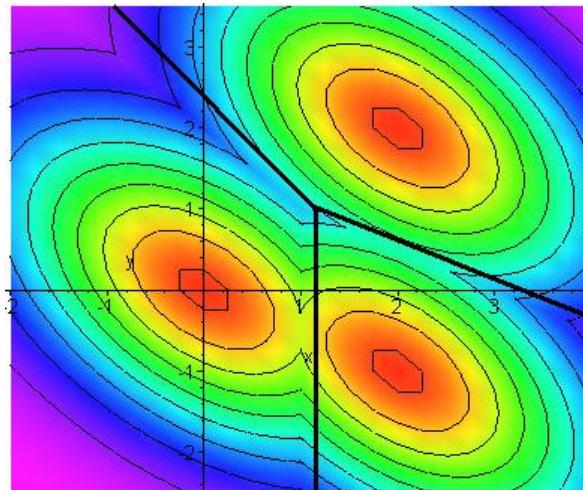


Рис. 6.7. Пример линейных дискриминантных функций.

Пример 6.3. Пусть даны три класса, данные которых описываются нормальным распределением с параметрами

$$\mu_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \mu_2 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \quad \mu_3 = \begin{pmatrix} 2 \\ -1 \end{pmatrix} \text{ и } \Sigma_1 = \begin{pmatrix} 2 & -1 \\ -1 & 1 \end{pmatrix}, \Sigma_2 = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}, \Sigma_3 = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix},$$

и априорные вероятности выпадения классов

$$P(c_1) = \frac{5}{12}, P(c_2) = \frac{1}{4}, P(c_3) = \frac{1}{3}.$$

В соответствии с (6.4), выпишем дискриминантные функции

$$g_1(x) = -\frac{1}{2}x_1^2 - x_1 x_2 - x_2^2 - 0.875468,$$

$$g_2(x) = -\frac{1}{4}x_1^2 - \frac{1}{4}x_2^2 + x_1 + x_2 - 2.693147,$$

$$g_3(x) = -\frac{1}{3}x_1^2 + \frac{1}{3}x_1 x_2 - \frac{1}{3}x_2^2 + \frac{8}{5}x_1 - \frac{4}{3}x_2 - 2.882639.$$

Тогда функция, разделяющая классы c_1 и c_2 будет равна

$$g_1(x) = g_2(x) \Rightarrow -\frac{1}{4}x_1^2 - x_1 x_2 - \frac{3}{4}x_2^2 - x_1 - x_2 + 1.81767 = 0.$$

Функция, разделяющая классы c_1 и c_3 будет иметь вид

$$g_1(x) = g_3(x) \Rightarrow -\frac{1}{6}x_1^2 - \frac{4}{3}x_1 x_2 - \frac{2}{3}x_2^2 - \frac{5}{3}x_1 + \frac{4}{3}x_2 + 2.00717 = 0.$$

И, наконец,

$$g_2(x) = g_3(x) \Rightarrow \frac{1}{12}x_1^2 + \frac{1}{12}x_2^2 - \frac{1}{3}x_1 x_2 + \frac{7}{3}x_2 - \frac{1}{6}x_1 + 0.18949 = 0.$$

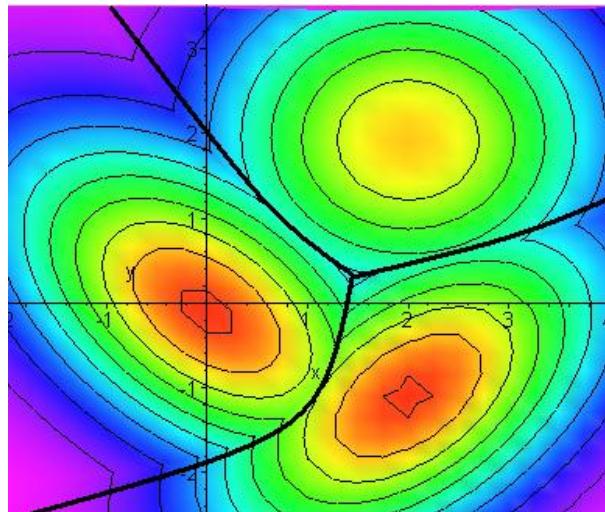


Рис. 6.7. Пример квадратичных дискриминантных функций

7. Использование генетических алгоритмов для построения векторного классификатора

В данном параграфе мы рассмотрим метод классификации, основанный не на понятии расстояния между элементами, а на критерии близости, построенном на вычислении косинуса угла между двумя векторами. Все построения приведем на примере классификации текстов ([49], [126]).

Первый шаг состоит в первичной обработке данных – построение множеств статистики для имеющихся классов. Для построения множества статистики последовательно обрабатываются все множества словоформ $b^\nu, \nu = 0, \dots, M-1$, принадлежащие одному классу $B = \{b^\nu\}_{\nu=0}^{M-1}$. По множеству словоформ каждого обрабатываемого текста b^ν строится множество уникальных (неповторяющихся) словоформ и их счетчики – $(w_i^\nu, n_i^\nu) (i = 0, \dots, N^\nu - 1)$. Здесь N^ν – количество уникальных словоформ для текста b^ν . После этого данные для каждого множества отдельно нормируются

$$\bar{n}_i^\nu = \frac{n_i^\nu}{\sqrt{\sum_{j=0}^{N^\nu-1} (n_j^\nu)^2}} (i = 0, \dots, N^\nu - 1).$$

Затем, упорядочиваем все слова для каждого документа в одном и том же порядке (сам порядок слов не существенен, главное, чтобы слова в каждой из структур $(w_i^\nu, n_i^\nu) (i = 0, \dots, N^\nu - 1)$ шли в одном и том же порядке) и находим сумму всех векторов $n_i(B) = \sum_{j=0}^{M-1} \bar{n}_i^\nu (i = 0, \dots, N(B))$ (где $N(B)$ – количество уникальных словоформ для класса B в целом) и нормируем ее единицей

$$\bar{n}_i(B) = \frac{n_i(B)}{\sqrt{\sum_{j=0}^{N(B)} (n_j(B))^2}}.$$

Для полученной центральной точки класса формируем множество статистики, записывая в него значения $(w_i(B), \bar{n}_i(B)) (i = 0, \dots, N(B))$.

Для построения центрального вектора классов $\{B^\mu\}_{\mu=0}^{K-1}$, где каждый класс B^μ описывается своим центральным вектором $(w_i(B^\mu), \bar{n}_i(B^\mu))_{i=0, \dots, N(B^\mu)}$, нужно найти их сумму, просуммировав все координаты из всех суммируемых векторов для каждого значения словоформы, то есть для словоформы ω получаем координату

$$n(\omega) = \sum_{\mu=0}^{K-1} \left\{ \bar{n}_i(B^\mu) \middle| w_i(B^\mu) = \omega, i = 0, \dots, N(B^\mu) \right\},$$

следовательно, нужно составить список уникальных словоформ по всем центральным векторам классов $\{B^\mu\}_{\mu=0}^{K-1}$ и просуммировать их координаты. Результатом будет множество, состоящее из уникальных (неповторяющихся) словоформ и их координат

$$(w_i(\{B^\mu\}_{\mu=0}^{K-1}), n_i(\{B^\mu\}_{\mu=0}^{K-1}))_{i=0, \dots, N(\{B^\mu\}_{\mu=0}^{K-1})},$$

где $N(\{B^\mu\}_{\mu=0}^{K-1})$ - количество уникальных словоформ множества классов $\{B^\mu\}_{\mu=0}^{K-1}$. Остается пронормировать полученные координаты

$$\bar{n}_i(\{B^\mu\}_{\mu=0}^{K-1}) = \frac{n_i(\{B^\mu\}_{\mu=0}^{K-1})}{\sqrt{\sum_{j=0}^{N(\{B^\mu\}_{\mu=0}^{K-1})} (n_j(\{B^\mu\}_{\mu=0}^{K-1}))^2}},$$

и, полученный вектор $(w_i(\{B^\mu\}_{\mu=0}^{K-1}), \hat{n}_i(\{B^\mu\}_{\mu=0}^{K-1}))_{i=0, \dots, N(\{B^\mu\}_{\mu=0}^{K-1})}$ будет центральным вектором множества $\{B^\mu\}_{\mu=0}^{K-1}$.

Идеально сформированной классификацией векторного метода является такой набор классов $\{B^\mu\}_{\mu=0}^{K-1}$, для которого выполняется следующее условие:

$\forall b \in B^\mu, \mu = 0, \dots, K-1$ имеет место соотношение

$$\langle \bar{n}(b), \bar{n}(B^\mu) \rangle < \langle \bar{n}(b), \bar{n}(B^\nu) \rangle, \nu \neq \mu. \quad (7.1)$$

Рассмотрим вектор Λ (управление) размерностью $N(B^\mu)$, координаты которого принимают только одно из двух допустимых значений

$$\lambda_i = \begin{cases} 0, \\ 1. \end{cases}$$

Через Λb обозначим прямое произведение векторов Λ и b , то есть

$$\Lambda b = (\lambda_0 \bar{n}_0(b), \lambda_1 \bar{n}_1(b), \dots, \lambda_{N(B^\mu)} \bar{n}_{N(B^\mu)}(b)).$$

Управление Λ будем называть допустимым на классе $B^\mu = \{b^k\}_{k=0}^{M-1}$, если выполняется условие

$$\left\langle \overline{\Lambda}^n(b^k), \overline{\Lambda}^n(B^\mu) \right\rangle < \left\langle \overline{\Lambda}^n(b^k), \overline{n}(B^\nu) \right\rangle, \nu \neq \mu, k = 0, 1, \dots, M-1. \quad (7.2)$$

Допустимое управление, для которого имеет место это соотношение и при этом

$$\sum_{k=0}^{M-1} (\Lambda b^k)^2 \rightarrow \max, \text{ называется оптимальным.}$$

Если для $\nu \neq \mu$ множество допустимых управлений вырождено, то класс $B^\mu = \{b^k\}_{k=0}^{M-1}$ определен некорректно, то есть он неразделим с классом B^ν .

Задача нахождения оптимального управления классическими методами достаточно сложна, поэтому для ее решения мы применим генетические алгоритмы.

В нашем случае используется одноточечный кроссинговер (Single-point crossover), который моделируется следующим образом - пусть имеются две родительские особи с хромосомами $X = \{x_i, i \in \{0, \dots, L\}\}$ и $Y = \{y_i, i \in \{0, \dots, L\}\}$. Случайным образом определяется точка внутри хромосомы (точка разрыва), в которой обе хромосомы делятся на две части и обмениваются ими. После процесса воспроизведения происходят мутации (mutation). Это достигается за счет того, что изменяется случайно выбранный ген в хромосоме.

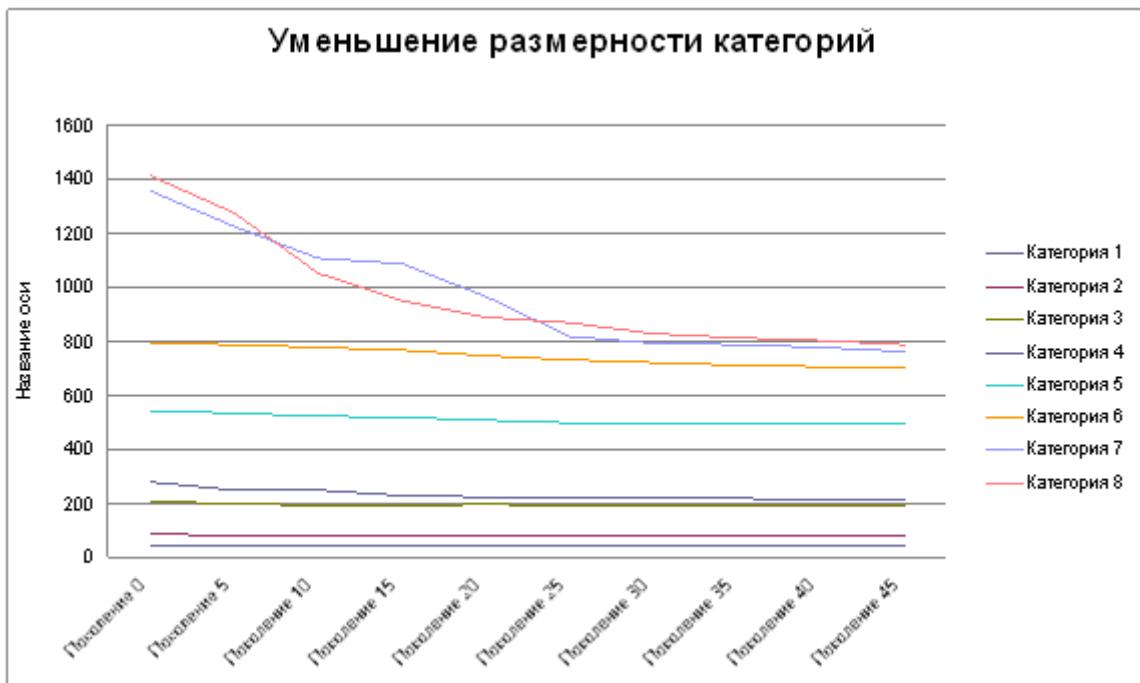


Рис. 7.1. Диаграмма уменьшения размерности категорий при использовании генетических алгоритмов.

Для создания новой популяции нами использован элитарный отбор (Elite selection). Создается промежуточная популяция, которая включает в себя как родителей, так и их потомков. Члены этой популяции оцениваются, а за тем из них выбираются N самых лучших (пригодных), которые и войдут в следующее поколение.

Результат применения генетического алгоритма к задаче сокращения размерности класса, приведен на рис. 7.1.

Заметим, что векторный метод в качестве критерия качества использует величину скалярного произведения ортов, таким образом, класс единичных векторов (документов) ограничен на сфере окружностью с центром в конце центрального вектора класса. Так как срезы сферы по окружности не могут плотно упаковать всю поверхность единичной сферы, то появляется множество точек (ортов), которые принципиально не могут попасть ни в один класс. Таким образом, возникает необходимость разбить множество точек на единичной сфере, так, чтобы элементы этого разбиения плотно упаковывали всю поверхность единичной сферы, то есть позволяли однозначно классифицировать любой документ.

Использование диаграмм Вороного в задаче классификации текстов

Для любого центра системы $\{A\}$ можно указать область пространства, все точки которой ближе к данному центру, чем к любому другому центру системы. Такая область называется многогранником Вороного или областью Вороного. К многограннику Вороного обычно относят и его поверхность. В трехмерном пространстве область Вороного любого центра i системы $\{A\}$ есть выпуклый многогранник, в двумерном — выпуклый многоугольник. Формально многоугольники Вороного T_i определяются следующим образом:

$$T_i = \left\{ x \in R^2 : d(x, x_i) < d(x, x_j) \forall j \neq i \right\}.$$

Построение аппроксимации опирается на фундаментальное свойство для произвольно заданных n точек множества S на плоскости. Для любого узла из n на плоскости существует множество натуральных соседей N . Понятие натуральных соседей тесно связано с разбиением области ячейками Вороного. Для непустой ячейки

Вороного $V(R), R \subset S$ натуральные соседи для $r \in R$ - вершины треугольников Делоне, инцидентных к $V(R)$.

Двумерный многогранник Вороного показан на рисунке 7.2. Плоскости Вороного, которые породили грани у данного многогранника, называются образующими плоскостями Вороного, а соответствующие центры системы — геометрическими соседями данного центра i . Среди геометрических соседей различают основные (естественные) и не основные. Для первых - середина отрезка, соединяющая его с центральным узлом, лежит на грани многогранника Вороного. Для вторых — вне грани и, следовательно, вне самого многогранника.

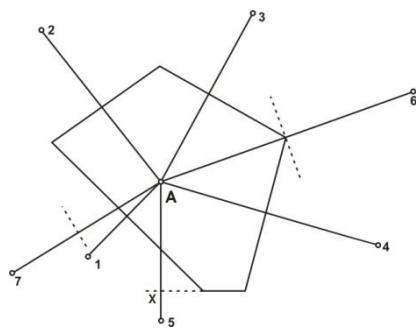


Рис. 7.2. Многогранник Вороного для центра i двумерной системы.

Многогранники Вороного, построенные для каждого центра системы $\{A\}$, дают мозаику многогранников - разбиение Вороного (рисунок 6.3). Многогранники Вороного системы $\{A\}$ не входят друг в друга и заполняют пространство, будучи смежными по целым граням. Разбиение пространства на многогранники Вороного однозначно определяется системой $\{A\}$ и, наоборот, однозначно ее определяет.

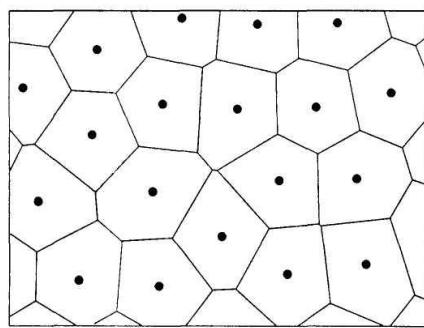


Рис. 7.3. Диаграмма Вороного на плоскости.

Используя конструкцию диаграмм Вороного применительно к точкам на многомерной единичной сфере, получаем разбиение всех ортов документов на естествен-

ные классы. Границами классов будут являться гиперплоскости, разделяющие сферические многогранники Вороного. При этом к одному классу будут относиться точки на единичной сфере (концы ортов документов), которые по отношению ко всем гиперплоскостям, ограничивающим данный класс, лежат с одной ее стороны, что и центральный вектор этого класса.

Проверка существующей классификации на корректность

Пусть проверяются на корректность разбиения классы документов C_ν и C_μ .

Для соответствующих орт (центральных векторов) \hat{C}_ν , \hat{C}_μ строим вектор разности

$$\vec{\Delta}_{\nu,\mu} = \hat{C}_\nu - \hat{C}_\mu = \{\hat{n}^\nu(w_i) - \hat{n}^\mu(w_i)\}$$

и вектор суммы

$$\vec{\Xi}_{\nu,\mu} = \frac{1}{2}(\hat{C}_\nu + \hat{C}_\mu) = \frac{1}{2}\{\hat{n}^\nu(w_i) + \hat{n}^\mu(w_i)\}.$$

Конец вектора полусуммы численно совпадает с координатами этого вектора.

Обозначим его через $\Xi_{\nu,\mu}$. Проведем через точку $\Sigma_{\nu,\mu}$ плоскость с нормальным вектором $\vec{\Delta}_{\nu,\mu}$

$$\Omega_{\nu,\mu} = \langle \vec{\Delta}_{\nu,\mu} \cdot (P - \Xi_{\nu,\mu}) \rangle = 0. \quad (7.3)$$

Эта плоскость разделяет классы. Для того, чтобы метод корректно разделял классы, нужно, чтобы все точки (документы) одного класса находились с одной стороны плоскости, то есть, если $b \in C_\nu$, то

$$\langle \vec{\Delta}_{\nu,\mu} \cdot (\hat{C}_\nu - \Xi_{\nu,\mu}) \rangle \langle \vec{\Delta}_{\nu,\mu} \cdot (\hat{b} - \Xi_{\nu,\mu}) \rangle \geq 0.$$

Точки, в которых это условие не выполняется нужно рассмотреть на вопрос принадлежности к категории C_μ .

Возможна ситуация, когда категории имеют непустое пересечение. Например, категория «АВТОР» содержит документы по математике (этого автора) и категория «НАУКА» содержит документы по математике того же автора. В этом случае нужно выделить непустое пересечение этих категорий и, в дальнейшем, либо эту категорию локализовать, либо проводить адресацию в обе категории.

Для решения этой проблемы предлагается следующий метод. Рассмотрим категории C_ν и C_μ . Разделим их плоскостью (3), и все точки, лежащие с одной стороны, соберем в новые две категории C_ν^* и C_μ^* .

Пусть

$$d(B, \Omega_{\nu,\mu}) = \frac{|\langle \vec{\Delta}_{\nu,\mu} \cdot (B - \Xi_{\nu,\mu}) \rangle|}{|\vec{\Delta}_{\nu,\mu}|}$$

расстояние от точки $B = \{b_i\}$ до плоскости $\Omega_{\nu,\mu}$.

Если выполняется условие (то есть, после отсечения данных обе категории отодвигаются друг от друга)

$$\begin{cases} d(C_\nu^*, \Omega_{\nu,\mu}) - d(C_\nu, \Omega_{\nu,\mu}) > 0, \\ d(C_\mu^*, \Omega_{\nu,\mu}) - d(C_\mu, \Omega_{\nu,\mu}) > 0, \end{cases}$$

то категории C_ν и C_μ имеют непустое пересечение \tilde{C} , которое можно определить следующим образом, $b \in \tilde{C}$ если $b \in C_\nu$ и при этом

$$\langle \vec{\Delta}_{\nu,\mu} \cdot (\hat{C}_\nu - \Xi_{\nu,\mu}) \rangle \langle \vec{\Delta}_{\nu,\mu} \cdot (\hat{b} - \Xi_{\nu,\mu}) \rangle < 0,$$

или, если $b \in C_\mu$, то

$$\langle \vec{\Delta}_{\nu,\mu} \cdot (\hat{C}_\mu - \Xi_{\nu,\mu}) \rangle \langle \vec{\Delta}_{\nu,\mu} \cdot (\hat{b} - \Xi_{\nu,\mu}) \rangle < 0.$$

Естественно, что для метода, построенного на диаграммах Вороного также актуальна задача сокращения размерности классов. Для этой цели, так же, как и в предыдущем случае, использовались генетические алгоритмы.

Сравнительный анализ применения различных методов классификации к тестовой базе документов Reuters [117] приведен на следующих диаграммах.

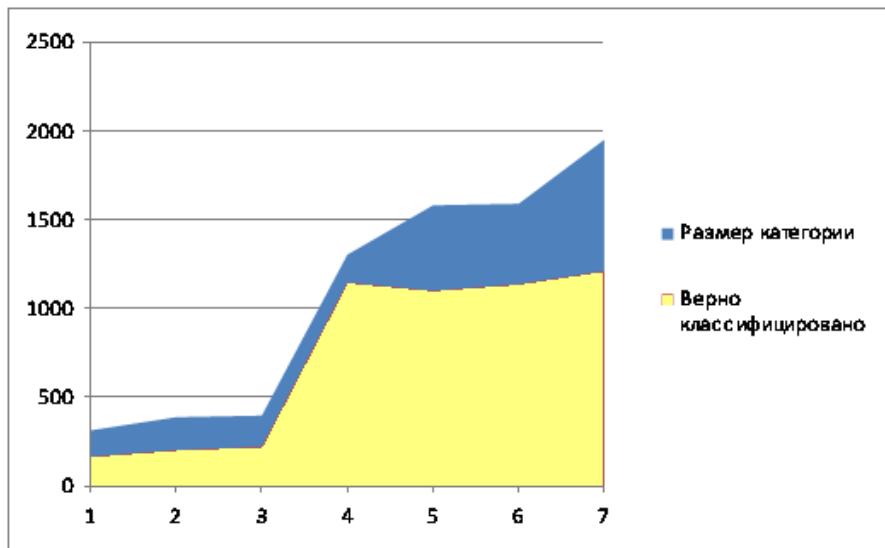


Рис. 7.4. Результат применения алгоритма Байеса.

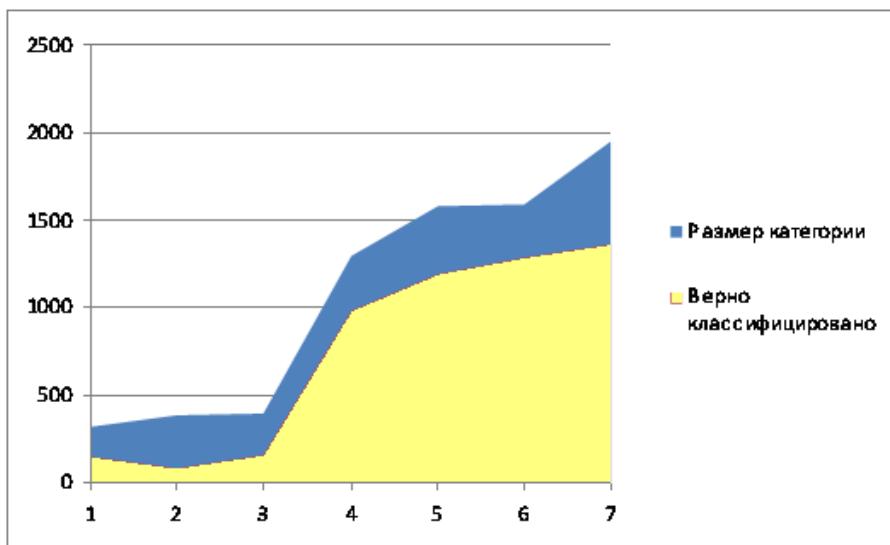


Рис. 7.5. Результат применения векторного алгоритма.

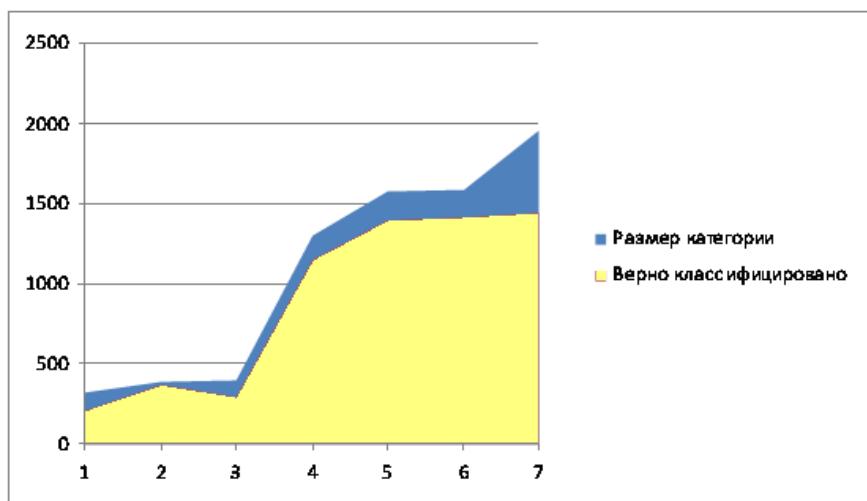


Рис. 7.6. Результат применения алгоритма основанного на диаграммах Вороного.

Таким образом, для тестовой базы Reuters при условии попадания в класс не менее 90% документов, удалось сократить размерность классов от 10% до 50%.

8. Метод опорных векторов (SVM — support vector machines)

Метод опорных векторов (SVM — support vector machines) — это набор алгоритмов «обучения с учителем», которые достаточно эффективно используются в задачах классификации. В их основе лежат работы В.Н. Вапника и А.Я. Червоненкиса (см. [7], [8],[132]). Этот метод принадлежит к семейству линейных классификаторов ([128]).

В предыдущем разделе нами были рассмотрены самые простые дискриминантные функции, реализующие линейный классификатор. Его можно записать в виде $g(x) = w^t x + w_0$, где

$$g(x) > 0 \Rightarrow x \in \text{Class}[1] \text{ и } g(x) < 0 \Rightarrow x \in \text{Class}[2].$$

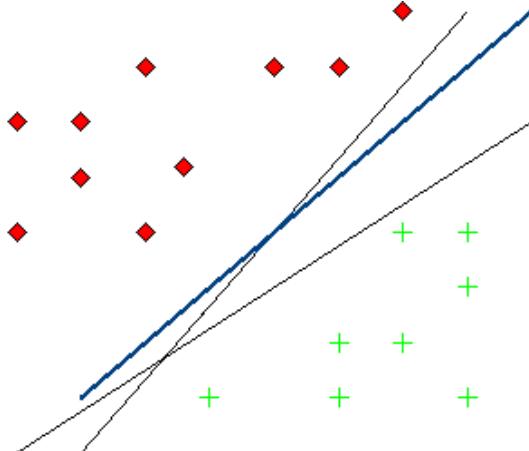


Рис.8.1. Линейные дискриминантные функции.

Таким образом разделяющая (дискриминантная) функция описывается уравнением $g(x) = 0$. Расстояние от точки до разделяющей функции $g(x) = 0$ равно

$$\frac{|w^t x + w_0|}{\|w\|}.$$

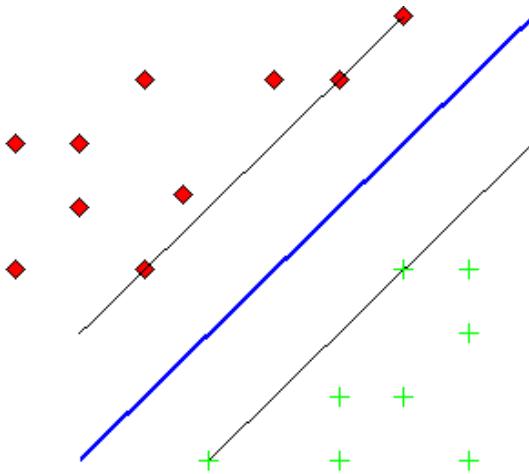


Рис. 8.2. Максимальный разделяющий коридор.

Пусть x_i лежит на замыкании границы, то есть $|w^t x_i + w_0| = 1$. Границу, ширину разделяющей полосы, нужно сделать как можно больше. Учитывая, что замыкание границы удовлетворяет условию $|w^t x_i + w_0| = 1$, тогда расстояние от x_i до $g(x) = 0$ равно

$$\frac{|w^t x + w_0|}{\|w\|} = \frac{1}{\|w\|},$$

таким образом, ширина разделяющей полосы равна $\frac{2}{\|w\|}$.

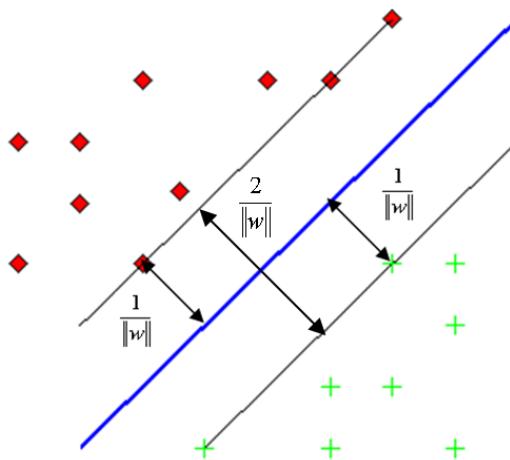


Рис. 8.3. Иллюстрация построения опорных векторов.

Для того, чтобы исключить точки из разделяющей полосы, выпишем условие принадлежности к одному из классов

$$\begin{cases} w^t x_i + w_0 \geq 1 & \text{если } x_i \text{ принадлежит первому классу,} \\ w^t x_i + w_0 \leq -1 & \text{если } x_i \text{ принадлежит второму классу.} \end{cases}$$

Введем индексную функцию

$$\begin{cases} u_i = 1 & \text{если } x_i \text{ принадлежит первому классу,} \\ u_i = -1 & \text{если } x_i \text{ принадлежит второму классу.} \end{cases}$$

Таким образом, задача выбора разделяющей функции, порождающей коридор наибольшей ширины можно записать в виде

$$J(w) = \frac{1}{2} \|w\|^2 \rightarrow \min$$

при условии $u_i(w^t x_i + w_0) \geq 1$ для всех i .

Так как целевая функция является квадратичной функцией, то у данной задачи существует единственное решение.

Согласно теореме Куна-Таккера это условие эквивалентно следующей задаче

$$L(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j u_i u_j x_i^T x_j \rightarrow \max \quad (8.1)$$

при условии, что $\alpha \geq 0 \forall i$ и $\sum_{i=1}^n \alpha_i u_i = 0$, где $\alpha = \{\alpha_1, \dots, \alpha_n\}$ новые переменные. Перепишем $L(\alpha)$ в матричном виде

$$L(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix}^T H \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix},$$

где коэффициенты матрицы H вычисляются следующим образом

$$H_{i,j} = u_i u_j x_i^T x_j .$$

Задача $L(\alpha) \rightarrow \max$ решается методами квадратичного программирования.

После нахождения оптимальных $\alpha = \{\alpha_1, \dots, \alpha_n\}$ для каждого i выполняется одно из двух условий

- $\alpha_i = 0$ (i соответствует неопорному вектору);
- $\alpha_i \neq 0$ и $u_i(w^t x_i + w_0 - 1) = 0$ (i соответствует опорному вектору);

Тогда может быть найдено w из соотношения $w = \sum_{i=1}^n \alpha_i u_i x_i$ и значение w_0 можно найти, учитывая, что для любого $\alpha_i > 0$ и $\alpha_i [u_i(w^t x_i + w_0) - 1] = 0$

$$w_0 = \frac{1}{u_i} - w^t x_i.$$

После чего, наконец, получаем дискриминантную функцию

$$g(x) = \left(\sum \{\alpha_i u_i x_i | x_i \in S\} \right)^T x + w_0.$$

Заметим, что суммирование проводится не по всем векторам, а только по множеству S , которое представляет собой множество опорных векторов $S = \{x_i | \alpha_i \neq 0\}$.

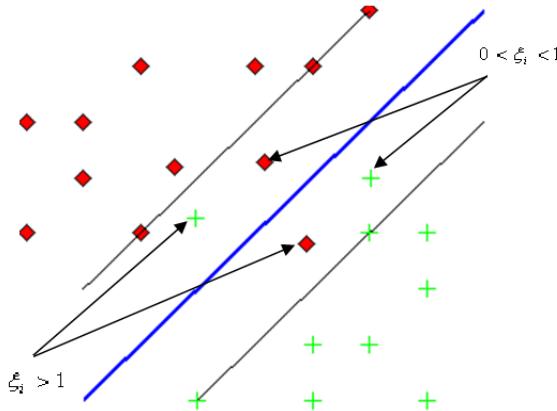
К сожалению, описанный алгоритм реализуем только для линейно разделимых множеств, что само по себе встречается достаточно нечасто. Приведем модернизацию этого алгоритма для случая линейно неразделимых множеств.

Введем дополнительные переменные ξ_i , которые характеризуют величину ошибки на каждом объекте x_i . Введем в функционал цели штраф за суммарную ошибку:

$$\begin{cases} \frac{1}{2} \|w\|^2 + \lambda \sum_{i=1}^n \xi_i \rightarrow \min, \\ u_i (w^t x_i + w_0) \geq 1 - \xi_i, i = 1, \dots, n, \\ \xi_i \geq 0, i = 1, \dots, n, \end{cases} \quad (8.2)$$

здесь λ — параметр настройки метода, который позволяет регулировать отношение между максимизацией ширины разделяющей полосы и минимизацией суммарной ошибки.

Величина штрафа ξ_i для соответствующего объекта x_i зависит от расположения объекта x_i относительно разделяющей полосы. Так, если x_i лежит с противоположной стороны дискриминантной функции, то будем считать величину штрафа $\xi_i > 1$, если x_i лежит в разделяющей полосе, но со стороны своего класса, то соответствующий вес будет $0 < \xi_i < 1$. Для идеального случая будем считать $\xi_i < 0$.



Rис. 8.4. Точки, к которым применяются штрафы.

Тогда задачу (8.2) можно переписать в виде

$$J(w, \xi_1, \dots, \xi_n) = \frac{1}{2} \|w\|^2 + \beta \sum_{i=1}^n I(\xi_i > 0) \rightarrow \min,$$

то есть в процессе минимизации участвуют элементы, которые не представляют собой идеальный случай. Здесь

$$I(\xi_i > 0) = \begin{cases} 1, & \xi_i > 0, \\ 0, & \xi_i \leq 0, \end{cases}$$

при выполнении условий $u_i(w^t x_i + w_0) \geq 1 - \xi_i$ и $\xi_i \geq 0$. Здесь постоянная β является весом, учитывающим ширину полосы. Если β мало, то мы позволяем расположить относительно много элементов в неидеальной позиции, то есть, в разделяющей полосе. Если β велико, то мы требуем наличия малого количества элементов в неидеальной позиции, то есть в разделяющей полосе.

К сожалению, задача минимизации является достаточно сложной, ввиду разрывности $I(\xi_i)$. Вместо этого мы рассмотрим минимизацию величины

$$J(w, \xi_1, \dots, \xi_n) = \frac{1}{2} \|w\|^2 + \beta \sum_{i=1}^n \xi_i \text{ при ограничениях } \forall i$$

$$\begin{cases} u_i(w^t x_i + w_0) \geq 1 - \xi_i, \\ \xi_i \geq 0. \end{cases}$$

Используя теорему Куна-Таккера, отсюда получаем

$$L(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j u_i u_j x_i^T x_j \rightarrow \max \quad (8.3)$$

при условии, что $0 \leq \alpha_i \leq \beta, \forall i$ и $\sum_{i=1}^n \alpha_i u_i = 0$.

Найдем w из соотношения $w = \sum_{i=1}^n \alpha_i u_i x_i$. Значение w_0 также можно найти, учитывая, что для любого $0 \leq \alpha_i \leq \beta$ и $\alpha_i [u_i (w^t x_i + w_0) - 1] = 0$.

Другой идеей метода опорных векторов (в случае невозможности линейного разделения классов), является переход в пространство большей размерности, в котором такое разделение возможно.

Для решения задачи нелинейной классификации с помощью линейного классификатора нужно:

- Спроектировать данные x в пространство более высокой размерности с помощью отображения $\varphi(x)$.
- Найти линейную дискриминантную функцию для данных $\varphi(x)$.
- Окончательная нелинейная дискриминантная функция может быть записана в виде

$$g(x) = w^t \varphi(x) + w_0.$$

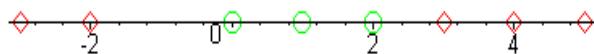
В 2D дискриминантная функция линейная

$$g\left(\begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix}\right) = [w_1 \quad w_2] \begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} + w_0.$$

В 1D дискриминантная функция нелинейная

$$g(x) = w_1 x + w_2 x^2 + w_0.$$

Для перевода данных в пространство более высокой размерности используют так называемые ядерные функции.



$$\downarrow \varphi(x) = (x, x^2)$$

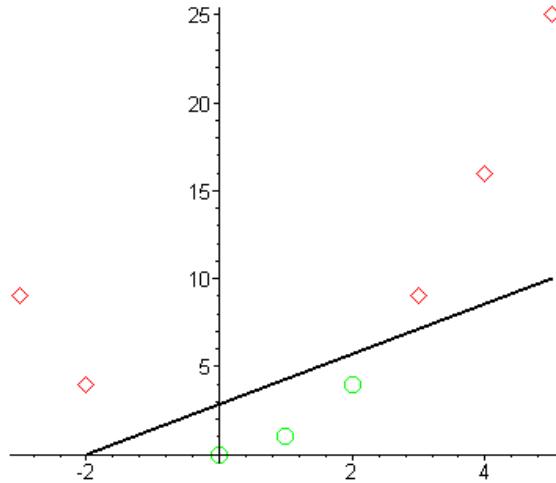


Рис. 8.6. Пример линейного разделения множеств при переходе в пространство более высокой размерности.

Запишем экстремальную задачу метода опорных векторов в виде (8.3)

$$L(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j u_i u_j x_i^T x_j \rightarrow \max .$$

Заметим, что оптимизация зависит от произведения $x_i^T x_j$. Если мы переведем x_i в пространство более высокой размерности используя отображение $\varphi(x)$, то нужно вычислять аналогичное произведение в пространстве более высокой размерности $\varphi(x_i)^T \varphi(x_j)$.

Идея метода состоит в том, что нужно найти ядерную функцию $K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$ и максимизировать целевую функцию

$$L(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j u_i u_j K(x_i, x_j) \rightarrow \max .$$

Рассмотрим пример и возьмем ядерную функцию в виде $K(x, y) = (x^T y)^2$. Выясним какое отображение $\varphi(x)$ соответствует этой ядерной функции.

$$\begin{aligned} K(x, y) = (x^T y)^2 &= \left(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix} \begin{bmatrix} y^{(1)} \\ y^{(2)} \end{bmatrix} \right)^2 = (x^{(1)} y^{(1)} + x^{(2)} y^{(2)})^2 = \\ &= (x^{(1)} y^{(1)})^2 + 2(x^{(1)} y^{(1)}) (x^{(2)} y^{(2)}) + (x^{(2)} y^{(2)})^2 = \\ &= \begin{bmatrix} (x^{(1)})^2 & \sqrt{2} x^{(1)} x^{(2)} & (x^{(2)})^2 \end{bmatrix} \begin{bmatrix} (y^{(1)})^2 & \sqrt{2} y^{(1)} y^{(2)} & (y^{(2)})^2 \end{bmatrix}^T . \end{aligned}$$

Таким образом, $\varphi(x) = \begin{bmatrix} (x^{(1)})^2 & \sqrt{2} x^{(1)} x^{(2)} & (x^{(2)})^2 \end{bmatrix}$.

Выбор ядерной функции является достаточно сложным.

Рассмотрим пример [133].

Класс[1]: $x_1=[1,-1]$, $x_2=[-1,1]$.

Класс[2]: $x_3=[1,1]$, $x_4=[-1,-1]$.

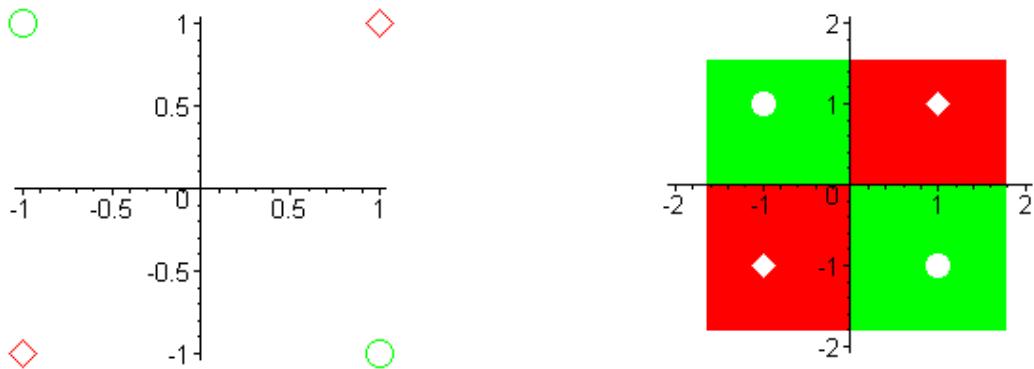


Рис. 8.7. Пример линейно неразделимых множеств.

Для построения нелинейной дискриминантной функции используем ядерную функцию вида

$$K(x_i, x_j) = (x_i^T x_j + 1)^2.$$

Отображение, соответствующее этой функции имеет вид

$$\varphi(x) = \begin{bmatrix} 1 & \sqrt{2}x^{(1)} & \sqrt{2}x^{(2)} & \sqrt{2}x^{(1)}x^{(2)} & (x^{(1)})^2 & (x^{(2)})^2 \end{bmatrix}.$$

Далее нужно максимизировать функцию

$$L(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j u_i u_j (x_i^T x_j + 1)^2 \rightarrow \max$$

при выполнении ограничений

$$\alpha_i \geq 0, \alpha_1 + \alpha_2 - \alpha_3 - \alpha_4 = 0.$$

Перепишем задачу в виде

$$L(\alpha) = \sum_{i=1}^4 \alpha_i - \frac{1}{4} \alpha^T H \alpha,$$

где $\alpha = [\alpha_1 \quad \alpha_2 \quad \alpha_3 \quad \alpha_4]^T$ и $H = \begin{pmatrix} 9 & 1 & -1 & -1 \\ 1 & 9 & -1 & -1 \\ -1 & -1 & 9 & 1 \\ -1 & -1 & 1 & 9 \end{pmatrix}$.

Для нахождения максимума, найдем производную по неизвестным и приравнивая нулю эти производные, найдем значения неизвестных, на которых достигается максимум целевой функции.

$$\frac{d}{d\alpha} L(\alpha) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 9 & 1 & -1 & -1 \\ 1 & 9 & -1 & -1 \\ -1 & -1 & 9 & 1 \\ -1 & -1 & 1 & 9 \end{pmatrix} \alpha = 0.$$

Решая эту систему, получаем $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = \frac{1}{4}$,

$$w = \sum_{i=1}^4 \alpha_i u_i \varphi(x_i) = \frac{1}{4} (\varphi(x_1) + \varphi(x_2) - \varphi(x_3) - \varphi(x_4)) = [0 \ 0 \ 0 \ -\sqrt{2} \ 0 \ 0]$$

и, наконец, нелинейная дискриминантная функция будет иметь вид

$$g(x) = w \varphi(x) = \sum_{i=1}^6 w_i \varphi_i(x) = -\sqrt{2} (\sqrt{2} x^{(1)} x^{(2)}) = -2x^{(1)} x^{(2)}.$$

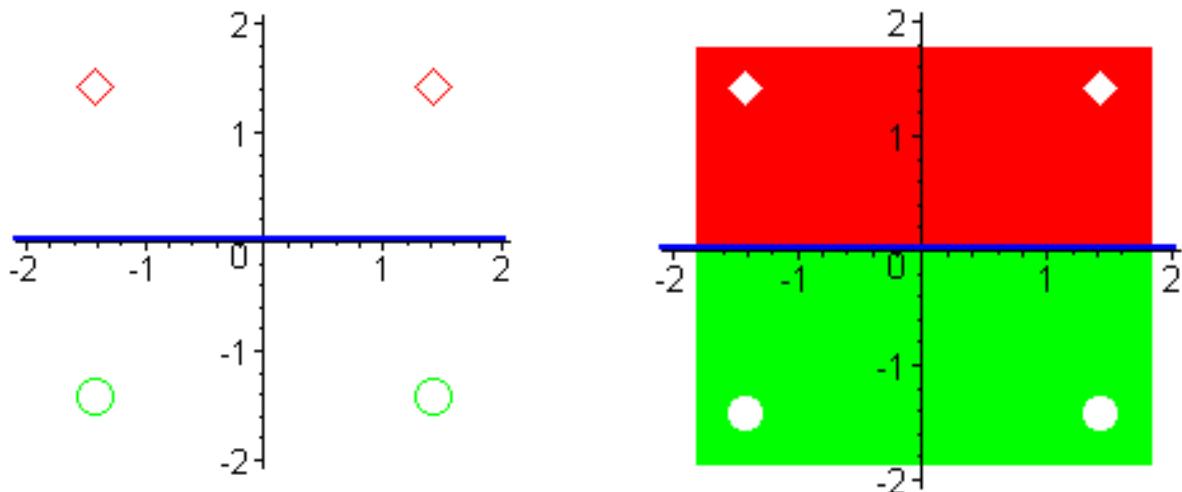


Рис.8.8. Линейная разделимость множеств, полученная в результате использования полиномиального неоднородного ядра.

В заключение приведем несколько наиболее распространенных ядер, используемых для разделения кластеров:

- Полиномиальное однородное ядро $K(x_i, x_j) = (x_i^T x_j)^d$.
- Полиномиальное неоднородное ядро $K(x_i, x_j) = (x_i^T x_j + 1)^d$.
- Радиальная функция Гаусса $K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$.

9. Визуализация многомерных данных

Человек достаточно хорошо анализирует двумерные, или, в крайнем случае, трехмерные данные, однако большинство информации, которая требует анализа, имеет существенно большую размерность. Что же делать? Для того, чтобы помочь исследователю провести анализ имеющихся данных, используют методы визуализации, сводя многомерные данные к дву- или трехмерным (см., например, [11]). Данный параграф посвящен некоторым таким методам.

Достаточно часто информация задана в виде квадратной таблицы удаленностей объектов друг от друга. На пересечении i -ой строки и j -ого столбца в такой таблице стоит значение, определяющее отличие (или сходство) i -го и j -го объекта. Такое представление информации характерно для различного рода исследований, когда человеку предлагается оценивать сходство или различие в некоторой системе объектов или понятий.

Таким образом, изначально каждому объекту не сопоставляется никакой координаты в многомерном пространстве, и, представить такие данные в виде некоторой геометрической интерпретации достаточно сложно. Задача *многомерного шкалирования* (см., например, [133]) заключается в том, чтобы так сконструировать распределение данных в пространстве, чтобы расстояния между объектами соответствовали значениям, заданным в матрице удаленностей. Возникающие при этом координатные оси могут быть интерпретированы как некоторые факторы, значения которых определяют различия объектов между собой, например, главные направления, определяемые с помощью МГК (PCA). В таком случае, поставив в соответствие каждому объекту пару координат, получим способ визуализации данных.

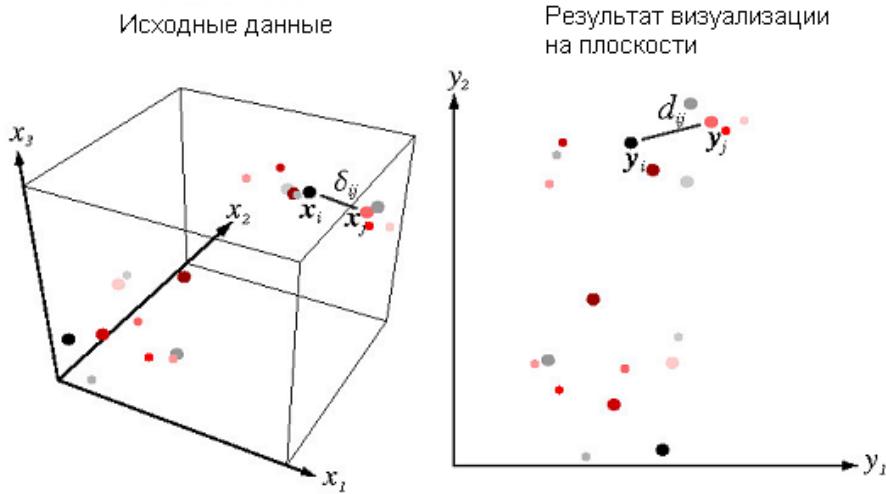


Рис. 9.1. Иллюстрация многомерного шкалирования.

В линейном методе метрического шкалирования применяется метод главных компонент, но не к исходной матрице расстояний, а к *дважды центрированной матрице*, у которой среднее значение чисел в любой строке и столбце равно нулю. Дважды центрированная матрица однозначно вычисляется по исходной. После этого, существует возможность определить размерность пространства, обеспечивающего *точное воспроизведение* матрицы удаленностей, либо определить эффективную размерность конструируемого пространства признаков, которая обеспечит воспроизведение матрицы удаленностей с заданной точностью.

В нелинейных методах размерность пространства задается изначально, и, с помощью градиентных методов оптимизируется функционал качества, описывающий меру искажения матрицы удаленностей.

Формализуем постановку задачи. Итак, пусть даны точки x_1, x_2, \dots, x_n в пространстве размерности k . Расстояние между точками x_i и x_j обозначим через $\delta_{i,j}$. Найдем точки y_1, y_2, \dots, y_n в пространстве меньшей размерности (2 или 3) так, чтобы расстояние между ними, равное $d_{i,j}$, было поставлено в соответствии с $\delta_{i,j}$. В идеале хотелось бы получить прямое соответствие $d_{i,j} = \delta_{i,j}$, но в нетривиальном случае при переходе в пространство меньшей размерности получить такое равенство невозможно. Но как-то эту задачу нужно решать, и, одним из возможных методов нахождения зависимости $d_{i,j}$ от

$\delta_{i,j}$ является сведение к минимизации некоторой функции цели. В качестве такой задачи можно использовать минимизацию среднеквадратической ошибки

$$J(y) = \frac{\sum_{i < j} (d_{i,j} - \delta_{i,j})^2}{\sum_{i < j} \delta_{i,j}^2}$$

Минимизация функции цели является нетривиальной задачей. Для ее решения можно использовать метод градиентного спуска.

$$\nabla_{y_k} J(\delta) = \frac{2}{\sum_{i < j} \delta_{i,j}^2} \sum_{j \neq k} (d_{k,j} - \delta_{k,j}) \frac{(y_k - y_j)}{d \sum_{i < j} d_{i,j}}$$

Координаты x_1, x_2, \dots, x_n выбираются так, чтобы получить наибольшую вариацию.

Заметим, что многомерное распределение данных, отображенное на плоскость, не решает всех вопросов визуализации исходных данных. Совершенно естественна идея отображать на двумерную карту не только сами точки данных, но и разнообразную информацию, сопутствующую исходным данным, например, положение точек в исходном пространстве, плотности различных подмножеств, другие непрерывно распределенные величины, заданные в исходном пространстве признаков. Все приводит к идее более эффективного использования всей первичной информации для отображения как количественной, так и качественной информации.

Наконец, после того, как данные нанесены на двумерную плоскость, хотелось бы, чтобы появилась возможность расположить на двумерной плоскости те данные, которые не участвовали в настройке отображения. Это позволило бы, с одной стороны, использовать полученную картину для построения различного рода экспертных систем и решать задачи распознавания образов, с другой - использовать ее для восстановления данных с пробелами. В результате приходим к идее использования для визуализации данных и извлечения информации некоторого вспомогательного объекта, называемого *картой*. Этот объект представляет ограниченное двумерное нелинейное многообразие, вложенное в многомерное пространство данных таким образом, чтобы служить моделью данных, то есть форма и расположение такого многообразия должна отражать основные особенности распределения множества точек данных.

Простой пример карты данных - плоскость первых двух главных компонент. Как уже отмечалось выше, среди всех двумерных плоскостей, вложенных в пространство, именно плоскость первых двух главных компонент служит оптимальным экраном, на котором можно отобразить основные закономерности, присущие данным. В качестве другой, еще более простой (но не оптимальной) карты можно использовать любую координатную плоскость любых двух выбранных координат.

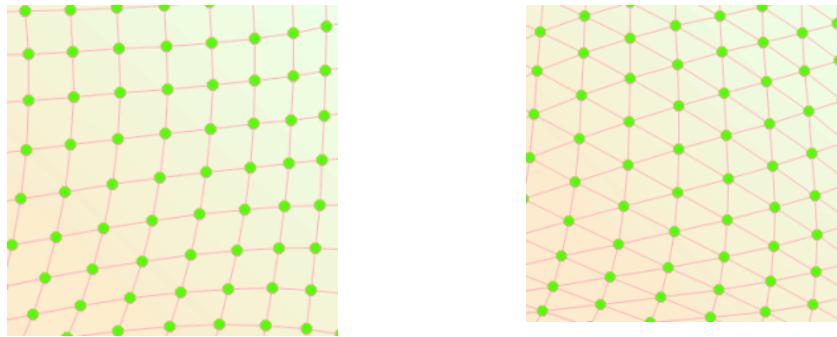
Самоорганизующиеся карты Кохонена (SOM)

Цель, как и ранее - проецировать имеющиеся данные в пространство меньшей размерности с сохранением соответствия расстояния между имеющимися точками (объектами).

Карты Кохонена [125] производят картографию от многомерного входа на 1D или 2D решетку узлов с использованием идеологии нейронных сетей. Важной особенностью этой картографии является сохранение топологии, между исходными данными и нейронами, на которые они проецируются. Самоорганизующиеся карты Кохонена представляют собой нейронную сеть на основе жадного алгоритма, не требующего обучения. Заметим, что идея SOM основана на организации функционирования головного мозга - активизация нейрона при восприятии информации приводит к возбуждению участков в соседних частях мозга.

Перейдем к подробному изложению идеологии SOM. Нейросетевая архитектура, предложенная Тойво Кохоненом [103] для автоматической кластеризации (классификации без учителя), основана на идее использования информации о взаимном расположении нейронов, которые образуют решетку. Сигнал в такую нейросеть поступает сразу на все нейроны, а веса соответствующих синапсов интерпретируются как координаты положения узлов, и, выходной сигнал формируется по принципу «победитель забирает все» - то есть ненулевой выходной сигнал имеет нейрон, ближайший (в смысле весов синапсов) к подаваемому на вход объекту. В процессе обучения веса синапсов настраиваются таким образом, чтобы узлы решетки «располагались» в местах локальных сгущений данных, то есть описывали кластерную структуру облака данных, с другой стороны, связи между нейронами соответствуют отношениям соседства между соответствующими кластерами в пространстве признаков.

Изначально SOM представляет сетку из узлов, соединенных между собой связями. Кохонен рассматривал два варианта соединения узлов - с прямоугольной и гексагональной сеткой. В прямоугольной сетке каждый узел соединен с четырьмя соседними, а в гексагональной - с шестью ближайшими узлами.

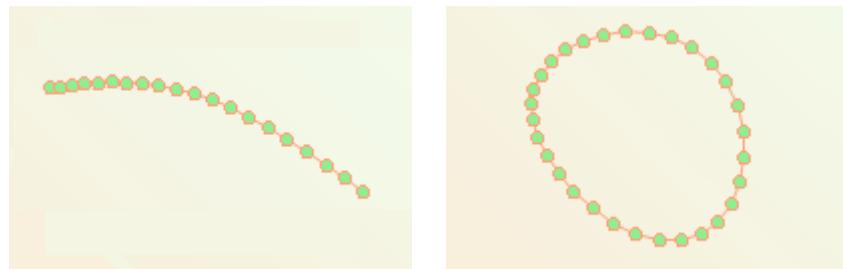


Нейроны в прямоугольной решетке – каждый нейрон имеет 4 ближайших соседа

Нейроны в гексагональной решетке – каждый нейрон имеет 6 ближайших соседей

Рис. 9.2. Примеры использования двумерных сеток нейронов Кохонена.

Заметим, что среди наиболее популярных топологий нейронных сетей Кохонена следует отметить следующие случаи



Линейный слой единичной высоты

Круговой слой единичной высоты

Рис. 9.3. Одномерные сети Кохонена.

а также,

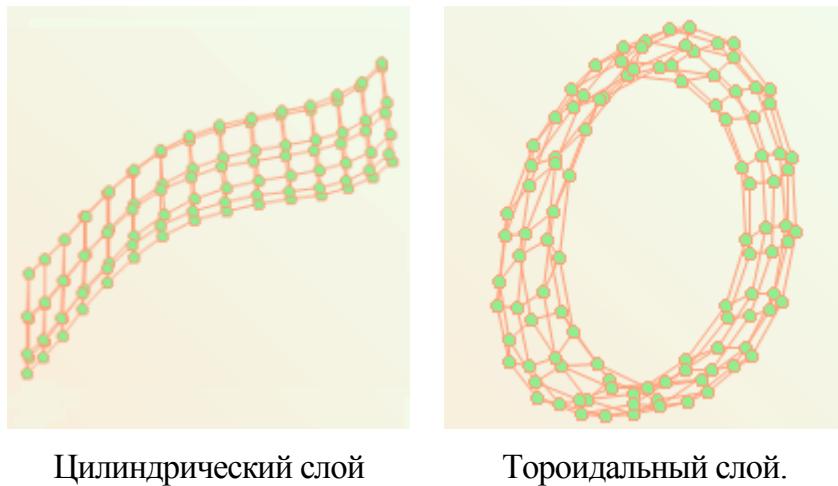


Рис. 9.4. Двумерные сети Кохонена.

Инициализация карты Кохонена

В основе SOM лежит связная структура нейронов, конкурирующих между собой при воздействии сигнала. Каждому нейрону поставлен в соответствие вес w_{ij} , число весов равно размерности входного сигнала.

Существует несколько способов начальной инициализации карты, путем присвоения значений всем векторам весов нейронов w_{ij} следующими способами:

1. Задание всех координат случайными числами.
2. Присвоение вектору веса значения случайного наблюдения из входных данных.
3. Выбор векторов веса из линейного пространства, натянутого на главные компоненты набора входных данных.

Следует заметить, что каждый нейрон после инициализации становится неподвижен на карте, т.е. вектор p не изменяется в течение всего обучения.

Алгоритм обучения

Алгоритм обучения производится по итерациям.

Пусть t - номер итерации. Положим, что инициализация имела номер итерации 0 . Далее, выполняются следующие операции:

1. Выбираем случайный вектор $x(t)$ из набора входных значений.
2. Находим расстояние до всех векторов весов всех нейронов карты. Для данной операции выбирается какая-нибудь мера, чаще всего используется среднеквадратичное отклонение. Ищем нейрон, наиболее близкий ко входному значению $x(t)$:

$$d(x(t), w_{\nu\mu}(t)) < d(x(t), w_{ij}(t)),$$

где $w_{\nu\mu}(t)$ - вектор веса нейрона победителя (MBU, Winner) $M_{\nu\mu}(t)$, $w_{ij}(t)$ - вектор веса, $d(x(t), w_{ij}(t))$ - некая мера отклонения.

В случае, если вышеуказанному условию удовлетворяют несколько нейронов, то нейрон-победитель выбирается случайным образом.

После этого узлы начинают перемещаться в пространстве. Однако, узел перемещается не один, а увлекает за собой определенное количество ближайших узлов из некоторой окрестности на карте. Поясним сказанное: если радиус окрестности равен 1, то вместе с ближайшим узлом по направлению к x двигаются 4 его соседа по карте, в случае прямоугольной сетки, и 6 соседей, в случае гексагональной сетки.

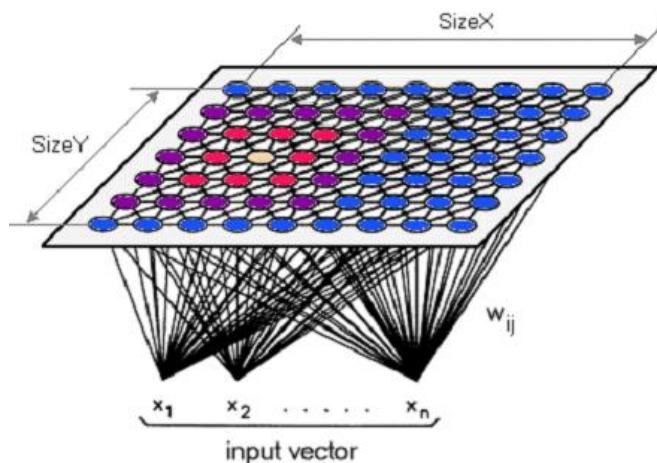


Рис. 9.5. Самоорганизующиеся карта Кохонена (SOM).

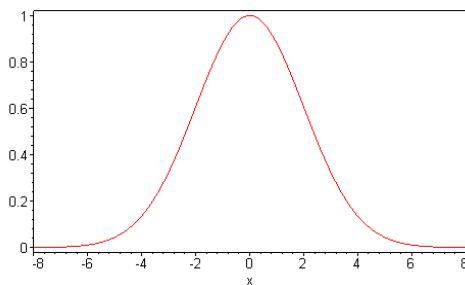
В настройке карты различают два этапа - этап грубой (*ordering*) и этап тонкой (*fine-tuning*) настройки. На первом этапе выбираются большие значения окрестностей, и, движение узлов носит колективный характер - в результате карта «расправляется» и грубым образом отражает структуру данных; на этапе тонкой настройки радиус окрестности равен 1-2 и настраиваются уже индивидуальные положения узлов.

О характере движения следует заметить следующее: обычно он настраивается так, чтобы среди всех двигающихся узлов наиболее сильно смещался центральный - ближайший к точке данных - узел, а остальные испытывали тем меньшие смещения, чем дальше они от центра узла.

Характер движения формируется так называемыми затухающими функциями соседства (neighborhood function). Если это не так, и все соседи из окружения испыты-

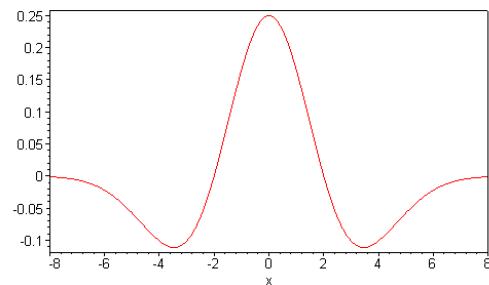
вают равные смещения, то такая функция соседства называется пузырьковой (bubble), и, для нее характерно большее число актов перемещения для обучения и менее гладкая сетка.

Рассмотрим подробнее определение меры соседства нейронов и изменение весов нейронов в карте. Выбираем меру соседства - функцию $h(t)$. Обычно, в качестве функции $h(t)$ используется гауссовская функция:



$$h(x) = a \exp\left(-\frac{(x-b)^2}{2c^2}\right)$$

Функция Гаусса



$$h(x) = a \left(1 - \left(\frac{(x-b)^2}{c^2}\right)\right) \exp\left(-\frac{(x-b)^2}{2c^2}\right)$$

Мексиканская шляпа (нормированная вторая производная функции Гаусса)

Рис.9.6. Примеры функций соседства.

Пусть $h(t)$ функция Гаусса, тогда «мера соседства» между нейронами $M_{i,j}$ и $M_{v,\mu}$ будет иметь вид

$$h_{i,j}^{v,\mu}(t) = \alpha(t) \exp\left(-\frac{\|r_{i,j} - r_{v,\mu}\|^2}{2\delta^2(t)}\right),$$

где $0 < \alpha(t) < 1$ — обучающий сомножитель, монотонно убывающий с каждой последующей итерацией (то есть определяющий приближение значения векторов веса нейрона-победителя и его соседей к наблюдению; чем больше шаг, тем меньше уточнение); $r_{i,j}, r_{v,\mu}$ — координаты узлов $M_{i,j}(t)$ и $M_{v,\mu}(t)$ на карте; $\delta(t)$ — монотонно убывающий сомножитель, регулирующий количество соседей в зависимости от итерации.

Параметры α , δ и их характер убывания определяются экспертным путем. Более простой способ задания функции соседства: $h_{i,j}^{\nu,\mu}(t) = \alpha(t)$, если $M_{i,j}(t)$ находится в окрестности $M_{\nu,\mu}(t)$ заранее заданного радиуса, и 0 - в противном случае.

Функция $h(t)$ равна $\alpha(t)$ для нейрона-победителя и уменьшается с удалением от него.

Следующий этап - вычисление ошибки карты. Изменяя вектора весов по формуле:

$$m_{i,j}(t) = m_{i,j}(t-1) + h_{i,j}^{\nu,\mu}(t)(x(t) - m_{i,j}(t-1))$$

Таким образом, все узлы, являющиеся соседями нейрона-победителя, приближаются к рассматриваемому наблюдению.

Наконец, рассмотрим выбор условия остановки процесса формирования карты.

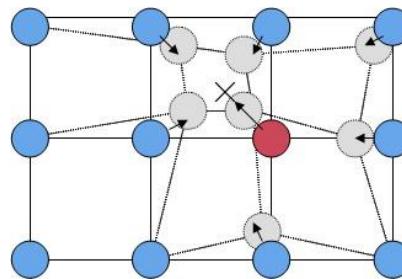


Рис.9.7. Схема движения нейронов.

Для определения критерия остановки в большинстве случаев используется ошибка карты, например, как среднее арифметическое расстояние между наблюдениями и векторами веса соответствующих им нейрона-победителя:

$$\frac{1}{N} \sum_{i,j}^N \|x_{i,j} - w_{i,j}\|,$$

где N - количество элементов набора входных данных.

Алгоритм повторяется определенное число тактов. На первом этапе число тактов выбирается порядка тысяч, на втором - десятка тысяч (понятно, что число шагов может сильно изменяться в зависимости от задачи).

Отметим, что предложенный алгоритм не использует явно никакого критерия оптимизации. Хотя ясно, что, по крайней мере, будет уменьшаться среднее расстояние от каждой точки данных до ближайшего узла карты. Средний квадрат такого расстоя-

ния служит критерием качества построенной карты. В этом пакете, как правило, строится несколько десятков карт, из которых выбирается лучшая согласно упомянутому критерию.

В результате действия алгоритма строится карта, то есть двумерная сетка узлов, размещенных в многомерном пространстве. Для того, чтобы изобразить их положение используются различные средства. Одно из них - такое раскрашивание карты, когда цвет отражает расстояние между узлами.

Приведем несколько примеров.

Всемирная карта бедности

На странице (<http://www.cis.hut.fi/nrc/worldmap.html>) Helsinki University of Technology Laboratory of Computer and Information Science) приведен пример использования SOM для построения некоего комплексного экономического показателя, по которому раскрашена обычная географическая карта. В результате, страны со сходной экономической ситуацией изображены на карте цветами, близкими по спектру.

Для построения этой карты рассмотрено 39 особенностей (индикаторов), описывающих различные факторы качества жизни, например, уровень системы здравоохранения, образовательные услуги, качество питания и пр. Государства, для которых имеющиеся значения индикаторов подобны, образовывают различные кластеры, которые на карте автоматически кодировались различными цветами. В результате этого процесса, каждой стране было автоматически назначено некоторое цветовое описание, характеризующее уровень бедности – от желтого цвета благополучных стран до фиолетового.

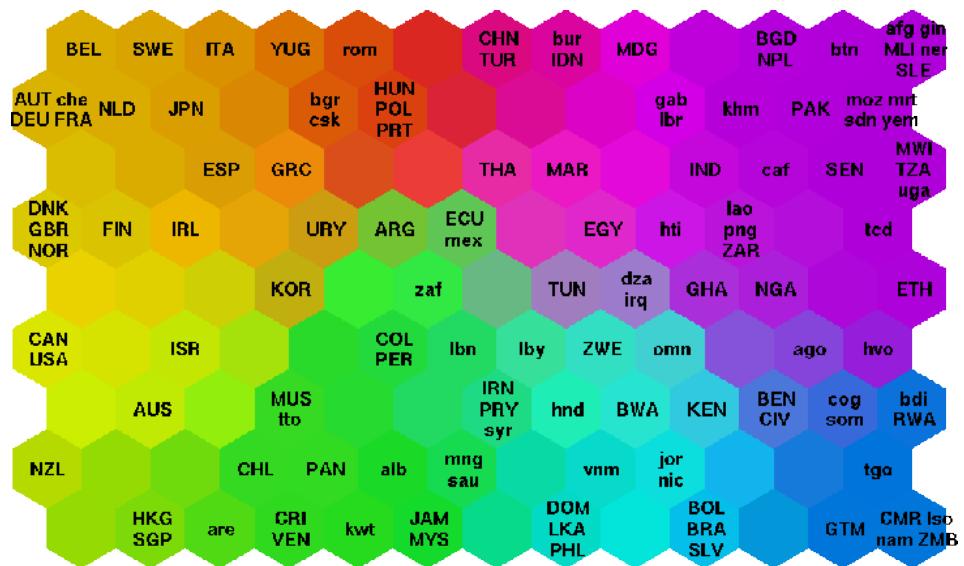


Рис. 9.8. Распределение цветов карты бедности мира.

На данной карте каждая страна окрашена в цвет согласно уровню бедности. В серый цвет окрашены страны, для которых (на время исследования) необходимая статистика была либо неполной, либо недоступной.

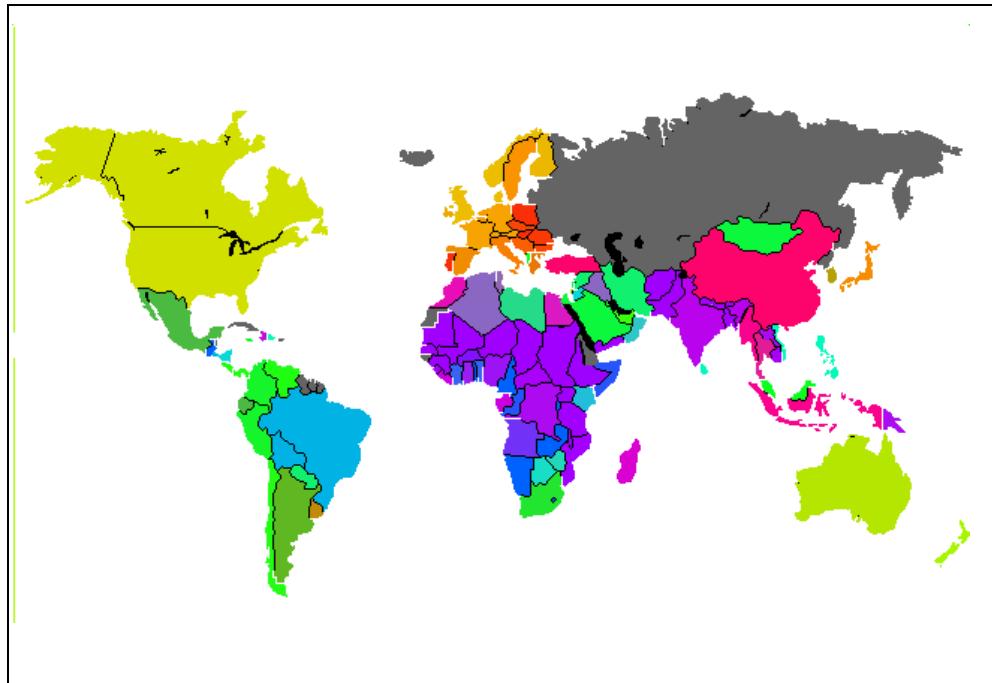


Рис. 9.9. Карта бедности мира.

Рассмотрим еще один пример использования сетей Кохонена – решение задачи коммивояжера.

Цель задачи коммивояжера состоит в том, чтобы найти кратчайший путь движения коммивояжера, который начинается из любого города, проходит через все города на карте продаж и возвращается в исходный город. Формально, мы должны найти кратчайший гамильтонов цикл неориентированного взвешенного графа, содержащего в качестве узла - город, а расстояние между двумя городами является весом ребра, соединяющего соответствующие узлы. Здесь мы попытаемся найти приближенное решение этой задачи с использованием самоорганизующихся карт Кохонена. Исходные данные состоят из координат (X и Y) всех городов на карте продаж. Сеть Кохонена состоит из кольцевого слоя.

На следующих скриншотах проиллюстрированы этапы применения SOM к решению задачи коммивояжера. На первой иллюстрации приведена исходная карта продаж. На последней – полученная траектория движения коммивояжера, полученный путь является наиболее коротким среди всех возможных маршрутов, соединяющих все города на карте продаж. На второй и третьей иллюстрациях приведены промежуточные результаты.

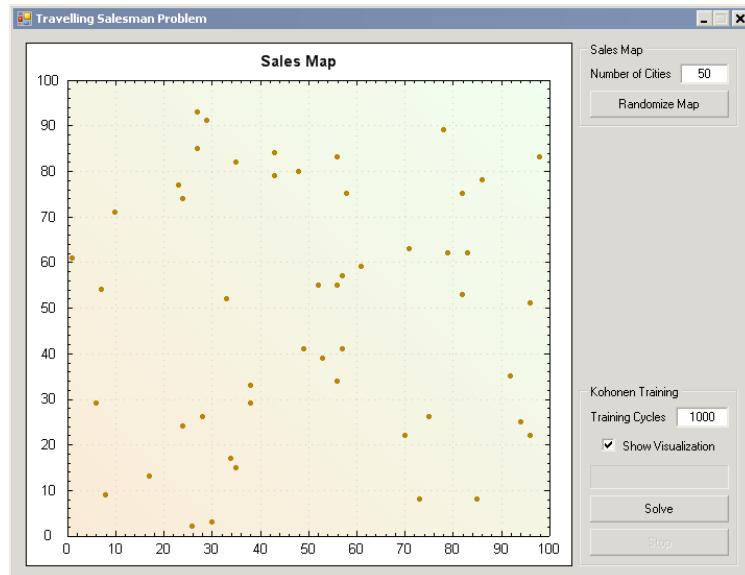


Рис. 9.10. Карта продаж.

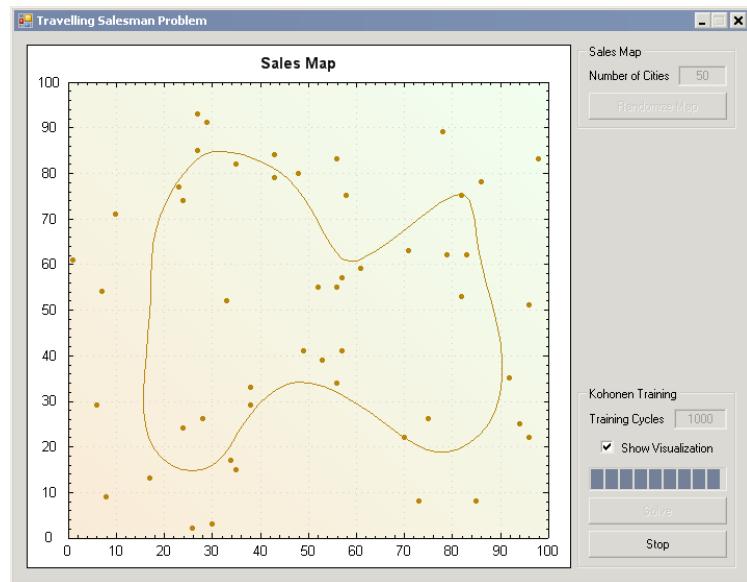


Рис. 9.11. Траектория движения после 100 итераций.

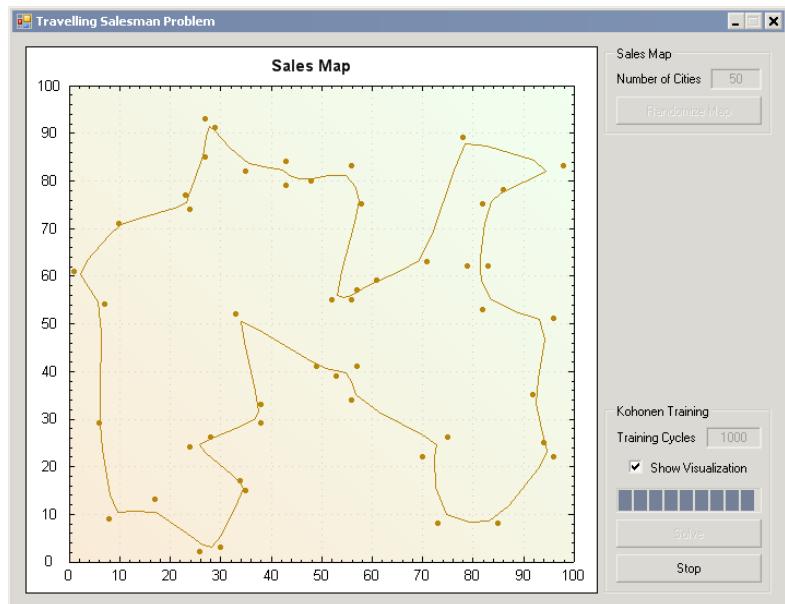


Рис. 9.12. Траектория движения после 700 итераций.

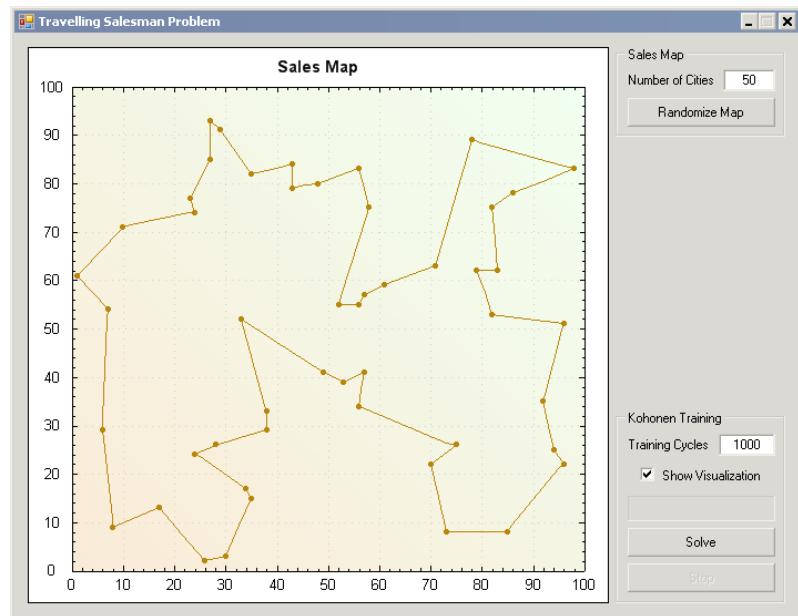


Рис. 9.13. Решение задачи коммивояжера.

10. Рекомендующие системы

Цель рекомендующей системы (см., например, [121]) состоит в создании значимых рекомендаций пользователям относительно набора предметов или продуктов (далее будем использовать термин сервис), которые могут их заинтересовать. Это могут быть предложения книг, товаров или фильмов. Разработка таких рекомендаций зависит от области и конкретных характеристик данных. Например, фильму на сайте Netflix часто приставляют рейтинги по шкале от 1 (не нравится) до 5 (понравилось). Такой источник данных записей позволяет реализовать взаимодействие между пользователями и элементами запроса. Кроме того, используя такие атрибуты, как демографические данные и описание продукта, система может иметь доступ к конкретному пользователю и пункту конкретного профиля. Рекомендующие системы отличаются по способу анализа этих источников данных для исследования связи между пользователями и элементами запроса, которые могут быть использованы для выявления хорошо коррелирующих пар клиент-сервис. В основе рекомендующих систем лежит коллаборативная фильтрация системы анализа истории взаимодействия [94] и content-фильтрация [54] на основе системы атрибутов сервисов (товаров или услуг). Кроме того, существуют гибридные методы [85], использование которых представляет собой попытку объединения обеих этих конструкций.



Рис.10.1. Общая структура рекомендующей системы.

Получение рекомендаций из надежных источников является одним из важнейших компонентов естественного процесса принятия решений. Растущее потребление, поддержанное развитием Всемирной Сети, приводит к тому, что покупателям представлен более широкий круг выбора сервисов, в то время как продавцы сталкиваются с проблемой персонализации своих рекламных кампаний. Рекомендующие системы эволюционируют, чтобы удовлетворить потребности покупателей и продавцов за счет автоматизации рекомендаций на основе анализа данных.

Термин «коллаборативная (совместная) фильтрация» [65] был введен в контексте первой коммерческой рекомендующей системы, которая была разработана для адресной рекомендации новостей определенному кругу пользователей. Мотивация состояла в том, чтобы пользователь не получал документы, которые ему не интересны. Совместная фильтрация анализирует данные об использовании товаров или услуг (сервисов) разными пользователями, с целью выявления хорошо подобранный пары пользователь-сервис. В соответствии с методологией фильтрации контента, которая уходит корням к информационному поиску, рекомендации не являются "совместными", в том смысле, что предложения, высказанные пользователями, явным образом не

используют информацию всей пользовательской базы, а отталкиваются от атрибутов сервиса (товара, услуги).

Первые методы для рекомендующей системы были основаны на простой статистике корреляции и прогнозного моделирования.

Дальнейшие исследования были вызваны публичной доступностью данных в Интернете, и интересом, связанным с возросшей популярностью электронной коммерции. Всплеск интереса к этой проблеме вызвала компания Netflix, специализирующаяся на онлайн потоковом видео и DVD аренде, которая выложила для свободного доступа крупный набор данных, содержащий 100 млн. рейтингов, примерно на полмиллиона пользователей, и несколько тысяч названий фильмов, после чего объявила открытый конкурс на лучший алгоритм коллоквиривной фильтрации.

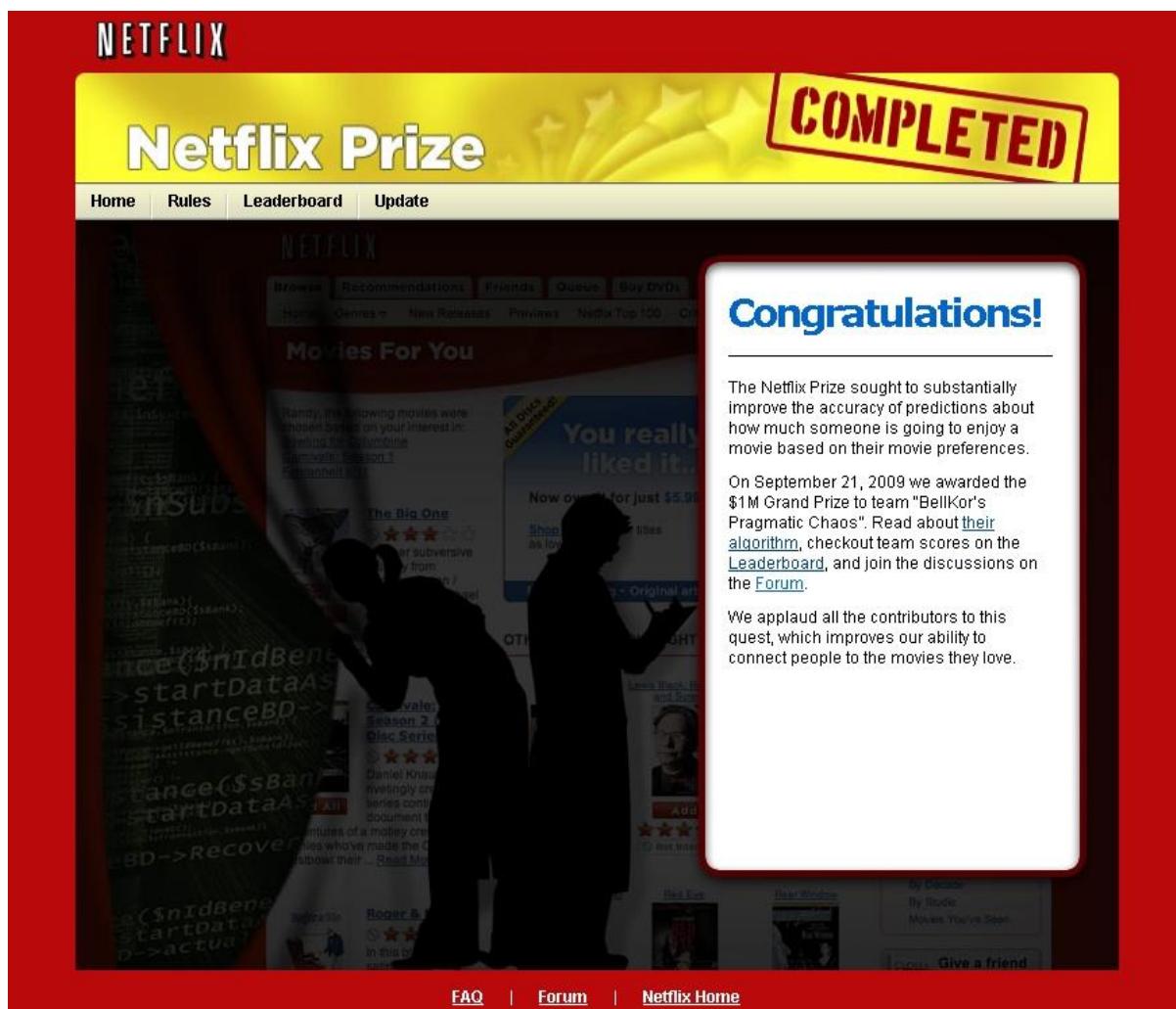


Рис. 10.2. Сообщение Netflix об окончании конкурса.

В настоящее время, рекомендующие системы остаются областью активных исследований, в которых пересекаются разделы математической статистики, машинного обучения, интеллектуального анализа данных, и пр. Приложения методов рекомендующих систем осуществляются в различных областях, начиная от веб-страниц, рекомендующих музыку, книги, фильмы и другие потребительские товары, до интерпретации политических выборов.

Общая структура рекомендующей системы представлена на рис. 10.3. Известные предпочтения пользователя представлены в виде матрицы из N пользователей и M сервисов (изделий, элементов, услуг), где каждой ячейке соответствует рейтинг данного сервиса, установленный пользователем U .

		Items					
		1	2	...	i	...	m
Users	1	5	3		1	2	
	2		2				4
	:		5				
	u	3	4		2	1	
	:					4	
	n			3	2		
		a	3	5	?	1	

Рис. 10.3. Матрица рейтингов $r_{u,i}$, соответствующих пользователю u и объекту i .

Матрица рейтингов является очень разреженной, поскольку большинство пользователей заполняет всего несколько пунктов. Особенностью является то, что под разреженностью матрицы понимается не традиционно математическое определение (большое количество коэффициентов матрицы равно нулю), а отсутствие в большей части ячеек какой-либо информации. Задачей рекомендационной системы является предсказание рейтинга пользователей, то есть заполнение пустых ячеек.

Множество подходов к рекомендующим системам можно классифицировать следующим образом:

- Совместная или коллаборативная фильтрация (CF - Collaborative Filtering): в системах CF, пользователю рекомендуется элементы на основе прошлых коллективных рейтингов всех пользователей.

- Content-ориентированные (CB – Content Based) рекомендации: эти подходы рекомендуют сервисы, аналогичные по содержанию сервисам, которые понравились пользователю в прошлом, или соответствуют предопределенным атрибутам предпочтений пользователя.
- Гибридные подходы: эти методы сочетают содержание приведенных подходов.

Совместная фильтрация

Совместная фильтрация (CF) системы работает путем реализации обратной связи с пользователями в виде оценки элементов данной области и использованием сходства в рейтинге поведения среди нескольких подобных пользователей. Другими словами, CF методы основаны на близости соседей и моделей [121].

Рассмотрим простую модель рекомендующей системы. Близость соседей является основой методов, в случае, если группа пользователей выбирается с учетом их сходства с активным пользователем, и, взвешенное сочетание их рейтингов используется для получения прогнозов этого пользователя.

Такого рода алгоритмы основываются на следующих шагах:

1. Назначение веса для всех пользователей в отношении сходства с активным пользователем.
2. Выбор k пользователей, которые имеют наибольшее сходство с активным пользователем - соседей.
3. Вычисление прогноза от взвешенной комбинации выбранных оценок соседей.

На шаге 1, вес $w_{a,u}$ является мерой сходства между пользователем u и активным пользователем a . Наиболее часто используемой мерой сходства является коэффициент корреляции Пирсона между рейтингами двух пользователей:

$$w_{a,u} = \frac{\sum_{i \in I} (r_{a,i} - \bar{r}_a)(r_{u,i} - \bar{r}_u)}{\sqrt{\sum_{i \in I} (r_{a,i} - \bar{r}_a)^2} \sqrt{\sum_{i \in I} (r_{u,i} - \bar{r}_u)^2}},$$

где I - набор элементов, оцененных пользователями, $r_{u,i}$ рейтинг данного сервиса i пользователем u , и \bar{r}_u - средняя оценка рейтинга пользователя u .

Что касается второго шага, то этому вопросу мы уделили внимание, рассматривая методы кластеризации.

На шаге 3, прогнозы, как правило, рассчитываются как средневзвешенные отклонения от соседей:

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u \in K} (r_{u,i} - \bar{r}_u) \times w_{a,u}}{\sum_{u \in K} w_{a,u}},$$

где $p_{a,i}$ предсказание для активных пользователей a по сервису i , $w_{a,i}$ - сходство между пользователями a и u , и K - соседи или окрестность, набор пользователей, наиболее близких к активному пользователю.

Кроме того, если рассматривать рейтинги пользователей как векторы, то оценить их сходство можно основываясь на косинусе угла между ними:

$$w_{a,u} = \cos(\vec{r}_a, \vec{r}_u) = \frac{\vec{r}_a \cdot \vec{r}_u}{\|\vec{r}_a\|_2 \|\vec{r}_u\|_2} = \frac{\sum_{i=1}^m r_{a,i} r_{u,i}}{\sqrt{\sum_{i=1}^m r_{a,i}^2} \sqrt{\sum_{i=1}^m r_{u,i}^2}}.$$

Заметим, что эмпирические исследования установили, что корреляция Пирсона обычно работает лучше. Есть и другие меры сходства, в том числе среднеквадратическая ошибка, энтропия и пр.

Выбор элемента на основе совместной фильтрации: При применении к миллионам пользователей и сервисов, использование традиционных методов из-за вычислительной сложности не эффективно. В качестве альтернативы, предлагается схема совместной фильтрации элемент-к-элементу, где соответствие определяется не по подобным пользователям, а по рейтингу аналогичных элементов. На практике, этот подход приводит к более быстро реагирующей рекомендационной системе, и часто приводит к улучшению прогнозов пользовательского поведения.

При таком подходе, сходство между парами элементов i и j вычисляется с использованием корреляции Пирсона:

$$w_{i,j} = \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r}_i)^2} \sqrt{\sum_{u \in U} (r_{u,j} - \bar{r}_j)^2}},$$

где U является множеством всех пользователей, которые оценили оба элемента i и j , $r_{u,i}$ рейтинг пользователя u по элементу i , и \bar{r}_i средний рейтинг элемента i между пользователями.

Теперь, рейтинг по элементу i пользователю a может быть предсказан с помощью взвешенной оценки:

$$p_{a,i} = \frac{\sum_{j \in K} r_{a,j} w_{i,j}}{\sum_{j \in K} |w_{i,j}|},$$

где K окрестность множества k элементов по оценке a , наиболее близких к сервису i .

Заметим, что при измерении сходства между пользователями, элементы, которые были оценены всеми (понравилось или не нравится), не так полезны, как менее распространенные элементы. Чтобы учесть этот факт используют понятие обратной частоты пользователя, которое рассчитывается следующим образом $f_i = \log n/n_i$, где n_i количество пользователей, которые оценили элемент i из общего числа n пользователей. Чтобы применить обратную частоту пользователя при использовании сходства на основе CF, оригинальный рейтинг i умножается на f_i . Основанием этого подхода является то, что элементы, которые поголовно понравились, оцениваются чаще, чем другие.

В случае использования больших массивов пользователей и сервисов, высокую эффективность проявили латентные (скрытые) модели и факторизация матричных моделей. В отличие от методов, основанных на локализации окрестности соседей, которые генерируют рекомендации на статистических понятиях сходства между пользователями или между элементами, латентные факторные модели базируются на том, что сходство между пользователями и предметами индуцируется некоторыми скрытыми структурами данных меньшей размерности. Например, рейтинг, который пользователь дает фильму, по всей вероятности, зависит от нескольких неявных факторов, таких как пользовательский вкус в жанрах кино. Матричной факторизацией называется класс удачных моделей скрытых факторов, где пользователи и сервисы представлены в виде неизвестных вектор-функций (вектор-столбцы) $w_u, h_i \in \Re^k$ вдоль k скрытых размерностей. Произведение $w_u^T h_i$ является приближением матрицы известных рейтингов $r_{u,i}$. Для решения этой задачи традиционно используется метод наименьших квадратов

$$J(W, H) = \sum_{(u,i) \in L} (r_{u,i} - w_u^T h_i)^2,$$

где $W = [w_1 \dots w_n]^T$ матрица размера $n \times k$, $H = [h_1 \dots h_m]$ является $k \times n$ матрицей, а L есть множество пар пользователь-элемент, для которых известны рейтинги. В тривиальном случае, когда все рейтинги пар пользователь-элемент известны, целевая функция будет иметь вид

$$J(W, H) = \|R - WH\|_{fro}^2,$$

где R обозначает $n \times m$ матрицу полностью известных пар пользователь-сервис и $\|\bullet\|_{fro}$ — норма Фробениуса. Решение этой задачи дается сингулярным разложением. Однако, в реальных условиях, когда большинство рейтингов пользователь-элемент неизвестно, такое глобально оптимальное решение не может быть получено в явном виде, поэтому требуется оптимизация невыпуклой целевой функции $J(W, H)$. В этом случае целевая функция представляет собой особую форму взвешенных потерь, т.е.

$$J(W, H) = \|S \otimes (R - WH)\|_{fro}^2,$$

где \otimes обозначает поэлементное произведение, и S является бинарной матрицей с коэффициентами, равными единице для известной пары пользователь-элемент L , и нулю в противном случае. Стандартное решение предполагает использование градиентных методов или процедур, подобных итерационной модификации метода наименьших квадратов, когда H определяется при фиксированном W , и, наоборот, пока не достигается критерий сходимости. Подобный подход был рассмотрен нами в разделе, посвященном методу наименьших квадратов.

После нахождения W и H , их произведение WH обеспечивает приближенное восстановления рейтинг-матрицы, откуда рекомендации могут быть считаны непосредственно.

Различный выбор функций цели, регуляризация и дополнительные ограничения модели вызвали большой интерес к методам факторизации матриц. Можно утверждать, что для дискретных рейтингов, квадрат ошибки не самый естественный метод.

Другой класс методов использует факторизацию матрицы при условии неотрицательности ограничений, налагаемых на W и H . Рейтинг поведения каждого пользователя может рассматриваться посредством проявления различных ролей, например, присутствием в группах пользователей связанных интересами или общением, что приводит к функции цели вида

$$J(W, H) = \sum_{(i,u) \in L} r_{u,i} \log \frac{r_{u,i}}{w_u^T h_i} - r_{u,i} + w_u^T h_i.$$

В недавно завершившемся конкурсе Netflix с призом в миллион долларов победил метод факторизации матриц [55]. Окончательную победу принес сложный ком-

плекс различных моделей, в состав которых входит несколько улучшений основных матричных моделей факторизации. К ним относятся:

1. Учет дополнительных конкретных пользователей и конкретных сервисов для учета систематических погрешностей в рейтингах, например, использование того факта, что популярные фильмы в среднем получают более высокие оценки.
2. Использование временной динамики рейтинга

$$J(W, H) = \sum_{(u,i) \in L} (r_{u,i}(t) - b_u(t) - \hat{r} - w_u^T(t)h_i)^2,$$

где t обозначает отсчет времени, и W включает в себя зависящую от времени размерность пространства пользователей.

Во многих случаях, в отличие от рейтингов «нравится - не нравится», доступны только неявные предпочтения, например, записи транзакций, содержащие информацию о продукции, приобретенной клиентами, или навигации клиента на сайте продавца. Эта информация может быть использована для формирования отрицательных примеров, что позволит не предлагать клиентам услуги или товары, в которых они не нуждаются. Такого рода информацию трудно собрать и поддерживать, с учетом быстро меняющейся бизнес-среды. Метод факторизации матриц может быть использован для решения таких задач, если ввести вес $c_{u,i}$, характеризующий степень доверия, который может принимать отрицательные значения

$$J(W, H) = \sum_{(u,i) \in L} c_{u,i} (r_{u,i} - w_u^T h_i)^2.$$

Смесь скрытых профилей. Пусть $Y = \{y_m\}_{m=1}^M$ набор сервисов и $U = \{u_n\}_{n=1}^N$ множество зарегистрированных пользователей. Матрица рейтингов R имеет размерность $N \times M$. Рейтинг установленный пользователем u_n сервису y_m обозначим $r_{n,m}$.

Наблюдаемые рейтинги хранятся в списке $D = \{(u, y, r)\}_{i=1}^L$. Заметим, что не все элементы имеют рейтинг для того или иного пользователя. Опишем вероятностные смеси модели скрытых профилей. Идея заключается в представлении рейтинга поведения пользователей простой порождающей моделью. Рассмотрим две версии вероятностного подхода к смеси. Первая предполагает, что существуют группы пользователей, имеющих по существу тот же поведенческий рейтинг. Вторая модель позволяет каждому пользователю иметь свой собственный профиль рейтинга, но этот профиль

представлен в виде смеси нескольких типичных профилей рейтингов. Заметим, что рейтинг профиля определяется распределением рейтингов на полном наборе сервисов, на основании $P(r, y|\theta)$, где $r = (r_1, r_2, \dots, r_M)^T$ и $y = (y_1, y_2, \dots, y_M)^T$ с значением $y_m = 1$, если элемент m получил рейтинг и $y_m = 0$ в случае $r_m = \emptyset$. При прогнозировании рейтинга, обычно, известно на какой элемент нужно делать прогноз, и, таким образом, достаточно определить условную вероятность рейтинга профиля $P(r|y; \theta)$, считая, что распределение параметризовано по θ .

Вначале рассмотрим рейтинг, построенный на бинарной информации – было ли взаимодействие между пользователем u_n и элементом y_m , или нет. В этом случае рейтинг профиля определяется через $P(y|\theta)$ – вероятность получения (по имеющейся информации) рейтинга для каждого элемента. В этом случае, используя распределение Бернулли, получаем

$$P(y|\theta) = \prod_m \beta_m^{y_m} (1 - \beta_m)^{1-y_m}.$$

Множество параметров $\theta \equiv (\beta_1, \dots, \beta_M)^T$ и $0 \leq \beta_M \leq 1$ определяет вероятность того, что пользователь взаимодействовал с y_m .

Альтернативой распределению Бернулли является полиномиальное распределение. В этом случае существует дополнительное ограничение между параметрами: $\theta \equiv \beta = (\beta_1, \dots, \beta_M)^T$ при условии $\sum_m \beta_m = 1$ и $\beta_m \geq 0$. Тогда вероятность профиля будет определяться следующим образом

$$P(y|\theta) \propto M n(y|\beta, V) = V! \prod_m \beta_m^{y_m} = V! \prod_{y_m=1} \beta_m,$$

где $V = \sum_m y_m$ – число элементов, имеющих рейтинг. Интересно, что, для полиномиального распределения, элементы, не имеющие рейтинга, не вносят непосредственный вклад в вероятность. Это делает процесс вычисления вероятности гораздо более быстрым и более эффективным на больших рейтинг-матрицах. Знак пропорциональности показывает, что проводя нормализацию, необходимо принимать во внимание ограничения $y_m \in \{0,1\}$. Заметим, что этот фактор не влияет на оптимизацию параметров.

В более общем случае полиномиальное распределение подходит для рейтингов, представляющих целые числа $r \in \{0, 1, 2, \dots\}$. Так же, как и в предыдущем случае, мы имеем

$$P(r|\theta) = Mn(r|\beta, \sum_m r_m).$$

Вектор y отсутствует в этом выражении, так как информация об отсутствующих рейтингах уже закодирована в r .

Принцип определения рейтинга профилей остается таким же для явных рейтингов, т.е. когда рейтинги кодируются с учетом мнения пользователей. Явные рейтинги могут принимать значения в упорядоченном множестве и быть либо дискретными $r \in \{1, 2, \dots, r_{\max}\}$, либо непрерывными $r \in [r_{\min}, r_{\max}]$. Отсутствующие значения рейтинга будем ассоциировать с $r = \emptyset$. Рассмотрим сначала простой случай, когда требуется только условное распределение рейтинга $P(r|y;\theta)$. Можно было бы определить профиль, получая рейтинги, например, с гауссовских распределений. Так, если использовать параметризацию $\theta \equiv \{(\mu_m, \sigma_m)\}_{m=1}^M$, получаем вероятность профиля

$$P(r|y;\theta) = \prod_{y_m=1} N(r_m|\mu_m, \sigma_m),$$

где произведение берется только на множестве наблюдаемых оценок. На практике, рейтинги всегда вынуждены принимать значения из ограниченного интервала, и, тогда имеет смысл использовать ограниченные распределения, подобные бета распределению (которое имеет два параметра). Кроме того, если рейтинги могут быть выбраны из дискретного набора, биномиальное распределение или полиномиальное распределения были бы более естественны, чем распределение Гаусса.

Теперь, с учетом схемы прогнозирования, т.е. информации $P(r|y;\theta)$, можно объединить в профиль неявные и явные рейтинги. Стандартным подходом является сочетание полиномиального распределения по недостающей информации и гауссовых распределений для имеющихся значений рейтинга. Следовательно, мы имеем $\theta \equiv \{(\beta_m, \mu_m, \sigma_m)\}$ и вероятность профиля

$$P(r|y;\theta) = P(r|y; \{(\mu_m, \sigma_m)\}) P(y|\beta) \propto \prod_{y_m=1} \beta_m N(r_m|\mu_m, \sigma_m).$$

Заметим, что если целью является только точное предсказание рейтингов пар пользователь-сервис, т.е. определение хорошей оценки $P(r|u,y)$, то, вероятно, будет мало пользы от того, что мы примем во внимание отсутствие рейтингов в модели.

Скрытый профиль кластеризации

Идея скрытого профиля кластеризации состоит в том, что каждому пользователю ставится в соответствие один профиль рейтинга среди множества типичных профилей. Предположим, что существует K моделей профилей рейтингов. Каждый профиль имеет свою собственную параметризацию, следовательно, полный набор профилей параметров есть $\theta = \{\theta_k\}_{k=1}^K$. Пользователю ставится в соответствие вектор индексов $z = [z_1, \dots, z_K]$ и $z_k = 1$ если пользователь соответствует k -му профилю и $z_k = 0$ в противном случае. Тогда вероятность того, что пользователь соответствует профилю, будет иметь вид

$$P(r, y|u; z, \theta) = \prod_k P(r, y|\theta_k)^{z_k}.$$

Content-ориентированные рекомендации

Коллаборационная фильтрация рекомендует использовать только пользовательские рейтинги, что делает возможным получение прогнозов без учета специфики отдельных пользователей или сервисов. Однако, можно сделать более персонализированные рекомендации, зная о пользователе больше, например, имея демографическую информацию о клиенте или о том, какой кино-жанр он предпочитает. Content-основанные рекомендации относятся к подходам, которые обеспечивают предсказание поведения клиента путем сравнения содержания описания элемента (сервиса) с учетом интересов пользователей.

Данная схема отражает общие принципы контент-фильтрации.

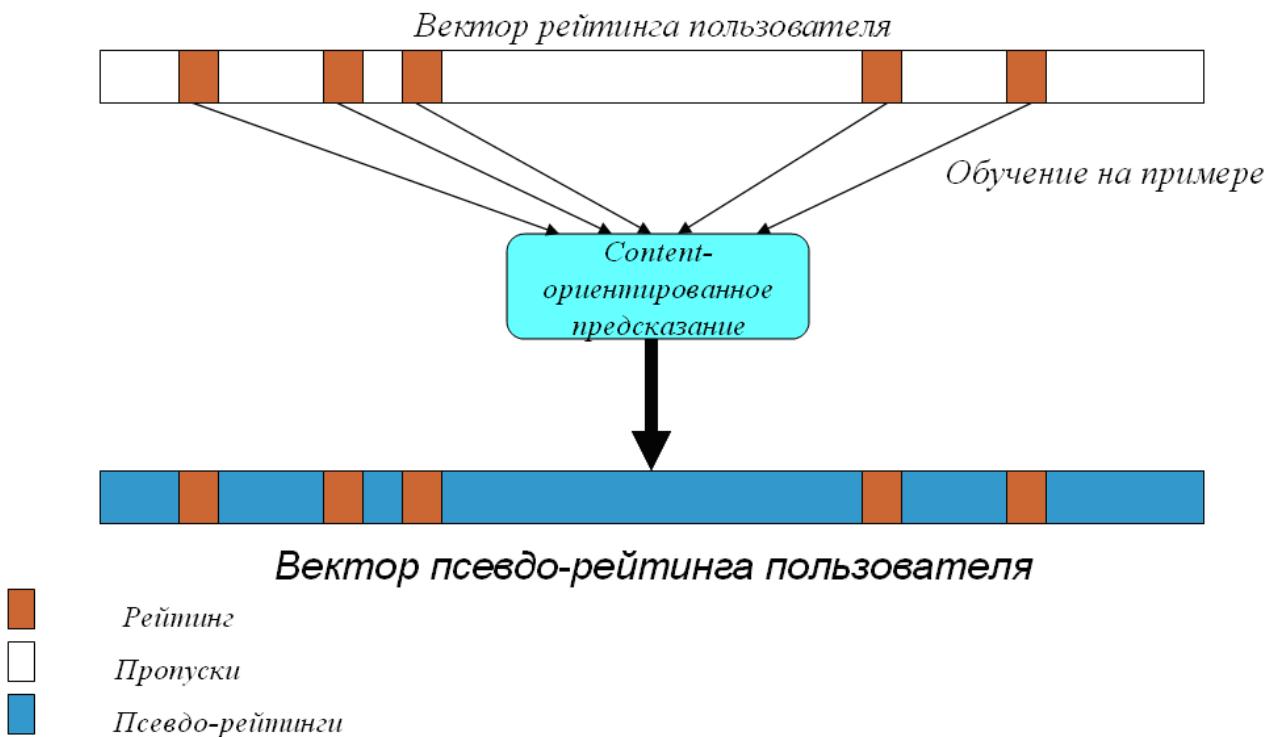


Рис.10.4. Общая схема content-фильтрации.

Многочисленные исследования в этой области сосредоточены на нахождении рекомендаций товаров или услуг с соответствующим текстовым контентом, например, используя веб-страницы товаров или услуг, содержащие описания и отзывы пользователей. Таким образом, можно относиться к этой проблеме, как к задаче поиска информации. Можно относиться к задаче, связанной с пользовательскими предпочтениями, как к информационному запросу, и, рейтинг документа соотносить с релевантностью запроса. Тогда документам в каждой рейтинговой категории ставится в соответствие TF-IDF вектор (см. раздел, посвященный классификаторам), после чего проводится их усреднение, что позволяет получить прототип вектора для каждой категории пользователей. Для классификации нового документа соответствующий вектор сравнивается с прототипом каждой категории. Предсказание рейтинга основано на сравнении значения косинуса угла между этими векторами.

Альтернативой подходов, основанных на информационном поиске, является использование стохастических моделей, например, наивной байесовской классификации, построении дерева решений, нейронных сетей и пр.

Профили пользователей. Профиль интересов пользователя используется в большинстве рекомендационных систем. Этот профиль может состоять из ряда различных типов информации. Мы сосредоточимся на двух типах информации:

1. Модель пользовательских настроек, т.е. описание типов элементов, описывающих интересы пользователей. Есть много возможных альтернативных представлений этого описания, но общим является наличие функции, которая для любого элемента прогнозирует вероятность того, насколько пользователь заинтересован в этом сервисе. Для повышения эффективности рекомендаций, эта функция может быть использована для получения n элементов, которые, скорее всего, представляют интерес для пользователя.
2. Истории взаимодействия пользователя с рекомендациями системы. Это может включать хранение элементов, которые пользователь просматривал вместе с другой информацией о действиях клиента (например, пользователь приобрел товар или рейтинг, указанный пользователем). Другой информацией может быть история запросов, введенных пользователем.

В пользовательских настройках, система рекомендации обеспечивает интерфейс, который позволяет пользователям построить представление о своих собственных интересах. Для этого часто используются флагшки, что позволяет пользователю выбрать то или иное известное значение атрибутов, например, выбрать любимые разделы новостей, или жанр фильмов. В других случаях, html-форма позволяет пользователю вводить слова, например, имя музыканта или автора, которые интересны пользователю. Как только пользователь ввел эту информацию в базу данных, процесс поиска использует их для нахождения элементов, которые отвечают критериям, определенным клиентом, и отображает их пользователю. При этом существует несколько ограничений системы пользовательских настроек. Во-первых, они требуют усилий со стороны пользователя, и нельзя рассчитывать на большое число пользователей, которые проделают эту работу. Во-вторых, настройки системы не обеспечивают способ определения порядка (рейтинга) элементов.

Рекомендационная система, основанная на истории пользователя, должна иметь свои правила рекомендации других продуктов. Например, система может содержать правило, которое рекомендует продолжение книги или фильма для клиентов, которые приобрели ранее книгу или фильм из этой серии. Другое правило может рекомендовать новый компакт-диск для пользователей, которые приобрели ранее диски этого исполнителя. В некоторых ситуациях целесообразно рекомендовать товар поль-

зователю, если он его приобрел ранее, а, в других ситуациях, это не так. Например, система должна по-прежнему предлагать элементы, которые изнашиваются или расходуются, например, такие как лезвия бритвы или картридж, в то же время мало рекомендуя мобильный телефон или WEB-камеру.

Обучения модели пользователя. Создание модели поведения клиента на основе пользовательской истории, является одной из форм обучения рекомендационной системы. Рассмотрим классификацию алгоритмов обучения. Такие алгоритмы являются ключевым компонентом построения системы рекомендаций. Как уже отмечалось, алгоритм обучения включает функцию, которая будет обеспечивать оценку вероятности того, что пользователь оплатит товар (услугу). Эта вероятность может быть использована для сортировки списка рекомендаций. Традиционные алгоритмы машинного обучения предназначены для работы на структурированных данных. При работе с документами, текст сначала преобразуется в структурированные данные, используя небольшое подмножество терминов в качестве атрибутов. Приведем обзор нескольких наиболее популярных алгоритмов.

Дерево принятия решений. Дерево принятия решений строится с использованием рекурсивно секционированных данных, на которых проходит обучение. В случае использования текстов, документы разделяются последовательно на подгруппы до образования тех подгрупп, в которых содержатся только экземпляры одного класса. Как правило, в качестве критерия отбора используется наличие или отсутствие характерного слова или фразы. Деревья решений хорошо изучены при использовании структурированных данных.

Поведение ближайшего соседа. Алгоритм ближайшего соседа просто хранит все этапы и уровни подготовки данных. Для того, чтобы классифицировать новые, непомеченные пункты, алгоритм сравнивает их со всеми сохраненными элементами и, используя сходство, определяет "ближайшего соседа" и сводит рекомендации к действиям ближайших соседей. Использование алгоритма ближайшего соседа зависит от типа данных. Достаточно часто этот алгоритм используется для структурированных данных и евклидовой метрики, или же для векторной модели и значения косинуса угла в качестве критерия схожести. Достаточно эффективно использование

этого алгоритма для сравнения текстов по одной тематике. Часто этот подход используется для персонализации новостей.

Релевантное соответствие и алгоритм Rocchio. Так как при использовании модели векторного пространства, успех поиска документов зависит от способности пользователя создавать запросы, выбирая множество ключевых слов, то методы, которые помогают пользователям постепенно уточнить запросы, основанные на предыдущих результатах поиска, лежат в центре внимания многих исследований. Эти методы обычно называют отношением обратной связи (релевантное соответствие). Алгоритм Rocchio [119] является широко используемым алгоритмом обратной связи. Алгоритм основан на модификации запроса через взвешенные прототипы соответствующих и не соответствующих документов. Подход формирует два прототипа документа - по всем соответствующим и не соответствующим документам, что формулируется следующим образом

$$Q_{i+1} = \alpha Q_i + \beta \sum_{rel} \frac{D_i}{|D_i|} - \gamma \sum_{nonrel} \frac{D_i}{|D_i|}.$$

Здесь Q_i пользовательский запрос на итерации i и α, β, γ параметры, которые контролируют влияние исходного запроса и двух прототипов на изменение в результате запроса. Основная идея состоит в постепенном перемещении вектора запроса в направлении кластеров соответствующих документов и вдали от второстепенных документов.

Линейные классификаторы. Алгоритмы, основанные на использовании линейных границ, т.е. гиперплоскостей, разделяющих множества решений в многомерном пространстве, называются линейными классификаторами (см. раздел, посвященный дискриминантным функциям). Есть большое количество алгоритмов, которые попадают в эту категорию, и многие из них успешно применяются для классификации задач текста.

Вероятностные методы и наивный байесовский классификатор. В отличие от векторной модели, не имеющей достаточного теоретического обоснования, вероятностные подходы классификации в своей основе имеют хорошую теоретическую базу. На данный момент наивный байесовский классификатор (см. раздел, посвящен-

ный классификаторам) признан как исключительно хорошо работающий алгоритм классификации текстов.

Итак, нам нужно найти наиболее вероятную гипотезу $h \in H$ при условии наличия данных D , то есть нужно максимизировать $P(h|D)$. Для достижения этого будем использовать теорему Байеса

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

то есть требуется найти $h = \arg \max_{h \in H} P(h|D) = \arg \max_{h \in H} P(D|h)P(h)$, а если гипотезы равновероятны, то $h = \arg \max_{h \in H} P(D|h)$.

Сравним различные варианты упрощенного алгоритма Байеса - полиномиальную модель и модель Бернулли. Обе модели основаны на предположении, что текстовые документы генерируют вероятностную модель, описываемую смесью

$$P(d_i|\theta) = \sum_{j=1}^{|C|} P(c_j|\theta)P(d_i|c_j; \theta).$$

Здесь, каждому классу c_i соответствует компонент смеси, который параметризуется непересекающимися подмножествами из θ . Как только из обучающих данных получены параметры $\hat{\theta}$, апостериорная вероятность принадлежности тестового документа классу может быть определена в соответствии с теоремой Байеса

$$P(c_j|d_i; \hat{\theta}) = \frac{P(c_j|\hat{\theta})P(d_i|c_j; \hat{\theta})}{P(d_i|\hat{\theta})}.$$

Многомерная модель Бернулли и полиномиальные модели отличаются по способу определения величины $P(d_i|c_j; \theta)$.

Пусть $V = \{w_t\}_{t=1}^{|V|}$ – словарь. Тогда документ d_i – это вектор длины $|V|$, состоящий из битов $B_{i,t}$. $B_{i,t} = 1$ если слово w_t встречается в документе d_i , иначе $B_{i,t} = 0$.

Многомерная формулировка распределения Бернулли для задач классификации текста предполагает, что каждый документ представляется в виде бинарного вектора над пространством всех слов из словаря V . Каждый элемент $B_{i,t}$ этого вектора указывает появляется ли слово хотя бы раз в документе. Следуя наивному Байесу, вероят-

ность каждого слова, входящего в документ не зависит от других слов и $P(d_i | c_j; \theta)$

можно представить в виде произведения:

$$P(d_i | c_j; \theta) = \prod_{i=1}^{|V|} (B_{i,t} P(w_t | c_j; \theta) + (1 - B_{i,t}) (1 - P(w_t | c_j; \theta))).$$

Для обучения такого классификатора нужно обучить вероятности $P(w_t | c_j; \theta)$.

Пусть дан набор документов $D = \{d_i\}_{i=1}^{|D|}$ которые уже распределены по классам c_j , дан

словарь $V = \{w_t\}_{t=1}^{|V|}$ и мы знаем биты $B_{i,t}$ (знаем документы).

Тогда можно подсчитать оптимальные оценки вероятностей $P(w_t | c_j; \theta)$ того, что то или иное слово встречается в том или ином классе (при помощи лапласовой оценки):

$$P(w_t | c_j; \theta) = \frac{1 + \sum_{i=1}^{|D|} B_{i,t} P(c_j | d_i)}{2 + \sum_{i=1}^{|D|} P(c_j | d_i)}.$$

Априорные вероятности классов можно вычислить следующим образом

$$P(c_j) = \frac{1}{|D|} \sum_{i=1}^{|D|} p(c_j | d_i)$$

и классификация будет происходить как

$$\begin{aligned} c &= \arg \max_j P(c_j) P(d_i | c_j; \theta) = \\ &= \arg \max_j \left(\frac{1}{|D|} \sum_{i=1}^{|D|} P(c_j | d_i) \right) \prod_{t=1}^{|V|} (B_{i,t} P(w_t | c_j; \theta) + (1 - B_{i,t}) (1 - P(w_t | c_j; \theta))) \\ &= \arg \max_j \left(\log \left(\sum_{i=1}^{|D|} P(c_j | d_i) \right) + \sum_{t=1}^{|V|} \log (B_{i,t} P(w_t | c_j; \theta) + (1 - B_{i,t}) (1 - P(w_t | c_j; \theta))) \right) \end{aligned}$$

В отличие от бинарного представления документа многомерной моделью Бернулли, полиномиальная формулировка учитывает частоту выпадения слова. Эта модель предполагает, что документы генерируются последовательностью независимых испытаний с полиномиальным распределением вероятностей. Пусть дан набор документов $D = \{d_i\}_{i=1}^{|D|}$, которые уже распределены по классам c_j , дан словарь $V = \{w_t\}_{t=1}^{|V|}$ и мы знаем $N_{i,t}$ число вхождений слова w_t в документе d_i . Опять таки, наивный Байес позволяет определить $P(d_i | c_j; \theta)$ на основании вероятностей встречи отдельных слов:

$$P(d_i | c_j; \theta) = P(|d_i|) \prod_{t=1}^{|d_i|} P(w_t | c_j; \theta)^{N_{i,t}}.$$

При этом, вероятность $P(w_t | c_j; \theta)$ может быть оценена следующим образом:

$$P(w_t | c_j; \theta) = \frac{1 + \sum_{i=1}^{|D|} B_{i,t} P(c_j | d_i)}{|V| + \sum_{s=1}^{|V|} \sum_{i=1}^{|D|} N_{i,s} P(c_j | d_i)}.$$

Априорные вероятности классов можно подсчитать как

$$P(c_j) = \frac{1}{|D|} \sum_{i=1}^{|D|} P(c_j | d_i)$$

Тогда классификация будет происходить как

$$\begin{aligned} c &= \arg \max_j P(c_j) P(d_i | c_j; \theta) = \\ &= \arg \max_j \left(\frac{1}{|D|} \sum_{i=1}^{|D|} P(c_j | d_i) \right) P(|d_i|) |d_i|! \prod_{t=1}^{|V|} \frac{1}{N_{i,t}!} P(w_t | c_j; \theta)^{N_{i,t}} \\ &= \arg \max_j \left(\log \left(\sum_{i=1}^{|D|} P(c_j | d_i) \right) + \sum_{t=1}^{|V|} N_{i,t} \log P(w_t | c_j; \theta) \right). \end{aligned}$$

Опыт показывает, что полиномиальная формулировка Байеса эффективнее многомерной модели Бернулли.

Гибридные подходы. Для того чтобы использовать сильные стороны обоих подходов и получить более эффективную рекомендующую систему, следует использовать гибридные подходы. Один простой подход состоит в том, что на основе контентных и совместных методов фильтрации составляются отдельные списки рекомендаций, а затем объединяются, чтобы получить окончательный список весовым объединением двух прогнозов, где вес возрастает с количеством пользователей, выбирающих данный элемент.

Интересной идеей является преобразование разреженных матриц рейтингов пользователей в заполненные матрицы, а затем, используя методы CF, получение рекомендаций. В частности, можно использовать наивный байесовский классификатор для обучения на документах, описывающих рейтинг каждого пользователя, и, использовать его для заполнения недостающих рейтингов. В результате получаем матрицу псевдо-рейтингов, используя которую, находим соседей, похожих на активного пользователя, и получаем прогнозы с помощью корреляции Пирсона. Существуют другие

гибридные подходы, основанные на совместной фильтрации, но и поддерживающие профиль каждого пользователя.

Качество рекомендующей системы можно оценить путем сравнения рекомендации для тестового набора известных рейтингов пользователей. Наиболее часто используемые метрики - это средняя абсолютная ошибка (MAE-Mean Absolute Error) - определяется как средняя абсолютная разница между прогнозируемым и фактическим рейтингами:

$$MAE = \frac{\sum_{\{u,i\}} |p_{u,i} - r_{u,i}|}{N},$$

где $p_{u,i}$ - это предсказанный рейтинг пользователя u по элементу i , $r_{u,i}$ - фактический рейтинг, и N общее количество рейтингов, полученных в ходе тестирования.

Другой, традиционно используемой метрикой, является среднеквадратичная (RMSE- Root Mean Squared Error), которая делает больший акцент на большие абсолютные ошибки, и определяется по формуле:

$$RMSE = \sqrt{\frac{\sum_{\{u,i\}} (p_{u,i} - r_{u,i})^2}{N}} .$$

Использование этих метрик приводит к рассмотрению всех элементов в равной степени. Однако, для большинства рекомендующих систем, главной задачей является как можно более точное предсказание поведения пользователя. В контексте информационного поиска (IR), выявление хороших на фоне плохих элементов можно рассматривать как различия между "соответствующими" и "имеющими значение" элементами. Здесь могут использоваться другие критерии, например, F-мера сбалансированной ошибки, τ -корреляция Кендалла и пр.

Отметим несколько проблем.

Новинки и новые пользователи создают серьезную проблему для рекомендующей системы. В совокупности эти проблемы называют проблемой «холодного запуска». Первая из этих задач возникает в системе совместной фильтрации, когда сервис не может быть рекомендован, пока какой-нибудь пользователь не оценил его раньше. Это касается не только новинок, но и может скрыть существующие сервисы, что особенно пагубно для пользователей с особенными (экзотичными) вкусами. Проблему новых пользователей вообще трудно решать, поскольку без предварительного пред-

почтения пользователя не представляется возможным найти подобных пользователей или построить профиль.

Другой проблемой является мошенничество. Как только рекомендующие системы коммерческих веб-сайтов начали играть значительную роль в воздействии на рентабельность, так сразу это привело к участию недобросовестных поставщиков в различных формах мошенничества с целью получить от рекомендующих систем результат в их пользу. Как правило, они пытаются надуть популярность собственной продукции (Push атаки) или понизить рейтинги своих конкурентов (Nuke атак). Такие атаки обычно предусматривают создание фиктивных профилей, и используют информацию о работе системы.

Отметим еще одну область применения приведенных методов – это «анализ клиентских сред» (см. [3]).

Анализ клиентских сред

Клиентская среда – это совокупность клиентов (пользователей, субъектов), регулярно пользующихся фиксированным набором сервисов (товаров, ресурсов, предметов, объектов). Предполагается, что действия клиентов протоколируются в электронном виде. Примерами действий являются: использование сервиса или покупка товара, оценивание (рейтингование) сервиса или товара, обращение за информацией, оплата услуг, выбор тарифного плана, участие в маркетинговой акции, получение бонуса от компании, отказ от обслуживания, и т.д.

Анализ клиентских сред, АКС (customer environment analysis, CEA) – это технология обработки протоколов действий клиентов, позволяющая эффективно вычислять взаимно согласованные оценки сходства клиентов и сервисов и использовать их для решения таких бизнес-задач, как автоматизация маркетинговых исследований, формирование направленных предложений клиентам, персонализация сервисов, повышение удовлетворённости и лояльности клиентов, более эффективное привлечение и удержание клиентов.

Технология АКС может быть использована для построения рекомендующих систем (recommender system), персонализации предложений (targeting, direct

marketing) и управления взаимоотношениями с клиентами (customer relationship management, CRM).

Наиболее близкими к АКС направлениями являются коллаборативная фильтрация (collaborative filtering, CF) и анализ соответствий (correspondence analysis). АКС имеет два основных отличия:

- АКС нацелен на получение взаимно согласованных оценок сходства клиентов и сервисов. Клиенты и сервисы рассматриваются как равноправные, двойственные сущности. Любой анализ, сделанный относительно клиентов, может быть перенесён на сервисы, и наоборот. Методы коллаборативной фильтрации, особенно простые, не допускают такой двойственности.
- АКС рассматривает весь комплекс задач и методов, связанных с дальнейшим использованием полученных оценок сходства для визуализации, кластеризации, классификации и прогнозирования поведения клиентов; в конечном итоге - для решения перечисленных выше бизнес-задач. Работы по коллаборативной фильтрации в большинстве случаев ограничиваются узкими постановками задач - предсказания рейтингов или формирования рекомендаций.

Примеры клиентских сред. Клиентские среды возникают в самых разных сферах бизнеса, и не только бизнеса. Можно говорить о клиентских средах производителей товаров, дилерских сетей, сетей супермаркетов, операторов связи, эмитентов пластиковых карт, библиотек, интернет-магазинов, поисковых машин, социальных сетей, форумов, блогов и т. д.

Возможны и такие приложения АКС, в которых сами термины «клиенты» и «сервисы» едва ли применимы, например анализ текстов или анализ результатов парламентских выборов. Однако математические методы обработки данных остаются теми же.

Торговые сети. «Сервисами» являются товары, «клиентами» - постоянные покупатели, имеющие дисконтную карту. «Действия клиентов» - это покупки товаров.

Примеры задач:

- Сделать клиенту направленное предложение тех товаров, которые ему с большой вероятностью понравятся. Персональное предложение может печататься с обратной стороны чека или выводиться на специальном терминале по запросу клиента.

- Вовремя подсказать клиенту, где находится новый товар, о котором мало кто знает, но который с большой вероятностью заинтересует данного клиента.

Операторы сотовой связи. «Сервисами» являются различные услуги (типы соединений), «клиентами» - абоненты сети. «Действия клиентов» - это звонки различных типов (входящие, исходящие, междугородние, международные, SMS, MMS, и т. д.), платежи, подключения и отключения услуг, смены тарифных планов, обращения в сервисный центр, и т. д.

Примеры задач:

- Прогнозирование ухода клиентов (churn prediction), на основе сходства с уже ушедшими клиентами.
- Сегментация клиентской базы, выделение целевых групп клиентов.
- Выявление схожих услуг при формировании пакетных предложений.
- Выявление необычного или потенциально опасного поведения клиентов (fraud detection).

Интернет-магазины книжной, аудио и видео продукции. «Сервисами» являются товары (книги, диски, фильмы, и т. д.), «клиентами» - постоянные покупатели. Действия клиентов - это либо покупки товаров, либо оценки (рейтинги) товаров.

Примеры задач:

- Предсказать рейтинги товаров для данного пользователя и предложить ему список товаров, наиболее интересных для него.
- Предложить персональную скидку на совместную покупку нескольких товаров (cross-selling).
- Вовремя информировать клиента о появлении новых интересных для него товаров (up-selling).

Поисковые машины. «Сервисы» - это страницы или документы, предлагаемые в качестве результатов поиска, «клиенты» - пользователи поисковой машины. «Действия клиентов» - это переходы со страницы результатов поиска к найденному документу. В данном приложении технология АКС примыкает к анализу веба (web mining), точнее, к анализу поведения пользователей веба (web usage mining).

Примеры задач:

- Ранжировать результаты поиска в таком порядке, чтобы в начале списка оказались документы, с большой вероятностью интересные для данного пользователя.

- Разместить на странице адресную рекламу, предлагая данному пользователю посетить сайты, с большой вероятностью интересные именно ему, именно в данный момент.
- Найти для данного сайта список наиболее близких к нему сайтов (например, для автоматической генерации страницы полезных ссылок).
- Найти для данного сайта список сайтов, наиболее близких к нему относительно данного пользователя (для автоматической генерации персонализированного списка рекомендуемых ссылок).

Парламентские выборы. Здесь в роли «сервисов» выступают политические партии, «клиентами» являются субъекты федерации, территориальные избирательные округа или избирательные участки. «Действия клиента» - это голоса избирателей, отданные партиям.

Задачи связаны в основном с интерпретацией результатов выборов:

- Отранжировать партии по сходству относительно любой заданной партии.
- Отранжировать регионы по сходству относительно любого заданного региона.
- Понять и визуализировать (например, с помощью карты сходства) политический спектр партий.
- Выделить схожие партии, «перетягивающие» голоса друг у друга.
- Выделить регионы, в которых данная партия могла бы перетянуть голоса у других партий.

Анализ текстов. В данном случае «сервисами» являются ключевые слова или выражения, «клиентами» - тексты. «Действие клиента» соответствует тому, что данное ключевое слово встречается в данном тексте.

Примеры задач:

- Автоматическая классификация и рубрикации больших объемов текстовых документов или новостных потоков.
- Поиск документов по сходству с данным документом.
- Поиск наиболее полных и релевантных документов по данной теме.

Социальные сети. В простейшем случае «сервисами» являются страницы (записи в блоге, личные страницы пользователей, разделы форума), «клиентами» - пользователи социального сервиса. Действия клиента - посещение страницы, просмотр сообщений, создание собственных сообщений, добавление/удаление друзей, и т. д. Социальные сети являются более сложным примером клиентской среды, поскольку в них приходится применять анализ текстовой информации. В общем случае имеется

уже не два типа взаимосвязанных сущностей (клиенты и сервисы), а три: пользователи, страницы и ключевые слова.

Примеры задач:

- Персональное предложение интересных для данного пользователя страниц, форумов, контактов.
- Автоматическая персонализированная классификация и рубрикация страниц, форумов, контактов.
- Поиск единомышленников (like-minded people), похожих людей (neighbours).

Finis coronat opus

Список литературы

1. Айвазян С.А. Классификация многомерных наблюдений / С.А. Айвазян, Э.Н. Бежаева, О.В. Староверов . – М., 1974 . – 238 с.
2. Айвазян С.А. Прикладная статистика: Основы моделирования и первичная обработка данных / С.А. Айвазян, И.С. Енюков, Л.Д. Мешалкин . – М. : Финансы и статистика, 1983 . – 471 с.
3. Анализ клиентских сред
(http://www.machinelearning.ru/wiki/index.php?title=Анализ_клиентских_сред)
4. Аркадьев А.Г. Обучение машины классификаций объектов / А.Г.Аркадьев, Э.М. Браверманн . – М. : Наука, 1971 . – 192 с.
5. Башмаков А.И. Интеллектуальные информационные технологии: Учебное пособие / А.И. Башмаков, И.А. Башмаков . – М. : Изд-во МГТУ им. Н.Э. Баумана, 2005 . – 304 с.
6. Бериков В.Б. Современные тенденции в кластерном анализе / В.Б. Бериков, Г.С. Лбов (<http://www.ict.edu.ru/ft/005638/62315e1-st02.pdf>)
7. Вапник В.Н. Теория распознавания образов. Стохастические проблемы обучения / В.Н. Вапник, А.Я. Червоненкис . – М. : Наука, 1974.
8. Вапник В.Н. Восстановление зависимостей по эмпирическим данным / В.Н. Вапник . – М. : Наука, 1979.
9. Воронцов К.В. Машинное обучение : курс лекций / К.В.Воронцов
([http://www.machinelearning.ru/wiki/index.php?title=Машинное_обучение_\(курс_лекций,_К.В.Воронцов\)](http://www.machinelearning.ru/wiki/index.php?title=Машинное_обучение_(курс_лекций,_К.В.Воронцов)))
10. Воронцов К.В. Анализ клиентских сред: выявление скрытых профилей и оценивание сходства клиентов и ресурсов / К.В. Воронцов, В.А. Лексин.
(<http://www.machinelearning.ru/wiki/images/4/49/VoronLeksin07mmro.pdf>)
11. Выявление и визуализация метрических структур на множествах пользователей и ресурсов Интернет / К.В. Воронцов, К.В. Рудаков, В.А. Лексин, А.Н. Ефимов // Искусств. Интеллект . – Донецк . – 2006 . – №2 . – С. 285–288.
12. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности / Г.К. Вороновский, К.В. Махотило, С.Н. Петрашев, С.А. Сергеев . – Харьков : Основа, 1997 . – 112 с.
13. Дебок Г. Анализ финансовых данных с помощью самоорганизующихся карт / Г. Дебок, Т. Кохонен . – Изд. Альпина, 2001 . – 317 с.
14. Добров Б.В. Автоматическая рубрикация полнотекстовых документов по классификаторам сложной структуры / Б.В. Добров, Н.В. Лукашевич // Восьмая национальная конференция по искусственному интеллекту. КИИ-2002. 7-12 октября 2002, Коломна . – М. : Физматлит, 2002 . – Т.1 . – С.178-186.

15. Дюран М.Б. Кластерный анализ / М.Б. Дюран . – М. : Финансы и статистика, 1977 . – 128 с.
16. Дягилева А.В. Статистическая модель рубрикации текстов на примере сообщений СМИ / А.В. Дягилева, С.Л. Киселев, Н.В. Сомин // Дистанционное образование . – 1998 . – №7 . – С. 16-21.
17. Журавлëв Ю.И. Об алгебраическом подходе к решению задач распознавания или классификации / Ю.И. Журавлëв // Проблемы кибернетики . – 1978 . – Т. 33 . – С. 5–68.
18. Журавлев Ю.И . Алгоритмы вычисления оценок и их применение / Ю.И. Журавлев . – М. : Фан, 1989 . – 119 с.
19. Журавлев Ю.И. Об алгебраических методах в задачах распознавания и классификации / Ю.И. Журавлев // Распознавание. Классификация. Прогноз. Математические методы и их применение . – М. : Наука, 1989 . – Вып. 1 . – С. 9-16.
20. Интернет-математика 2007: Сборник работ участников конкурса . – Екатеринбург : Изд-во Урал. ун-та, 2007 . – 224 с. (<http://company.yandex.ru/grant>)
21. Каминский Л.С. Измерение связи (корреляция) / Л.С. Каминский . – Л.: Изд-во Ленингр. унив., 1962.
22. Ким Дж.О. Факторный, дискриминантный и кластерный анализ / Дж.О.Ким, Ч.У. Мьюллер, У.Р. Клекка . – М. : Финансы и статистика, 1989 . – 216 с.
23. Классификация и кластер / Под ред. Дж. Вэн Райвин; Пер. с англ. под ред. Ю.И. Журавлева . – М. : Мир, 1978.
24. Кнут. Д. Искусство программирования для ЭВМ / Д. Кнут . – М. : Мир, 1977 . – Т.2 . – 724 с.
25. Коллаборативная фильтрация
[\(\[http://www.machinelearning.ru/wiki/index.php?title=Коллаборативная_фильтрация\]\(http://www.machinelearning.ru/wiki/index.php?title=Коллаборативная_фильтрация\)\)](http://www.machinelearning.ru/wiki/index.php?title=Коллаборативная_фильтрация)
.
26. Лем С. Солярис. Эдем. Непобедимый / С. Лем . – М. : АСТ, 2003.
27. Лигун А.А. Математическая обработка результатов эксперимента : Методические указания / А.А. Лигун, А.Д. Малышева . – Днепродзержинск : ДИИ, 1982 . – 47 с.
28. Лигун А.О. Комп'ютерна графіка (Обробка та стиск зображень) / А.О. Лигун, О.О. Шумейко; ОКВНЗ «П «Стратегія» . – Дніпропетровськ : Видав. Біла К.О., 2010 . – 114 с.
29. Линник В.И. Метод наименьших квадратов и основы математико-статистической теории обработки наблюдений / В.И. Линник . – М. : Физматгиз, 1962.
30. Лоусон Ч.Численное решение задач метода наименьших квадратов / Ч. Лоусон, Р. Хенсон . – М. : Наука, 1986 . – 232 с.
31. Мандель И.Д. Кластерный анализ / И.Д. Мандель . – М. : Финансы и статистика, 1988 . – 176 с.

32. Метод наименьших квадратов / Wikipedia. (http://ru.wikipedia.org/wiki/Метод_наименьших_квадратов)
33. Методы и модели анализа данных: OLAP и Data Mining / А.А. Барсегян, М.С. Куприянов, В.В. Степаненко, И.И. Холод .– «БХВ-Петербург», 2004 .– 336 с.
34. Прикладная статистика: Классификация и снижение размерности / С.А. Айвазян, В.М. Бухштабер, И.С. Енюков, Л.Д. Мешалкин .– М. : Финансы и статистика, 1989 .– 450 с.
35. Райзен Дж. В. Классификация и кластер / Дж. Вэн Райзен // Труды науч.семинара .– М. : Мир, 1980
36. Сеченов Д.А. Методика обработки экспертных оценок / Д.А. Сеченов, А.В. Письменов, М.Д. Скубилин // Технология и конструирование в электронной аппаратуре .– 2000 .– № 2-3 .– С. 36-39.
37. Система команд виртуальной машины AVIDA (<http://alife.ccp14.ac.uk/avida/~charles/avida/manual/cpu.html>).
38. Сотник С.Л. Пространственный кроссовер для задач оптимизации выбора узлов при помощи генетического алгоритма / С.Л. Сотник // Математичне моделювання .– 2007. (http://www.sotnyk.com/Articles/spatial_cross.pdf)
39. Технология АКС-анализ клиентских сред / ЗАО “Форексис” .– 2005. (<http://www.forecsys.ru/cea.php>)
40. Тимохин В.Н. Применение ЭВМ для решения задач распознавания образов / В.Н. Тимохин .– Л. : изд-во Ленингр. Ун-та, 1983 .– 216 с.
41. Ту Дж. Принципы распознавания образов / Дж. Ту, Р. Гонсалес .– М., 1978 .– 410 с.
42. Ферстер Э. Методы корреляционного и регрессионного анализа / Э. Ферстер, Б. Ренц .– М. : Финансы и статистика, 1983 .– 299 с.
43. Фильтрация спама по Байесу. (<http://www.ritlabs.com/ru/solutions/bayesian.php>)
44. Хартиган Дж. А. Задачи связанные с функциями распознавания в кластер-анализе / Дж.А. Хартиган .– М. : Мир, 1989 .– 230 с.
45. Шлезингер М. Десять лекций по статистическому и структурному распознаванию / М. Шлезингер, В. Главач .– Киев : Наукова думка, 2004.
46. Шлезингер М. И. О самопроизвольном различении образов // Читающие автоматы .– Киев : Наукова думка, 1965.
47. Шумейко А.А.Об использовании вероятностных методов распределения служебных слов при проведении экспертизы письменной речи / А.А. Шумейко, С.А. Усенко // Слідча діяльність: проблеми теорії та практики .– Дніпропетровськ : Дніпропетровський державний університет внутрішніх справ, 2008 .– С. 112-117.
48. Шумейко А. А. Об использовании распределения служебных слов при проведении экспертизы письменной речи / А.А. Шумейко, С.Л. Сотник // Інформаційна безпека .– 2009 .– №2 (2) .– С. 96-99.

49. Шумейко А.А. Использование генетических алгоритмов в задачах классификации текстов / А.А. Шумейко, С.Л. Сотник, М.В. Лысак // Системные технологии . – 2009 . – 1(60) . – С. 125-138.
50. Шумейко А.А. Итерационный метод построения векторного классификатора / А.А. Шумейко, С.Л. Сотник // Методи математичного моделювання . – 2009 . – 1(20) . – С. 7-11.
51. Шумейко А.А. Использование агломеративной кластеризации для автоматической рубрикации текстов / А.А. Шумейко, С.Л. Сотник // Системные технологии . – 2011 . – 3(74) . – С. 131-137.
52. Archambeau C. Mixtures of probabilistic principal component analyzers / C. Archambeau, N. Delannay, M. Verleysen // Elsevier, Neurocomputing . – 71 (2008) . – P. 1274-1282.
53. Baker L.D. Distributional Clustering of Words for Text Classification In ACM SIGIR 98 / L.D. Baker, A.K. McCallum . – 1998.
54. Basu C. Recommendation as classification: Using social and content-based information in recommendation / C. Basu, H. Hirsh, W. Cohen // In Recommender System Workshop . – 1998 . – P. 11-15.
55. Bell R. Matrix factorization techniques for recommender systems / R. Bell, Y. Koren, C. Volinsky // IEEE Computer . – 2009 . – 42(8) . – P. 30–37.
56. Berry M. Lecture Notes in Data Mining / M. Berry, M. Browne // Word Scientific . – 2006 . – 220 p.
57. Bezdek J.C. Pattern Recognition With Fuzzy Objective Function Algorithms / J.C. Bezdek . – NY : Plenum Press, 1981.
58. Biesbroek R. Genetic Algorithm Tutorial. 4.1 Mathematical foundations / Robin Biesbroek . – 1999.
59. Boser B.E. A training algorithm for optimal margin classifiers / B.E. Boser, I.M. Guyon, V.N. Vapnik // In D. Haussler, editor, 5th Annual ACM Workshop on COLT . – Pittsburgh, PA : ACM Press , 1992 . – P. 144-152.
60. Brusilovsky P. The Adaptive Web. Methods and Strategies of Web Personalization / P. Brusilovsky, A. Kobsa, W. Nejdl (Eds.) . – Verlag Berlin Heidelberg : Springer, 2007 . – 763 p.
61. Burges C. A Tutorial on Support Vector Machines for Pattern Recognition / C.A. Burges (<http://www.music.mcgill.ca/~rfergu/adamTex/references/Burges98.pdf>)
62. Chakrabarti S. Mining The Web Discovering Knowledge From Hypertext Data / S. Chakrabarti . – Morgan Kaufmann Publishers, 2004.
63. Chan P. Spectral k-way ratio cut partitioning / P.Chan, M. Schlag, J.Zien // IEEE Trans. CAD-Integrated Circuits and Systems . – 1994 . – 13 . – P. 1088-1096.
64. Classification using Naive Bayes. (http://www.resample.com/xlminer/help/NaiveBC/classNB_intro.htm)
65. Cohen William W. Collaborative Filtering / W.W. Cohen (http://www.ccs.neu.edu/home/jaa/CSG339.06F/Lectures/collab_filter.pdf)

66. Collaborative filtering: Fallacies and insights in measuring similarity / P. Symeonidis, A. Nanopoulos, A. Papadopoulos, Y. Manolopoulos // PKDD Workshop on Web Mining . – Berlin, Germany . – 2006.
67. Collaborative filtering (http://en.wikipedia.org/wiki/Collaborative_filtering)
68. Cristianini N. Introduction to support vector machines / N. Cristianini N., J. Shawe-Taylor . – Cambridge University Press, 2000.
69. Dadachova E. Ionizing radiation: how fungi cope, adapt, and exploit with the help of melanin / Ekaterina Dadachova, Arturo Casadevall // Current Opinion in Microbiology . – 2008 . – 11 . – P. 525-531.
70. De Jong K.A. An analysis of the behavior of a class of genetic adaptive systems / K.A. De Jong // Unpublished PhD thesis . – University of Michigan, Ann Arbor, 1975 . – (Also University Microfilms No. 76-9381).
71. De Jong K.A. An Analysis of the Interacting Roles of Population Size and Crossover / K.A. De Jong, W.M. Spears // Proceedings of the International Workshop «Parallel Problems Solving from Nature» . – 1990 . – (PPSN'90).
72. De Jong K.A. A formal analysis of the role of multi-point crossover in genetic algorithms / K.A. De Jong, W.M. Spears // Annals of Mathematics and Artificial Intelligence . – 1992 . – no. 5(1).
73. Deb K. Understanding Interactions Among Genetic Algorithm Parameters / K. Deb, S. Agrawal . – 1998.
74. Delannay N. Collaborative filtering with interlaced generalized linear models / N. Delannay, M. Verleysen // Elsevier, Neurocomputing . – 71 (2008) . – P. 1300-1310.
75. Dempster A. Maximum likelihood from incomplete data via the EM algorithm / A. Dempster, N. Laird, D. Rubin // Journal of the Royal Statistical Society . – (Series B) . – 1977 . – 39(1) . – P. 1–38.
76. Dhillon I. Kernel k-means, spectral clustering and normalized cuts / I. Dhillon, Y. Guan, B. Kulis // Proc. 10th Intl. Conf. on Knowledge Discovery and Data Mining, 2004.
77. Diday E. Clustering analysis / E. Diday, J.C. Simon // In: Digital Pattern Recognition / K. S. Fu, Ed. Springer . – Verlag, Secaucus, NJ . – P. 47–94.
78. Domingos P. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss / P. Domingos, M. Pazzani // Kluwer Academic Publishers, Machine Learning . – 1997 . – 29 . – P. 103–130.
79. Dorigo M. Optimization, Learning and Natural Algorithms / M. Dorigo // PhD thesis . – Italy : Politecnico di Milano, 1992.
80. Fabrizio Sebastiani. Machine Learning in Automated Text Categorization / Fabrizio Sebastiani. (<http://nmis.isti.cnr.it/sebastiani/Publications/ACMCS02.pdf>)
81. Fern X.Z. Clustering ensembles for high dimensional data clustering / X.Z. Fern, C.E. Brodley // In Proc. International Conference on Machine Learning . – 2003 . – P. 186-193.

82. Fisher R.A. The Use of Multiple Measurements in Taxonomic Problems / R.A. Fisher // Annals of Eugenics .– 1936 .– 7 .– P. 179-188.
83. Fraley C. How many clusters? Which clustering method? Answers via model-based cluster analysis / C. Fraley, A. Raftery // The Computer Journal .– 1998 .– 41 .– P. 578–588.
84. Genetic programming. An Introduction / W. Banzhaf, P. Nordin, R. E. Keller, F. D. Francone .– San Francisco, California : Morgan Kaufmann Publishers, Inc., 1998.
85. Ghazanfar M.A. An Improved Switching Hybrid Recommender System Using Naive Bayes Classifier and Collaborative Filtering / Mustansar Ali Ghazanfar, Adam Prugel-Bennett // Proceeding of the International Multi Conference of Engineers and Computer Scientists .– Hong Kong .– Vol 1.
86. Goldberg D.E. A Practical Schema Theorem for Genetic Algorithm Design and Tuning / David E. Goldberg, Kumara Sastry .– 2001.
87. Goldberg D.E. Genetic Algorithms in Search, Optimization and Machine Learning / D.E. Goldberg .– Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
88. Gordon A. Classification / A. Gordon .– Chapman and Hall, London, 1999.
89. Gowda K.C. Agglomerative clustering using the concept of mutual nearest neighborhood / K.C. Gowda, G. Krishna // Pattern Recognition .– 1977 .– V. 10 .– P. 105–112.
90. Cramer N.L. A representation for the adaptive generation of simple sequential programs / N.L. Cramer // Proceedings of an International Conference on Genetic Algorithms and the Applications .– Pittsburg, PA, : Carnegie-Mellon University, 1985 .– P. 183-187.
91. Hartigan J. Clustering algorithms / J. Hartigan .– NY : John Wiley and Sons, 1975.
92. Hofmann T. Latent Semantic Models for Collaborative Filtering / T. Hofmann // ACM Transactions on Information Systems .– January 2004 .– Vol. 22, No. 1 .– P. 89–115.
93. Holland J. H. Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence / J. H. Holland .– London : Bradford book edition, 1994 .– 211 p.
94. Item-based collaborative filtering recommendation algorithms / B.M. Sarwar, G. Karypis, J.A. Konstan, J. Reidl // WorldWideWeb .– 2001 .– P. 285–295.
95. Jain A.K. Data clustering: a review / A.K. Jain, M.N. Murty, P.J. Flynn // ACM Computing Surveys .– 1999 .– V.31, N 3 .– P. 264-323.
96. Jain A.K. Data Clustering / A.K. Jain, M.N. Murty, P.J. Flynn // A Review (<http://www.csee.umbc.edu/nicholas/clustering/p264-jain.pdf>)
97. John G. Estimating continuous distributions in Bayesian classifiers / G. John, P. Langley // Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence .– Montreal, Canada : Morgan Kaufmann, 1995 .– P. 338–345.
98. Journal of Economic Literature Classification System. (www.aeaweb.org/journal.html).

99. Kleinberg J. Two algorithms for nearest-neighbor search in high dimensions / Jon Kleinberg. (<http://citeseer.ist.psu.edu/kleinberg97two.html>)
100. Kogan J. Clustering Large and High Dimensional data / J. Kogan, C. Nicholas, M. Teboulle (<http://www.csee.umbc.edu/nicholas/clustering/tutorial.pdf>)
101. Kohavi R. Scaling up the accuracy of naive-Bayes classifiers: A decision-tree hybrid / R. Kohavi // Proceedings of the Second International Conference on Knowledge Discovery and Data Mining .– Portland, OR : AAAI Press, 1996 .– P. 202–207.
102. Kohavi R. Improving simple Bayes (technical report) / R. Kohavi R., B. Becker, D. Sommerfield; Data Mining and Visualization Group, Silicon Graphics Inc., Mountain View, CA .– 1997 (<ftp://starry.stanford.edu/pub/ronnyk/-impSBC.ps.z>)
103. Kohonen Teuvo. (<http://users.ics.tkk.fi/teuvo/>)
104. Kononenko I. Comparison of inductive and naive Bayesian learning approaches to automatic knowledge acquisition / I. Kononenko, I. // B. Wielinga (Ed.) Current Trends in Knowledge Acquisition .– Amsterdam, The Netherlands : IOS Press, 1990.
105. Kononenko I. Semi-naive Bayesian classifier / I. Kononenko // Proceedings of the Sixth European Working Session on Learning .– Porto, Portugal : Springer-Verlag, 1991 .– P. 206–219.
106. Koza John R. Genetic programming: on the programming of computers by means of natural selection / John R. Koza // A Bradford book .– London : The MIT Press, 1992.
107. Koza J.R. Hierarchical genetic algorithms operating on populations of computer programs / J.R. Koza // Proceedings of the Eleventh International Joint Conference on Artificial Intelligence IJCAI-89 .– San Francisco, CA : Morgan Kaufmann, 1989 .– volume 1 .– P. 768-774.
108. Langley P. An analysis of Bayesian classifiers / P. Langley, W. Iba, K. Thompson // Proceedings of the Tenth National Conference on Artificial Intelligence .– San Jose, CA : AAAI Press, 1992 .– P. 223–228.
109. Lawson C.L. Solving Least Squares Problems (Classics in Applied Mathematics) / C.L. Lawson, R.J. Hanson .– Society for Industrial Mathematics, 1987 .– 350 p.
110. Least-squares method / Wikipedia (http://en.wikipedia.org/wiki/Least_squares)
111. Marlin B. Modeling user rating profiles for collaborative filtering / B. Marlin // Neural Information Processing Systems (NIPS-16) .– MIT Press, 2004.
112. Mercer D.P. Clustering large datasets / D.P. Mercer (<http://ldc.usb.ve/~mcuriel/Cursos/WC/Transfer.pdf>)
113. Mitchell M. An Introduction to Genetic Algorithms / M. Mitchell .– Cambridge, MA : The MIT Press, 1996.
114. Naive Bayes algorithm for learning to classify text. (<http://www-2.cs.cmu.edu/afs/cs/project/theo-11/www/naive-bayes.html>)
115. Pazzani M. Constructive induction of cartesian product attributes / M. Pazzani // Information, Statistics and Induction in Science, 66–77. World Scientific .– 1996.

116. Poncelet P. Data Mining Patterns: New Methods and Applications / P. Poncelet, M. Teisseire, F. Masseglia // Information science reference, Hershey .– New York, 2008 .– 307 p.
117. Reuters-21578 text categorization test collection.
(www.daviddlewis.com/resources/testcollections/reuters21578/)
118. Rocchio J. Relevance Feedback in Information Retrieval, in Salton / J. Rocchio // The SMART Retrieval System: Experiments in Automatic Document Processing .– Prentice-Hall, 1971 .– Chapter 14 .– P. 313-323.
119. Rocchio J. J. Document retrieval systems - optimization and evaluation / J.J. Rocchio // Ph.D. Thesis .– Harvard University, 1966.
120. Salton G. Term-Weighting Approaches. Automatic Text Retrieval / G. Salton, C. Buckley // Information Processing and Management .– 1988 .– 24, 5 .– P. 513-523.
121. Sammut Claude. Encyclopedia of Machine Learning / Claude Sammut, Geoffrey I. Webb (Eds.) .– New York : Springer, 2011 .– 1059 p.
122. Scholarpedia. Kohonen network
(http://www.scholarpedia.org/article/Kohonen_network)
123. Scholkopf B. Kernel Principal Component Analysis, Advances in Kernel Methods-Support Vector Learning / B. Scholkopf, A. Smola, K. Muller .– 1999.
124. Sebastiani F. Machine Learning in Automated Text Categorization / F. Sebastiani // ACM Computing Surveys .– 2002 .– vol. 1 .– P. 1-47.
125. Self-organizing map / Wikipedia (http://en.wikipedia.org/wiki/Self-organizing_map)
126. Shumeyko A.A. Using Genetic Algorithms for Texts Classification Problems / A.A. Shumeyko, S.L. Sotnik // Anale. Seria Informatica. – 2009. – Vol.VII fasc.1 .– P. 325-340.
127. Soraya Rana. Examining the Role of Local Optima and Schema Processing in Genetic Search / Soraya Rana .– 1999.
128. Support Vector machine / Wikipedia
(http://en.wikipedia.org/wiki/Support_vector_machine)
129. Thorsten Joachims “SVM-light” Support Vector Machine”.
(http://www.cs.cornell.edu/People/tj/svm_light/)
130. Tipping M.E. Probabilistic principal component analysis / M.E. Tipping, C.M. Bishop // Journal of the Royal Statistical Society .– 1999 .– B 61 .– P. 611–622.
131. The Copy Number Variation (CNV) Project. (<http://www.sanger.ac.uk/humgen/cnv/>)
132. Vapnik V. The Nature of Statistical Learning Theory / V. Vapnik .– Springer-Verlag, 1999.
133. Veksler O. Course “Pattern Recognition”/ Prof. Olga Veksler; The University of Western Ontario Department of Computer Science (http://www.csd.uwo.ca/faculty/olga/Courses/Winter2006/CS434_654b/index.html)

134. Villa N. Support Vector Machine For Functional Data Classification / N. Villa, F. Rossi // European Symposium on Artificial Neural Networks, Bruges (Belgium), 2005 . – P. 467-472.
135. Whitley D. A Genetic Algorithm Tutorial / Darrel Whitley // Statistics and Computing . – 1994 . – (4).
136. Whitley D. An Overview of Evolutionary Algorithms: Practical Issues and Common Pitfalls / Darrel Whitley // Journal of Information and Software Technology. – 2001.
137. Wu C.F.G. On the convergence properties of the EM algorithm / C.F.G. Wu // The Annals of Statistics. – 1983. – no. 11 . – P. 95–103. (<http://citeseer.ist.psu.edu/78906.html>.)
138. Yang Y. A comparative study on feature selection in text categorization / Y. Yang, J. Pedersen // In: Proc. of ICML-97, 14th International Conf. On machine Learng .– Nashville, USA, 1997 . – P. 412-420.
139. Yang Y. A re-examination of text categorization methods / Y. Yang, X. Liu // Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval / M.A.Hearst, F.Gey, R.Tong (eds.) .– New York, Berkeley : ACM Press, 1999 . – P. 42-49.

Содержание

ПРЕДИСЛОВИЕ К ЭЛЕКТРОННОМУ ИЗДАНИЮ.....	- 3 -
ВВЕДЕНИЕ	- 4 -
1. БАЗОВАЯ ИНФОРМАЦИЯ	- 7 -
Кое-что необходимое из линейной алгебры	- 7 -
Кое-что необходимое из теории вероятностей	- 9 -
2. МЕТОД НАИМЕНЬШИХ КВАДРАТОВ.....	- 20 -
Линеаризация при методе наименьших квадратов.....	- 26 -
3. МЕТОД ГЛАВНЫХ КОМПОНЕНТ	- 32 -
Итерационный алгоритм вычисления главных компонент.....	- 39 -
Оптимальный переход от модели RGB к оптимальной трехкомпонентной модели.....	- 42 -
4. МЯГКИЕ ВЫЧИСЛЕНИЯ В ОБРАБОТКЕ ДАННЫХ	- 46 -
Введение в мягкие вычисления	- 46 -
Эволюционные вычисления	- 48 -
Краткая история.....	- 48 -
Генетический алгоритм	- 51 -
Простой пример реализации ГА	- 54 -
Ближе к реальности, или пространственный кроссовер.....	- 59 -
Генетическое программирование.....	- 71 -
To be, or not to be...	- 74 -
ИНТЕЛЛЕКТ СТАИ.....	- 82 -
Задача коммивояжера и муравьиный алгоритм	- 88 -
5. МЕТОДЫ КЛАСТЕРИЗАЦИИ.....	- 96 -
Кластеризация	- 97 -
Общие понятия.....	- 97 -
ИЕРАРХИЧЕСКИЕ МЕТОДЫ.....	- 100 -
Агломеративные алгоритмы	- 100 -
Дивизимные алгоритмы	- 102 -
НЕИЕРАРХИЧЕСКИЕ АЛГОРИТМЫ	- 103 -
Метод k-средних.....	- 103 -
Fuzzy k-means	- 106 -
Кластеризация Гюстафсона-Кесселя.....	- 107 -
FOREL (ФОРмальный Элемент)	- 107 -

Метод корреляционных плеяд	- 110 -
6. КЛАССИФИКАТОРЫ	112 -
Стохастические классификаторы	115 -
Использование теоремы Байеса (<i>Bayes</i>) для принятия решения	116 -
Наивный байесовский классификатор	119 -
ЕМ-АЛГОРИТМ	130 -
Линейный дискриминантный анализ	137 -
7. ИСПОЛЬЗОВАНИЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ДЛЯ ПОСТРОЕНИЯ ВЕКТОРНОГО КЛАССИФИКАТОРА....	143 -
8. МЕТОД ОПОРНЫХ ВЕКТОРОВ (SVM — SUPPORT VECTOR MACHINES)	152 -
9. ВИЗУАЛИЗАЦИЯ МНОГОМЕРНЫХ ДАННЫХ.....	161 -
Самоорганизующиеся карты Кохонена (SOM)	164 -
10. РЕКОМЕНДУЮЩИЕ СИСТЕМЫ	175 -
Совместная фильтрация	179 -
CONTENT-ОРИЕНТИРОВАННЫЕ РЕКОМЕНДАЦИИ.....	186 -
Анализ клиентских сред	195 -
СПИСОК ЛИТЕРАТУРЫ.....	200 -
СОДЕРЖАНИЕ	209 -