Федеральное государственное бюджетное
образовательное
учреждение высшего профессионального образования
“Тверской государственный университет”
Кафедра информатики

А.М. Дехтярь, М.И. Дехтярь

АЛГОРИТМЫ ИЗВЛЕЧЕНИЯ ЗНАНИЙ ИЗ ДАННЫХ

Учебное пособие

Тверь — 2012

Оглавление

Глава 1

Введение

Настоящие лекции посвящены базовым алгоритмам сравнительно новой области информатики, имеющей два различных англоязычных названия: Knowledge Discovery in Data (KDD) и Data Mining. Первое из них можно перевести как "Извлечение (обнаружение) знаний из данных". Перевод второго сложнее, так как mining буквально означает добычу (раскопку) руды. Его пытались переводить на русский язык как добычу или раскопку данных, информационную проходку данных, интеллектуальный анализ данных и т.п. Однако часто сохраняют и англоязычное название этой области (см. [29, 30, 34]). Некоторые авторы пытаются различить эти области, относя те или иные разделы к одной из них, но мы будем считать оба названия эквивалентными.

Интенсивное развитие Data Mining с первой половины 1990-х годов объясняется быстрым ростом объемов накапливающейся в базах данных (БД) информации и пониманием, что в ней скрыты полезные знания о соответствующих предметных областях.

KDD (Data Mining) является мультидисциплинарной областью, включающей подходы и методы из следующих областей информатики и математики:

- **Базы данных (БД):** в KDD обрабатываются *очень большие наборы данных,* хранящихся, как правило в современных базах данных. Поэтому методы теории и практики БД, связанные с эффективным размещением, хранением и обработкой больших объемов данных в БД используются в KDD.

- **Статистика:** является традиционной областью анализа данных. Она обеспечивает методологию проведения экспериментов и оценки их результатов. Ряд методов статистики используется в качестве базовых *строительных блоков* в процедурах KDD. Многие методы KDD основаны также на методах *теории вероятностей.*

- **Искусственный интеллект (ИИ):** *машинное обучение* является разделом ИИ, который рассматривает алгоритмы извлечения знаний из опыта[1]. Понятия *обучения с учителем* (*классификации*) и *обучения без учителя* (*кластеризации*), являющиеся сейчас важнейшими в *Data Mining*, происходят из машинного обучения и ИИ.

[1] Tom Mitchell, *Machine Learning*, McGraw Hill, 1997.

- **Визуализация:** *сама по себе является мультидисциплинарной областью* , она занимается средствами ясного и понятного пользователям представления информации. Для различных задач, решаемых методами KDD используются различные графические и нерафические формы визуализации знаний, автоматически извлеченных из больших наборов данных.

- **Лингвистика:** *обработка естественного языка* обеспечивает KDD "строительными блоками"для анализа текстовых данных.

- **Алгоритмика:** важна для KDD, поскольку обработка больших объемов данных требует использования специальных структур данных и ряда известных и новых эффективных алгоритмов.

Кратко процесс извлечения знаний из данных можно описать как последовательность трех следующих этапов:

1. **Подготовка данных.** Выбор источников данных, преобразование "сырых"данных в данные подходящего формата, очистка и фильтрация данных.

2. **Обнаружение и извлечение знаний.** На этом этапе к данным, полученным после предыдущего этапа, применяется один из алгоритмов KDD.

3. **Заключительная обработка.** Результаты работы алгоритма KDD анализируются, при необходимости фильтруются, оцениваются и визуализируются.

В известной степени, содержание дисциплины KDD напоминает содержание кулинарной книги. Под одной обложкой в этой области собрано множество различных проблем, связанных лишь идеей поиска интересной информации в больших собраниях данных. Традиционно эти проблемы относят к следующим разделам KDD:

- **Оперативная аналитическая обработка данных (On-Line Analytical Query Processing (OLAP)) и хранилища данных (Data Warehousing)**: расширение приложений БД, ориентированное на выполнение задач анализа больших БД с деловой информацией.

- **Поиск ассоциативных правил.** Поиск ассоциативных образцов в наборах транзакций (*потребительских корзин).*

- **Классификация (обучение с учителем).** Определение того, какому классу (классам) объектов принадлежат входные данные, основанное на ранее полученной информации о существующих классах объектов.

- **Кластеризация (Обучение без учителя).** Анализ совокупности данных с целью выделения в них групп (кластеров), основанного на подобии, "близости" этих данных.

- **Совместная фильтрация и рекомендующие системы (Collaborative Filtering and Recommender Systems).** Выработка рекомендаций и предсказаний на основе сходства образцов, обнаруженного в данных.

- **Информационный поиск (Information Retrieval).** Поиск в совокупности текстовых документов таких документов, которые *релевантны*, т.е. хорошо соответствуют, запросам пользователей.

- **Анализ связей в социальных сетях.** Анализ структуры графов с целью выделения в них "важных" вершин и компонент.

Перечисленные разделы во многом нашли свое отражение в списке 10 наиболее значительных и оказавших наибольшее влияние на развитие Data Mining алгоритмов, составленном на международной конференции IEEE по Data Mining, состоявшейся в декабре 2006г. [27]. Приведем список этих алгоритмов в порядке их "важности", установленном на конференции, с указанием разделов, к которым они относятся:

- *C4.5* – классификация,
- *k-Means* – кластеризация,
- *Support Vector Machine - SVM (метод опорных векторов)* – классификация,
- *Apriori* – поиск ассоциативных правил,
- *Expectation-Maximization - EM (максимизация ожидания)* – кластеризация,
- *PageRank* – анализ связей и ранжирование Web-страниц,
- *AdaBoost* – классификация,
- *kNN (k ближайших соседей)* – классификация,
- *Naive Bayes* – классификация, информационный поиск,
- *Classification and Regression Trees - CART* – классификация.

Эти алгоритмы затем более подробно были изложены в монографии [26]. Они и некоторые их новые варианты составили основу настоящего курса. Мы включили в него также несколько других алгоритмов, играющих важную роль в решении задач перечисленных выше областей Data Mining. Кроме того, два раздела пособия " Совместная фильтрация и рекомендующие системы" и "Основы анализа связей в социальных сетях " содержат основные алгоритмы, относящиеся к этим двум сравнительно новым областям KDD.

Настоящие лекции были подготовлены на основе курсов по алгоритмам для KDD и Data Mining, неоднократно читавшихся в 2005 - 2012 годах в Тверском государственном университете, Университете Кентуки (University of Kentucky, Lexington, США) и Калифорнийском политехническом государственном университете в Сан Луис Обиспо (California Polytechnic State University, San Luis Obispo, США).

Глава 2

Поиск ассоциативных правил

Задача поиска ассоциативных правил или ассоциативных образцов занимает одно из центральных мест в KDD. Исторически, эта задача рассматривалась в контексте учета продаж товаров в предприятиях розничной торговли. Каждый магазин имеет заранее известный конечный ассортимент товаров[1]. Каждый покупатель, приходя в магазин, приобретает определенный набор товаров, оставляя магазину информацию о покупке в виде копии торгового чека. Администрацию магазина или сети магазинов интересуют какие-либо закономерности совместного приобретения товаров. Такими закономерностями, например, могут являться частые покупки вместе определенных сортов хлеба и масла, или пива и вяленой рыбы[2]. Эти закономерности, называемые *ассоциативными правилами*, могут быть использованы в разных целях, направленных на увеличение объемов продаж и доходности магазина; например, для более эффективного размещения товаров на полках магазинов или для определения товаров для проведения рекламных акций. Подобное происхождение этой задачи нашло отражение в используемой терминологии.

В обзоре [5] перечислены более 70 алгоритмов поиска ассоциативных правил, которые появились с 1993 по 2005 гг. В этой главе мы подробно рассмотрим лишь несколько таких алгоритмов: Apriori, AprioriTid, FP-growth, genRules, оказавших наибольшее влияние на всю область. Будут также указаны некоторые идеи и направления их возможных уточнений и расширений.

2.1 Основные определения

Покупательские корзины (Market Baskets) и транзакции. Рассмотрим конечное множество $I = \{i_1, \ldots, i_m\}$. Будем называть элементы этого множества *объектами (items)*.

Покупательская корзина или **транзакция** — это некоторое подмножество t множества I: $t \subseteq I$.

[1] Который, как правило, не меняется, или меняется редко и незначительно.

[2] Бывают и значительно менее очевидные связи между покупаемыми товарами. Один из пионеров использования ассоциативных правил в маркетинговых целях в начале 1990х годов, американская компания Osco Drugs, владеющая сетью аптек, обнаружила, что по четвергам мужчины, приходящие в их магазины часто покупают вместе пиво и детские подгузники.

Совокупность покупательских корзин или транзакций вида $T = \{t_1, \ldots, t_n\}$ ($t_i \subseteq I, 1 \leq i \leq n$) назовем *базой данных (БД) покупательских корзин* или *БД транзакций* над I.

Ассоциативные правила. Пусть I — это множество объектов . **Ассоциативное правило** это *импликация* вида

$$X \longrightarrow Y,$$

где $X \subset I$, $Y \subset I$ и $X \cap Y = \emptyset$.

Содержательно, такое правило можно трактовать как утверждение о том, что *достаточно часто, покупка всех объектов их X сопровождается приобретением всех объектов из Y.*

Поддержка и достоверность ассоциативных правил. Пусть I – множество объектов, а $T = \{t_1, \ldots, t_n\}$ — некоторая БД транзакций над I. Пусть $R : X \longrightarrow Y$ – некоторое ассоциативное правило.

Поддержка (support) правила R в БД T это доля транзакций $t_i \in T$, содержащих $X \cup Y$:

$$Supp_T(X \longrightarrow Y) = \frac{|\{t_i \in T | X \cup Y \subseteq t_i\}|}{n}.$$

Достоверность (confidence) правила R в БД T это доля транзакций $t_i \in T$, содержащих X и Y, среди всех транзакций, содержащих X:

$$Conf_T(X \longrightarrow Y) = \frac{|\{t_i \in T | X \cup Y \subseteq t_i\}|}{|\{t_j \in T | X \subseteq t_j\}|}.$$

Для заданного набора объектов X его **поддержка** определяется как доля транзакций $t_i \in T$, которые его содержат:

$$Supp_T(X) = \frac{|\{t_i \in T | X \subseteq t_i\}|}{n}.$$

- **Поддержка** ассоциативного правила определяет его *охват*: как много транзакций (или какой процент всего набора транзакций) затрагивает это правило. *Разумеется, хотелось бы находить ассоциативные правила с высокой поддержкой, поскольку такие правила будут говорить о* ***типичных, обычно встречающихся*** *транзакциях или покупательских корзинах.*

- **Достоверность** правила определяет его *предсказательную силу*, т.е. то, как часто оно встречается среди затрагиваемых транзакций. *Мы хотим находить ассоциативные правила с* ***высокой достоверностью****, так как такие правила задают прочные связи между группами объектов.*

Задача поиска ассоциативных правил. Для заданных множества объектов I, набора транзакций T и двух чисел: minSup и minConf найти **все ассоциативные правила для** T, имеющие поддержку не менее minSup и достоверность не менее minConf.

Замечание: в такой формулировке задача поиска правил требует выдачи **всех** найденных правил. Имеются также варианты этой задачи, в которых требуется возвращать некоторое подмножество всех обнаруженных правил.

Переборное решение задачи поиска ассоциативных правил. Имеется очевидый переборный алгоритм решения этой задачи:

Алгоритм **ARM_BRUTE_FORCE(T, I, minSup, minConf)**

foreach $X \subseteq I$ **do**
  foreach $Y \subseteq I$ **do**
    if $X \cap Y = \emptyset$ **then**
      $s := Supp(T, X \to Y)$;
      $c := Conf(T, X \to Y)$;
      if $(s > minSup)$ **and** $(c > minConf)$ **then** выдать $(X \to Y)$;
      end_if
    end_if
  endfor
endfor

Оценим сложность этого переборного алгоритма. I содержит m элементов, поэтому внешний цикл выполнит 2^m итераций. Для каждого $X \subset I$, внутренний цикл будет выполняться для $2^{m-|X|}$ различных выборов множества Y, что в среднем дает $2^{\frac{m}{2}}$ итераций. Вычисление функций $Supp()$ и $Conf()$ можно выполнить за полиномиальное время от $|T|$. Отсюда общую сложность алгоритма `ARM_BRUTE_FORCE` можно оценить для некоторого полинома Pol как

$$O(2^{1.5m} \cdot Pol(|T| + |I|)),$$

т.е. время этого алгоритма может быть экспоненциальным по отношению к размеру задачи (обычно размер БД $|T|$ существенно меньше числа всех возможных транзакций 2^m).

2.2 Алгоритм Apriori

2.2.1 Порождение частых наборов алгоритмом Apriori

Алгоритм Apriori, предложенный в 1994г. двумя группами авторов R. Agrawal, R. Sricant [3] и H. Mannila, H.Toivonen, A.I. Verkamo [18], был первым эффективным алгоритмом поиска ассоциативных правил.

Алгоритм Apriori предназначен для поиска часто встречающихся в потребительских корзинах наборов объектов.

Частые наборы. Пусть I — множество объектов, T — БД транзакций. Скажем, что набор $X \subseteq I$ является **частым набором** в T относительно уровня поддержки minSup, если $Supp_T(X) >$ minSup.

```
Алгоритм Apriori(T, I, minSup)
begin
  n := |T|;
  F_1 := {{i}|i ∈ I; Supp_T({i}) ≥ minSup}; //первый проход
  k := 2;
  repeat  //основной цикл
     C_k := candidateGen(F_{k-1}, k − 1);     //кандидаты в частые наборы
     foreach c ∈ C_k do count[c] := 0;  //инициализация счетчиков
     foreach t ∈ T do
        foreach c ∈ C_k do
           if c ⊆ t then count[c] + +;
        endfor
     endfor;
     F_k := {c ∈ C_k | count[c]/n ≥ minSup};
     k := k + 1;
  until F_k == ∅;
  return F := ∪_{i=1}^{k-1} F_i;
end
```

Рис. 2.1: Алгоритм Apriori для поиска частых наборов

Принцип Apriori (называемый также **свойством замкнутости вниз**):

> Если X частый набор в T, то **всякое его непустое подмножество также является частым набором** в T.

На этом принципе основана вся стратегия алгоритма **Apriori**.

В чем польза этого принципа? Всякий алгоритм поиска частых наборов это, по-существу, некоторый алгоритм поиска в пространстве всех возможных наборов. **Принцип Apriori** позволяет во многих случаях исключить из рассмотрения большое число наборов, поскольку, если известно, что множество X *не* является частым набором, то и **всякое надмножество** X также не будет частым набором!

Идея алгоритма Apriori состоит в том, что последовательно строятся все частые наборы размера 1, затем — все частые наборы размера 2, затем — размера 3 и т.д. При этом, при определении частых наборов размера $i+1$ используются ранее построенные частые наборы размера i.

Алгоритм Apriori включает две части. На рис. 2.1 показана основная часть. В ней на каждой итерации вызывается функция candidateGen(), генерирующая кандидатов в частые наборы. Она представлена на рис. 2.2.

Функция candidateGen(). На k-ом этапе алгоритм **Apriori** строит частые наборы размера k. Начиная со 2-го этапа вначале вызывается процедура вычисления функции candidateGen(). На этапе k ей передается построенный на предыдущем этапе список F_{k-1} всех частых наборов размера $k-1$. Далее перебираются объединения всевозможных пар наборов из F_{k-1}, имеющие размер k, и отбраковываются те из них, которые содержат подмножества размера $k-1$, не входящие в F_{k-1}. Нетрудно установить справедливость следующего утверждения.

```
Function candidateGen(F,k)
begin
  C := ∅;
  foreach f1, f2 ∈ F do
    if |f1 ∪ f2| == k + 1 then
      c := f1 ∪ f2;                                  // шаг объединения
      flag := true;
      foreach s ⊆ c, |s| = k − 1, do     // шаг исключения
        if s ∉ F then flag := false;
      endfor;
      if flag == true then C := C ∪ c;
    endif
  endfor;
  return C;
end
```

Рис. 2.2: Порождение кандидатов в частые наборы.

Лемма 2.1. *Для каждого $k \geq 2$ вызов функции candidateGen$(F_{k-1}, k-1)$ возвращает множество наборов C_k, содержащее все частые наборы размера k.*

Доказательство. Во-первых, в основном цикле в качестве кандидатов рассматриваются все частые наборы размера k, так как каждый из них является объединением любых своих двух разных поднаборов размера $k-1$ (по принципу Apriori такие поднаборы также являются частыми, т.е. принадлежат F_{k-1}). Во-вторых, набор c размера k не попадает в список кандидатов только в том случае, когда он содержит некоторый поднабор s размера $k-1$, не входящий в F_{k-1}, т.е. не являющийся частым. Но тогда и c не является частым. Таким образом, возвращаемый результат C_k содержит все частые наборы размера k.

Теорема 2.1. *Алгоритм Apriori по БД транзакций T, множеству объектов I и степени поддержки minSup строит множество всех частых наборов F.*

Доказательство. Достаточно заметить, что для каждого набора $c \in C_k$ после внутреннего цикла значение $count[c]$ равно числу вхождений этого набора в транзакции из T. Тогда в F_k попадают все частые наборы из C_k и, с учетом леммы 2.1, получаем, что F_k состоит из всех частых наборов размера k. Результат F объединяет все такие наборы.

2.2.2 О сложности алгоритма Apriori

Сложность в худшем случае можно оценить как $O(2^N)$, где N — это размер входа. Причина в том, что **в наихудшем сценарии все** 2^m возможных наборов могут оказаться частыми и должны быть рассмотрены алгоритмом.

Однако в "типичных"случаях алгоритм работает достаточно эффективно, поскольку покупательские корзины, как правило, являются **редкими**. Это означает, что на практике относительно немного наборов, особенно больших размеров, могут оказаться частыми.

Сложность относительно данных, оцениваемая числом обращений к базе данных, у алгоритма **Apriori** относительно невелика. Он использует $\min(K+1, m)$ просмотров входного множества транзакций, где K — размер самого большого частого набора.

Рабочая память, требуемая алгоритму **Apriori** невелика. Поскольку каждая транзакция $t \in T$ анализируется независимо от других, то в каждый момент достаточно хранить в оперативной памяти небольшое число транзакций.

Отметим также, что алгоритм **Apriori** можно использовать для поиска частых множеств **заранее ограниченного размера.** Для этого достаточно останавливаться после итерации основного цикла, порождающей все частые множества нужного размера.

2.2.3 Улучшение эффективности алгоритма Apriori

Рассмотрим несколько способов ускорения работы алгоритма **Apriori**.

Хеширование. Метод хеширования может уменьшить размер множества наборов-кандидатов C_k. Например, во время просмотра транзакции $t \in T$ при построении частых наборов размера 1 можно перебрать все подмножества t размера 2 и занести каждое из них в хеш-таблицу. При этом для каждого такого набора ведется подсчет числа вхождений. Наборы, число вхождений которых меньше уровня поддержки, могут удаляться из списка кандидатов.

Часто используется представление множества C_k кандидатов в частые наборы с помощью хеш-дерева. Предполагается, что все транзакции и наборы упорядочены в лексикографическом порядке (они могут рассматриваться как слова в алфавите I). Фиксируется некоторая хеш-функция h на I. Внутренние вершины хеш-дерева содержат хеш-таблицы с указателями на их сыновей. Листья содержат сами наборы-кандидаты из C_k (или указатели на них). Если число наборов-кандидатов в листе v глубины d начинает превосходить некоторый порог, то этот лист становится внутренней вершиной и строки его хеш-таблицы начинают указывать на листья глубины $d+1$. При этом набор $i_1, \ldots, i_d, \ldots, i_k$, находящийся в v, перемещается в лист w, указатель на который находится в строке $h(i_d)$ хеш-таблицы из v. Хеш-таблица для кандидатов из C_k позволяет эффективно обрабатывать очередную транзакцию t в Apriori и обнаруживать все входящие в нее наборы из C_k. Пусть $t = j_1, j_2, \ldots, j_r$. В корне дерева хешируются все объекты t и определяются сыновья корня, на которые указывают ссылки $h(j_1), h(j_2), \ldots, h(j_r)$. Если достигнута внутренняя вершина v по некоторой ссылке $h(j_p)$, то осуществляются переходы к ее сыновьям по ссылкам $h(j_{p+1}), \ldots, h(j_r)$. При достижении листа каждый находящийся в нем набор c проверяется на вхождение в t, и при успешной проверке увеличивается счетчик $count[c]$. Заметим, что если переход в лист осуществлялся по некоторой ссылке $h(j_p)$, то достаточно проверять вхождение в t для объектов из $c \cap \{j_{p+1}, \ldots, j_r\}$. Очевидно, что наборы в листьях, которые не достигаются в этом процессе не входят в t.

Уменьшение числа транзакций. Транзакция, которая не содержит ни одного набора размера k, не может содержать частых наборов размера $(k+1)$. Поэтому ее можно отметить и не рассматривать на последующих итерациях основного алгоритма.

Разбиение данных. На первом этапе все транзакции из T разбиваются на p подмножеств. Для каждого из них находятся все *локально частые наборы*. Каждый глобально частый набор должен быть локально частым хотя бы для одного из подмножеств разбиения. Поэтому объединение всех локально частыых наборов образует множество кандидатов для глобально частых наборов. На втором этапе транзакции просматриваются еще один раз и для каждого из этих кандидатов определяется его "глобальная"поддержка. Размеры

подмножеств, на которые разбивается T, выбираются так, чтобы обработка каждой части могла выполняться в оперативной памяти.

Работа с выборкой транзакций. Здесь идея в том, чтобы случайно выбрать небольшое подмножество S транзакций из T и искать частые наборы в нем. Чтобы не упустить некоторых кандидатов в частые наборы, при их поиске в S уменьшают уровень минимальной поддержки. После получения списка F^S частых наборов в S оставшаяся часть транзакций $T - S$ используется, чтобы найти реальную поддержку выделенных кандидатов из F^S. Для надежности можно выполнить еще один проход по набору транзакций, чтобы выявить частые наборы, пропущенные при первом проходе.

2.2.4 Алгоритм AprioriTid

Алгоритм AprioriTid является одним из вариантов алгоритма Apriori. Его идея состоит в замене на k-ом этапе базы данных T на новое множество данных $\overline{C}_k$. Элементы этого множества имеют вид $< TID, CID >$, где при $k > 1$ CID содержит идентификаторы потенциально частых k-наборов, входящих в транзакцию с идентификатором TID. При $k = 1$, множество $\overline{C}_1$ соответствует исходной базе T (хотя логически каждый объект i заменяется на множество $\{i\}$). Элементом $\overline{C}_k$, соответствующим транзакции t, является $< t.TID, \{c \in \overline{C}_k \mid c \subseteq t\} >$.

Преимущество этого алгоритма над обычным Apriori состоит в том, что не все транзакции из T могут попасть в $\overline{C}_k$ и, кроме того, для попавшей туда $t \in T$ ее список кандидатов $t.CID$ может оказаться меньше чем у Apriori.

Алгоритм AprioriTid представлен на рис. 2.3. В нем предполагается, что в каждом наборе c все объекты упорядочены и $c[k]$ обозначает k-ый элемент этого набора.

Сделаем несколько замечаний о возможной реализации этого алгоритма. Каждое множество $\overline{C}_k$ хранится в списковой структуре. Каждый набор-кандидат $c_k \in C_k$ включает кроме поля с числом вхождений еще два поля: РОДИТЕЛИ и РАСШИРЕНИЯ. Поле РОДИТЕЛИ содержит идентификаторы двух $(k-1)$-наборов, соединением которых получен c_k. Поле РАСШИРЕНИЯ содержит идентификаторы $(k+1)$-наборов-кандидатов, которые расширяют c_k. Поле РОДИТЕЛИ в c_k заполняется при его порождении двумя наборами f_{k-1} и g_{k-1}, тогда же идентификатор c_k добавляется в поле РАСШИРЕНИЯ этих двух наборов. Поле $t.CID$ элемента t из $\overline{C}_k$ содержит идентификаторы всех $(k-1)$-наборов-кандидатов, входящих в $t.TID$. В каждом из этих кандидатов c_{k-1} поле РАСШИРЕНИЯ задает множество идентификаторов T_k всех k-наборов-кандидатов, которые расширяют c_{k-1}. А поле РОДИТЕЛИ каждого $c_k \in T_k$ – идентификаторы двух наборов, породивших c_k. Если эти наборы имеются в $t.CID$, то c_k входит в транзакцию $t.TID$ и поэтому добавляется к C_t.

AprioriTid тратит дополнительное время на вычисление $\overline{C}_k$, но при больших k всё $\overline{C}_k$ обычно помещается в оперативной памяти. Можно заключить, что Apriori работает быстрее на первых проходах при малых k, а AprioriTid более эффективен при больших k. Поэтому естественно объединить эти алгоритмы в один AprioriHybrid, который вначале работает как Apriori, а затем переключается на AprioriTid, когда понимает, что на очередном проходе $\overline{C}_k$ поместится в оперативной памяти.

Алгоритм AprioriTid(T, I, minSup)
begin
$n := |T|$;
$F_1 := \{\{i\} | i \in I; Supp_T(\{i\}) \geq minSup\}$; //первый проход
$\overline{C}_1 := T$; $k := 2$;
repeat //основной цикл
C_k := candidateGen(F_{k-1}, $k-1$); //кандидаты в частые k-наборы
$\overline{C}_k := \emptyset$;
foreach $t \in \overline{C}_{k-1}$ **do**
// определение наборов-кандидатов в C_k, содержащихся
// в транзакции с идентификатором $t.TID$
$C_t := \{c \in C_k \mid (c - c[k]) \in t.CID \ \wedge (c - c[k-1]) \in t.CID\}$;
foreach $c \in C_t$ **do** $count[c]++$ **endfor**;
if $(C_t \neq \emptyset)$ **then** $\overline{C}_k := \overline{C}_k \cup < t.TID, C_t >$
endfor;
$F_k := \{c \in C_k \mid \frac{count[c]}{n} \geq$ minSup$\}$;
$k := k + 1$;
until $F_k == \emptyset$;
return $F := \cup_{i=1}^{k-1} F_i$;
end

Рис. 2.3: Алгоритм AprioriTid для поиска частых наборов

2.3 Порождение и оценка интересности ассоциативных правил

2.3.1 Извлечение правил из частых наборов

Алгоритм Apriori находит все **частые наборы** во входном множестве транзакций. Кроме того, для каждого такого набора c вычислено число $count[c]$ транзкций, содержащих этот набор. Имея частые наборы и их поддерки, несложно породить все ассоциативные правила с заданными значения ми минимальной поддержки minSup и минимальной достоверности minConf.

Идея состоит в том, чтобы для частого набора f размера больше 1 рассмотреть все его разбиения на две части α и $(f - \alpha)$. Тогда правило вида

$$(f - \alpha) \longrightarrow \alpha$$

будет иметь поддержку не меньше minSup. Поэтому достаточно проверить, что степень достоверности этого правила $Conf_T((f - \alpha) \longrightarrow \alpha) = \frac{count(f)}{count(f-\alpha)}$ не меньше minConf. Реализующий эту идею алгоритм представлен на рис. 2.4.

Теорема 2.2. *Алгоритм genRules по множеству всех частых наборов F, значениям их частот count[] и минимальной степени надежности minConf выдает множество H всех ассоциативных правил, имеющих поддержку не менее minSup и достоверность не менее minConf.*

Алгоритм genRules(F, $count[]$, minConf) // F - частые наборы,
//$count[f]$ - число вхождений f в T
begin
$H = \emptyset$;
foreach $f \in F$ такого, что $|f| \geq 2$ **do**
foreach $\alpha \subset f$ **do**
if $Conf_T((f - \alpha) \longrightarrow \alpha) = \frac{count(f)}{count(f-\alpha)} \geq$ minConf **then**
$H := H \cup \{f - \{s\} \longrightarrow \{s\}\}$
endfor
endfor;
return H

Рис. 2.4: Порождение ассоциативных правил по частым наборам.

2.3.2 Оценка интересности правил

Алгоритмы порождения ассоциативных правил часто выдают сотни и тысячи правил, большая часть которых не представляет интереса с практической точки зрения. Например, если у правила $A \to B$ поддержка заключения $Supp(B)$ больше, чем достоверность, т.е. $Conf(A \to B) < Supp(B)$ или $Supp(A \cup B) \leq Supp(A) \cdot Supp(B)|$, то вероятность случайно обнаружить в транзакции набор B больше, чем это предсказывается правилом $A \to B$.

Поэтому для дополнительной оценки интересности правила часто используют величину **подъемной силы (lift)** правила:

$$Lift_T(A \to B) = \frac{Supp_T(A \cup B)}{Supp_T(A) \cdot Supp_T(B)}$$

Если эта величина больше 1, то вероятность случайно обнаружить набор B в T меньше, чем по предсказанию правила $A \to B$, а если — меньше, то наоборот. В последнем случае более разумно правило вида $A \to не B$, так как его улучшение $Lift(A \to не B) > 1$. Но правила с отрицательными заключениями большого практического значения не имеют.

Величина *подъемной силы* достаточно широко используется для выделения интересных правил. Однако, правило, которое имеет небольшую поддержку и высокую подъемную силу, может представлять меньший интерес, чем другое правило с большей поддержкой и меньшей подъемной силой, поскольку это другое правило применимо в большем числе случаев. Пятецкий-Шапиро предложил в 1991г. еще одну меру интересности ассоциативных правил, названную *рычагом (leverage) правила*, которая объединяет объем и силу действия правила:

$$Leverage_T(A \to B) = Supp_T(A \to B) - Supp_T(A) \cdot Supp_T(B).$$

Это величина равна разности между частотой вхождения обеих частей правила в БД и оценкой частоты их совместного вхождения в предложении их взаимной независимости. Чем больше *рычаг* правила, тем более интересным оно представляется.

Очевидно, что правило $A \to B$, условие которого A содержит непустое подмножество A' такое, что $Supp_T(A \to B) = Supp_T(A - A' \to B)$ является излишним. Правило называется *продуктивным (productive)*, если величина его *улучшения (improvement)*, определяемая равенством

$$improvement(A \to B) = Conf_T(A \to B) - \max_{C \subset A}(Conf_T(C \to B)),$$

больше 0.

Для излишних правил улучшение не может быть больше 0, поэтому условие продуктивности устраняет все такие правила. Оно также устраняет правила, в условия которых входят объекты, которые при заданных остальных объектах условий взаимно независимы с заключениями правил.

2.4 Форматы данных для поиска ассоциативных правил

2.4.1 Транзакции как редкие вектора

В типичных случаях при данном (достаточно большом) списке объектов I и списке рассматриваемых **покупательских корзин/транзакций** $T = \{t_1, \ldots, t_n\}$ каждая транзакция содержит немного объектов: $|I| >> |t_i|$.

Представление плотными векторами. Есл I не велико, то БД T можно представить как множество ***плотных двоичных векторов***:

```
0 1 0 0 0 1 0 1
0 0 0 1 1 1 0 1
0 1 0 1 1 1 0 1
1 1 0 0 0 1 0 1
0 1 0 0 0 0 0 1
1 1 0 0 0 0 1 0
```

В этом примере $|I| = 8$. Каждый элемент $t_i \in T$ представляется двоичным вектором длины 8. Например, первый вектор задает транзакцию $\{i_2, i_6, i_8\}$.

Достоинства:

- Регулярность представления.
- Удобно для реляционных БД.

Недостатки:

- Неэффективное использование памяти.

Представление редкими векторами. Если I велико, то плотные вектора будут содержать много нулей и потребуют ненужных затрат при чтении и обработке. В этом случае используется **представление редкими векторами**. Например, приведенное выше множество представляется следующим образом:

```
2,6,8
4,5,6,8
2,4,5,6,8
1,2,6,8
2,8
1,2,7
```

Здесь векторы содержат информацию только о непустых столбцах.

Достоинства:

- Эффективное использование памяти.
- Универсальность.
- Простые операции над векторами выполняются достаточно легко.

Недостатки:

- Не очень подходит для реляционных БД.
- Записи различной длины.

2.4.2 Реляционные БД как базы данных транзакций

БД транзакций представлены выше как двоичные вектора. Однако в реляционных базах данных многие атрибуты ("объекты") не являются двоичными и могут иметь несколько значений.

Пример 2.1. *Пусть* $I = \{CSC365, CSC366, CSC480, CSC437, CSC408, CSC466, CSC481, CSC409\}$ *— перечень 8 предметов, изучаемых на факультете Информатики. Пусть в БД транзакций имеются сведения о предметах, посещенных каждым из 6 студентов. В простом случае — это списки предметов, например:*

Транзакции	Двоичные вектора
$\{CSC365, CSC366, CSC480\}$	1,1,1,0,0,0,0,0
$\{CSC408, CSC409\}$	0,0,0,0,1,0,0,1
$\{CSC365, CSC366, CSC408, CSC409\}$	1,1,0,0,1,0,0,1
$\{CSC480, CSC437, CSC481\}$	0,0,1,1,0,0,1,0
$\{CSC480, CSC481\}$	0,0,1,0,0,0,1,0
$\{CSC365, CSC480, CSC481\}$	1,0,1,0,0,0,1,0

Используя эту БД, можно находить правила, связывающие предметы, выбираемые студентами для изучения. Например, $CSC365 \to CSC366$.

В более сложном случае БД может содержать оценки, полученные студентами по изученным предметам. Предположим, что студент может получить одну из следующих оценок: A,B,C,F. Соответствующие результаты можно представить в следующей **реляционной БД**:

Студент	365	366	480	437	408	466	481	409
1	A	B	B	NULL	NULL	NULL	NULL	NULL
2	NULL	NULL	NULL	NULL	A	NULL	NULL	A
3	C	C	NULL	NULL	B	NULL	NULL	B
4	NULL	NULL	B	C	NULL	NULL	C	NULL
5	NULL	NULL	A	NULL	NULL	NULL	A	NULL
6	C	NULL	B	NULL	NULL	NULL	B	NULL

Для такой БД естественно интересоваться поиском ассоциативных правил вида:

> Студенты с оценкой C по CSC365 имеют обыкновение изучать также CSC408 и получают по нему оценку B .

Преобразование реляционной БД в БД транзакций. Пусть $R = (A_1, \ldots A_s)$ — схема реляционной БД (таблицы). Пусть область каждого атрибута A_i конечна, т.е. $dom(A_i) = \{a_{i1}, \ldots, a_{il}\}$. По заданной схеме R и множеству наборов $T = \{t_1, \ldots, t_n\}$ над ней мы построим множество объектов I_R и БД транзакций $\hat{T} = \{\hat{t_1}, \ldots, \hat{t_n}\}$ следующим образом:

- Множество объектов $I_R = \{(A_1, a_{11}), \ldots, (A_1, a_{1l}), (A_2, a_{21}), \ldots, (A_2, a_{2l}), \ldots, (A_s, a_{sl})\}$.

  Таким образом, каждый объект из I_R представляет пару вида *(имя атрибута, значение)*.

- Набор значений (запись или строка) реляционной таблицы $t = (b_1, b_2, \ldots, b_s)$ преобразуется в двоичный вектор
  $\hat{t} = (x_{11}, \ldots, x_{1l}, x_{21}, \ldots, x_{2l}, \ldots, x_{s1}, \ldots x_{sl})$,
  где $x_{1b_1} = x_{2b_2} = \ldots x_{sb_s} = 1$, а все остальные $x_{ij} = 0$.

Алгоритм Apriori для реляционных БД. После преобразования реляционных данных в БД транзакций можно применять для поиска частых наборов **модифицированный вариант алгоритма Apriori**. В нем следует сделать следующие изменения в процедуре candidateGen() порождения кандидатов:

- При создании списка кандидатов в частые наборы на этапе объединения нужно порождать **только те наборы, в которых для каждого исходного атрибута** $A_i, (1 \leq i \leq s)$ **имеется не более одного столбца** $(A_i, a_j), (1 \leq j \leq il)$ **со значением 1** .

Например, приведенная выше БД с оценками студентов преобразуется в следующую **БД транзакций**.

- Множество объектов I:

$$\begin{aligned} I \;=\; \{&CSC365A, CSC365B, CSC365C, CSC365F,\\ &CSC366A, CSC366B, CSC366C, CSC366F,\\ &CSC480A, CSC480B, CSC480C, CSC480F,\\ &CSC437A, CSC437B, CSC437C, CSC437F,\\ &CSC408A, CSC408B, CSC408C, CSC408F,\\ &CSC466A, CSC466B, CSC466C, CSC466F,\\ &CSC481A, CSC481B, CSC481C, CSC481F,\\ &CSC409A, CSC409B, CSC409C, CSC409F\} \end{aligned}$$

- Шесть записей в реляционной таблице превращаются в шесть двоичных векторов (для удобства мы сгруппировали столбцы, относящиеся к одному предмету):

```
 365       366       480       437       408       466       481      409
1,0,0,0, 0,1,0,0, 0,1,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0
0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 1,0,0,0, 0,0,0,0, 0,0,0,0, 1,0,0,0
0,0,1,0, 0,0,1,0, 0,0,0,0, 0,0,0,0, 0,1,0,0, 0,0,0,0, 0,0,0,0, 0,1,0,0
```

```
0,0,0,0, 0,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,0, 0,0,0,0, 0,0,1,0, 0,0,0,0
0,0,0,0, 0,0,0,0, 1,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 1,0,0,0, 0,0,0,0
0,0,1,0, 0,0,0,0, 0,1,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,1,0,0, 0,0,0,0
```

- В терминах редких векторов эта БД транзакций выглядит так:

```
1,6,10
17,29
3,6,18,30
10,15,27
9,25
3,10,26
```

(Заметим, что в этом случае представление с помощью редких векторов занимает **существенно** меньше места)

Обработка числовых параметров. Приведенное выше преобразование применимо в ситуациях, когда области значений всех атрибутов конечны и сравнительно невелики. Если значения некоторых атрибутов являются *числовыми* или принадлежат *большим дискретным или непрерывным областям*, то эти области нужно вначале **дискретизировать**:

- Если область атрибута A непрерывна (или является большой числовой областью), то процесс **дискретизации** включает выбор **небольшого числа интервалов значений** и замену атрибута A в исходной БД на новый атрибут A_d, значения которого являются интервалами, содержащими соответствующие исходные значения.

Пример 2.2. *Рассмотрим, например, реляционную схему,*

$R = (СтажВФирме,\ Зарплата,\ Должность,\ Отдел),$

описывающую данные сотрудников некоторой фирмы.

Предположим, что *СтажВФирме* принимает целые значения от 0 до 30, а *Зарплата* находится в пределах от 20 000 до 110 000. Пусть также *Должность* может принимать значения {ассистент, кассир, менеджер, ст. менеджер , зав.отделом}, а значениями атрибута *Отдел* могут быть {торговый, производственный, аналитический}. Рассмотрим следующую небольшую таблицу:

СтажВФирме	Зарплата	Должность	Отдел
5	70,000	менеджер	торговый
23	105,000	зав.отделом	аналитический
2	36,000	кассир	производственный
3	60,000	ассистент	аналитический
16	85,000	ст. менеджер	производственный

Вначале дискретизируем атрибуты *СтажВФирме* и *Зарплата*:

СтажВФирме Интервал	Дискретное значение
0 — 3	малый
4 — 10	средний
11 — 20	ветеран
20 — 30	старожил

Зарплата	Интервал	Дискретное значение
20,000 — 39,999		низкая
40,000 — 64,999		ниже-средней
65,000 — 84,999		выше-средней
85,000 — 110,000		высокая

После этого заменим эти два атрибута в БД на Стаж-дискр и Зарплата-дискр:

Стаж-дискр	Зарплата-дискр	Должность	Отдел
средний	выше-средней	менеджер	торговый
старожил	высокая	зав.отделом	аналитический
малый	низкая	кассир	производственный
малый	ниже-средней	ассистент	аналитический
ветеран	высокая	ст. менеджер	производственный

В этой БД имеются четыре различных атрибута, каждый из которых имеет сравнительно небольшую область значений. Далее она может быть преобразована в БД транзакций так, как это было описано выше.

2.5 Поиск частых наборов без порождения кандидатов. Алгоритм FP-tree

2.5.1 Дерево частых наборов

В работе [12] был предложен алгоритм построения частых наборов, не использующий предварительного перечисления кандидатов, который основан на использовании специальной структуры данных — деревьев частых наборов ЧН-деревьев (FN-trees).

Пусть $I = \{i_1, \ldots, i_m\}$ – множество объектов, а $T = \{t_1, \ldots, t_n\}, t_i \subseteq I (1 \leq i \leq n)$ – некоторая БД транзакций. Для данного уровня поддержки minSup частыми называются такие наборы (подмножества) объектов A из I, которые входят в не менее чем $\xi = n \cdot minSup$ транзакций БД T.

Пусть объект i входит в $k(i)$ транзакций из T. Выделим в I лишь частые объекты, т.е. такие $i \in I$, для которых $k(i) \geq \xi$. Упорядочим это множество по убыванию числа вхождений в T и обозначим получившуюся последовательность через $FList$:
$FList = a_1, \ldots, a_j, a_{j+1}, \ldots, a_r$, где $k(a_j) \geq k(a_{j+1}) \geq \xi$ для $1 \leq j < r$. Удалим теперь все нечастые объекты из всех транзакций и получим новую БД транзакций T'. Будем считать, что объекты в транзакциях из T' упорядочены в соответствии с порядком в списке $FList$. Поскольку в частые наборы могут входить лишь объекты из $FList$, то множества частых наборов в T и в T' совпадают. Определим теперь графовую структуру *дерево частых наборов (ЧН-дерево) (FP-tree)* , которая в достаточно сжатом виде будет содержать информацию о транзакциях из T', достаточную для порождения всех частых наборов.

Определение 2.1. ***Дерево частых наборов (или ЧН-дерево)*** *это размеченный граф, имеющий следующую структуру.*

1. Он включает одну вершину-корень, помеченную как "null", некоторое множество объектных префиксных поддеревьев, *корни которых являются сыновьями корня "nul", и таблицу-заголовок, включающую список частых объектов FList.*

2. Каждая вершина в объектных префиксных поддеревьях, кроме обычных для деревьев

ссылок на отца и сыновей, содержит еще три поля: item-name*, с именем объекта этой вершины,* count *– счетчик, хранящий число транзакций T', префиксы которых совпадают с путем в данную вершину, и* node-link *– указатель на следующую вершину в дереве с тем же именем объекта (null, если такой вершины нет).*

3. Каждый элемент в таблице-заголовке состоит из двух полей: item-name *– имя объекта и* head of node-link *– голова списка, которая содержит ссылку на первую вершину дерева, имеющую то же имя.*

Алгоритм построения ЧН-дерева
Вход: БД транзакций T и минимальная поддержка ξ.
Выход: ЧН-дерево (дерево частых наборов).

1. За один проход по БД T определить все частые объекты и их частоту. Содать список $FList$ этих объектов, отсортированный по убыванию их частот.

2. Создать корень ЧН-дерева r и пометить его как "null".

3. Для каждой транзакции $t \in T$ выполнять:
составить список частых объектов t в порядке убыывания их частот, заданном $FList$;
пусть это будет список $[p|P]$;
$insert\_tree([p|P], r)$.

Этот алгоритм использует следующую рекурсивную процедуру вставки объектов транзакции в ЧН-дерево.

Procedure $insert\_tree([p|P], v)$
Вход: список объектов $[p|P]$, вершина ЧН-дерева v.

if у вершины v есть сын w такой, что $w.item\text{-}name = p$
then $w.count := w.count + 1$
else
создать новую вершину w и сделать ее сыном v;
$w.item\text{-}name := p$;
$w.count := 1$;
вставить w в список вершин с именем p
end_if;
if $P \neq \emptyset$ **then** $insert\_tree(P, w)$.

Пример 2.3. *Рассмотрим базу транзакций T_1 представленную в первых двух столбцах следующей таблицы.*

Номер покупки	Покупка (транзакция)	Упорядоченные частые объекты
1	f, a,c, d, g, i, m, p	f, c, a, m
2	a, b, c, f, l, m,o	f, c, a, b, m
3	b, f, h, j, o	f, b
4	b, c,k, s	c, b
5	a, c, e, f, m, n	f, c, a, m

Таблица 2.1: База транзакций T_1

Пусть уровень минимальной поддержки minSup равен 0.6, тогда число транзакций, содержащих частый набор, должно быть не меньше $\xi = 3$. Тогда за первый проход по базе

определится список частых объектов $FList$:
$(f:4),(c:4),(a:3),(b:3),(m:3)$. Остальные объекты имеют меньшую частоту и не могут попасть в частые наборы. В третьем столбце таблицы показаны частые объекты пяти транзакций, упорядоченные по убыванию частоты.

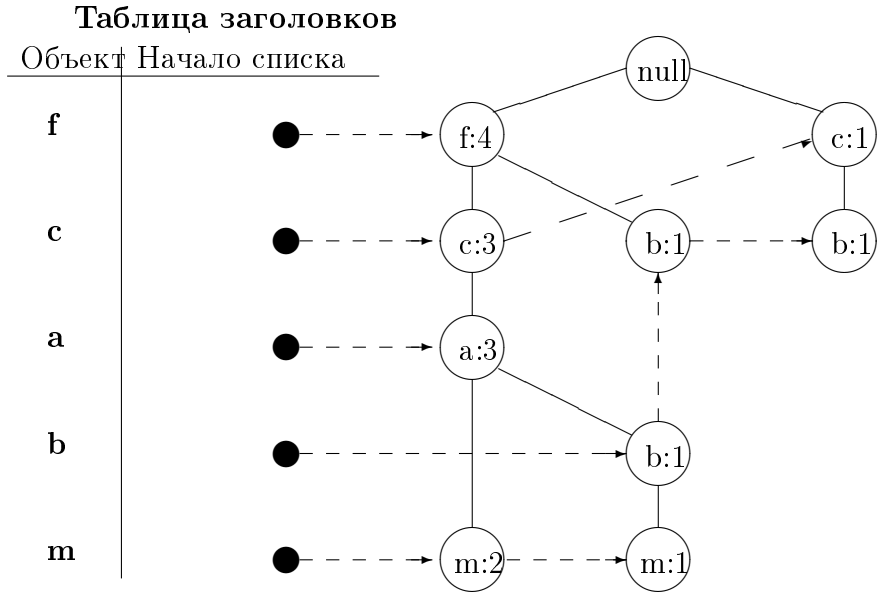

Рис. 2.5: ЧН-дерево для базы транзакций T_1

Замечание о сложности: алгоритм построения ЧН-дерева использует всего два прохода по БД транзакций T. Для вставки одной транзакции $t \in T$ в дерево процедуре $insert\_tree$ достаточно $O(|freq(t)|)$ шагов, где $freq(t)$ – это подмножество частых объектов в t.

Оценим размеры ЧН-дерева.

Лемма 2.2. *Число вершин-объектов ЧН-дерева не превосходит* $\sum_{t\in T}|freq(t)|$, *а высота ЧН-дерева не больше* $\max\{|freq(t)| \mid t \in T\}$.

2.5.2 Алгоритм построения всех частых наборов

Для простоты будем далее ассоциировать пути в ЧН-дереве с именами объектов в их вершинах.

Лемма 2.3. *Пусть* $i_1 i_2 \dots i_k$ *— путь в ЧН-дереве,* $c_k = v.count$ *для последней вершины этого пути* v, c'_k *— сумма значений поля count у всех сыновей вершины* v. *Тогда число вхождений транзакции* $\{i_1 i_2 \dots i_k\}$ *в БД* T *равно* $(c_k - c'_k)$.

Определим *условную базу наборов для* i_k или *базу наборов* $T|\, i_k$ следующим образом: для каждого пути $P = i_1 i_2 \dots i_k$ в ЧН-дереве из корня в вершину v, помеченную объектом $i_k \in I$, поместим в эту базу транзакцию $i_1 i_2 \dots i_{k-1}$ в количестве $v.count$ экземпляров. Построенное по этой базе ЧН-дерево называется *условным* i_k*-ЧН-деревом* или *ЧН-деревом* $|\, i_k$. Отметим, что это дерево содержит всю информацию, необходимую для определения частых наборов, включающих i_k и более частые чем i_k объекты.

Объект	Условная база наборов	Условное ЧН-дерево
m	$\{(fca:2),(fcab:1)\}$	$\{(f:3,c:3,a:3)\}\|m$
b	$\{(fca:1),(f:1),(c:1)\}$	$\emptyset$
a	$\{(fc:3)\}$,	$\{(f:3,c:3)\}\|a$
c	$\{(f:3)\}$	$\{(f:3)\}\|c$
f	$\emptyset$	$\emptyset$

Таблица 2.2: Условные базы наборов

Для объектов базы транзакций T_1 из примера 2.3 условные базы наборов и соответствующие условные ЧН-деревья приведены в таблице 2.2.

Процесс формирования условных баз наборов и соответствуюших условных ЧН-деревьев можно рекурсивно продолжить на наборы α, включающие несколько подряд идущих в ЧН-дереве объектов. Если $\alpha = i, \alpha'$, то *условная базу наборов* $T|\alpha$ получается из $T|\alpha'$ как $(T|\alpha')|i$. По этой базе, как и выше, строится *условное α-ЧН-дерево.*

Лемма 2.4. *Пусть α – набор объектов из БД T, $B = T|\alpha$ – условная база для α и β – некоторый набор объектов из B. Тогда поддержка $(\alpha \cup \beta)$ в T равна поддержке β в B.*

Из этой леммы непосредственно выводится два следующих утверждения.

Следствие 1 (о частых наборах). *Пусть α – набор объектов из БД T, $B = T|\alpha$ – условная база для α и β – некоторый набор объектов из B. Тогда набор $(\alpha \cup \beta)$ является частым набором в T тогда и только тогда, когда β является частым набором в B.*

Следствие 2 (о частых наборах в одной ветви). *Пусть α – набор объектов из БД T и условное ЧН-дерево $D_{T|\alpha}$ представляет из себя одну ветвь P. Тогда множество всех частых наборов T, содержащих α, состоит из всех наборов вида $(\alpha \cup \beta)$, где β – подмножество объектов на ветви P такое, что* $\min\{b.count \mid b \in \beta\} \geq \xi$.

Эти два утверждения лежит в основе алгоритма построения всех частых наборов по ЧН-дереву. Опишем процедуру построения всех частых наборов, содержащих данный набор α по условному α-ЧН-дереву.

Procedure FP-growth$(D_{T|\alpha}, \alpha, \xi)$
Вход: БД транзакций $T|\alpha$, представленная условным α-ЧН-деревом $D_{T|\alpha}$, набор α и минимальная поддержка ξ.
Выход: Полное множество частых наборов, содержащих α .

(1) **if** в $D_{T|\alpha}$ одна ветвь P
(2) **then foreach** подмножества вершин β ветви P
(3) **if** $s = supp(\alpha \cup \beta) = \min\{b.count \mid b \in \beta\} \geq \xi$
(4) **then** выдать набор $(\alpha \cup \beta)$ с поддержкой s
(5) **else foreach** объекта i из $D_{T|\alpha}$
(6) { создать образец $\beta = (\alpha \cup i)$;
(7) построить условную базу наборов $T|\beta$;
(8) построить условное β-ЧН-дерево $D_{T|\beta}$;
(9) **if** $D_{T|\beta} \neq \emptyset$
(10) **then** FP-growth$(D_{T|\beta}, \beta, \xi)$
(11) }

При выполнении этого алгоритма на ЧН-дереве D_{T_1} (см. рис. 2.5) для БД транзакций T_1 из примера 2.3 для объекта m будет построено условное m-ЧН-дерево $D_{T_1|m} = \{(f : 3, c : 3, a : 3)\}$, состоящее из одной ветви. Для него будут выданы в стр. 4 частые наборы $(m, f) : 3, (m, c) : 3, (m, a) : 3, (m, f, c) : 3, (m, f, a) : 3, (m, c, a) : 3, (m, f, c, a) : 3$, содержащие m. Для объекта b его условное b-ЧН-дерево $D_{T_1|b}$ пусто, т.е. не существует частых наборов размера > 1, содержащих b. Для объекта a будет построено условное a-ЧН-дерево $D_{T_1|a} = \{(f : 3, c : 3)\}$, состоящее из одной ветви, из которого в результат попадут наборы $(a, f) : 3, (a, c) : 3, (a, f, c) : 3$. Условное ЧН-дерево для c также состоит из одной ветви: $\{(f : 3)\}$, для которой будет порожден частый набор $(c, f) : 3$. Пустота условного ЧН-дерева для f означает, что новых частых наборов с этим объектом нет.

Теорема 2.3. *Пусть D – это ЧН-дерево, построенное по БД транзакций T и минимальной поддержке ξ. Тогда вызов процедуры FP-growth$(D, \emptyset, \xi)$ завершится выдачей всех частых наборов с указанием их частот.*

Корректность алгоритма FP-growth следует непосредственно из следствий 1 и 2.

Что касается сложности, то она зависит от структуры БД транзакций. Отметим, что в стр. 7 при построении условной базы наборов $T|(\alpha \cup i)$ используется список всех вхождений объекта i в дерево $T|\alpha$. При этом объекты в цикле из стр. 5 - 11 перебираются по таблице заголовков списков в порядке "снизу-вверх т.е. по возрастанию их частоты. Тогда время построения $T|(\alpha \cup i)$ можно оценить как $O(|T|(\alpha \cup i)|)$. Время построения дерева $D_{T|\beta}$ также линейно относительно его размера. При этом в процессе продвижения от более редких объектов к более частым, как правило, размеры деревьев быстро уменьшаются. Эксперименты, проведенные с алгоритмами Apriori и FP-growth, показали, что второй из них дает более компактное представление БД и работает быстрее.

2.5.3 Алгоритм COFI-дерево

Алгоритм COFI-дерево (Co-Occurence Frequent Item Tree), предложенный в работе [6], во многих случаях делает более эффективным порождение всех частых наборов по ЧН-дереву. ЧН-дерево для этого алгоритма должно быть двусвязным, т.е. позволять двигаться по дереву как снизу вверх, так и сверху вниз. Этого нетрудно добиться, произведя небольшие изменения в приведенном выше алгоритме построения ЧН-дерева. Для каждого частого объекта i создается свое i-COFI дерево и по нему порождаются все частые наборы, включающие i.

Определение 2.2. ***i-COFI дерево*** *как и ЧН-дерево включает две структуры: таблицу-заголовок и собственно размеченное дерево.*

1. Каждый элемент в таблице-заголовке состоит из трех полей: item-name *– имя объекта,* item-count *– число вхождений объекта в транзакции совместно с i и* head of node-link *– голова списка, которая содержит ссылку на первую вершину дерева, имеющую то же имя.*

2. Каждая вершина в дереве, кроме обычных для деревьев ссылок на отца и сыновей, содержит еще четыре поля: item-name *– имя объекта этой вершины,* supCount *– счетчик, хранящий число транзакций T', содержащих набор объектов на пути из корня в данную вершину,* participationCount *– счетчик, используемый в процессе порождения частых наборов, и* node-link *– указатель на следующую вершину в дереве с тем же именем объекта (null, если такой вершины нет).*

3. Поле item-name *у корня равно i, а поле* supCount *содержит общее число транзакций, включающих i.*

В таблицу-заголовок i-COFI дерева входят лишь объекты, предшествующие i в списке FList.

Алгоритм COFI-tree-creation построения i-COFI дерева
Вход: ЧН-дерево D, список FList, минимальная поддержка ξ и объект i.
Выход: i-COFI дерево D_i.

1. Используя *FList*, создать таблицу-заголовок для дерева D_i с объектами, предшествующими (более частыми чем) i в *FList*. Все поля *item-count* вначале равны 0, а поля *head of node-link* пусты (равны NIL).

2. Создать корень r i-COFI дерева D_i с *item-name* равным i и нулевыми счетчиками.

3. **for each** v ∈D с именем v.item-name= i **do**
{ k := v.count;
curD := v; // текущая вершина в D
curDi := r; // текущая вершина в D_i
// Вставка в D_i набора на пути из v в корень в D:
while curD $\neq$ *null* **do**
{ j := curD.item-name;
if у вершины curDi есть сын w: w.item-name= j
then w.supCount := w.supCount+k;
else {создать новую вершину w и сделать ее сыном curDi;
w.item-name := j;
w.supCount := k;
w.participationCount := 0;
вставить w в список вершин с именем j
};
j.item-count := j.item-count +k;
curDi := w;
curD := curD.ОТЕЦ
}
}

Пусть D_i – это i-COFI дерево, построенное алгоритмом COFI-tree-creation по ЧН-дереву базы транзакций T. Следующая лемма выделяет основные свойства этого дерева.

Лемма 2.5. *(1) Для каждого объекта j в таблице-заголовке дерева D_i поле j.item-count равно поддержке $Supp_T(\{i,j\})$ набора $\{i,j\}$ в исходной БД T.*
(2) Для любой вершины v дерева D_i, путь в которую из корня помечен объектами $i, j_1, \ldots, j_k$, поле v.supCount равно $Supp_T(\{i, j_1, \ldots, j_k\})$, т.е. поддержке набора $\{i, j_1, \ldots, j_k\}$ в исходной БД T.

Заметим, что из п. (1) этой леммы следует, что те объекты j, для которых в таблице-заголовке значение поля j.item-count меньше минимальной поддержки ξ, не могут входить в частые наборы вместе с i. Поэтому их можно игнорировать при генерации частых наборов по дереву D_i.

Следующий алгоритм по i-COFI дереву и минимальной поддержке ξ строит все частые наборы, включающие объект i.

Алгоритм COFI-tree-mining
Вход: i-COFI дерево D_i, минимальная поддержка ξ.
Выход: Список L_i всех частых наборов, включающих i.
Вспомогательная структура данных: список C кандидатов в частые наборы; его элементы содержат два поля: itemset – набор и Scount – текущая поддержка этого набора.

1. Упорядочить элементы таблицы-заголовка D_i по значениям поля item-count.

2. Для каждого объекта j в этой таблице, у которого j.item-count $< \xi$, используя список с головой j.head of node-link, удалить из D_i все вершины v с $v.item-name = j$, а затем удалить запись j из таблицы.

3. Пусть оставшиеся в таблице-заголовке объекты расположены в порядке: $i_1, \ldots, i_t$.

$C := \emptyset$;
for p=1 **to** t **do**
$j := i_p$;
for each v $\in D_i$ с именем v.item-name= j **do**
{ k := v.supCount - v.participationCount;
if k > 0
then { пусть $w_0 = r, w_1, \ldots, w_l = v$ – путь из корня r в v;
пусть $i, j_1, \ldots, j_l = j$ – имена объектов на этом пути;
for s=0 **to** l **do**
$w_s.participationCount := w_s.participationCount + k$
endfor;
for each $\alpha \subseteq \{j_1, \ldots, j_l\}$ **do**
{$\beta := \alpha \cup \{i\}$;
if $\beta \notin C$
then $C := C \cup (\beta, 0)$
endif;
$\beta.Scount := \beta.Scount + k$
}
endif
}
endfor;
$L_i := \emptyset$;
for each $\beta \in C$ **do**
if $\beta.Scount \geq \xi$
then $L_i := L_i \cup \beta$)
endif
endfor;
return L_i.

Теорема 2.4. *Пусть D_i – это i-COFI дерево, построенное по ЧН-дереву для БД транзакций T и минимальной поддержки ξ. Тогда алгоритм COFI-tree-mining(D_i, ξ) завершится выдачей списка L_i всех частых наборов для T, содержащих объект i и другие объекты из D_i.*

На рис. 2.6 показано m-COFI дерево для базы транзакций T_1, построенное алгоритмом *COFI-tree-creation* по ее ЧН-дереву на рис. 2.5. Для каждой его вершины указаны три поля в формате *item-name*:*supCount*:*participationCount* .

При вызове алгоритма *COFI-tree-mining* будет удалена вершина $b:1:9$, а затем после выполнения внутреннего цикла в п.3 для $j = f$ и вершины с меткой $f:2:0$ список C будет состоять из пар $(\{f,m\},2), (\{c,m\},2), (\{a,m\},2), (\{f,c,m\},2), (\{f,a,m\},2), (\{f,a,c,m\},2)$, а m-COFI дерево примет вид, показанный на рис. 2.7. После этого внутренний цикл будет выполнен для вершины с меткой $f:1:0$ и в списке C у всех элементов поля SCount увеличатся на 1, список пар примет вид $(\{f,m\},3), (\{c,m\},3), (\{a,m\},3), (\{f,c,m\},3), (\{f,a,m\},3)$, $(\{f,a,c,m\},3)$. После этого у каждой вершины v m-дерева разность

$$v.supCount - v.participationCount = 0 .$$

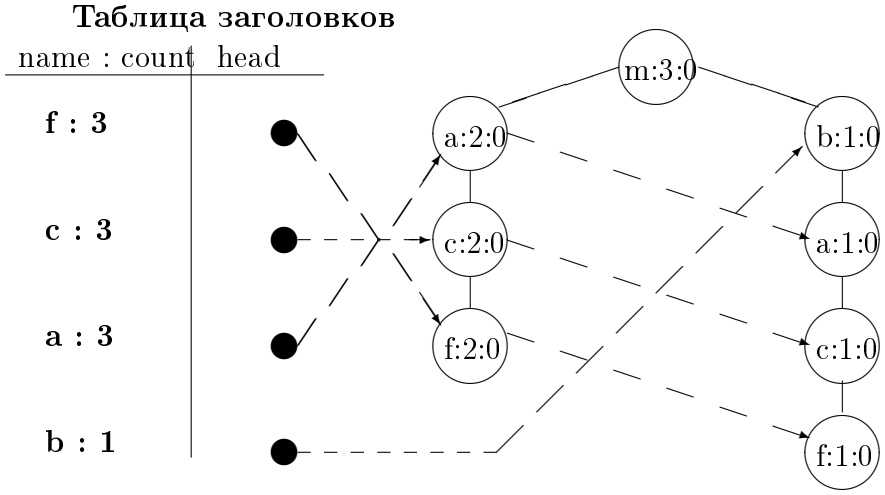

Рис. 2.6: m-COFI дерево для базы транзакций T_1

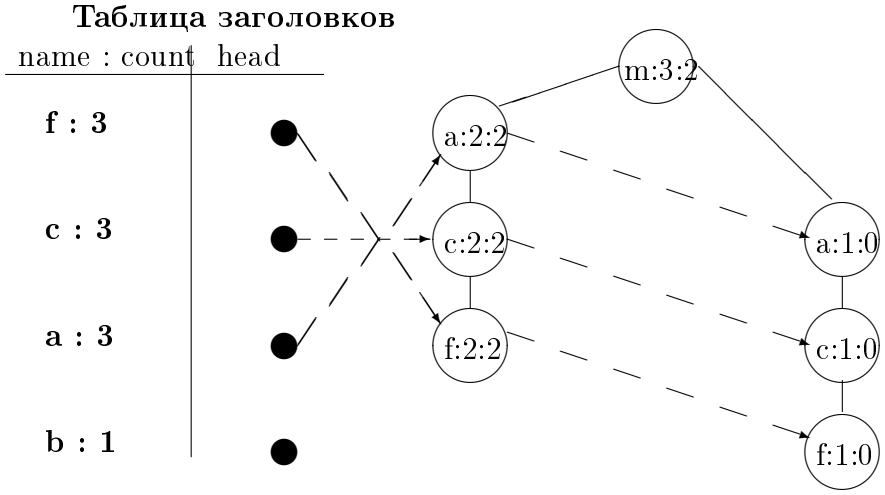

Рис. 2.7: m-COFI дерево после обработки вершины $f : 2 : 0$.

Поэтому дальше список C не меняется и все входящие в него наборы попадут в L_m и далее – в результат.

2.6 Задачи

Задача 2.1. *Доказать, что алгоритм* Apriori *может построить по базе данных транзакций T все частые ней наборы с заданным уровнем поддержки minsup.*

Задача 2.2. *Докажите , что для любых БД T и трех наборов объектов X, Y и Z справедливо следующее соотношение*

$$Supp_T(X \cup Y \cup Z) \geq Supp_T(X \cup Y) + Supp_T(X \cup Z) - Supp_T(X).$$

Задача 2.3. *(а) Доказать, что все непустые подножества частых наборов также являются частыми наборами.*

(b) Доказать, что поддержка любого непустого подмножества A' набора A не меньше поддержки A.

(c) Пусть частый набор A содержит подмножества B и C и $B \subset C$. Доказать, что надежность правила $B \to (A \setminus B)$ не может быть больше надежности правила $C \to (A \setminus C)$.

(d) Один из вариантов Apriori разбивает БД T на p непересекающихся подмножеств $T_1, \ldots, T_p$. Докажите, что каждый частый набор в T должен быть частым набором хотя бы в одном из T_i $(1 \leq i \leq p)$.

Задача 2.4. *В п. 2.3 приведен алгоритм* genRules *порождения ассоциативных правил по частым наборам. Предложите более эффективный алгоритм и объясните, почему он более эффективен. (Указание: используйте пункты (b) и (c) из предыдущей задачи.*

Задача 2.5. *Используя алгоритмы* Apriori *и* AprioriTid, *найдите все частые наборы в БД T, представленной на Табл. 1, и сравните их эффективность. Пусть minsup = 0.3.*

ТАБЛИЦА 1. БД покупок T

N	*Покупка*
T01	*Сыр, Молоко, Яйца*
T02	*Яблоки, Сыр, Апельсины*
T03	*Яблоки, Хлеб, Сыр, Апельсины, Лук*
T04	*Хлеб, Яйца, Апельсины*
T05	*Сыр, Молоко, Лук*
T06	*Яблоки, Сыр, Яйца, Апельсины*
T07	*Хлеб, Сыр, Апельсины, Лук*
T08	*Сыр, Яйца, Лук*
T09	*Хлеб, Сыр, Яйца, Лук*
T10	*Хлеб, Сыр, Лук*

Задача 2.6. *Используя алгоритмы* FP_Tree *и* FP_Growth *построить ЧН-дерево и найти все частые наборы в БД T, представленной на Табл. 1 (при minsup = 0.3). Построить по ним ассоциативные правила с уровнем надежности не менее minconf=0.6. Для каждого из них определите параметры "интересности": подъемную силу (lift), рычаг (leverage) и величину улучшения (improvement).*

Задача 2.7. *База данных T содержит 5 транзакций. Пусть minsup = 60*

ТАБЛИЦА 2. БД покупок T

N	*Покупка*
T100	*M, O, N, K, E, Y*
T200	*D, O, N, K, E, Y*
T300	*M, A, K, E*
T400	*M, U, C, K, E, Y*
T500	*C, O, E, N, K, I*

(a) Используя алгоритмы Apriori и FP_Tree + FP_Growth *найдите все частые наборы в БД T. Сравните эффективность этих двух алгоритмов.*

(b) Перечислите все ассоциативные правила с указанными поддержкой и надежностью, которые на русском языке можно сформулировать в виде:
"Если покупается X и покупается Y, то, как правило, покупается и Z."

Задача 2.8. *Докажите лемму 2.3:*
Пусть $i_1 i_2 \ldots i_k$ — путь в ЧН-дереве, $c_k = v.count$ для последней вершины этого пути

v, c'_k — *сумма значений поля count у всех сыновей вершины v. Тогда число вхождений транзакции* $\{i_1 i_2 \dots i_k\}$ *в БД T равно* $(c_k - c'_k)$.

Задача 2.9. *Докажите лемму 2.4:*
Пусть α *– набор объектов из БД T,* $B = T|\alpha$ *– условная база для* α *и* β *– некоторый набор объектов из B. Тогда поддержка* $(\alpha \cup \beta)$ *в T равна поддержке* β *в B.*

Задача 2.10. *Пусть* D_i *– это i-COFI дерево, построенное алгоритмом COFI-tree-creation по ЧН-дереву базы транзакций T. Докажите лемму 2.5:*
(1) Для каждого объекта j в таблице-заголовке дерева D_i *поле j.item-count равно поддержке* $Supp_T(\{i, j\})$ *набора* $\{i, j\}$ *в исходной БД T.*
(2) Для любой вершины v дерева D_i*, путь в которую из корня помечен объектами* $i, j_1, \dots, j_k$*, поле v.supCount равно* $Supp_T(\{i, j_1, \dots, j_k\})$*, т.е. поддержке набора* $\{i, j_1, \dots, j_k\}$ *в исходной БД T.*

Задача 2.11. *Докажите теорему 2.4:*
Пусть D_i *– это i-COFI дерево, построенное по ЧН-дереву для БД транзакций T и минимальной поддержки* ξ*. Тогда алгоритм COFI-tree-mining*(D_i, ξ) *завершится выдачей списка* L_i *всех частых наборов для T, содержащих объект i и другие объекты из* D_i.

Задача 2.12. *Используя алгоритм* FP_Tree*, построить ЧН-дерево для БД T, представленной на Табл. 1 (при minsup = 0.3). Используя алгоритм COFI-дерево найти все частые наборы. Построить по ним ассоциативные правила с уровнем надежности не менее minconf=0.6. Для каждого из них определите параметры "интересности": подъемную силу (lift), рычаг (leverage) и величину улучшения (improvement).*

Задача 2.13. *Предположим, что большая сеть магазинов имеет распределенную БД, находящуюся в 10 местах. Транзакции, в каждой из этих 10 компонент имеют одинаковый формат:* $T_j : i_1, \dots i_m$*, где* T_j *– это идентификатор транзакции, а* $i_k (1 \le k \le m)$ *– это товары, купленные в этой транзакции. Предложите алгоритм поиска глобальных ассоциативных правил для такого случая. Этот алгоритм не должен требовать передачи всех данных в одно место и, по возможности, должен минимизировать накладные расходы на передачу данных по сети.*

Задача 2.14. *Предположим, что для большой БД транзакций D были определены и запомнены частые наборы (с их поддержкой). Затем к ней добавилось некоторое множество новых транзакций* ΔD*. (Например, к продажам за месяц добавились данные о продажах за день.) Как в этом случае эффективно пересчитать множество ассоциативных правил с теми же параметрами минимальной поддержки и надежности?*

Задача 2.15. *Рассмотренные нами алгоритмы выявления ассоциативных правил работают с покупательскитми корзинами, в которых не учитывается кратность (количество, размер) купленных товаров (например, четыре пирожных, два кг сахара, и т.п.). Как можно находить частые наборы с учетом кратности закупаемых товаров? Это должно позволить генерировать ассоциативные правила вида:*
"Если куплено 3 бутылки красного вина и 2 пачки масла, то будет куплен 1кг сыра".
Предложите модификации алгоритмов Apriori и FP-growth для выявления ассоциативных правил "с кратностью".

Задача 2.16. *Пусть покупательские корзины в БД T имеют формат:* $T_j : i_1, \dots i_m, sum_j$*, где* T_j *– это идентификатор транзакции,* i_k $(1 \le k \le m)$ *– это товары, купленные в этой транзакции, а* sum_j *– это стоимость всей покупки. Предположим, что менеджера интересуют только ассоциативные правила вида:*

"Если куплен товар X, то сумма всей покупки не меньше Y руб."
Предложите эффективный алгоритм для извлечения правил этого вида (с заданными уровнями поддержки и надежности).

Глава 3

Классификация/Обучение с учителем

3.1 Основные понятия

Данные. Рассмотрим множество атрибутов $A = \{A_1, \ldots, A_n\}$ и еще один выделенный атрибут категориального типа(т.е. атрибут, имеющий конечную область значений) C, который назовем **атрибутом класса** или **атрибутом категории**.

Пусть $dom(C) = \{c_1, \ldots, c_k\}$. Каждое значение c_i будем называть **меткой класса** или **меткой категории.**

Обучающий набор данных — это некоторая реляционная таблица D, каждая строка которой содержит значения атрибутов некоторого объекта.

Можно рассматривать таблицы двух видов:

1. **Обучающая выборка данных (training (data)set)**. Таблица D имеет схему $(A_1, \ldots, A_n, C)$, т.е. *для каждого объекта задана метка его класса.*
2. **Тестовая выборка данных (test (data)set)**. Таблица D имеет схему $(A_1, \ldots, A_n)$, т.е. *метки классов, которым принадлежат объекты, представленные в D, неизвестны.*

Задача классификации. Для заданной (обучающей) выборки данных D построить **функцию классифиции/предсказания** , которая правильно определяет метку класса для каждой записи из D.

В качестве синонимов термина **функция классифиции** используются также термины **функция предсказания, модель классифиции** или просто **классификатор (classifier)**.

Обучение с учителем (supervised learning) соответствует первой форме таблиц, поскольку в этом случае обучающая выборка содержит метки классов, проставленные "учителем". При этом имеется возможность сравнивать результаты предсказаний полученного классификатора с метками учителя.

3.2 Методы классификации

Для решения задачи классификации был предложен целый ряд методов. Перечислим основные из них.

Деревья решений. Представляют древообразные классификаторы. (главное достоинство – хорошая читабельность!)

Ассоциативные правила. Ассоциативные правила, заключения которых являются метками классов.

Naïve Bayes. Оценивается вероятность того, что некоторая запись принадлежит каждому из классов.

Метод опорных векторов (Support Vector Machines (SVM)). Линейные (и нелинейные) модели для различения двух классов.

Нейронные сети. Графические модели, которые, исходя из обучающей выборки, строят "выделяющую классы функцию".

В этом разделе мы рассмотрим алгоритмы, относящиеся к первым трем из указанных методов.

3.3 Деревья решений

Впервые деревья решений были предложены Ховилендом и Хантом (Hoveland, Hunt) в конце 50-х годов прошлого века. Самая ранняя и известная работа Ханта и др., в которой излагается суть деревьев решений - "Эксперименты в индукции"("Experiments in Induction") - была опубликована в 1966 году.

Классификаторы, основанные на деревьях решений, являются **простыми** и **эффективными**.

Деревья решений. Пусть $A = \{A_1, \ldots, A_n\}$ — набор атрибутов и C — метки классов. Пусть $dom(C) = \{c_1, \ldots, c_k\}$. **Дерево решений** над A и C — это размеченное дерево $T = \langle V, E \rangle$ такое, что

1. Каждая **внутренняя вершина** $v \in V$ помечена некоторым атрибутом $A_i \in A$.
2. Каждый **лист** $v_l \in V$ помечен некоторой меткой класса $c_i \in dom(C)$.
3. Каждое ребро $e = (v, v')$, у которого метка начала $label(v) = A_i$, помечено некоторым значением $a \in dom(A_i)$, причем ребра, выходящие из одной вершины, помечены разными значениями.
4. Ни один из атрибутов $A_i \in A$ не встречается на одной ветви более одного раза.

В качестве классификатора **дерево решений** можно использовать следующим образом:

- Рассмотрим запись $t = (a_1, a_2, \ldots, a_n)$.
- Начнем с корня r дерева решений T. Пусть $label(r) = A_i$. Найдем ребро $e = (r, v)$ такое, что $label(e) = t(A_i) = a_i$, и перейдем в вершину v.

- Для каждой из далее посещенных внутренних вершин v будем выходить по ребру, помеченному значением атрибута $label(v)$ из записи t.
- При достижении листа l, его метка $label(l)$ и будет классом $class(t)$ записи t.

3.4 Алгоритм C4.5: построение деревьев решений

3.4.1 Алгоритм C4.5 – общая схема

Алгоритм C4.5 был первоначально предложен Куинленом (Quinlan) в 1984г.([22]).

Вход/Выход **Алгоритм C4.5** имеет три входных параметра и выдает дерево решений:

Имя	I/O	Объяснение
D	вход	обучающая выборка
A	вход	список атрибутов
threshold	вход	порог прироста информации
T	выход	построенное дерево решений

Идея алгоритма. **Алгоритм C4.5** является рекурсивным алгоритмом, определяемым по индукции. У него имеется три вида основных шагов:

1. Проверка условий остановки. Алгоритм может остановиться по двум причинам:
   (a) все записи в D принадлежат одному и тому же классу c. В этом случае алгоритм создает дерево, состоящее из одной вершины, и приписывает ей метку класса c.
   (b) $A = \emptyset$: не осталось требующих рассмотрения атрибутов. В этом случае создается дерево, состоящее из одной вершины, и ей приписывается метка с именем класса, в который входит максимальное число записей из D.
2. Выбор атрибута для разбиения. Алгоритм выбирает атрибут A_i, который используется для разбиения выборки. В некоторых ситуациях, выбрать хороший атрибут для разбиения не удается. Тогда алгоритм работает так же, как на шаге 1.(b). Если атрибут выбран, алгоритм переходит к шагу 3.
3. Построение дерева. Алгоритм работает следующим образом:
   (a) Создает вершину дерева r, помеченную A_i.
   (b) Разбивает выборку D на $|dom(A_i)|$ подмножеств $D_1, \ldots D_{|dom(A_i)|}$ в соответствии со значениями атрибута A_i, и рекурсивно вызывает себя для каждого подмножества D_j с уменьшенным множеством атрибутов $A - \{A_i\}$.
   (c) Создает $|dom(A_i)|$ ребер из r в корни деревьев $T_1, \ldots, T_{|dom(A_i)|}$, построенных в результате рекурсивных вызовов. Помечает каждое из этих ребер соответствующим значением из $dom(A_i)$.
   (d) Возвращает построенное дерево.

3.4.2 Алгоритм C4.5 – псевдокод

Псевдокод **Алгоритма C4.5** приведен на рис. 3.1.

```
Алгоритм C45(D, A, T, threshold);
begin
        // Шаг 1: проверка условий останова
 if for all d ∈ D: class(d) = c_i then
    создать лист r;
     label(r) := c_i;
     T := r
 else if  A = ∅  then
    c := наиболее_частый_класс(D);
    создать лист r;
    label(r) := c
 else  //Шаг 2: выбор атрибута для разбиения
    A_g := selectSplittingattribute(A,D, threshold);
     if  A_g = NULL  then   //нет подходящего для разбиения атрибута
       создать лист r;
        label(r) := c_i;
        T := r
     else   // Шаг 3: Построение дерева
        создать вершину r;
        label(r) := A_g;
        foreach  v ∈ dom(A_g) do
           D_v := {t ∈ D | t[A_g] = v};
           if  D_v ≠ ∅  then
              C45(D_v, A − {A_g}, T_v, threshold);      //рекурсивный вызов
              связать  r  с корнем T_v  ребром, помеченным  v
           endif
       endfor
    endif
 endif
end
```

Рис. 3.1: Алгоритм C4.5 индуктивного построения дерева решений.

```
function selectSplittingattribute(A,  D,threshold);      //использует прирост
информации
begin
  p0 := enthropy(D);
  for each  A_i ∈ A do
     p[A_i] := enthropy_{A_i}(D);
     Gain[A_i] = p0 − p[A_i];     //вычисляем прирост информации
 endfor
  best := arg(findMax(Gain[]));
  if  Gain[best] >threshold  then return best
  else  return  NULL; end
```

Рис. 3.2: Функция selectSplittingattribute(), использующая меру прироста информации

```
function selectSplittingattribute(A, D,threshold);   //использует коэффициент
прироста информации
begin
  p0 := enthropy(D);
  for each  A_i ∈ A do
     p[A_i] := enthropy_{A_i}(D);
     Gain[A_i] := p0 − p[A_i];     //вычисляем прирост информации
     gainRatio[A_i] := Gain[A_i]/enthropy(A_i);     //вычисляем коэффициент
прироста информации
  endfor
  best := arg(findMax(gainRatio[]));
  if  Gain[best] >threshold   then return best
  else   return  NULL; end
```

Рис. 3.3: Функция selectSplittingattribute(), использующая коэффициент прироста информации

3.4.3 Выбор атрибута для разбиения

Алгоритм C4.5. основан на вызове внешней функции, которая должна определять на каждом шаге атрибут, по которому следует производить ращепление данных. Прежде чем рассмотреть основные способы выбора таких атрибутов, напомним понятие информационной энтропии.

Информационная энтропия.

Информационная энтропия — мера хаотичности информации, неопределённость появления какого-либо символа первичного алфавита. При отсутствии информационных потерь численно равна количеству информации на символ передаваемого сообщения.

Например, в последовательности букв, составляющих какое-либо предложение на русском языке, разные буквы появляются с разной частотой, поэтому неопределённость появления для некоторых букв меньше, чем для других.

К. Шеннон сформулировал требования к измерению энтропии:

- мера должна быть непрерывной; то есть изменение значения величины вероятности на малую величину должно вызывать малое результирующее изменение функции;
- в случае, когда все варианты (буквы в приведённом примере) равновероятны, увеличение количества вариантов (букв) должно всегда увеличивать значение функции;
- должна быть возможность сделать выбор (в нашем примере – букв) за два шага, в которых значение функции конечного результата должно являться суммой функций промежуточных результатов.

Шеннон показал, что единственная функция, удовлетворяющая этим требованиям, т.е. информационная энтропия для независимых случайных событий X с n возможными состояниями (от 1 до n), i-ое из которых имеет вероятность $p(i)$, определяется (с точностью до мультипликативной константы) по формуле:

$$H(X) = -\sum_{i=1}^{n} p(i) \log_2 p(i)$$

.

Энтропия измеряется в **битах**. При вычислении энтропии по этой формуле предполагается, что $0 \cdot \log_2(0) = 0$.

Математические свойства энтропии

1. Неотрицательность: $H(X) \geq 0$. Равенство 0 получается при отсутствии неопределенности, когда для некоторого i вероятность $p(i) = 1$, а для остальных она равна 0.
2. Ограниченность: $H(X) \leq \log_2 |X|$. Равенство достигается в случае, когда все элементы из X равновероятны, т.е. $H(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}) = \log_2 n$.
3. Если X и Y независимы, то $H(XY) = H(X) + H(Y)$.
4. Энтропия — выпуклая вверх функция распределения вероятностей элементов.
5. Если X и Y имеют одинаковое распределение вероятностей элементов, то $H(X) = H(Y)$.

Использование информационной энтропии для выбора атрибута.

Рассмотрим реляционную выборку D над списком атрибутов $A = \{A_1, \ldots, A_n, C\}$, где C — это атрибут класса. Пусть $dom(C) = \{c_1, \ldots c_k\}$. Пусть $D_i = \{t \in D | class(t) = c_i\}$. Тогда, $D = D_1 \cup D_2 \cup \ldots \cup D_s$.

Через $Pr(C = c_i)$ обозначим вероятность того, что случайно выбранная запись $t \in D$ имеет метку класса c_i, т.е.

$$Pr(C = c_i) = \frac{|D_i|}{|D|}.$$

В соответствии с общим определением **энтропия** выборки D относительно C равна

$$H(D) = -\sum_{i=1}^{k} Pr(C = c_i) \cdot \log_2(Pr(C = c_i)).$$

Прирост информации (Information Gain). Идея: мы хотим выбрать атрибут, который позволит разбить выборку D на наиболее **простые (pure)** подмножества. Для этого введем меру **прироста информации**. Определим *энтропию D после разбиения, использующего атрибут A_i* с областью $dom(A_i) = \{v_1, \ldots, v_s\}$, следующим образом:

$$H_{A_i}(D) = \sum_{j=1}^{s} \frac{|D_j|}{|D|} \cdot H(D_j),$$

где $D_j = \{t \in D | t[A_i] = v_j\}$.

Прирост информации (*information gain*), достигаемый при таком разбиении, это разность между энтропией D до и после разбиения:

$$Gain(D, A_j) = H(D) - H_{A_j}(D).$$

Коэффициент прироста информации *(information gain ratio).* — это нормализованный вариант меры прироста информации:

$$gainRatio(D, A_j) = \frac{Gain(D, A_j)}{-\sum_{j=1}^{s}\left(\frac{|D_j|}{|D|}\cdot \log_2 \frac{|D_j|}{|D|}\right)}$$

(т.е., мы нормализуем прирост информации энтропией самого разбиения.)

На рис. 3.2 и 3.3 приведены два варинта функции selectSplittingattribute(). Первый из них для определения атрибута для разбиения использует меру **прироста информации**, а второй — **коэффициент прироста информации**.

Оба алгоритма работают следующим образом:

1. Вычислить энтропию текущей выборки данных.
2. Для каждого доступного атрибута вычислить энтропию выборки, получаемой после разбиения по этому атрибуту.
3. Найти атрибут с наибольшим *приростом информации/ коэффициентом прироста информации.*
4. Если для этого атрибута *прирост информации/ коэффициент прироста информации* превышает заданный порог (threshold), то выдать этот атрибут. В противном случае, выдать `NULL`, поскольку ни один атрибут не приведет к существенному увеличению энтропии.

3.4.4 Работа с непрерывными атрибутами в алгоритме C4.5.

В том виде, в котором он представлен в пункте 3.4.2, **Алгоритм C4.5** не может быть применен к наборам данных с непрерывными атрибутами. Чтобы можно было строить деревья решений для задач классификации с такими атрибутами, нужно несколько изменить процедуру выбора лучшего разбиения.

Пусть D – это обучающая выборка над списком атрибутов $A = \{A_1, \ldots, A_n\}$. Пусть $A_i \in A$ является **непрерывным атрибутом**.

Бинарное разбиение D по значению α атрибута A_i это пара $D^- \subseteq D$, $D^+ \subseteq D$, такая что:

1. $D^- \cup D^+ = D$
2. $D^- \cap D^+ = \emptyset$
3. $(\forall d \in D^-) d[A_i] \leq \alpha$;
4. $(\forall d \in D^+) d[A_i] > \alpha$;

Идея. На каждом шаге алгоритма C4.5 для каждого непрерывного атрибута A_i ищется **бинарное разбиение** с наилучшим приростом информации (или коэффициентом прироста информации). Отметим, что энтропия бинарного разбиения D по значению α атрибута A_i определяется следующим образом

$$H_{A_i,\alpha}(D) = -\frac{|D^-|}{|D|}\cdot H(D^-) - \frac{|D^+|}{|D|}\cdot H(D^+).$$

```
function selectSplittingAttribute(A, D,threshold);   //использует прирост информации
begin
  p0 := enthropy(D);
  for each  A_i ∈ A do
     if A_i непрерывный then
        x := findBestSplit(A_i, D);
        p[A_i] := enthropy_{A_i,x}(D)
     else
        p[A_i] := enthropy_{A_i}(D)
     endif;
     Gain[A_i] = p0 − p[A_i]      //вычислить прирост информации
 endfor;
  best := arg(findMax(Gain[]));
  if  Gain[best] >threshold   then return best
  else   return  NULL;
end
```

```
Функция  findBestSplit(A_i,   D)      //находит наилучшее бинарное разбиение для
непрерывного атрибута
begin
  определить множество  Val_i = {x_1, ..., x_r}  значений  A_i  в  D;
  инициализировать массивы counts_1[1, r], ..., counts_k[1, r];
  инициализировать массив Gain[];
  p0 := enthropy(D);
  foreach d ∈ D do    //подсчет частоты классов
     for j = 1 to k do
        Пусть class(d) = c_j;
        Пусть d[A_i] = x_l;
         counts_j[l] := counts_j[l] + 1
     endfor
  endfor;
  foreach x_l ∈ Val_i do
     //вычисление энтропии бинарного разбиения по x_l
     Gain[x_l] := p0 − enthropy(D, A_i, x_l, counts_1, ..., counts_k)
  endfor;
  best := arg(findMax(Gain[x_l]));
  return  best;
end
```

Рис. 3.4: Модифицированный вариант функции selectSplittingAttribute() с поиском наилучших бинарных разбиений для непрерывных атрибутов. .

Тогда прирост информации, получаемый при бинарном разбиении A_i по α равен

$$Gain_{A_i,\alpha}(D) = H(D) - H_{A_i,\alpha}(D).$$

Поиск наилучшего бинарного разбиения. Новый вариант функции selectSplitting Attribute() приведен на рис. 3.4.

- В случае непрерывности атрибута A_i новая функция selectSplittingAttribute() вызывает функцию findBestSplit(), представленную на том же рисунке.
- Чтобы найти наилучшее бинарное разбиение, мы
  - проходим всю выборку D и определяем список $Val_i = \{x_1, \ldots, x_r\}$ всех значений A_i.
    Заметим, что несмотря на непрерывность области $dom(A_i)$, D содержит только конечное множество различных значений A_i;
  - для каждого $x_l \in Val_i$ и класса c_j определяем число $counts_j[l]$ примеров из D со значением атрибута A_i равным x_l, отнесенных к классу c_j;
  - для каждого значения x атрибута A_i в D находим энтропию $H_{A_i,x}(D)$;
  - находим x с наибольшим приростом информации и возвращаем его в качестве результата.

Требуется сделать еще одно изменение в алгоритме **C4.5**:

- если на текущем шаге для разбиения D выбирается **категориальный атрибут**, то он, как и раньше, удаляется из списка атрибутов, передаваемых в следующий вызов C4.5.
- если же для разбиения D выбирается **непрерывный атрибут**, то он остается в списке атрибутов, передавемых в рекурсивный вызов C4.5.

3.5 Оценки качества классификаторов

Алгоритмы построения классификаторов являются эвристическими, то есть их результаты не обязательно оказываются правильными во всех случаях. В то же время, нужно уметь понимать, как сравнивать работу разных алгоритмов на одном наборе данных, а также, как оценивать точность одной построенной функции-классификатора. Ниже, мы рассматриваем ряд способов оценки точности и качества работы алгоритмов классификации.

3.5.1 Меры точности классификации/предсказаний

Обозначения. Пусть T — это классификатор, построенный **некоторым** алгоритмом обучения с учителем по заданной **обучающей выборке** D.

Пусть D' — это некоторая **проверочная выборка**, ***из данных с тем же распределением, что и*** D.

Пусть $t \in D'$. Черз $T(t)$ обозначим **метку класса**, определяемую для записи t классификатором T.

Через $class(t)$ обозначим **настоящую** метку класса t.

Через D_{true} обозначим *множество всех проверочных примеров, для которых рассматриваемый классификатор дает правильные результаты*:

$$D_{true} = \{t \in D' | T(t) = class(t)\}$$

Через D_{error} обозначим *множество всех проверочных примеров, для которых рассматриваемый классификатор определяет класс неверно*:

$$D_{error} = \{t \in D' | T(t) \neq class(t)\}$$

Точность распознавания. **Точность** классификатора T определяется как доля правильных предсказаний:

$$accuracy(T) = \frac{|D_{true}|}{|D'|}.$$

Степень ошибки. **Степень ошибки (error rate)** классификатора T определяется как доля неверных предсказаний:

$$errorRate(T) = 1 - accuracy(T) = \frac{|D_{error}|}{|D'|}.$$

Меры точности для бинарной классификации

Бинарные классификаторы. Многие классификаторы являются **бинарными**, т.е. атрибут класса C имеет лишь два значения. Задача классификации для $dom(C) = \{c_1, \ldots c_k\}$, $k > 2$, очевидно, может быть сведена к k задачам классификации для атрибутов классов $C_1, C_2, \ldots, C_k$ таких, что $dom(C_i) = \{0, 1\}$. $C_i = 1$ означает, что $C = c_i$.

Ошибки классификации. Рассмотрим задачу бинарной классификации для атрибута класса C, $dom(C) = \{0, 1\}$, где $C = 1$ интерпретируется как *"запись принадлежит классу C"*, а $C = 0$ интерпретируется как *"запись не принадлежит классу C"*, а Пусть T — некоторый классификатор для C. D' — проверочная выборка. Для записи $t \in D'$ имеется четыре возможности классификации:

1. **Правильный положительный** *(true positive)*: $T(t) = class(t) = 1$;
2. **Правильный отрицательный** *(true negative)*: $T(t) = class(t) = 0$;
3. **Неверный положительный** *(false positive)*: $T(t) = 1; class(t) = 0$;
4. **Неверный отрицательный** *(false negative)*: $T(t) = 0; class(t) = 1$;

Выделяют **два рода ошибок классификации**:

1. **Ошибка первого рода** *(type I error)*: **ошибочное доверие**, или **ложный (лишний) положительный**: классификатор неверно классифицирует запись как принадлежащую классу C.

2. **Ошибка второго рода** *(type II error)*: **ошибочный пропуск** , или **ложный (лишний) отрицательный**: классификатор неверно классифицирует запись как НЕ принадлежащую классу C.

Введем следующие обозначения:

1. D_{TP} : множество всех **правильных положительно** расклассифицированных записей из D'; $TP = |D_{TP}|$;
2. D_{TN} : множество всех **правильных отрицательно** расклассифицированных записей из D'; $TN = |D_{TN}|$;
3. D_{FP} : множество всех **неверно положительно** расклассифицированных записей из D'; $FP = |D_{FP}|$;
4. D_{FN} : множество всех **неверно отрицательно** расклассифицированных записей из D'; $FN = |D_{FN}|$;

Матрица неточности (Confusion Matrix). Информация о качестве **бинарного классификатора** обычно представляется в виде **матрицы неточности**:

	Классифицируются положительно	Классифицируются отрицательно
Фактически положительные	TP	FN
Фактически отрицательные	FP	TN

Точность (precision). **Точность** классификатора оценивается как процент правильно *положительно* расклассифицированных записей во множестве всех положительно расклассифицированных записей:

$$precision(T) = \frac{TP}{TP + FP}.$$

Эта величина определяет *насколько точно классификатор выбирает положительные примеры*, она достигает 100%, когда классификатор не допускает *неверных положительных* ошибок.

Возврат (Recall). **Возврат** классификатора оценивается как процент правильно *положительно* расклассифицированных записей во множестве всех фактически положительных записей:

$$recall(T) = \frac{TP}{TP + FN}.$$

Величина возврата определяет *насколько точно классификатор идентифицирует все положительные записи*, она достигает 100%, когда классификатор не допускает *неверных отрицательных* ошибок.

Замечание: **Точность (Precision)** и **возврат (recall)** имеют смысл для оценки качества классификатора **только**, когда они используются совместно.

Легко построить пример классификатора, имеющего 100%-ю точность: выбор *T(t) = 0* для всех $t \in D'$ гарантирует это. Но **возврат** у этого классификатора равен 0.

Также легко определить классификатор со 100%-ым возвратом: выбор *T(t) = 1* для всех $t \in D'$ гарантирует это. Но этот классификатор будет иметь **малую точность.**

Мера PF. Эта мера определяется как

$$PF(T) = \frac{FP}{FP + TN}.$$

PF оценивает *степень ошибочной классификации*: процент записей, не принадлежащих классу C, которые были **неверно расклассифицированы**.

Мера F. Мера **F** — это гармоническое среднее точности и возврата:

$$F(T) = \frac{2}{\frac{1}{precision(T)} + \frac{1}{recall(T)}} = \frac{2 \cdot precision(T) \cdot recall(T)}{precicion(T) + recall(T)}.$$

В некоторых случаях одна из двух величин – точность или возврат – является более важной чем другая. Меру **F** легко можно изменить в пользу любой из них. Приведенная ниже мера F_2 предполагает, что возврат вдвое более значим чем точность. А мера $F_{0.5}$ предполагает, что точность вдвое более значима чем возврат.

$$F_2(T) = \frac{5 \cdot precision(T) \cdot recall(T)}{4 * precision(T) + recall(T)}.$$

$$F_{0.5}(T) = \frac{1.25 \cdot precision(T) \cdot recall(T)}{0.25 * precision(T) + recall(T)}.$$

Следующая формула определяет меру F_β, в которой β представляет важность возврата по отношению к точности:

$$F_\beta(T) = \frac{(1 + \beta^2) \cdot precision(T) \cdot recall(T)}{\beta^2 * precision(T) + recall(T)}.$$

3.5.2 Построение и оценка классификатора

Обычно имеется обучающая выборка D и требуется по ней построить классификатор. Но если использовать при построении классификатора все записи из D, то не окажется независимого способа проверки его качества.

Отложенное множество. Разобьем D на две части: $D = D_{train} \cup D_{test}$; $D_{train} \cap D_{test} = \emptyset$.

D_{test} называют **отложенной выборкой.**

Построим классификатор T, используя в качестве обучающей выборки D_{train}. А для проверки T используем отложенную выборку D_{test}.

Способы построения **отложенной выборки**:

- **Случайный выбор.** Зафиксировать некоторый процент x и случайно выбрать $x\%$ записей D в качестве D_{test}.

  Обычно около 90% записей D используется для обучающей выборки и оставшиеся 10% — для отложенной.

- **Временной срез.** Если D состоит из "старых" и "новых" данных, то в обучающую выборку можно поместить все "старые" данные, а отложенную выборку составить из "новых" данных (например, в тех случаях, когда новые записи появляются ежедневно).

- **Повторный случайный выбор.** Этот способ применяется, когда выборка D мала.
  - Зафиксировать некоторое число повторов M.
  - Выполнить M раз случайный выбор отложенной выборки из D. Построить классификатор по оставшейся выборке D_{train}. Для каждой из отложенных выборок определить `точность` построенного классификатора.
  - Вычислить окончательную `точность` как среднее значение `точности` по всем M отложенным выборкам.
    Повторный случайный выбор позволяет устранить или, по крайней мере, уменьшить влияние счастливых случайностей на результаты работы алгоритма классификации.

- **Перекрестная проверка.** Это вариант повторного случайного выбора, при котором случайный выбор записей происходит только один раз, а классификатор строится несколько раз.
  - Зафиксировать число n – количество *срезов* данных из D.
  - Используя случайный выбор, разбить D на n *срезов* равного (или почти равного) размера.
  - n раз выполнить построение классификатора. На шаге i использовать срез D_i в качестве отложенной выборки, а остальные $n-1$ срезов в качестве обучающей.

C4.5. и избыточная точность

Сверхточность. Пусть $D_{training}$ это обучающая выборка, а D_{test} – проверочная выборка. Пусть f – классификатор, построенный по $D_{training}$.

> f ***избыточно точен***, *если существует другой классификатор* f', *который имеет меньшую точность чем* f *на* $D_{training}$, *но более высокую точность чем* f *на* D_{test}.

Причины избыточной точности:

- *неточные данные* (например, неверные метки классов);
- *случайные обстоятельства* (например, обучающая выборка не является репрезентативной для рассматриваемой области);
- *сложность модели* (например, очень много атрибутов, некоторые из которых не нужны для классификации).

Как бороться с избыточной точностью. Два основных подхода:

- **Предварительное сокращение** или **ранняя остановка**. Например, *третье условие остановки* в алгоритме C4.5, отказ от выбора разбивающего атрибута в связи с тем, что не презвойден порог улучшения, может раньше оборвать построение дерева, используя заданный пользователем `пороговое значение` параметра.
- **Заключительное сокращение** или **сокращение построенного дерева**. При этом подходе допускается чтобы построенный классификатор был избыточно точен, но затем он проверяется на избыточную точность специальным алгоритмом *сокращения*.

3.6 Алгоритм CART

Алгоритмы конструирования деревьев решений состоят из этапов "построение"или "создание"дерева (tree building) и "сокращение"дерева (tree pruning). В ходе создания дерева решаются вопросы выбора критерия расщепления и остановки обучения (если это предусмотрено алгоритмом). В ходе этапа сокращения дерева решается вопрос отсечения некоторых его ветвей.

3.6.1 Общая схема алгоритма

Алгоритм CART – Classification and Regression Tree (Дерево Классификации и регрессии), решает задачи классификации и регрессии. Он разработан в 1974-1984 годах четырьмя профессорами статистики - Leo Breiman (Berkeley), Jerry Friedman (Stanford), Charles Stone (Berkeley) и Richard Olshen (Stanford) [4]. Алгоритм CART предназначен для построения бинарного дерева решений, т.е. каждой внутренней вершине дерева приписан некоторый тест, от истинности или ложности которого зависит переход к одному из двух сыновей вершины: левому (left) или правому (right).

Атрибуты набора данных могут иметь как дискретные нечисловые, так и числовые (порядковые) значения.

Основная идея при построении дерева состоит в выборе среди всех возможных разбиений такого, для которого результирующие вершины-потомки были бы наиболее однородными ("чистыми").

Для категориального атрибута A_i с $|dom(A_i)| = r$ имеется $(2^r - 1)$ вариантов выбора списка значений A_i, по которому может произойти разбиение. Тесты для таких атрибутов имеют вид "$A_i \in B$? где B – произвольное непустое подмножество $dom(A_i)$. С учетом эквивалентности тестов для B и $dom(A_i) \setminus B$ получаем $2^{r-1} - 1$ различных вариантов тестов. Если A_i — это числовой атрибут с r значениями, то возможно $(r - 1)$ разбиение по A_i с помощью тестов вида "$A_i \leq a$?".

Построение дерева решений начинается с корня. Для каждой вершины последовательно выполняются следующие шаги.

Шаг 1. Поиск наилучшего разбиения для атрибута.

Для каждого непрерывного или целочисленного атрибута A_i отсортировать его значения по возрастанию. Затем, начиная с самого большого значения, рассмотреть для каждого значения a атрибута A_i тест вида "$t(A_i) \leq a$? при выполнении которого запись t переходит к левому потомку, а если тест не выполнен, то — к правому. Выбрать среди этих тестов тот, который максимизирует определяемый ниже *критерий разбиения*.

Для каждого категориального атрибута A_i и для каждого подмножества $B \subseteq dom(A_i)$ рассматриваем тест вида "$t(A_i) \in B$? при выполнении которого запись t переходит к левому потомку, а если тест не выполнен, то — к правому. Выбрать среди этих тестов тот, который максимизирует определяемый ниже *критерий разбиения*.

Шаг 2. Поиск наилучшего разбиения для вершины.

Среди наилучших разбиений, найденных на шаге 1, выбрать то, которое максимизирует *критерий разбиения*.

Шаг 3. Если *критерии остановки* не выполнены, то провести разбиение по тесту, найденному на шаге 2.

3.6.2 Критерии разбиения для CART

Будем использовать ниже следующие обозначения:
$D = \{t_1, \ldots, t_N\}$ – входная обучающая выборка,
$D_j = \{t \in D \mid t[C] = c_j\}$ – множество примеров из класса c_j в выборке D ,
$\pi(j) = \frac{|D_j|}{|D|}$ – априорная вероятность класса c_j,
$D(v)$ – множество обучающих примеров в вершине v дерева,
w_i – вес примера t_i,
f_i – вес частотности, связанный с примером t_i,
$p(j, v)$ – вероятность класса c_j в вершине дерева v,
$p(v)$ – вероятность подмножества примеров, попавших в вершину дерева v,
$p(j|v)$ – вероятность примера из класса c_j при условии, что он попал в вершину v,
$C(i|j)$ – стоимость ошибки классификации при отнесении примера из класса c_j к классу c_i.

Для класса c_j и вершины дерева v положим
$N_{w,j} = \sum_{t_i \in D_j} w_i f_i$, и $N_{w,j}(v) = \sum_{t_i \in D(v)_j} w_i f_i$. $N_{w,j}(v)$ – это взвешенное число примеров класса c_j во входной выборке, а $N_{w,j}(v)$ – это взвешенное число примеров класса c_j в выборке, попавшей в вершину v. Если все веса и частоты равны 1, то $N_{w,j} = |D_j|$, $N_{w,j}(v) = |D(v)_j|$.

В вершине v вероятности $p(j, v)$, $p(v)$ и $p(j|v)$ определяются следующими формулами:

$$p(j, v) = \frac{\pi(j) N_{w,j}(v)}{N_{w,j}},$$

$$p(v) = \sum_{j=1}^{k} p(j, v),$$

$$p(j|v) = \frac{p(j,v)}{p(v)} = \frac{p(j,v)}{\sum_{j=1}^{k} p(j,v)}.$$

В простом случае, когда все веса и частоты равны 1, эти формулы упрощаются:
$p(j, v) = \frac{\pi(j)|D(v)_j|}{|D_j|} = \frac{|D(v)_j|}{|D|}$,
$p(v) = \frac{|D(v)|}{|D|}$,
$p(j|v) = \frac{|D(v)_j|}{|D|} \cdot \frac{|D(v)|}{|D|}$.

Критерий Gini

Мера Gini неоднородности ("нечистоты") в вершине v определяется как

$$Gini(v) = \sum_{i,j} C(i|j)p(i|v)p(j|v).$$

Критерий Gini для выбора разбиения в вершине v состоит в максимальном уменьшении неоднородности, определяемом для разбиения s как

$$\Delta Gini(s, v) = Gini(v) - p_L Gini(v_L) - p_R Gini(v_R),$$

где p_L и p_R – вероятности передачи примера к левому v_L и правому v_R потомкам вершины v, соответственно. Они определяются как $p_L = p(v_L)/p(v)$ и $p_R = p(v_R)/p(v)$.

В простых случаях, когда все стоимости ошибок $C(i|j)$ одинаковы, в качестве меры неоднородности выборки D рассматривают величину

$$Gini(D) = 1 - \sum_{j=1}^{k} p_j^2,$$

где $p_j = |D_j|/|D|$ – вероятность (частота) класса c_j в D. Тогда, если при разбиени s выборка $D(v)$ из N примеров разбивается на две части D_L и D_R с числом примеров в каждой N_L и N_R, соответственно, то показатель качества разбиения будет равен:

$$\Delta Gini(s, v) = Gini(v) - \frac{N_L}{N} Gini(v_L) - \frac{N_R}{N} Gini(v_R).$$

Наилучшим считается то разбиение s, для которого $\Delta Gini(s, v)$ максимально. Если через l_i (r_i) обозначить число примеров из класса c_i в левом (правом) потомке v_L (v_R) вершины v, то дело сведется к минимизации выражения

$$\frac{N_L}{N}(1 - \sum_{i=1}^{k}(\frac{l_i}{N_L})^2) + \frac{N_R}{N}(1 - \sum_{i=1}^{k}(\frac{r_i}{N_R})^2),$$

что эквивалентно максимизации выражения

$$\tilde{G} = \frac{1}{N_L}\sum_{i=1}^{k} l_i^2 + \frac{1}{N_R}\sum_{i=1}^{k} r_i^2.$$

Иногда в алгоритме CART используют другие критерии разбиения Twoing и Ordered Twoing.

Критерий Twoing определяется как

$$\Delta Gini(s, v) = p_L p_R [\sum_j |p(j|v_l) - p(j|v_R)|]^2.$$

Определение критерия Ordered Twoing более сложно и здесь не приводится.

3.6.3 Критерии остановки

Критерии остановки должны определять, когда следует прекращать процесс построения дерева. Для этого используются следующие правила.

Построение дерева прекращается

- когда вершина стала однородной ("чистой"), т.е. все примеры в ней принадлежат одному классу;
- когда все примеры в вершине имеют одинаковые значения всех оставшихся атрибутов A_i;
- когда глубина вершины достигла заранее заданного максимального значения;
- когда число примеров в вершине не превосходит заранее заданного минимального значения;
- когда разбиение вершины приведет к появлению потомка с числом примеров меньшим, чем заранее заданное минимальное число примеров у потомка;
- когда наилучшее разбиение s^* вершины v приводит к величине улучшения $\Delta Gini(s^*, v)$ меньшей заранее заданного уровня минимально допустимого улучшения.

3.7 Пример: покупка дома

Предположим, что некоторая семья хочет купить дом. Существенными для осмотра и выбора дома она считает следующие параметры: количество комнат, наличие полуподвала (Подвал), тип планировки (Планировка) и географическое местоположение (Место). В следующей таблице содержится список домов, которые уже были выбраны для осмотра.

Номер	Комнаты	Подвал	Планировка	Место	Посещение
1	3	Нет	традиционная	Юг	**Нет**
2	3	Есть	традиционная	Юг	**Да**
3	3	Есть	открытая	Север	**Нет**
4	3	Есть	традиционная	Север	**Нет**
5	3	Нет	открытая	Север	**Нет**
6	3	Есть	традиционная	Юг	**Да**
7	3	Есть	открытая	Юг	**Нет**
8	3	Нет	традиционная	Юг	**Нет**
9	4	Нет	традиционная	Юг	**Да**
10	4	Есть	открытая	Север	**Нет**
11	4	Есть	открытая	Юг	**Да**
12	4	Нет	традиционная	Север	**Нет**
13	4	Нет	открытая	Юг	**Да**
14	4	Есть	открытая	Юг	**Да**
15	4	Нет	традиционная	Север	**Нет**
16	4	Есть	открытая	Север	**Нет**

Наша цель – построить дерево решений, определяющее по характеристикам дома будут ли его осматривать.

Алгоритм C4.5. для выборки продаваемых домов

Шаг 1. Определение корня дерева. Вход: вся выборка D, атрибуты: {Комнаты, Подвал, Планировка, Место}.

Замечание: Условие останова на шаге 1 не выполнено.

Вычисление прироста информации.

$Pr(\text{Посещение} = \text{Да}) = \frac{6}{16} = 0.375.$
$Pr(\text{Посещение} = \text{Нет}) = \frac{10}{16} = 0.625.$

$enthropy(D) = -\frac{3}{8} \cdot \log_2 \frac{3}{8} - \frac{5}{8} \cdot log_2 \frac{5}{8} = 0.9544$

Комнаты. $D_1 = \{1, 2, 3, 4, 5, 6, 7, 8\}$; $D_2 = \{9, 10, 11, 12, 13, 14, 15, 16\}$.
$enthropy(D_1) = -\frac{2}{8} \log_2 \frac{2}{8} - \frac{6}{8} \log_2 \frac{6}{8} = 0.811$
$enthropy(D_2) = -\frac{4}{8} \log_2 \frac{4}{8} - \frac{4}{8} \log_2 \frac{4}{8} = 1$
$enthropy_{\text{Комнаты}}(D) = \frac{8}{16} \cdot 0.811 + \frac{8}{16} \cdot 1 = 0.9055$

$$Gain_{\text{Комнаты}}(D) = 0.9544 - 0.9055 = 0.0489$$

Подвал. $D_1 = D_{\text{Есть}} = \{2, 3, 4, 6, 7, 10, 11, 14, 16\}$; $D_2 = D_{\text{Нет}} = \{1, 5, 8, 9, 12, 13, 15\}$.
$Pr_{D_1}(\text{Посещение= Да}) = \frac{4}{9}$
$Pr_{D_2}(\text{Посещение= Да}) = \frac{2}{7}$
$enthropy(D_1) = -\frac{4}{9} \log_2 \frac{4}{9} - \frac{5}{9} \log_2 \frac{5}{9} = 0.99107$
$enthropy(D_2) = -\frac{2}{7} \log_2 \frac{2}{7} - \frac{5}{7} \log_2 \frac{5}{7} = 0.8631$
$enthropy_{\text{Подвал}}(D) = \frac{9}{16} \cdot 0.99107 + \frac{7}{16} \cdot 0.8631 = 0.9350$

$$Gain_{\text{Подвал}}(D) = 0.9544 - 0.9350 = 0.0193$$

Планировка. $D_1 = D_{\text{традиционная}} = \{1, 2, 4, 6, 8, 9, 12, 15\}$;
$D_2 = D_{\text{открытая}} = \{3, 5, 7, 10, 11, 13, 14, 16\}$.

$Pr_{D_1}(\text{Посещение= Да}) = \frac{3}{8}$
$Pr_{D_2}(\text{Посещение= Да}) = \frac{3}{8}$
$enthropy(D_1) = -\frac{3}{8} \log_2 \frac{3}{9} - \frac{5}{8} \log_2 \frac{5}{8} = 0.9544$
$enthropy(D_2) = -\frac{3}{8} \log_2 \frac{3}{9} - \frac{5}{8} \log_2 \frac{5}{8} = 0.9544$
$enthropy_{\text{Планировка}}(D) = \frac{8}{16} \cdot 0.9544 + \frac{8}{16} \cdot 0.9544 = 0.9544$

$$Gain_{\text{Планировка}}(D) = 0.9544 - 0.9544 = 0$$

Место. $D_1 = D_{\text{Север}} = \{3, 4, 5, 10, 12, 15, 16\}$; $D_2 = D_{\text{Юг}} = \{1, 2, 6, 7, 8, 9, 11, 13, 14\}$.

$Pr_{D_1}(\text{Посещение= Да}) = \frac{0}{7} = 0$
$Pr_{D_2}(\text{Посещение= Нет}) = \frac{6}{9}$
$enthropy(D_1) = -\frac{0}{7} \log_2 \frac{0}{7} - \frac{7}{7} \log_2 \frac{7}{7} = 0$
$enthropy(D_2) = -\frac{6}{9} \log_2 \frac{6}{9} - \frac{3}{9} \log_2 \frac{3}{9} = 0.918$
$enthropy_{\text{Место}}(D) = \frac{7}{16} \cdot 0 + \frac{9}{16} \cdot 0.918 = 0.516$

$$Gain_{\text{Место}}(D) = 0.9544 - 0.516 = 0.438$$

Результат шага 1: Место дает наилучший прирост информации.

Разбиваем выборку по атрибуту Место:
$D_1 = D_{\text{Север}} = \{3, 4, 5, 10, 12, 15, 16\}$;
$D_2 = D_{\text{Юг}} = \{1, 2, 6, 7, 8, 9, 11, 13, 14\}$.

Шаг 2: Вход: $D_1 = \{3, 4, 5, 10, 12, 15, 16\}$, атрибуты: {Комнаты, Подвал, Планировка}.

Замечание: Эта выборка **однородна**: ни один дом из D_1 не посещался.

Класс(D_1) = Нет.

Шаг 3: Вход: $D_2 = D_{\text{Юг}} = \{1, 2, 6, 7, 8, 9, 11, 13, 14\}$. Атрибуты: {Комнаты, Подвал, Планировка}.

$enthropy(D_2) = 0.918$

Комнаты. $D_{21} = D_{\text{Юг, 3к}} = \{1, 2, 6, 7, 8\}$; $D_{22} = D_{\text{Юг,4к}}\{9, 11, 13, 14\}$.

$Pr_{D_{21}}(\text{Посещение= Да}) = \frac{2}{5}$, $Pr_{D_{22}}(\text{Посещение= Да}) = \frac{4}{4}$.
$enthropy(D_{21}) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.9709$
$enthropy(D_{22}) = -\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4} = 0$
$enthropy_{\text{Комнаты}}(D_2) = \frac{5}{9}\cdot 0.9709 + \frac{4}{9}\cdot 0 = 0.5394$

$$Gain_{\text{Комнаты}}(D) = 0.918 - 0.5394 = 0.3786$$

Подвал. $D_{21} = D_{\text{Юг,Есть}} = \{2, 6, 7, 11, 14\}$; $D_{22} = D_{\text{Юг,Нет}} = \{1, 8, 9, 13\}$.

$Pr_{D_{21}}(\text{Посещение= Да}) = \frac{4}{5}$
$Pr_{D_{22}}(\text{Посещение= Да}) = \frac{2}{4}$
$enthropy(D_{21}) = -\frac{4}{5}\log_2\frac{4}{5} - \frac{1}{5}\log_2\frac{1}{5} = 0.7219$
$enthropy(D_{22}) = -\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4} = 1$
$enthropy_{\text{Подвал}}(D) = \frac{5}{9}\cdot 0.7219 + \frac{4}{9}\cdot 1 = 0.8455$

$$Gain_{\text{Подвал}}(D) = 0.918 - 0.8455 = 0.0725$$

Планировка. $D_{21} = D_{\text{Юг,традиционная}} = \{1, 2, 6, 8, 9\}$;
$D_{22} = D_{\text{Юг, открытая}} = \{7, 11, 13, 14\}$.

$Pr_{D_{21}}(\text{Посещение= Да}) = \frac{3}{5}$
$Pr_{D_{22}}(\text{Посещение= Да}) = \frac{3}{4}$
$enthropy(D_{21}) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.9709$
$enthropy(D_{22}) = -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 0.8112$
$enthropy_{\text{Подвал}}(D) = \frac{5}{9}\cdot 0.9709 + \frac{4}{9}\cdot 0.8112 = 0.8999$

$$Gain_{\text{Подвал}}(D) = 0.918 - 0.8999 = 0.0181$$

Результат шага 3: Комнаты дает наилучший прирост информации.

Шаг 4: $D_{21} = D_{\text{Юг, 3к}} = \{1, 2, 6, 7, 8\}$; Атрибуты: {Подвал, Планировка}. $enthropy(D_{21}) = 0.7219$

Подвал. $D_{211} = D_{\text{Юг, 3к, Есть}} = \{2, 6, 7\}$; $D_{212} = D_{\text{Юг, 3к, Нет}} = \{1, 8\}$.

$Pr_{D_{211}}(\text{Посещение= Да}) = \frac{2}{3}$
$Pr_{D_{212}}(\text{Посещение= Да}) = \frac{0}{2} = 0$
$enthropy(D_{211}) = -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} = 0.918$
$enthropy(D_{212}) = -\frac{0}{2}\log_2\frac{0}{2} - \frac{2}{2}\log_2\frac{2}{2} = 0$
$enthropy_{\text{Подвал}}(D) = \frac{3}{5}\cdot 0.918 + \frac{2}{5}\cdot 0 = 0.5509$

$$Gain_{\text{Подвал}}(D) = 0.7219 - 0.5509 = 0.1709$$

Планировка. $D_{211} = D_{\text{Юг,3к,традиционная}} = \{1, 2, 6, 8\}$; $D_{212} = D_{\text{Юг,3к,открытая}} = \{7\}$.

$Pr_{D_{211}}(\text{Посещение= Да}) = \frac{2}{4}$
$Pr_{D_{221}}(\text{Посещение= Да}) = \frac{0}{1} = 0$
$enthropy(D_{211}) = -\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4} = 1$

$enthropy(D_{212}) = -\frac{0}{1}\log_2\frac{0}{1} - \frac{0}{1}\log_2\frac{0}{1} = 0$
$enthropy_{\text{Подвал}}(D) = \frac{4}{5}\cdot 1 + \frac{1}{5}\cdot 0 = 0.8$

$$Gain_{\text{Подвал}}(D) = 0.7219 - 0.8 = -0.0781$$

Результат шага 4: Подвал дает наилучший прирост информации

Шаг 5: $D_{211} = D_{\text{Юг, 3к, Yes}} = \{2, 6, 7\}$; Атрибуты: {Планировка}. $enthropy(D_{211}) = 0.918$
Планировка. $D_{2111} = D_{\text{Юг,3к, Есть, традиционная}} = \{2, 6\}$; $D_{2112} = D_{\text{Юг, 3к, Есть, открытая}} = \{7\}$.
$Pr_{D_{2111}}(\text{Посещение= Да}) = \frac{2}{2} = 1$
$Pr_{D_{2112}}(\text{Visited= Yes}) = \frac{0}{1} = 0$
$enthropy(D_{2111}) = -\frac{2}{2}\log_2\frac{2}{2} - \frac{0}{2}\log_2\frac{0}{2} = 0$
$enthropy(D_{221}) = -\frac{0}{1}\log_2\frac{0}{1} - \frac{0}{1}\log_2\frac{0}{1} = 0$
$enthropy_{\text{Подвал}}(D) = \frac{2}{3}\cdot 0 + \frac{2}{3}\cdot 0 = 0$

$$Gain_{\text{Подвал}}(D) = 0.918 - 0 = 0.918$$

Результат шага 5: Планировка дает наибольший прирост информации.

Шаги 6 и 7: $D_{2111} = D_{\text{Юг,3к, Есть, традиционная}} = \{2, 6\}$ и
$D_{2112} = D_{\text{Юг, 3к, Есть, открытая}} = \{7\}$: список атрибутов исчерпан,
Класс$(D_{2111}) =$ Да.
Класс$(D_{2112}) =$ Нет.

Шаг 8: Вход: $D_{22} = D_{\text{Юг,4к}}\{9, 11, 13, 14\}$. Атрибуты: {Подвал, Планировка}.
$Pr_{D_{22}}(\text{Посещение= Да}) = 1$, следовательно, Класс$(D_{22}) =$ Да.

Результат: полученное дерево решений показано на рис. 3.5. По этому дереву легко построить словесное описание выведенного критерия осмотра:

> *Семья будет осматривать только дома, находящиеся на юге. Будут осматриваться все четырехкомнатные дома, а также трехкомнатные дома с традиционной планировкой, в которых имеется в наличии полуподвал.*

3.8 Классификация с помощью ассоциативных правил

Пусть как и выше $A = \{A_1, \ldots, A_n\}$ — набор атрибутов, а атрибут C представляет метки классов и $dom(C) = \{c_1, \ldots, c_k\}$. Назовем *атомарным условием* выражение вида $A_i = t$, где $t \in dom(A_i)$, если область $dom(A_i)$ конечна, и выражение вида $A_i \in t$, где $t = [a, b] \subseteq dom(A_i)$, если область атрибута A_i непрерывна. Атомарное условие $A_i = t$ *выполняется* на записи (примере) $X = (x_1, x_2, \ldots, x_i, \ldots, x_n)$, если $x_i = t$, атомарное условие $A_i \in t$ выполнено на том же примере, если $x_i \in t$. *Ассоциативное правило R для классификации* это правило вида

$$\alpha_1, \alpha_2, \ldots, \alpha_m \longrightarrow (C = c_j),$$

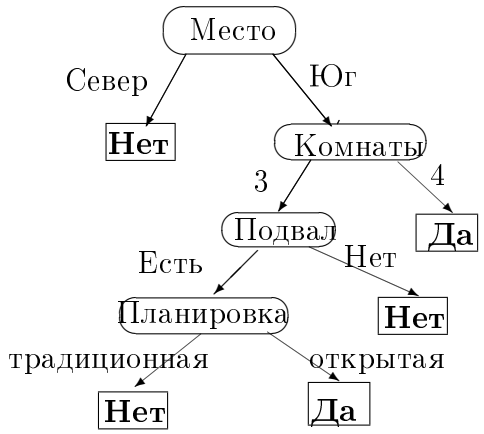

Рис. 3.5: Классифицирующее дерево, построенное алгоритмом C4.5

где все α_l в левой части являются атомарными условиями. Как обычно, левая часть правила (до стрелки) называется его *условием*, а правая – *заключением*. Такое правило *выполняется* на "обученном"примере $X = (x_1, x_2, \ldots, x_i, \ldots, x_n, c)$, если на X выполняется его условие, т.е. все атомарные условия $\alpha_l, (1 \leq l \leq m)$, и выполняется заключение, т.е. $c = c_j$. Таким образом, если R выполняется на объекте X, то оно его правильно классифицирует. Отметим, что запятые в условии правила понимаются как конъюнкции.

Надежностью $Conf_D(R)$ правила R на обучающей выборке D назовем долю примеров D, на которых выполнено R среди всех примеров, на которых выполнено условие R. Содержательно, эта величина характеризует точность правила. *Поддежка* $Supp_D(R)$ правила R на обучающей выборке D – это доля всех примеров D, на которых выполнено R. Нетрудно понять, что эти определения соответствуют определениям поддержки и надежности ассоциативных правил из гл. 2. Для классификации нового примера из множества правил обычно выбирается самое надежное правило, условие которого выполняется на этом примере. Нетрудно найти примеры, показывающие, что такой выбор может оказаться не самым удачным.

Любое классифицирующее дерево решений можно представить в виде набора классифицирующих правил. Каждой ветви дерева будет соответствовать правило, условие которого является конъюнкцией атомарных условий на ребрах этой ветви, а заключение выдает метку класса из листа, которым заканчивается ветвь.

Например, самой правой ветви дерева решений, показанного на на рис. 3.5, соответствует правило:

(Место = Юг), (Комнаты = 4) $\longrightarrow$ *(Посещение = Да).*

Его надежность равна 1, а поддержка – 0.25.

Считается, что правила могут быть более понятны пользователям классификаторов. Но для деревьев с большим числом листьев наборы механически построенных по ним правил также будут необозримы и потребуются специальные алгоритмы, позволяющие их уменьшить и упростить. Другой подход к классификации с помощью правил состоит в том, чтобы строить их непосредственно по обучающей выборке, минуя этап построения деревьев решений.

3.8.1 Алгоритм CBA

Алгоритм CBA, предложенный в работе [16], является одним из первых и простейших алгоритмов классификации с помощью ассоциативных правил. Его работа происходит в два этапа. На первом происходит генерация классифицирующих правил с помощью процедуры CBA-RG. Она, используя описанную выше модификацию алгоритма Apriori находит все частые наборы атомарных условий вида $< \alpha_1, \alpha_2, \ldots, \alpha_m, (C = c_j) >$ и выдает соответствующие им правила классификации, удовлетворяющие входным параметрам поддержки и надежности.

На втором этапе построения классификатора правила ранжируются следующим образом. Для данных правил R_1 и R_2 скажем, что R_1 имеет более *высокий ранг* чем R_2 (обозначение: $R_1 > R_2$) , если
(1) $Conf_D(R_1) > Conf_D(R_2)$, или
(2) $Conf_D(R_1) = Conf_D(R_2)$, но $Supp_D(R_1) > Supp_D(R_2)$, или
(3) $Conf_D(R_1) = Conf_D(R_2)$ и $Supp_D(R_1) = Supp_D(R_2)$, но правило R_1 порождено раньше правила R_2.

Правила сортируются в порядке убывания рангов, а затем рассматриваются по порядку. Если хотя бы один пример классифицируется правилом верно, то оно добавляется в классификатор, а все правильно классифицируемые им примеры удаляются из обучающей выборки. Этот процесс завершается при исчерпании правил или примеров выборки. Если в выборке остаются нерасклассифицированные примеры, то выбирается "класс по умочанию который содержит наибольшее их число. Этот класс используется в качестве ответа алгоритма при классификации нового примера, к которому нельзя применить ни одно правило классификатора. Затем алгоритм вычисляет общее число ошибок классификатора и число ошибок каждого правила на обучающей выборке. После этого находится правило с наименьшим числом ошибок и все правила после него удаляются из классификатора.

Пусть после процедуры CBA-RG получено множество правил $\mathbf{R}$. Следующий алгоритм CBA-CB выделяет из этого множества правила, образующие классификатор C.

Алгоритм CBA-CB

1. $C := \emptyset$; $\mathbf{R} := sort(\mathbf{R})$; // сортировка правил по рангам ;
2. **for each** $R \in \mathbf{R}$ (по порядку) **do**
3. $temp := \emptyset$;
4. **for each** $d \in D$ **do**
5. **if** на d выполнено условие R **then**
6. $temp := temp \cup \{d.id\}$;
7. **if** R выполнено на d **then** отметить R **endif**
8. **endif**
9. **enddo**;
10. **if** R отмечено **then**
11. вставить R в конец C;
12. удалить из D все примеры с идентификаторами в $temp$;
13. выбрать для текущего C класс c_R "по умолчанию";
14. вычислить общее число ошибок e_R для текущего C
15. **endif**
16. **enddo**;
17. найти первое правило R в C с наименьшим числом ошибок e_R;
18. удалить из C все правила, стоящие после R;
19. добавить правило, безусловно выдающее класс по умолчанию c_R;

20. **return**(C). // C – построенный классификатор.

Классификация новых объектов. После построения системы правил C алгоритм при классификации нового объекта ищет первое правило в списке C, условие которого выполняется на этом объекте, и присваивает объекту класс из заключения этого правила. Если такое правило в C отсутствует, то объекту прсваивается класс по умолчанию c_R, где R – это последнее правило в C.

3.8.2 Алгоритм CMAR

Алгоритм CMAR (Classification based onMultiple Association Rules) – алгоритм классификации, основанной на использовании многих ассоциативных правил. CMAR включает два этапа: порождение правил и классификацию. На первом этапе порождения правил CMAR вычисляет полное множество правил, имеющих заданные пороги поддержки и надежности. Затем CMAR удаляет некоторые из правил, оставляя подмножество наиболее подходящих для классификации правил.

На втором этапе при классификации нового объекта (примера) CMAR разбивает все множество правил, условия которых выполняются на этом объекте, на группы в соответствии с их заключениями, оценивает "силу"этих групп и предсказывает класс объекта в соответствии с предсказанием самой сильной группы.

На первом этапе для порождения правил с заданными уровнями поддержки и надежности используется алгоритм *FP-Tree+FP-growth*. При этом, как только обнаруживаем частый набор P, включающий атомарные условия на атрибуты, так сразу пытаемся породить связанные с ним ассоциативные правила вида $P \longrightarrow (C = c)$. Если оказалось, что такое правило имеет поддержку меньше заданного уровня, то не требуется рассматривать правила вида $P' \longrightarrow (C = c)$ для $P \subset P'$, так у них будет не большая поддержка.

Полученные правила помещаются в специальную структуру данных – *CR-дерево*. У него есть корень, от которого отходят ветви, представляющие правила. Атомарные условия в условиях правил упорядочиваются по их частоте в примерах. Правилу $p_1, p_2, \ldots, p_r \longrightarrow (C = c)$, имеющему поддержку sp и надежность cn, соответствует ветвь дерева $p_1 \rightarrow p_2 \rightarrow \ldots \rightarrow p_r$, в которой последняя вершина с меткой атомарного условия p_r имеет дополнительную метку (c, sp, cn), включающую метку класса в заключении правила, его поддержку и надежность. Заметим, что на одной ветви могут быть представлены несколько правил, если условия одних продолжают условия других. Кроме того, все одинаковые атомарные условия, входящие в CR-дерево, связываются в отдельный список. Головы всех таких списков находятся, как и у FP-деревев, в специальной таблице заголовков. CR-дерево позволяет эффективно работать с правилами на следующей стадии *устранения правил*.

На правилах вводится глобальный порядок, первые два условия которого совпадают с соответствующими условиями порядка в алгоритме CBA. А третье условие изменено на следующее:
(3) $conf(R_1) = conf(R_2)$ и $supp(R_1) = supp(R_2)$, но в условии R_1 используется меньше атомарных условий, чем в условии R_2.
Кроме того, скажем, что правило $R_1 = P \longrightarrow (C = c)$ является *обобщением* правила $R_2 = P' \longrightarrow (C = c')$, если P является подмножеством P'.

CMAR использует следующие способы устранения "лишних"правил.

Во-первых, более общие правила и правила с большей надежность используются для

устранения более специальных и менее надежных правил. Если R_1 является обобщением R_2 и имеет более высокий ранг, т.е. $R_1 > R_2$, то правило R_2 удаляется.

Это удаление происходит при вставке нового правила в CR-дерево. В этот момент запускается процедура поиска, которая проверяет можно ли устранить вставляемое правило или можно ли после вставки данного правила удалить какие-либо ранее вставленные в дерево правила.

Во-вторых, *выбираются только правила с положительной корреляцией* . Для каждого правила $R = P \longrightarrow (C = c)$ проверяется с помощью критерия χ^2 положительная корреляция между P и c [1].

Все правила, у которых значение χ^2 меньше заранее заданного порога, удаляются.

В-третьих, происходит удаление некоторых правил на основе *покрытия входной выборки (базы данных)*. При этом остаются правила, способные корректно классифицировать хотя бы один новый объект. А объекты, которые правильно классифицируются большим (превышающим заданный порог *threshold*) числом правил, удаляются из выборки. Более формально эта процедура выглядит следующим образом.

Алгоритм отбора правил на основе покрытия базы данных
Вход: входная выборка D, множество правил **R** и порог *threshold*.
Выход: R' – подмножество правил, используемых для классификации.

1. Отсортировать правила из **R** в порядке убывания рангов.
   Пусть после этого **R**= $R_1, \ldots, R_m$;
   R':= $\emptyset$;

2. **for each** $d \in D$ **do**
   $cover\_count(d) := 0$ //число правил, классифицирующих d
   endfor;

3. **while** $(D \neq \emptyset)$ **do**
   for $i = 1$ **to** m **do**
   $D_i := \{d \in D \mid R_i$ *выполняется на* $d\}$;
   if $D_i \neq \emptyset$ **then**
   R' := **R'** $\cup \{R\}$;
   for each $d \in D_i$ **do**
   $cover\_count(d) := cover\_count(d) + 1$;
   if $cover\_count(d) \geq threshold$ **then** $D := D \setminus \{d\}$ **endif**
   enddo
   endif
   enddo
   enddo;
   return(R').

[1]Напомним, как определяется значение теста χ^2. Предположим, что нас интересует независимость атрибутов (величин) A с областью $dom(A) = \{a_1, a_2, \ldots, a_k\}$ и B с областью $dom(B) = \{b_1, b_2, \ldots, b_r\}$. Данные, в которых появляются эти атрибуты, – это примеры нашей входной выборки D. Тогда значение χ^2 (называемое иногда статистикой Пирсона χ^2) определяется следующим образом: $\chi^2 = \sum_{i=1}^{k} \sum_{j=1}^{r} (o_{ij} - e_{ij})^2 / e_{ij}$, где o_{ij} – наблюдаемая частота пары $(A = a_i, B = b_j)$, т.е. $Supp_D(A = a_i, B = b_j)$, а e_{ij} – ее ожидаемая частота, определяемая формулой $e_{ij} = Supp_D(A = a_i) \times Supp_D(B = b_j)/|D|$. Содержательно, большое значение χ^2 означает “зависимость"между A и B.

Классификация новых объектов. После построения системы классифицирующих правил CMAR готов классифицировать новые объекты. Если все правила, условия которых выполняются на новом объекте, приписывают ему одинаковый класс, то, естественно, этот же класс и выдает CMAR.

Если метки классов, которые предписывают объекту правила, условия которых выполняются на нем, не совпадают, то CMAR разбивает все такие правила на группы в соответствии с приписываемыми классами. Затем для каждой группы определяется ее "сила"и в качестве ответа выдается класс, определяемый самой сильной группой правил.

Для определения силы группы используются следующие эмпирически найденные формулы.

Пусть $sup(c)$ – это число объектов обучающей выборки D, принадлежащих классу c, а $|D|$ – размер этой выборки. Для правила $R = P \longrightarrow (C = c)$ определяется величина $max\chi^2_R$:

$$max\chi^2_R = (\min\{supp_D(P), sup(c)\} - \frac{supp_D(P)sup(c)}{|D|})^2|D|e,$$

где

$$e = \frac{1}{supp_D(P)sup(c)} + \frac{1}{supp_D(P)(|D| - sup(c))} + \frac{1}{(|D| - supp_D(P))sup(c)} + \frac{1}{(|D| - supp_D(P))(|D| - sup(c))}$$

. После чего сила группы правил G определяется как сумма

$$\sum_{R \in G} \frac{\chi^2_R}{max\chi^2_R}\chi^2_R.$$

Таким образом, основные отличия CMAR от CBA состоят в использовании FP-tree вместо Apriori при построении всех классифицирующих правил, использовании для их представления специальной структуры – CR-дерева, немного другом алгоритме отбора правил для классификатора и, главное, – в применении для классификации нового объекта, не одного правила, а нескольких.

Авторы CMAR приводят в [14] экспериментальные данные, которые показывают, что этот алгоритм во многих случаях превосходит алгоритмы CBA и C4.5 по средней точности и эффективности.

3.9 Классификация методом Наивный Баейес

Как и в предыдущих случаях, мы рассматриваем ситуацию, когда имеется обучающая выборка данных D, состоящая из примеров (объектов) вида $d = (a_1, \ldots, a_n, c)$, где a_i – значение атрибута A_i из его области значений $dom(A_i)$ $(1 \leq i \leq n)$, а c – это метка класса-значение атрибута C из его области значений $dom(C) = \{c_1, \ldots, c_k\}$.

Задача классификации состоит в построении функции, которая корректно предсказывает по новому объекту $d = (a_1, \ldots, a_n)$ класс $c(d)$, которому этот объект принадлежит.

Один из способов предсказания *метки класса* $c(d)$ для объекта d состоит в **оценке вероятностей** $Pr(c(d) = c_1), Pr(c(d) = c_2), \ldots, Pr(c(d) = c_k)$ и **выборе в качестве предсказания класса, имеющего наибольшую вероятность**.

Замечание: вообще говоря, нам *не требуется знать* точные значения вероятностей, **достаточно знать их порядок (ранги)**.

Наивный Баейес – это метод оценки вероятностей (рангов) того, что некоторый объект принадлежит каждому из классов. Он оценивает $Pr(c(d) = c_i)$ через условную вероятность класса c_i при условии заданных значений атрибутов d:

$$Pr(c(d) = c_i) = Pr(c_i|d).$$

Пусть $d = (a_1, \ldots, a_n)$. Тогда,

$$Pr(c_i|d) = Pr(c_i|A_1 = a_1, A_2 = a_2, \ldots, A_n = a_n).$$

По определению **условной вероятности**:

$$Pr(X|Y) = \frac{Pr(X \wedge Y)}{Pr(Y)}.$$

Теорема Баейса для условных вероятностей:

$$Pr(X|Y) \cdot Pr(Y) = Pr(Y|X) \cdot Pr(X).$$

Когда $Pr(Y) \neq 0$ и $Pr(X) \neq 0$, теорему Баейса можно переформулировать:

$$Pr(X|Y) = \frac{Pr(Y|X) \cdot Pr(X)}{Pr(Y)}.$$

Шаг 1: (Применение теоремы Байеса) По теореме Байеса условная вероятность $Pr(c_i|d)$ наблюдения объекта класса c_i при данных значениях атрибутов d может быть определена как:

$$Pr(c_i|d) = \frac{Pr(d|c_i) \cdot Pr(c_i)}{Pr(d)},$$

или

$$Pr(c_i|d) = Pr(c_i|A_1 = a_1, \ldots A_n = a_n) = \frac{Pr(A_1 = a_1, \ldots, A_n = a_n|c_i) \cdot Pr(c_i)}{Pr(A_1 = a_1, \ldots A_n = a_n)}.$$

Здесь

- $P(d \mid c_i) = Pr(A_1 = a_1, \ldots, A_n = a_n|c_i)$ это вероятность наблюдения вектора d при условии, что классификатор присвоил ему класс c_i (т.е., вероятность нахождения d среди элементов класса c_i).
- $P(c_i)$ это вероятность того, что объект принадлежит классу c_i (априорная вероятность класса c_i).
- $P(d)$ это вероятность появления объекта d на входе (априорная вероятность объекта d).

Итак, мы свели задачу оценки $Pr(c_i|d)$ к задачам оценки $P(d|c_i)$, $Pr(c_i)$ и $Pr(d)$.

Шаг 2: упрощение. Нам нужно оценить три вероятности: $Pr(d \mid c_i), Pr(c_i)$ and $Pr(d)$. Заметим следующее:

1. **Оценка** $Pr(c_i)$**.** Эту вероятность можно оценить как долю объектов обучающей выборки D, принадлежащих классу c_i.

   Пусть $D_i = \{d \in D | c(d) = c_i\}$. Тогда

   $$Pr(c_i) = \frac{|D_i|}{|D|}.$$

2. **Оценка** $Pr(d)$**.** Объект d может принадлежать одному из k классов. Тогда $Pr(d)$ можно оценить как сумму условных вероятностей принадлежности d каждому из классов $c_1, \ldots, c_k$:

   $$Pr(d) = Pr(d|c_1) \cdot Pr(c_1) + \ldots Pr(d|c_k) \cdot Pr(c_k) = \sum_{i=1}^{k} Pr(d|c_i) \cdot Pr(c_i).$$

   Заметим, что вероятность $Pr(d)$ **неизменна в оценках** $Pr(c_1|d)$**, ...,** $Pr(c_k|d)$: Поэтому, *если мы интересуемся только ранжированием (относительным порядком) вероятностей*, то можем игнорировать $Pr(d)$ и сосредоточиться на оценке произведения

   $$Pr(d|c_i) \cdot Pr(c_i).$$

Шаг 3: "наивная"часть – *оценка условной вероятности):* Нам нужно оценить

$$Pr(d|c_i) = Pr(A_1 = a_1, \ldots, A_n = a_n|c_i).$$

"Наивная"часть метода состоит в **предположении условной независимости.**

Предположение условной независимости. Две случайных переменных X и Y называются **независимыми**, если для всех $x \in dom(X), y \in dom(Y)$,

$$Pr(X = x|Y = y) = Pr(X = x),$$

т.е., если на вероятность наблюдения любого значения X не влияет появление какого-то значения Y.

Следствием свойства независимости является то, что совместная вероятность двух случайных переменных равна призведению их вероятностей:

$$Pr(X = x \wedge Y = y) = Pr(X = x) \cdot Pr(Y = y).$$

Оценка $Pr(d|c_i)$ **с использованием предположения о независимости.** Мы применяем **предположение о независимости** к

$$Pr(d|c_i) = Pr(A_1 = a_1, \ldots, A_n = a_n|c_i).$$

А именно, мы предполагаем, что *каждый атрибут* A_i *не зависит от* $A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n$ *при заданном значении* C:

$$Pr(A_i = a_i|A_1 = a_1, \ldots A_{i-1} = a_{i-1}, A_{i+1} = a_{i+1}, \ldots, A_n, C = c_j) = Pr(A_i = a_i|C = c_j).$$

Следовательно,

$$Pr(A_1 = a_1, \ldots, A_n = a_n|c_i) = Pr(A_1 = a_1|c_i) \cdot \ldots \cdot Pr(A_n = a_n|c_i) = \prod_{j=1}^{n} Pr(A_j = a_j|c_i).$$

$Pr(A_j = a_j|c_i)$ это вероятность попадания объекта с $A_j = a_j$ в класс c_i.

Таким образом, мы свели задачу оценки условной вероятности $Pr(c_i|d)$ к задаче оценки семейства вероятностей $Pr(A_j = a_j|c_i)$.

Шаг 4: Оценка вероятностей. Вероятности $Pr(A_j = a_j|c_i)$ оцениваются следующим образом.

Пусть $D_i = \{d \in D \mid c(d) = c_i\}$ – это множество всех объектов класса c_i. Пусть $D_{ij} = \{d \in D_i \mid d.A_j = a_j\}$ – это множество всех объектов класса c_i со значением a_j атрибута A_j. Тогда оценим $Pr(A_j = a_j \mid c_i)$ как

$$Pr(A_j = a_j \mid c_i) = \frac{|D_{ij}|}{|D_i|},$$

т.е., как долю объектов класса c_i со значением a_j атрибута A_j.

Шаг 5: предсказание класса. Вычисляем оценки

$$Pr(d \mid c_1) \cdot Pr(c_1), \ldots, Pr(d \mid c_k) \cdot Pr(c_k),$$

в соответствии с шагами 1-4.

Предсказываем класс:

$$c(d) = arg \max_{i=1,\ldots,k} (Pr(d \mid c_i) \cdot Pr(c_i)).$$

Пример

Таблица 6.1 **Обучающая выборка: покупатели электроники.**

Номер	Возраст	Доход	Студент	Кредитный рейтинг	Покупатель компьютера
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle	medium	no	excellent	yes
13	middle	high	yes	fair	yes
14	senior	medium	no	excellent	no

Пусть C1 соответствует классу покупатель компьютера = yes, а C2 соответствует классу покупатель компьютера = no. Предположим, что мы хотим классифицировать объект $X = (возраст = youth,\ доход = medium,\ студент = yes,\ кредитный\ рейтинг = fair)$.

Для этого нужно максимизировать $P(X|C_i)P(C_i)$ для $i = 1, 2$.
$P(C_i)$ – априорная вероятность каждого класса – определяется по входной выборке:
$P(покупатель\ компьютера = yes) = 9/14 = 0.643$,
$P(покупатель\ компьютера = no) = 5/14 = 0.357$.
Для определения $P(X|C_i)$ для $i = 1, 2$ мы вычисляем следующие условные вероятности:
$P(возраст = youth \mid покупатель\ компьютера = yes) = 2/9 = 0.222$,
$P(возраст = youth \mid покупатель\ компьютера = no) = 3/5 = 0.600$,
$P(доход = medium \mid покупатель\ компьютера = yes) = 4/9 = 0.444$,
$P(доход = medium \mid покупатель\ компьютера = no) = 2/5 = 0.400$,

$P(студент = yes \mid покупатель\ компьютера = yes) = 6/9 = 0.667$,
$P(студент = yes \mid покупатель\ компьютера = no) = 1/5 = 0.200$,
$P(кредитный\ рейтинг = fair \mid покупатель\ компьютера = yes) = 6/9 = 0.667$,
$P(кредитный\ рейтинг = fair \mid покупатель\ компьютера = no) = 2/5 = 0.400$.
Используя эти вероятности, получаем:
$P(X|покупатель\ компьютера = yes) = P(возраст = youth \mid покупатель\ компьютера = yes)\times$
$P(доход = medium \mid покупатель\ компьютера = yes)\times$
$P(студент = yes \mid покупатель\ компьютера = yes)\times$
$P(кредитный\ рейтинг = fair \mid покупатель\ компьютера = yes)$
$= 0.222 \times 0.444 \times 0.667 \times 0.667 = 0.044$.
Аналогично, $P(X|покупатель\ компьютера = no) = 0.600 \times 0.400 \times 0.200 \times 0.400 = 0.019$.
Чтобы найти класс C_i, максимизирующий $P(X|C_i)P(C_i)$, мы вычисляем
$P(X|покупатель\ компьютера = yes)P(покупатель\ компьютера = yes) = 0.044 \times 0.643 = 0.028$ и
$P(X|покупатель\ компьютера = no)P(покупатель\ компьютера = no) = 0.019 \times 0.357 = 0.007$.
Следовательно, "Наивный Баейс"определит для X класс "покупатель компьютера = yes".

3.10 Алгоритм k-ближайших соседей (kNN)

C4.5. и **многие другие методы классификации** (нейронные сети, метод опорных векторов SVN, правило индукции) являются ***жадными***: они анализируют обучающую выборку и строят классификатор *до того, как получают данные для тестирования.*

Принцип **ленивых вычислений** состоит в том, чтобы *откладывать всякий анализ данных до того момента, как будет получен реальный запрос на классификацию.*

Для обучения с учителем **ленивые вычисления** означают, что **классификатор не строится** заранее до чтения данных из тестовой выборки.

Алгоритм k-ближайших соседей (kNN). kNN – это простой, но на удивление хорошо работающий алгоритм **ленивых вычислений**. Его идея состоит в следующем:

- Входом алгоритма является обучающая выборка $D_{training}$, пример d, который нужно классифицировать, и целое число $k > 1$.
- Алгоритм вычисляет *расстояние* между d и всеми примерами $d' \in D$.
- Алгоритм отбирает k **наиболее похожих на** или **самых близких к** d примеров из D: $d_1, \ldots, d_k$, $d_i \in D$.
- Алгоритм относит d к классу, которому принадлежит наибольшее число примеров из списка $d_1, \ldots, d_k$.

Меры близости/сходства расстояний. Расстояние или близость между двумя примерами можно измерять по-разному.

Замечание: Меры близости возрастают при увеличении схожести объектов а **расстояния** при увеличении схожести объектов уменьшаются.

1. **Эвклидово расстояние**. Если атрибуты у D непрерывные , то каждый пример $d \in D$ является N-мерным вектором и представляет некоторую точку в N-мерном пространстве . В этом случае естественно использовать Эвклидово расстояние:

$$d(d_1, d_2) = \sqrt{\sum_{i=1}^{n} (d_1[A_i] - d_2[A_i])^2}.$$

2. **Манхетенское расстояние**. Если у D числовые дискретные атрибуты, то может оказаться, что лучше использовать Манхетенское расстояние:

$$d(d_1, d_2) = \sum_{i}^{n} |d_1[A_i] - d_2[A_i]|.$$

3. **Близость косинусов**. Косинус угла между векторами может оценивать их "однонаправленность"Близость косинусов пренебрегает длиной векторов и измеряет лишь различие в их направлениях:

$$sim(d_1, d_2) = \cos(d_1, d_2) = \frac{d_1 \cdot d_2}{||d_1|| \cdot ||d_2||} = \frac{\sum_{i=1}^{n} d_1[A_i] \cdot d_2[A_i]}{\sqrt{\sum_{i=1}^{n} d_1[A_i]^2} \cdot \sqrt{\sum_{i=1}^{n} d_2[A_i]^2}}.$$

Если d_1 и d_2 *параллельны*, то $sim(d_1, d_2) = 1$. Если же d_1 и d_2 *ортогональны*, то $sim(d_1, d_2) = 0$.

3.11 Совместное обучение

3.11.1 Усиление простых классификаторов

Усиление простых классификаторов - подход к решению задачи классификации (распознавания), путём комбинирования примитивных *слабых* классификаторов в один *сильный*. Под <силой> классификатора в данном случае подразумевается эффективность (качество) решения задачи классификации. Алгоритм AdaBoost (от английских слов *адаптивность* и *усиление*) был описан в 1996 в работе Freund и Schapire [10]. Он был успешно использован во многих областях, в частности для задачи поиска лиц на изображении [15].

Типичный сценарий агрегации (bagging) (повторная выборка). Для исходной выборки D размера n **улучшенная (bootstrap) выборка** из D это выборка состоящая из n примеров, извлеченных **случайно (с возвратами)** из D.

Замечание: Некоторые примеры могут входить по несколько раз.

Усиленная агрегация для обучения с учителем. Пусть задана обучающая выборка D и $|D| = N$. Построим **объединенный (bagging) классификатор** для D следующим образом:

Этап обучения: По заданным D, числу k и обучающему алгоритму BaseLearner:

1. создать k **улучшенных копий** $D_1, \ldots, D_k$ выборки D.
2. Для каждой копии D_i создать классификатор f_i, используя алгоритм BaseLearner.

Этап тестирования: По построенным классификаторам $f_1, \ldots, f_k$ и тестируемому примеру d:

1. Вычислить $f_1(d), \ldots f_k(d)$.
2. Выдать в качестве $class(d)$ класс большинства ответов из $f_1(d), \ldots, f_k(d)$.

3.11.2 Алгоритм AdaBoost

Улучшение (Boosting) — это набор методов для порождения последовательности классификаторов, в которой каждый последующий классификатор пытается исправить ошибки предыдущих.

Идея. Улучшение применяется к некоторому алгоритму классификации, называемому базовым классификатором (BaseLearner). Этот классификатор часто называют слабым классификатором. На каждом шаге алгоритм изменяет веса записей обучающей выборки, увеличивая веса неверно расклассифицированных записей.

AdaBoost. Алгоритм **Ada**ptive **Boost**ing [10] (AdaBoost) приведен на рис. 3.6.

Algorithm AdaBoost(D, BaseLearner, k) **begin**
  foreach $d_i \in D$ **do** $D_1(i) = \frac{1}{|D|}$;
  for $t = 1$ **to** k **do** //основной цикл
    f_t :=BaseLearner(D_t);
    $e_t := \sum_{class(d_i) \neq f_t(d_i)} D_t(i)$;
    // f_t строится для минимизации e_t
    if $e_t > 0.5$ **then** // большая ошибка: возврат
      $t := t - 1$;
      break;
    endif
    $a_t := \frac{1}{2} \ln \frac{1-e_t}{e_t}$; //параметр для изменения весов
     foreach $d_i \in D$ **do** $D_{t+1}(i) := D_t(i) \cdot e^{-\alpha_t \cdot class(d_i) \cdot f_t(d_i)}$; //изменения весов примеров
    $Norm_t := \sum_{i=1}^{|D|} D_{t+1}(i)$;
    foreach $d_i \in D$ **do** $D_{t+1}(i) := \frac{D_{t+1}(i)}{Norm_t}$; //нормализованные новые веса
  endfor

$f_final(.) := sign(\sum_{t=1}^{k} a_t \cdot f_t(.)$
end

Рис. 3.6: **AdaBoost**: адаптивный улучшающий алгоритм. Это вариант для выбора одного из двух классов $Y = \{-1, +1\}$.

1) Каждому примеру из $d \in D$ присваивается некоторый вес. На первом шаге $w(d) = \frac{1}{|D|}$.
2) На каждом шаге i строится очередной классификатор f_i.
Ошибочно классифицируемым примерам $d \in D$, для которых $f_i(d) \neq class(d)$, присваивается больший вес. На следующем шаге алгоритм классификации будет "уделять больше внимания"таким примерам.
3) Окончательный классификатор получается путем совместного взвешенного голосования классификаторов $f_1, \ldots f_k$.

Слабые классификаторы. Некоторые классификаторы используют известные веса обучающих примеров, но большинство, например, **C4.5**, их не используют. Чтобы превратить классификатор типа **C4.5** в слабый классификатор, подходящий для алгоритма **AdaBoost**, его можно изменить следующим образом:

- На шаге t по заданной выборке D_t построим обучающую выборку того же размера D'_t. Элемент d_i будет выбираться для D'_t с вероятностью $D_t(i)$. Таким образом "ошибочные"элементы могут входить в D'_t по нескольку раз.

Если имеются несколько алгоритмов классификации $\mathcal{A}_1, \dots \mathcal{A}_k$ построивших классификаторы $f_1, \dots f_k$, то для их комбинирования можно использовать **прямое голосование**. В этом случае объединенный классификатор для каждого примера $d \in D$ будет возвращать метку класса c, выдаваемою наибольшим числом классификаторов: $|\{i \mid f_i(d) = c\}| \geq |\{i \mid f_i(d) = c'\}|$ при $c' \neq c$.

3.12 Задачи

Задача 3.1. *Требуется оценить сложность (в худшем случае) алгоритма калассификации на основе деревьев решений. Пусть дана обучающая выборка D с числом примеров $|D|$, а число атрибутов в примерах равно n. Покажите, что сложность построения дерева решений $O(n \times |D| \times \log(|D|))$.*

Задача 3.2. *Рассмотрим одномерную выборку данных из Табл. 1. В ней атрибут y задает класс соответствующего x.*

ТАБЛИЦА 1

x	1.5	2.5	3.5	4.5	5.0	5.5	5.75	6.5	7.5	10.5
y	+	+	-	-	-	+	+	-	+	+

(a) Вычислите класс точки $x = 5.5$ в соответствии с ее 1- 3-, 6- и 9-ближайшими соседями (используя голосование по большинству).

(b) Пусть k ближйших к x точек выборки это $x_1, \dots, x_k$. Припишем точке x_i вес $w_i = 1/d(x, x_i)^2$. Взвешенный вариант алгоритма k-ближайших соседей вычисляет для каждого класса c_j сумму $s_j = \sum_{x_i \in c_j} w_i$ и определяет класс в соответствии с максимальным значением s_j.

Вычислите класс точки $x = 5.5$ в соответствии с ее 1- 3-, 6- и 9-ближайшими соседями, используя взвешенный вариант алгоритма k-ближайших соседей.

Какой из вариантов (a) или (b) более устойчив к изменению k?

Задача 3.3. *В некоторых приложениях значениями атрибутов могут быть множества. Например, объект может быть окрашен в несколько цветов и для классификации объекта может оказаться важно рассматривать цвет со значением-множеством, а не как обычный атрибут с конечным числом значений. Предложите вид тестов, которые можно использовать для сравнения атрибутов-множеств и покажите, как этот вид тестов можно использовать при построении C4.5- дерева решений.*

Задача 3.4. *В таблице 2 приведены данные о голосовании 15 избирателей на выборах в США в 2008г. Каждый избиратель характеризуется следующими атрибутами:*
Партия, Пол, Религия, Доход, Образование, Возраст, Регион, Голосование(класс).
(Классы: 1 – Обама, 2 – Мак Кейн)

ТАБЛИЦА 2. Голосование

N	*П,По,Рл,Д,О,В,Рг,Г*
1	*1, 2, 2, 1, 2, 2, 3, 2*
2	*3, 2, 3, 6, 3, 4, 4, 2*
4	*3, 1, 2, 2, 2, 3, 1, 2*
5	*2, 2, 1, 3, 1, 3, 1, 1*
6	*1, 1, 2, 1, 2, 2, 3, 2*
7	*1, 2, 1, 2, 1, 2, 4, 1*
8	*1, 1, 1, 1, 1, 3, 2, 2*
9	*3, 1, 2, 3, 3, 1, 1, 1*
10	*3, 2, 1, 6, 2, 3, 1, 1*
11	*1, 2, 1, 4, 1, 3, 3, 1*
12	*2, 2, 2, 5, 2, 2, 2, 1*
13	*3, 2, 1, 1, 3, 3, 1, 1*
14	*1, 1, 2, 3, 2, 1, 2, 1*
15	*3, 2, 1, 6, 2, 1, 3, 1*

Используя алгоритм CART, построить классифицирующее дерево для этих данных. Используйте в качестве дополнительного критерия остановки требование, чтобы построение дерева прекращалось, если в вершине не более 3-х объектов.

Проверьте Ваш классификатор на следующих примерах.

1,2,1,1,1,3,3,1
2,1,2,4,2,3,2,1
3,2,3,1,1,2,1,1
2,1,1,2,2,3,1,2
2,1,2,6,3,2,3,2
1,2,1,6,3,1,2,1
3,2,3,2,2,3,1,1
1,2,3,6,2,2,4,1
Сколько ошибочных предсказаний он выдал?

Задача 3.5. *Докажите, что ошибка правила ближайшего соседа при некоторых разумных предположениях ограничена сверху удвоенной ошибкой метода Баейса.*

Задача 3.6. *Докажите, что ошибка общего kNN-метода асимптотически оценивается ошибкой метода Баейса и может использоваться для ее аппроксимации.*

Задача 3.7. *Рассмотрим пример с покупкой дома. Информация о доиах и их посещениях представлена в Таблице 3. Каждый дом характеризуется слдедующими атрибутами: количество комнат (Комнаты), наличие полуподвала (Подвал), тип планировки (Планировка) и географическое местоположение (Место).*

ТАБЛИЦА 3. Покупка дома

Номер	Комнаты	Подвал	Планировка	Место	Посещение
1	3	Нет	традиционная	Юг	**Нет**
2	3	Есть	традиционная	Юг	**Да**
3	3	Есть	открытая	Север	**Нет**
4	3	Есть	традиционная	Север	**Нет**
5	3	Нет	открытая	Север	**Нет**
6	3	Есть	традиционная	Юг	**Да**
7	3	Есть	открытая	Юг	**Нет**
8	3	Нет	традиционная	Юг	**Нет**
9	4	Нет	традиционная	Юг	**Да**
10	4	Есть	открытая	Север	**Нет**
11	4	Есть	открытая	Юг	**Да**
12	4	Нет	традиционная	Север	**Нет**
13	4	Нет	открытая	Юг	**Да**
14	4	Есть	открытая	Юг	**Да**
15	4	Нет	традиционная	Север	**Нет**
16	4	Есть	открытая	Север	**Нет**

Используя алгоритм CART, постройте классифицирующее дерево для этих данных. Используйте в качестве дополнительного критерия остановки требование, чтобы построение дерева прекращалось, если в вершине не более 3-х объектов.

Определите матрицу неточности (Confusion Matrix) для полученного классификатора, величину его точности (precision) и возврата (Recall).

Задача 3.8. *В таб. 4 приведены сведения о покупателях компьютеров.*

Таблица 4. Покупатели электроники.

Номер	Возраст	Доход	Студент	Кредитный рейтинг	Класс: покуп. комп.
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle	medium	no	excellent	yes
13	middle	high	yes	fair	yes
14	senior	medium	no	excellent	no

а) Используя алгоритм C4.5, построить классифицирующее дерево для этих данных. Используйте в качестве дополнительного критерия остановки требование, чтобы построение дерева прекращалось, если в вершине не более 3-х объектов.

б) Используя алгоритм "Наивный Баейс определите прогнозируемые классы 5 следующих покупателей и сравните с их настоящими классами, указаными в скобках. 1 (senior, low , yes, fair, (no))
2 (youth, medium, yes, fair, (yes))
3 (youth, medium, no, excellent, (no))
4 (middle , high, no, fair, (yes))

5 (senior, high, no, excellent, (no))

в) В каждом из вариантов (а) и (б) определите матрицу неточности (Confusion Matrix) для полученного классификатора, величину его точности (precision) и возврата (Recall).

Задача 3.9. *Одна из особенно привлекательных черт метода "Наивный Баейс"заключается в простоте его применения. Предложите способ последовательной модификации этого классификатора при появлении новых примеров в обучающей выборке.*

Глава 4

Кластеризация / Обучение без учителя

4.1 Основные понятия

Кластеризация . **Кластеризация** это процесс организации исходных данных – объектов, экземпляров, примеров, образцов, наблюдений, векторов свойств – в группы, элементы которых близки друг к другу или похожи в некотором смысле друг на друга.

Кластер. **Кластер** представляет из себя совокупность экземпляров данных, которые предполагаются *похожими* друг на друга и *отличными* от других экземпляров данных. Для классифицируемых элементов используются различные синонимы: **экземпляр данных**= **объект**=**пример** = **образец** = **точка** = **наблюдение**= **вектор свойств**. Мы будем чаще всего использовать термины "точка " и "объект".

Приведенное определение кластера весьма неформально. Имеется достаточно много различных видов кластеров (некоторые показаны на рис. 4.1). Решение задачи кластеризации для каждого из них требует разработки своих алгоритмов.

Набор данных или входная **выборка** для задачи **кластеризации** представляет из себя совокупность $D = \{x_1, \ldots, x_n\}$ точек данных над множеством атрибутов (свойств, измерений) $A = \{A_1, \ldots, A_M\}$. Для каждой пары точек x_i, x_j определено расстояние или мера близости $d(x_i, x_j)$. Матрица "расстояний" или "близости " $\mathcal{D} = [d(x_i, x_j)]$ размера $n \times n$ и служит основным входом алгоритмов кластеризации.

Замечание: Основное отличие между **кластеризацией** и **классификацией** состоит в том, что в задаче **классификации**каждый пример в в обучающей выборке включает значение атрибута **класса**, заранее заданное "учителем". При кластеризации классы объектов (примеров) заранее не заданы. Задача состоит *не в предсказании* класса для каждого объекта, а *в разбиении исходных точек на группы в соответствии с их предполагаемой близостью*.Поэтому *кластеризацию* называют *обучением без учителя.*

Пример. Рассмотрим четыре диаграммы данных, приведенне на рис. 4.1.

- Диаграмма (а) содержит пример трех легко отделимых кластеров с ясными границами и разделением их точек.

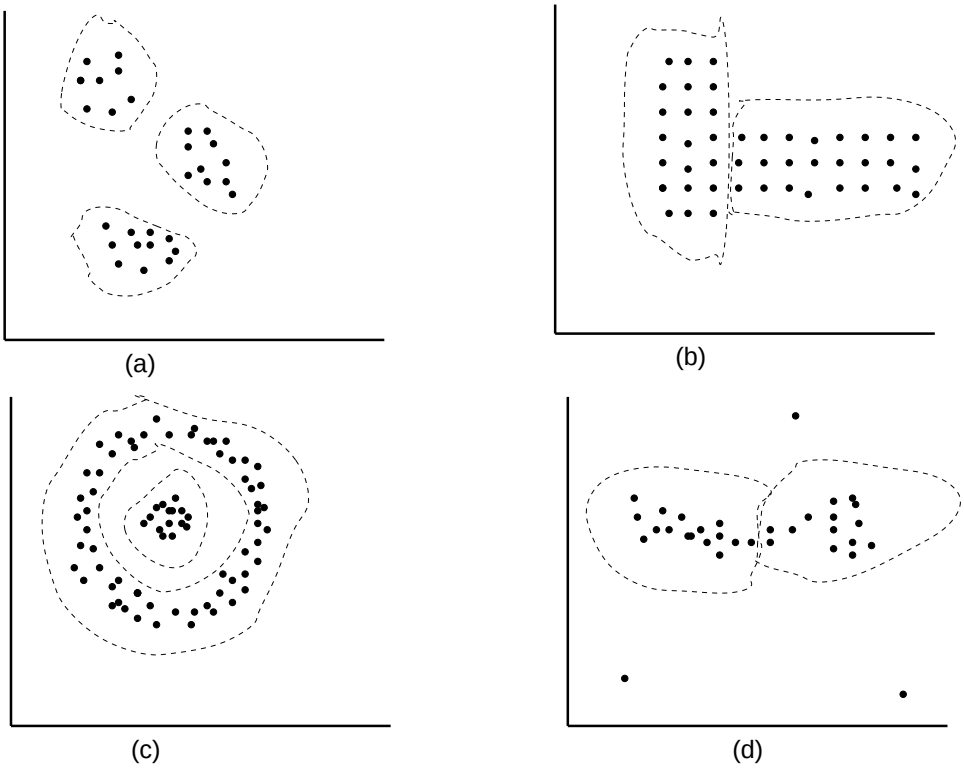

Рис. 4.1: Данные для кластеризации.

- На диаграмме (b) приведен пример двух кластеров, которые достаточно просто идентифицировать. Но границы этих кластеров и точное разбиение точек по ним нуждаются в дополнительном обсуждении и не всякий алгоритм кластеризации сможет корректно определить кластер каждой точки.
- На диаграмме (c) показан пример двух кластеров, для которых вхождение точек определяется не только близостью. Алгоритмы кластеризации, которые распределяют точки по кластерам, руководствуясь исключительно степенью их близости между собой, не смогут правильно идентифицировать кластеры-кольца.
- На диаграмме (d) показан пример двух кластеров с "мостом"между ними, что делает задачу кластеризации в этом случае трудной как для машин, так и для людей. На этой диаграмме также имются три *выброса - изолированных* точки, которые лежат вне кластеров. Если алгоритм кластеризации не сможет их выявить и отделить, то он не сможет из-за этого правильно определить границы кластеров.

Алгоритмы кластеризации

Имеется несколько различных “измерений”, по которым различаются алгоритмы кластеризации: алгоритмы разбиения – иерархические алгоритмы, однозначные – с перекрытием – нечеткие, полные – частичные.

Алгоритмы разбиения – это алгоритмы кластеризации, которые просто разбивают исходные наборы данных на непересекающиеся подмножества (кластеры) так, что каждый объект входит ровно в один кластер.

Иерархические алгоритмы – это алгоритмы кластеризации, которые создают иерархически организованные совокупности кластеров – *деревья (дендрограммы)*. Каждая вершина-кластер в дереве является объединением всех своих сыновей (подкластеров), а корень – это кластер, включающий все объекты. Конкретные кластеризации получаются при выборе сечений дерева.

Однозначные алгоритмы – это алгоритмы, которые помещают каждый объект в единственный класс. В ряде ситуаций объекту разумно приписать несколько классов. Алгоритмы, которые так поступают, называются *алгоритмами с перекрытием* или *неоднозначными*. При *нечеткой кластеризации* каждому объекту приписывается его степень принадлежности (число между 0 и 1) каждому классу.

Алгоритмы полной кластеризации – присваивают классы всем объектам, а при *частичной кластеризации* некоторым объектам класс может быть не назначен. Причинами могут быть шумы, выбросы, неинтересный фон и т.д.

4.2 Алгоритмы разбиения:семейство алгоритмов k-средних (k-Means)

Замечание об истории. Алгоритм кластеризации k-средних (k-Means) открывался и переоткрывался много раз в различных областях. Он появился в работах Lloyd (1957) [17], Forgey (1965)[9], Friedman и Rubin (1967) и McQueen (1967).

Применимость. Набор данных $D = \{x_1, \ldots, x_n\}$, $x_i = (x_{i1}, \ldots x_{iM}) \in \mathcal{R}^M$.

Для использования **алгоритма k-means** нужно, чтобы для области значений каждого атрибута A_i имелось понятие **среднего значения**.

Набросок алгоритма. На вход алгоритма поступает набор данных $D = \{x_1, \ldots x_n\}$ и целое число k — количество кластеров, на которые требуется разбить данные.

Алгоритм работает следующим образом.

1. Выбираются k начальных центров классов.
2. На каждом этапе для каждой точки данных вычисляется расстояние до каждого из центров и точка относится к кластеру **ближайшего к ней центра**.
3. Перевычисляются центры кластеров.
4. Этапы 2 и 3 повторяются пока процесс не сойдется, т.е. центры перестанут изменяться.

Детализация: центры кластеров.
Для выбора начальных центров можно использовать один из следующих способов:

- Выбрать случайно k точек из набора данных (их называют **семена**).
- Использовать следующую процедуру селекции SelectCentroids:
    1. Вычислить **центр** c всего набора D.

2. Первый центр m_1 – это такой $x \in D$, что $d(m_1, c) = \max_{x \in D}(d(x, c))$ (точка наиболее отстоящая от центра всего набора).
3. Выбрать m_2 так, чтобы $d(m_1, m_2) = \max_{x \in D}(d(m_1, x))$.
4. Выбрать m_i так, чтобы $\sum_{j=1}^{i-1}(d(m_j, m_i)) = \max_{x \in D} \sum_{j=1}^{i-1}(d(m_j, x))$.

- Выбрать подмножество $S \subset D$, $|S| > k$. Применить описанную выше процедуру к S. (Это помогает бороться с **выбросами**).
- Выбрать семена перед началом работы алгоритма (например, взять первые k точек из D).

Перевычисление центра:

- Пусть $m_{t,1}, \ldots m_{t,k}$ – это k центров на этапе $t > 1$. Пусть $C_{t,1}, \ldots, C_{t,k}$ – это k кластеров, построенных на этапе t. Новые центры $m_{t+1,1}, \ldots, m_{t+1,k}$ вычисляются как **средние** точек из кластеров, полученных на предыдущем этапе:

$$m_{t+1,i} = \frac{1}{|C_{t,i}|} \sum_{x \in C_{t,i}} x.$$

Критерии остановки. Алгоритм k-means может использовать любые из следующих критериев остановки (сходимости):

1. никакое (или минимальное) перераспределение точек между кластерами;
2. никакое (или минимальное) изменение центров кластеров;
3. несущественное уменьшение **суммы квадратов ошибок**:

$$SSE = \sum_{j=1}^{k} \sum_{x \in C_{t,j}} d^2(x, m_{t,j}).$$

Дисковый вариант алгоритма кластеризации k-means . На рис. 4.2 приведен псевдокод варианта алгоритма k-means, который работает с данными, находящимися во вторичной памяти (на диске).

Перечислим свойства этого варианта алгоритма кластеризации k-means.

- *Одно сканирование данных* на каждой итерации.
- *Малая память.* Алгоритм работает по принципу *одна запись в каждый момент*. Для сканирования данных достаточно иметь один блок диска.
- *Распределение записей по кластерам* не используется в коде, за исключением проверки условий останова. При необходимости его можно выполнять внутри самих данных. Это потребует выполнения на каждой итерации одной дополнительной операции ввода-вывода для каждого блока диска.
- В случае, когда проверка условия останова не требует рассмотрения всех точек в каждом кластере (например, базируется на смещении центров кластеров), для минимизации операций ввода-вывода можно изменить алгоритм следующим образом:

```
Алгоритм diskKMeans(D,k);
begin
  m[] := SelectInitialCentroids(D,k);
  repeat
    for j := 1 to k do
      s[j] := (0,0,0,...,0);   // s[] - семейство векторов размера dim(D)
      num[j] := 0;    // num[] - число точек в каждом кластере
      cl[j] := ∅;   // cl[] - текущие кластеры
    endfor
    foreach x ∈ D do
      cluster := arg min_{j=1,...,k}(dist(x, m_j)); //назначение кластера точке x
      cl[cluster] := cl[cluster] ∪ {x};
      s[j] := s[j] + x;
      num[j] := num[j] + 1;
    endfor
    for j := 1 to k do m[j] := s[j]/num[j];
  until isStoppingCondition(m[],cl[]) = true;
  output cl[];
end
```

Рис. 4.2: Дисковый вариант алгоритма кластеризации k-means.

- устранить из алгоритма $cl[]$. Кластеры для точек вычисляются, но *не запоминаются* после использования для изменений $num[]$ и $s[]$;
- после выполнения условия остановки запускается процесс кластеризации, который выполняется за один проход по D. В этом случае $cl[]$ используется только для вывода кластеров.

Этот вариант требует дополнительного прохода по набору данных в конце алгоритма, но при этом общее число операций ввода-вывода уменьшается в два раза.

Сила и слабость алгоритма кластеризации k-means

Достоинства. Основными достоинствами алгоритма k-means являются: его *простота* и *эффективность*

Время работы алгоритма можно оценить как $O(tkn)$, где $n = |D|$, k – число кластеров, а t – число итераций. *Число обращений к данным* не превосходит $t \cdot B(D)$, где $B(D)$ — число блоков диска с данными из D.

Недостатки и способы их преодоления.

- **Применимость.** Для области значений каждого атрибута должно быть определено *среднее значение* и задано правило его вычисления.

  Замечание. Мы привели вариант алгоритма k-means для точек в вещественном пространстве $\mathcal{R}^M$. Существует также вариант алгоритма k-means, называемый *кластеризация в k-режимах (k-modes clustering)*, который можно использовать для данных с *категориальными атрибутами*.

- **Необходимость параметра** k. Пользователь должен заранее "угадать"правильное значение k. Если k не совпадает с истинным числом кластеров дынных, то алгоритм k-means может выдавать плохие результаты.

  Пример. На рис. 4.3.(a) показано множество данных, состоящее из двух кластеров (кругов). Если запустить алгоритм k-means со значением $k = 3$, то один из созданных им кластеров будет содержать точки двух разных кругов, т.е. реальных кластеров.

  Замечание. Чтобы определить правильное число кластеров, можно запустить алгоритм k-means несколько раз с возрастающими значениями k. *Как правило, число кластеров не очень велико, поэтому эта процедура вполне выполнима.*

- **Чувствительность к начальному выбору центров**. Неудачно выбранные в начале центры могут привести к неверным результатам даже, в случаях хорошо выделяемых кластеров.

  Пример. Рассмотрим рис. 4.3.(b). Если в качестве исходных центров будут случайно выбраны точки одного кластера (они показаны в прямоугольниках), то результом кластеризации оказываются два неверных кластера, показанных в прямоугольниках со штриховыми границами. На рис. 4.3.(d) показаны результаты кластеризации алгоритма k-means в случае, когда исходные центры находятся в разных кластерах.

  Как справиться с чувствительностью алгоритма к начальному выбору центров.

  – *Случайный выбор* хорошо работает для больших значений $|D|$.

  – Использовать описанную выше процедуру SelectCentroids. Она пытается разместить начальные центры кластеров *чем подальше друг от друга.*

  – "Вручную"выбрать исходные семена до начала работы алгоритма k-means.

- **Чувствительность к выбросам.**

  > ***Выбросы*** *– это точки данных, расположенные очень далеко от других точек (т.е. изолированные точки).*

  Алгоритм k-means не выделяет выбросы. На каждом этапе выбросы входят в кластеры и могут оказать существенное и нежелаемое влияние на определение центров кластеров. Это может привести к неверному определению границ кластеров.

  Пример. Рассмотрим рис. 4.3.(c). Набор данных состоит из двух кластеров и одного выброса. Если выявить этот выброс, то оба кластера определяются корректно. Иначе, появление выброса в одном из кластеров смещает положение центра этого кластера и позволяет другому кластеру “аннексировать” некоторые чужие точки.

  Как справиться с чувствительностью алгоритма к выбросам.

  – *Выявление выбросов в процессе работы алгоритма k-means.* Находить точки, которые очень далеко отстоят от центров их кластеров и удалять их из рассмотрения. Для определения того, что точка расположена *чересчур далеко от центра кластера*, использовать некоторый заранее заданный *порог.*

  – *Случайный выбор точек.* Случайно выбрать некоторое подмножество данных и построить его кластеризацию. Вероятность попадания выбросов в это подмножество *очень мала.*

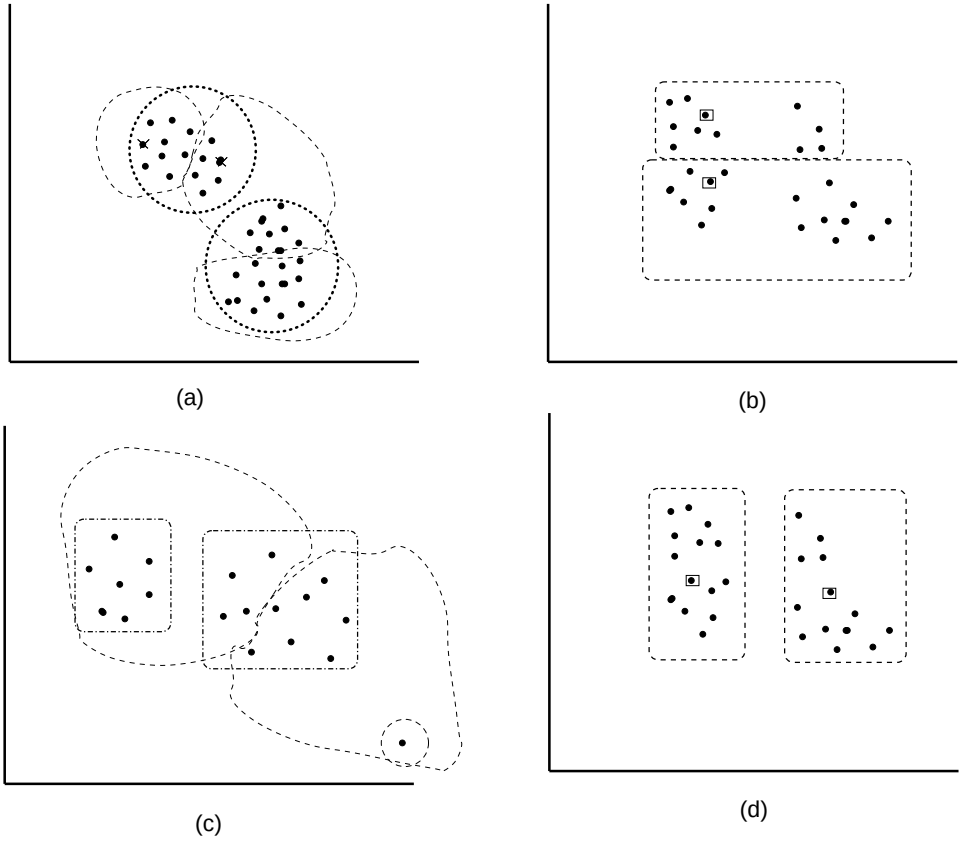

Рис. 4.3: Иллюстрация недостатков алгоритма кластеризации k-средних.

- **Не может правильно выделять кластеры с близкими центрами.**

  Если центры двух кластеров расположены рядом друг с другом, то алгоритм k-means не в состоянии правильно разделить эти кластеры.

  Пример. Рассмотрим рис. 4.1.(c). На нем центры двух кластеров – внешнего кольца и внутреннего ядра – расположены близко друг от друга. Поэтому вместо выделения *внутреннего и внешнего* кластеров алгоритм k-means выдаст разбиение на *левый - правый* или *верхний-нижний* или аналогичное разбиение (исходя из первоначального выбора положения центров).

  По-существу, алгоритм k-means *подменяет проблему разнесения точек по разным кластерам проблемой раделения центоров кластеров.* В нашем примере он будет стараться увеличивать расстояние между центрами кластеров.

  Что делать, если кластеры имеют сложную форму или близкие центры.

  Иногда это не является препятствием для кластеризации из-за каких-то специфических свойств входных данных. Однако, в общем случае, *не существует хороших способов справиться с этими трудностями.*

4.3 Представления кластеров

Для демонстрации результатов работы алгоритмов кластеризации используются разные представления кластеров. Выбор того или иного из них зависит от специфики рассматриваемой предметной области и предпочтений пользователей.

1) Кластер = набор точек. В этом варианте каждый кластер представляется как совокупность всех входящих в него точек.

- *Это представление хорошо* для приложений, заинтересованных в свойствах отдельных точек данных.
- *Это представление плохо* для приложений, заинтересованных в *простых*, *интуитивно понятных* описаниях обнаруженных кластеров.

2) Кластер = центр + радиус. В этом варианте каждый кластер задается как шар с заданными центром и радиусом.

- *Это представление хорошо*, так как является очень *сжатым* (и простым). Информация о нем может быть непосредственно получена из *алгоритма кластеризации k-means* и не требуется никаких дополнительных вычислений.
- *Это представление плохо*, так как является очень *грубым*. В частности, оно может привести к *частичному перекрытию кластеров*.

3) Кластер = Класс. Каждому полученному кластеру присваивается некоторое имя - *метка класса*. Эта метка добавляется в качестве значения нового атрибута ко всем точкам кластера. Для полученной новой обучающей выборки D' строится одним из алгоритмов обучения с учителем классификатор, обладающий объяснительной способностью (например, *дерево решений* или набор *ассоциативных правил*).

Кластер представляется как множество точек, которые относятся построенным классификатором к соответствующему классу.

- *Это представление хорошо* тем, что обладает *объяснительной способностью*. Можно четко сформулировать, что собой на самом деле представляет данный кластер.
- *Это представление плохо* тем, что его получение требует больших дополнительных затрат времени и других ресурсов.
- Кроме того, попытка получить *содержательно понятное* объяснение может оказаться неудачной.

4) Кластер = частые значения. В этом варианте каждый кластер представляется с помощью небольшого подмножества *часто встречающихся* или *типичных точек данных*.

- *Это представление хорошо*, потому что следует интуитивному принципу *"Все вот такое"*.
- Для выделения *хороших представителей кластера* может потребоваться дополнительная работа.
- Позволяет сравнивать новые точки данных.
- Но может также упустить некоторую важную информацию о структуре кластера (см.,например, рис. 4.1.(c)).

4.4 Меры близости и сходства атрибутов и объектов (точек)

4.4.1 Сходство и различие объектов

Для применения алгоритма k-means и других алгоритмов кластеризации требуется уточнить способ определения "расстояния" между значениями одного атрибута и между точками выборки. Для этого используются двойственные понятия: меры сходства и меры различия.

Сходство (similarity): измеряет то, насколько близки друг к другу две точки (два объекта, примера). Чем "ближе" они друг к другу, тем больше значение их схожести.

Несходство, различие (dissimilarity): измеряет то, насколько отличаются две точки (два объекта, примера) друг от друга. Различие велико, когда объекты очень разные, и мало, когда они близки.

Термин **соседство (proximity)**используют для *сходства* или для *различия*.

Мера близости, расстояние (distance metric) – это мера различия, удовлетворяющая следующим законам (аксиомам треугольной нормы):

- $d(x, y) \geq 0$; $d(x, y) = 0 \Leftrightarrow \ x = y$;
- $d(x, y) = d(y, x)$;
- $d(x, y) + d(y, z) \geq d(x, z)$.

Преобразования между мерами сходства и различия

Как правило, меру сходства можно "перевернуть" так, чтобы получить меру различия и наоборот.

Такое преобразование можно проводить по-разному. Например, если d измеряет расстояние, то в качестве соответстствующей меры близости можно использовать

$$s(x, y) = \frac{1}{d(x, y)}$$

или

$$s(x, y) = \frac{1}{d(x, y) + 0.5}$$

Если s – мера сходства со значениями между 0 и 1 (так называемая *степень сходства*), то соответствующие меры различия можно определить как

$$d(x, y) = 1 - s(x, y)$$

или

$$d(x, y) = \sqrt{(1 - s(x, y))}.$$

В общем случае, можно использовать *всякое монотонно убывающее отображение* для преобразования мер сходства в меры различия, а *всякое монотонно возрастающее отображение* можно использовать для преобразования в обратном направлении.

4.4.2 Метрики расстояний для числовых атрибутов

В случае *стандартного* представления исходной выборки каждый ее элемент можно рассматривать как вектор $\bar{x} = (x_1, \ldots, x_N)$ значений атрибутов с номерами $1, \ldots, N$.

Рассмотрим вначале варианты метрик для числовых атрибутов.

Евклидово расстояние.

$$d_E(\bar{x}, \bar{y}) = \sqrt{\left(\sum_{k=1}^{N}(x_k - y_k)^2\right)}.$$

Квадрат Евклидова расстояния

$$d_E(\bar{x}, \bar{y}) = \sum_{k=1}^{N}(x_k - y_k)^2.$$

Манхетенское расстояние или расстояние по Хеммингу.

$$d_m(\bar{x}, \bar{y}) = \sum_{k=1}^{N}|x_k - y_k|.$$

Расстояние Минковского является обобщением Евклидова и Манхетенского расстояний:

$$d_{M,\lambda}(\bar{x}, \bar{y}) = \left(\sum_{k=1}^{N}(x_k - y_k)^{\lambda}\right)^{\frac{1}{\lambda}}.$$

В частности,: $d_{M,1} = d_m$, $d_{M,2} = d_E$.

Интересно также
Расстояние Чебышева.

$$d_{M,\infty}(\bar{x}, \bar{y}) = \max_{k=1,\ldots,N}(|x_k - y_k|).$$

Все эти метрики являются *аддитивными* и *равнозначными. Аддитивность* означает что вклады разных атрибутов независимы и суммируются. *Равнозначность (commesurability)* означает, что числовые значения разных атрибутов вносят равный по значимости вклад в оценку расстояния между объектами. Например, равнозначными являются все три координаты точки в 3-х мерном пространстве.

Стандартизованное (нормализованное) Евклидово расстояние.

Во многих приложениях разные атрибуты могут иметь разные шкалы измерений. Если некоторые атрибуты не равнозначны, то можно попытаться "нормализовать" их, разделив значение на отклонение от среднего значения.

Стандартизация области. Каждая точка стандартизуется путем отображения в некоторую точку единичного куба. Значение каждого атрибута x_i точки $\bar{x} = (x_1, \ldots, x_N)$ стандартизуется следующим образом:

$$x_i' = \frac{x_i - \min_{\bar{y}\in D}(y_i)}{\max_{\bar{y}\in D}(y_i) - \min_{\bar{y}\in D}(y_i)}.$$

Стандартизация z-значения . Предположим, что значения атрибута распределены по нормальному закону. Нормализуем данные, используя среднее значение и среднеквадратичное отклонение.

Среднеквадратичное отклонение для i-го атрибута:

$$\hat{\sigma}_i = \sqrt{\left(\frac{1}{n-1}\sum_{i=j}^{n}(\bar{x_j}[i] - \mu_i)^2\right)},$$

где n – число примеров в наборе данных, а

$$\mu_i = \frac{\sum_{j=1}^{n}\bar{x_j}[i]}{n},$$

это среднее значение i-го атрибута.

Стаддартным z-значением вектора $x = (x_1, \dots x_N)$ является:

$$\hat{x} = (x'_1, \dots, x'_N) = \left(\frac{x_1 - \mu_1}{\hat{\sigma}_1}, \dots \frac{x_N - \mu_N}{\hat{\sigma}_N}\right)$$

Стандартизованным Евклидовым расстоянием тогда служит величина:

$$d_{SE}(\bar{x}, \bar{y}) = d_E(\hat{x}, \hat{y}).$$

Взвешенное расстояние.

Разные атрибуты могут иметь разную *важность* при определении расстояния. Часто эта важность задается с помощью *веса атрибута*. Для заданного вектора $w = (w_1, \dots, w_N)$ весов атрибутов взвешенное расстояние Минковского определяется как:

$$d_{WM,\lambda}(\bar{x}, \bar{y}) = \left(\sum_{k=1}^{N} w_k \cdot (x_k - y_k)^\lambda\right)^{\frac{1}{\lambda}}.$$

Отсюда можно извлечь формулы для взвешенного расстояния по Евклиду и по Хеммингу.

4.4.3 Меры близости для категориальных атрибутов

Расстояние между двоичными векторами

Рассмотрим вначале случай, когда все атрибуты объекта являются двоичными, т.е. принимают по два значения: $\{0, 1\}, \{+, -\}$, {да, нет } и т.п. Не ограничивая общности, будем считать, что все атрибуты используют значения $\{0, 1\}$. Тогда каждый объект определяется *двоичным вектором* – вектором вида $\bar{v} = (v_1, \dots, v_n) \in \{0, 1\}^n$, где n – это число атрибутов.

Матрица отличия (confusion matrix) для двоичных векторов. Пусть $\bar{x} = (x_1, \dots, x_n)$ и $\bar{y} = (y_1, \dots, y_n)$ – два двоичных вектора.

Для каждого атрибута $A_i,\ i = 1 \dots n$, возможны четыре случая:

N	x_i	y_i
(1)	1	1
(2)	1	0
(3)	0	1
(4)	0	0

Подсчитаем число атрибутов для каждого из этих случаев и объединим эти числа в **матрице отличия** вида:

	$x_i = 1$	$x_i = 0$
$y_i = 1$	A	B
$y_i = 0$	C	D

Симметричные атрибуты. Двоичный атрибут называется **симметричным**, если оба его значения 0 и 1 имеют одинаковую значимость (например, Мужчина и Женщина для атрибута Пол).

Если атрибуты двоичных векторов симметричны, то можно использовать следующие способы определения расстояний:

Расстояние по простому совпадению:

$$d_s(\bar{x}, \bar{y}) = \frac{B + C}{A + B + C + D}.$$

Взвешенное расстояние по совпадению:

$$d_{s,\alpha}(\bar{x}, \bar{y}) = \frac{\alpha \cdot (B + C)}{A + D + \alpha \cdot (B + C)},$$

или

$$d_{s,\alpha}(\bar{x}, \bar{y}) = \frac{B + C}{\alpha \cdot (A + D) + B + C}.$$

Несимметричные атрибуты. Двоичный атрибут **несимметричен**, если одно из его значений более важно чем другое (например, истина и ложь, присутствие и отсутствие). Предположим, что значение 1 более важно чем 0. Тогда растояние между веторами можно определить, игнорируя совпадения нулей.

Расстояние Жаккарда (Jaccard):

$$d_J(\bar{x}, \bar{y}) = \frac{B + C}{A + B + C}.$$

Взвешенное расстояние Жаккарда

$$d_{J,\alpha}(\bar{x}, \bar{y}) = \frac{\alpha \cdot (B + C)}{A + \alpha \cdot (B + C)}.$$

или

$$d_{J,\alpha}(\bar{x}, \bar{y}) = \frac{B + C}{\alpha \cdot A + B + C}.$$

Категориальные атрибуты (не двоичные)

В этом случае естественно определять **расстояние по простому совпадению**:

$$d_s(\bar{x}, \bar{y}) = \frac{n - q}{n},$$

где: n – число атрибутов в $\bar{x}$ и $\bar{y}$, а q – число атрибутов в $\bar{x}$ и $\bar{y}$ с совпадающими значениями.

4.4.4 Определение расстояния для объектов с зависимыми атрибутами

Иногда некоторые атрибуты *коррелируют* друг с другом (например, разные меры одного свойства). Если этого не учитывать, то такие атрибуты могут исказить расстояние между элементами.

Чтобы корректно определить расстояние в этом случае можно использовать *коэффициенты корреляции* и *коэффициенты ковариации*.

Ковариационная матрица

Коэффициент ковариации i-го и j-го атрибутов

$$cov(i,j) = \frac{1}{n}\sum_{k=1}^{n}(x_{ki} - \mu_i)(x_{kj} - \mu_j),$$

где μ_i и μ_j средние значения i-го и j-го атрибутов. Из этих коэффициентов составляется ковариационная матрица $C = (cov(i,j))$. Очевидно, матрица C симметрична.

Можно также стандартизировать коэффициенты ковариации. *Коэффициент корреляции* определяется как

$$\rho(i,j) = \frac{\sum_{k=1}^{n}(x_{ki} - \mu_i)(x_{kj} - \mu_j)}{(\sum_{k=1}^{n}(x - \mu_i)^2 \sum_{k=1}^{n}(x - \mu_j)^2)^{\frac{1}{2}}}.$$

Из этих коэффициентов можно образовать матрицу корреляции $\mathcal{S} = (\rho(i,j))$.

Эта матрица используется для определения **расстояние Маханолобиса**:

$$d_{MH}(\bar{x},\bar{y}) = (\bar{x} - \bar{y})^T \mathcal{S}^{-1}(\bar{x} - \bar{y}),$$

здесь $\bar{x}$, $\bar{y}$ рассматриваются как столбцы.

Отметим, что коэффициенты ковариации/корреляции учитывают только линейные зависимости между значениями атрибутов. Нелинейные связи ими не учитываются.

4.5 Иерархическая кластеризация

Иерархическая кластеризация включает методы построения **дендрограммы (dendrogram)** по входной выборке. В отличие от методов разбиения методы **иерархической кластеризации** не создают разбиения входных данных на непересекающиеся кластеры. Вместо этого данные организуются в дендрограмму на основе предполагаемой схожести.

Дендрограмма выборки – это **размеченное бинарное дерево** со следующими свойствами:

- Все **листья** помечены точками входной выборки D. Каждая точка D является меткой одного и только одного листа.
- **Внутренние вершины** дерева имеют метки, которые обычно являются **вещественными числами**.

- Каждая внутренняя вершина представляет кластер данных с определенной степенью близости. Метка такой вершины задает степень (порог) близости между данными двух *кластеров-потомков*.

Разрезав дендрограмму "горизонтально" можно получить кластеры. Таким образом, разбиение на кластеры определяется некоторым разрезом. А сам разрез задается значением некоторого **порога** схожести точек внутри кластера.

Разрез. **Разрез** определяется по **дендрограмме** и заданному **порогу** α следующим образом:

> Все вершины дендрограммы с метками большими α удаляются вместе с инцидентными им ребрами. Полученный **лес** представляет кластеры, полученные из денрограммы с порогом α.

Существует два вида алгоритмов иерархической кластеризации:

Дивизимные (делимые): эти алгоритмы начинают с рассмотрения всей выборки как единого кластера. Затем на каждом шаге они стараются расщепить рассматриваемый кластер на два и построить соответствующую часть денрограммы. **Дивизимные алгоритмы кластеризации** строят кластеры **сверху вниз**.

Агломеративные алгоритмы: эти алгоритмы вначале рассматривают каждую точку как отдельный кластер. На каждом шаге алгоритм старается слить два кластера в один и расширить дендрограмму вверх. **Агломеративные алгоритмы кластеризации** строят кластеры **снизу вверх**.

Ниже мы рассмотрим агломеративные алгоритмы.

4.5.1 Агломеративные алгоритмы иерархической кластеризации

Вход. Входная выборка $D = \{x_1, \dots x_n\}$.
Выход. **Дендрограмма** T для выборки D.

Идея алгоритма. Алгоритм работает следующим образом:

1. На шаге 1 каждой точке $x \in D$ присваивается ее собственный кластер.
2. На каждом шаге алгоритм вычисляет **матрицу расстояний** для текущего набора кластеров.
3. Затем он выбирает пару наиболее близких кластеров и объединяет их в один (увеличивая соответствующую часть дендрограммы).
4. Алгоритм завершает работу, когда все точки будут объединены в один кластер.

Алоритм. Псевдокод этого алгоритма приведен на рис. 4.4.

```
Алгоритм Agglomerative(D).
begin
  foreach x_i ∈ D do C_1i = {x_i};
  C_1 := {C_11, ..., C_1n}; /* C_i -список кластеров на i-ом шаге
  i := 1;
  while |C_i| > 1 do
      for j = 1 to |C_i| do
          for k = j + 1 to |C_i| do
              d[j, k] := dist(C_ij, C_ik);
          endfor
      endfor
      (s, r) := args min d[j, k]
      for j := 1 to |C_i| do
          if j ≠ r and j ≠ s then C_i+1,j := C_ij
          else if j = r then C_i+1,j := C_ir ∪ C_is;
  end while
end
```

Рис. 4.4: Алгоритм агломеративной кластеризации.

4.5.2 Расстояние между кластерами

При определении расстояния между кластерами используются разные методы. Перечислим основные из них.

Метод ближайших соседей. Расстояние между двумя кластерами определяется как расстояние между двумя ближайшими точками этих кластеров.

Метод далеких соседей. Расстояние между двумя кластерами определяется как **расстояние между двумя самыми далекими точками** этих кластеров.

Метод средней связи. Расстояние между двумя кластерами определяется как **среднее всех попарных** расстояний между их точками.

Центроидный метод. Расстояние между двумя кластерами определяется как **расстояние между их центрами**.

Метод Уорда (Ward). Расстояние между двумя кластерами определяется как **прирост суммы квадратов ошибок (расстояний до центров)**.

4.6 Алгоритм, основанный на плотности точек: DBSCAN

При кластеризации, основанной на плотности, алгоритмы пытаются выделить области высокой плотности точек, разделенные областями с малой их плотностью. Рассмотрим один

из простых алгоритмов этой группы – DBSCAN, предложенный в работе [7]. В этом алгоритме плотность оценивается как число точек, находящихся в круге определенного радиуса с центром в данной точке.

В зависимости от задаваемых пользователем параметров – радиуса ϵ и порога для числа точек в круге MinPts – все точки выборки разбиваются на три группы:

1) *точки ядра* – это точки, для которых не менее MinPts точек выборки находятся на расстоянии не больше ε;

1) *граничные точки* – это точки, не попавшие в ядро, но входящие в ϵ-окрестность одной или нескольких точек ядра;

1) *точки выбросов* – это точки выборки, не вошедшие в ядро и не являющиеся граничными.

Алгоритм DBSCAN объединяет в один кластер находящиеся на расстоянии $\leq \varepsilon$ точки ядра. Каждая граничная точка добавляется в кластер с соответствующей ей точкой ядра. Точки выбросов удаляются.

Алгоритм DBSCAN(D, ε, $MinPts$);
1. Разбить D на точки ядра, граничные точки и точки выбросов.
2. Удалить все точки выбросов.
3. Связать ребрами все точки ядра, находящиеся на расстоянии не больше ε.
4. Объединить в кластеры связные компоненты получившегося неориентированного графа.
5. Каждую граничную точку поместить в кластер, содержащий близкие для нее точки ядра. Если таких кластеров несколько выбрать тот, к точкам которого данная граничная точка ближе.

Рис. 4.5: Алгоритм кластеризации $DBSCAN$.

Как выбирать параметры $DBSCAN$?

Общий подход состоит в оценке расстояния, на котором находятся k ближайших к точке соседей. Назовем его k-*близостью*. Если плотность разных кластеров близка, то и у их точек k-близость будет мало различаться. Конечно, в выборке могут оказаться точки (например, выбросы), у которых k-близость имеет гораздо большее значение. Но в типичных случаях, если упорядочить точки по возрастанию k-близости, то будет видна граница, после которой идет резкий скачок этой величины. Если выбрать эту границу в качестве ε, а k в качестве $MinPts$, то точки с k-близостью $\leq \varepsilon$ попадут в ядро выборки, а остальные точки окажутся граничными или выбросами.

Конечно, и при этом подходе остается проблема выбора подходящего k. Например, для двумерных данных подходящим часто считают число $k = 4$.

Сложность алгоритма $DBSCAN$. Время работы алгоритма можно оценить как $O(|D| \times t(\varepsilon))$, где $t(\varepsilon))$ – это время поиска точек в ε-окрестности. Оно не превосходит $O(|D|)$. Поэтому в худшем случае время работы алгоритма $DBSCAN$ можно оценить как $O(|D|^2)$.

Что касается памяти, то она линейна – $O(|D|)$ даже для данных большой размерности, так как для каждой точки достаточно хранить ее тип (ядро, граница, выброс) и метку ее кластера.

4.7 Кластеризация данных большой размерности. Алгоритм CLIQUE

Рассмотренные выше алгоритмы ориентированы, в основном, на кластеризацию данных в пространствах небольшой размерности. Алгоритм CLIQUE (CLustering In QUEst) был предложен в работе [2] как метод кластеризации данных больших размерностей. Общая схема этого алгоритма состоит в том, что пространство данных разбивается на блоки (параллелепипеды), из них выделяются блоки с высокой плотностью точек входной выборки и кластеры образуют связные компоненты из таких блоков.

Пусть $A = \{A_1, A_2, \ldots, A_d\}$ – множество ограниченных линейно упорядоченных областей и $\mathbf{S} = A_1 \times A_2 \times \ldots \times A_d$ – соответствующее d-мерное пространство. Будем называть $A_1, \ldots, A_d$ измерениями или атрибутами S. Входная выборка $D = \{x_1, \ldots, x_m\}$ состоит из d-мерных точек $x_i = (x_{i1}, \ldots, x_{id})$.

Пространство $\mathbf{S}$ разбивается на непересекающиеся блоки-параллелепипеды. Они получаются путем разбиения каждого измерения на ξ интервалов равной длины (ξ является входным параметром). Каждый d-мерный блок u имеет вид $\{u_1, \ldots, u_d\}$, где $u_i = [l_i, h_i)$ – полуоткрытый интервал из разбиения A_i. *Плотностью* блока называется доля точек выборки, попавших в этот блок. Блок называется *плотным* , если его плотность не меньше τ (это еще один входной параметр).

Аналогично определяются блоки для всех подпространств исходного d-мерного пространства. Блок в k-мерном подпространстве $\mathbf{S}$, определяемом произведением $A_{t1} \times A_{t2} \times \ldots \times A_{tk}$, где $k < d$ и $t_i < t_j$ $i < j$, является произведением соответствующих интервалов $u_{t1} \times u_{t2} \times \ldots \times u_{tk}$.

Кластером назовем максимальное множество связанных плотных блоков в k-мерном подпространстве. Два блока u_1 и u_2 *связаны*, если у них есть общая грань или если существует третий k-мерный блок u_3 такой, что u_1 связан с u_3 и u_2 связан с u_3. Блоки $u_1 = \{r_{t1}, \ldots, r_{tk}\}$ и $u_2 = \{r'_{t1}, \ldots, r'_{tk}\}$ *имеют общую грань*, если для для $(k-1)$-го измерения их интервалы совпадают: $r_{ti} = r'_{ti}$ при $i \in \{1, 2, \ldots, k\} \setminus \{j\}$, а для оставшегося измерения tj либо $h_{tj} = l'_{tj}$, либо $h'_{tj} = l_{tj}$.

Рассматриваемый вариант задачи кластеризации можно сформулировать следующим образом:
по заданной входной выборке D, состоящей из d-мерных точек, и параметрам ξ и τ найти и эффективно представить кластеры во всех подпространствах исходного пространства.

Алгоритм CLIQUE решает эту задачу в три этапа:
1. Выделение подпространств, содержащих кластеры, и плотных блоков в них.
2. Идентификация кластеров.
3. Получение коротких описаний кластеров.

Этап 1. Определение плотных блоков.

На этом этапе требуется рассмотреть 2^d различных подпространств и для каждого k-мерного подпространства исследовать на плотность ξ^k блоков. Чтобы уменьшить объем выполняемой работы, алгоритм использует выраженный в следующей лемме принцип монотонности, аналогичный принципу Apriori.

Лемма 4.1. *Если набор точек C образует кластер в некотором k-мерном подпространстве, то в каждой $(k-1)$-мерной прекции этого пространства C также входит в какой-нибудь кластер .*

Обозначим через F_k множество плотных k-мерных блоков. Алгоритм последовательно

Алгоритм DenseBlock(D, ξ, τ)

1. **for** $i = 1$ **to** d **do**
2. **for** $j = 1$ **to** ξ **do**
3. $count(i : j) := 0$
4. **endfor endfor**;
5. **foreach** $x = (x_1, \ldots, x_d) \in D$ **do** //первый проход
6. **for** $i = 1$ **to** d **do**
7. **for** $j = 1$ **to** ξ **do**
8. **if** $x_i \in u_i^j$ **then** $count(i : j) + +$
9. **endfor endfor endfor**;
10. $F_1 := \emptyset$;
11. **for** $i = 1$ **to** d **do**
12. **for** $j = 1$ **to** ξ **do**
13. **if** $count(i : j)/|D| > \tau$ **then** $F_1 := F_1 \cup \{B(i : j)\}$
14. **endfor endfor**;
15. $k := 2$;
16. **repeat** //основной цикл
17. $C_k :=$ candidateGen($\mathsf{F_{k-1}}, \mathsf{k-1}$); //кандидаты в плотные k-блоки
18. **foreach** $B(t1 : j1, \ldots, tk : jk) \in C_k$ **do** $count(t1 : j1, \ldots, tk : jk) := 0$ **endfor**;
19. **foreach** $x \in D$ **do**
20. **foreach** $B(t1 : j1, \ldots, tk : jk) \in C_k$ **do**
21. **if** $Prj(x)_{t1,\ldots,tk} \in B(t1 : j1, \ldots, tk : jk)$
22. **then** $count(t1 : j1, \ldots, tk : jk) + +$
23. **endfor endfor**;
24. $F_k := \{B(t1 : j1, \ldots, tk : jk) \in C_k \mid \frac{count(t1:j1,\ldots,tk:jk)}{|D|} > \tau\}$;
25. $k := k + 1$
26. **until** $(F_k = \emptyset)$ **OR** $(k = d + 1)$;
27. **return** $F_1, F_2, \ldots, F_{k-1}$

Рис. 4.6: Алгоритм порождения плотных блоков DenseBlock

Function candidateGen(F,k)

1. $C := \emptyset$;
2. **foreach** $B(t1 : j1, \ldots, t(k-1) : j(k-1)), B(r1 : q1, \ldots, r(k-1) : q(k-1)) \in F$ **do**
3. **if** $(t1 = r1)$ **AND** $(j1 = q1)$ **AND** $\ldots$ **AND** $(t(k-2) = r(k-2))$
AND $(j(k-2) = q(k-2))$ **AND** $(t(k-1) < r(k-1))$
4. **then**
5. $B := B(t1 : j1, \ldots, t(k-1) : j(k-1), r(k-1) : q(k-1))$; `// шаг объединения`
6. $flag := true$;
7. **foreach** B'-подблока B размера $(k-1)$ **do** `// шаг исключения`
8. **if** $B' \notin F$ **then** $flag := false$;
9. **endfor**;
10. **if** $flag = true$ **then** $C := C \cup B$ **endif**
11. **endif**
12. **endfor endfor**;
13. **return** C.

Рис. 4.7: Порождение кандидатов в k-плотные блоки.

строит множества $F_1, F_2, \ldots, F_{k-1}, F_k, \ldots$. При построении F_k вначале с помощью F_{k-1} формируется множество C_k кандидатов в плотные k-мерные блоки, а затем неподходящие кандидаты отсеиваются.

Пусть каждое измерение A_i $(i = 1, \ldots, d)$ разбито на ξ подинтервалов $u_i^1, \ldots, u_i^\xi$, где $u_i^j = [l_i^j, h_i^j)$. Обозначим через $B(t1 : j1, \ldots, tk : jk)$ блок в k-мерном подпространстве $\mathbf{S}' = A_{t1} \times A_{t2} \times \ldots \times A_{tk}$, определяемый произведением $u_{t1}^{j1} \times u_{t2}^{j2} \times \ldots \times u_{tk}^{jk}$. Через $Prj(x)_{t1,\ldots,tk}$ обозначим проекцию точки x на подпространство $\mathbf{S}'$. Пусть $count(t1 : j1, \ldots, tk : jk)$ это число точек D, проекции которых на $\mathbf{S}'$ попали в блок $B(t1 : j1, \ldots, tk : jk)$, т.е. $count(t1 : j1, \ldots, tk : jk) = |\{x \in D \mid Prj(x)_{t1,\ldots,tk} \in B(t1 : j1, \ldots, tk : jk)\}|$. Алгоритм порождения плотных блоков DenseBlock в подпространствах размерностей 1, 2, $\ldots, d$ представлен на рис. 4.6, а процедура candidateGen(F,k) подготовки множества C_k кандидатов в k-мерные плотные блоки представлена на рис. 4.7.

Оценка времени 1-го этапа зависит от максимальной размерности k подпространства, в котором еще имеются плотные блоки. Тогда они имеются и в 2^k его подпространствах. Для каждой размерности $i \in [1, k]$ алгоритм делает в основном цикле проход по D. Тогда при $|D| = m$ получаем оценку времени $O(c^k + km)$.

Полученная оценка экспоненциальна по k и поэтому алгоритм будет мало пригоден для пространств большой размерности d. Чтобы сделать его более эффективным авторы предлагают использовать эвристический принцип ОМД (Описания Минимальной Длины) (по английски: MDL – Minimal Description Length – principle). В применении к нашей ситуации он выглядит следующим образом. Пусть имеется набор подпространств $S_1, S_2, \ldots, S_n$, содержащих плотные блоки. Найдем для каждого из них S_j его *покрытие* x_j – число, равное количеству точек входной выборки D, попавших в плотные блоки из S_j. После этого подпространства с большим покрытием остаются для дальнейшей обработки, а с малым – удаляются.

Предположим, что подпространства $S_1, S_2, \ldots, S_n$ упорядочены по убыванию величины их покрытия. Для $i \in [1, n]$ определим среднее покрытие первых i подпространств $\mu_I(i) = \lceil (\sum_{1 \le j \le i} x_j)/i \rceil$ и последних $(n-i)$ подпространств – $\mu_P(i) = \lceil (\sum_{i+1 \le n \le i} x_j)/(n-i) \rceil$. Число

битов для записи этих чисел равно $\log_2(\mu_I(i))$ и $\log_2(\mu_P(i))$, соответственно. С помощью этих средних можно задать все покрытия x_j, указав их разности с соответствующими средними. Тогда длина такого представления равна

$$CL(i) = \log_2(\mu_I(i)) + \sum_{1 \le j \le i} \log_2(|x_j - \mu_I(i)|) + \log_2(\mu_P(i)) + \sum_{i+1 \le n \le i} \log_2(|x_j - \mu_P(i)|).$$

Далее выбирается i_{min}, минимизирующее величину $CL(i)$ и все подпространства $S_j, j = i_{min}, \ldots, n$ удаляются из рассмотрения.

Эта операция не требует много времени, поскольку величины x_j могут вычисляться при определении плотных блоков в S_j, а после сортировки вычисление средних и $CL(i)$ можно провести за два прохода по i от 1 до n. Вместе с тем ясно, что примененный выше принцип ОМД мало обоснован и, как отмечает сами авторы, удаление подпространств с небольшим покрытием в малых размерностях может привести к нежелательным потерям кластеров в больших размерностях.

Этап 2. Поиск кластеров.

Входом второго этапа служит множество DB, состоящее из плотных блоков некоторого k-мерного подпространства S. Выходом будет разбиение DB на кластеры $D^1, \ldots, D^q$ так, что любые два блока в D^i связаны, а любые два блока $B \in D^i$ и $B' \in D^j$ при $i \neq j$ не связаны.

Определим неориентированный граф G_{DB}, вершинами которого являются блоки из DB, а ребра соединяют блоки, имеющие общую грань. Тогда из определения кластера следует, что кластеры соответствуют связным компонентам графа G_{DB}. Обнаружить и перечислить все связные компоненты можно, используя стандартный обход графа G_{DB} в глубину.

Каждый блок $B \in DB$ может быть связан в графе G_{DB} не более, чем с $2k$ соседними блоками. Поэтому время работы для каждого k-мерного подпространства S можно оценить как $O(2k|DB|)$. Заметим, что общее число плотных блоков в DB не очень велико, так как каждый из них содержит достаточно много точек входной выборки.

Авторы работы [2] включили в алгоритм CLIQUE и 3-ий этап, на котором для каждого кластера, заданного перечнем входящих в него плотных блоков, должно строиться "минимальное" представление в виде объединения непересекающихся k-мерных параллелепипедов. В общем случае эта задача является NP-полной уже для 2-мерного пространства. Поэтому предлагается некоторый эвристический жадный алгоритм, который мы здесь рассматривать не будем.

4.8 Алгоритм максимизации правдоподобия EM (Expectation-Maximization)

Алгоритм EM относится к классу статистических алгоритмов, которые основаны на предположении о том, что кластеры неплохо описываются некоторым семейством вероятностных распределений. Тогда задача кластеризации сводится к разделению смеси распределений по конечной выборке.

Основное предположение о вероятностной природе данных:
точки входной выборки D появляются случайно и независимо согласно вероятностному

распределению, представляющему собой смесь распределений

$$p(x) = \sum_{j=1}^{k} w_j p_j(x), \quad \sum_{j=1}^{k} w_j = 1,$$

где $p_j(x)$ — функция плотности распределения кластера j, w_j — неизвестная априорная вероятность появления объектов из кластера j.

В качестве распределений $p_j(x)$ чаще всего берут сферические нормальные распределения (гауссовские плотности). Многомерное Гауссовское распределение для кластера $j, j = 1\ldots., k$, задается его параметрами: m-мерным средним значением μ_j и $m \times m$ матрицей ковариаций Σ_j. Вероятность точки $x \in D$ для распределения кластера j определяется формулой

$$p_j(x) = \frac{1}{\sqrt{(2\pi)^m |\Sigma_j|}} exp(\frac{1}{2}(x-\mu_j)^T (\Sigma_j)^{-1}(x-\mu_j)),$$

где x и μ_j – векторы-столбцы, степень ^{T} означает транспонирование столбца в строку, $|\Sigma|$ – определитель матрицы Σ, а $(\Sigma)^{-1}$ – обратная матрица.

Работа алгоритма EM состоит в итеративном повторении двух шагов:
E-шаг: для каждой точки x и кластера j вычисляется вероятность того, что x попадает в кластер j:

$$p(j|x) = \frac{w_j p_j(x)}{p(x)}.$$

M-шаг: уточняются параметры каждого кластера μ_j, Σ_j, при этом существенно используются определенные на E-шаге вероятности $p(j|x)$.

Аалгоритм EM представлен на рис.4.8. Его входом служит выборка D, состоящая из N m-мерных точек, и предполагаемое число кластеров k. Критерием остановки алгоритма служит стабилизация логарифма степени правдоподобия. Выбор кластера для точки производится по максимальной вероятности попадания точки в кластер.

4.9 Задачи

Задача 4.1. *Для заданных 2-х объектов, представленных векторами (22, 1, 42, 10) и (20, 0, 36, 8):*
(a) вычислить Евклидово расстояние между этими объектами;
(b) вычислить Манхетенское расстояние между этими объектами;
(c) вычислить расстояние Минковского между этими объектами при $\lambda = 3$;
(d) вычислить расстояние Чебышева между этими объектами

Задача 4.2. *Опишите или изобразите 2-мерные данные, для которых алгоритм k-средних не является подходящим алгоритмом кластеризации.*

Задача 4.3. *Докажите или опровергните утверждение о том, что после остановки алгоритма k-средних расстояние между двумя точками одного кластера всегда меньше, чем расстояние между точками разных кластеров.*

Задача 4.4. *Для заданных 4-х объектов, представленных векторами (22, 1), (42, 10), (18, 5) и (20, 8):*
(a) определить их стандартные z-значения;
(b) построить матрицу стандартизованных Евклидовых расстояний между ними.

Алгоритм EM(D,k);
begin
Инициализация:
1. **for** $j := 1$ **to** k **do**
2. $\{w_j^0 := 1/k$;
3. μ_j^0 :=*случайная точка из* D;
4. Σ_j^0 :=*начальная матрица ковариаций для j-го кластера* $\}$;

textitОсновной цикл:
5. $t := 0$;
6. **repeat**
Перевычисление вероятностей точек во всех кластерах:
7. **foreach** $x \in D$ **do**
8. **for** $j := 1$ **to** k **do**
9. $p^t(j|x) = \frac{w_j^t p_j^t(x)}{p^t(x)}$;

Обновление параметров смеси распределений:
10. **for** $j := 1$ **to** k **do**
11. $\{w_j^{t+1} := \frac{1}{N} \sum_{x \in D} p^t(j|x)$;
12. $p_j := \sum_{x \in D} p^t(j|x)$;
13. $\mu_j^{t+1} := \frac{1}{p_j} \sum_{x \in D} x \cdot p^t(j|x)$;
14. $\Sigma_j^{t+1} := \frac{1}{p_j} \sum_{x \in D} p^t(j|x)(x - \mu_j^{t+1})(x - \mu_j^{t+1})^T$ $\}$;
15. $E^{t+1} := \sum_{x \in D} \log(\sum_{j=1}^{k} w_j^{t+1} \cdot p_j^{t+1}(x))$;
16. $t := t + 1$
17. **until** $|E^t - E^{t+1}| \leq \varepsilon$
Определение результата:
18. **foreach** $x \in D$ **do**
19. $claster(x) := maxarg_j p(j|x)$
end

Рис. 4.8: Алгоритм кластеризации EM.

Задача 4.5. *Предположим, что нужно разбить следующие 8 точек на три кластера:* $A_1(2, 10), A_2(2, 5), A_3(8, 4), B_1(5, 8), B_2(7, 5), B_3(6, 4), C_1(1, 2), C_2(4, 9)$.

Пусть расстояние будет Евклидовым. Предположим, что начальными центрами кластеров являются точки A_1, B_1 *и* C_1 *(или выбираются методом* SelectCentroids*). Используя алгоритм k-средних определить:*
(a) центры трех кластеров после первой итерации алгоритма;
(b) окончательное разбиение точек на кластеры;
(c) получившееся значение суммы квадратов ошибок SSE.

Задача 4.6. *Предложите условия, при которых методы кластеризации, основанные на плотности (в частности, DBSCAN), работают лучше методов, основанных на разбиении, и иерархических методов. Подтвердите свои предложения примерами.*

Задача 4.7. *Приведите пример интеграции разных методов кластеризации. Например, случаи, когда один метод можно использовать как предварительный шаг перед другим. Объясните, почему комбинация разных методов может привести к улучшению качества и эффективности кластеризации.*

Задача 4.8. *Для заданных 6 объектов, представленных векторами* $a = (4, 5), b = (2, 4), c = (3, 3), d = (3, 2), e = (1, 4), f = (5, 3)$*:*
(a) построить дендрограмму, используя метод ближайших соседей;
(b) построить дендрограмму, используя метод далеких соседей соседей;
(c) построить дендрограмму, используя метод средней связи;
(d) построить дендрограмму, используя метод центроидной связи;
(e) построить дендрограмму, используя метод Уорда.

Расстояние между объектами считать Евклидоввым (Манхетенским).

Задача 4.9. *Данные представляют 10 точек на плоскости:*
$p_1 = (0.0, 1.0), p_2 = (1.0, 2.0), p_3 = (2.0, 0.0), p_4 = (4.0, 8.0), p_5 = (5.0, 7.0), p_6 = (5.0, 9.0), p_7 = (6.0, 7.0), p_8 = (8.0, 4.0), p_9 = (9.0, 3.0), p_{10} = (10.0, 5.0)$.
Пусть расстояние будет Евклидовым (Манхетенским). Предположим, что начальными центрами кластеров являются точки p_1, p_4 *и* p_5 *(или выбираются методом* SelectCentroids*).*
Используя алгоритм k-средних определить:
(a) центры трех кластеров после первой итерации алгоритма;
(b) окончательное разбиение точек на кластеры;
(c) получившееся значение суммы квадратов ошибок SSE.

Задача 4.10. *В следующей таблице представлены данные о составе молока десяти разных видов животных.*

Вид	Вода %	Белок %	Жир %	Лактоза %
Лошади	*90.1*	*2.6*	*1.0*	*6.9*
Мулы	*90.0*	*2.0*	*1.8*	*5.5*
Ослы	*90.3*	*1.7*	*1.4*	*6.2*
Зебры	*86.2*	*3.0*	*4.8*	*5.3*
Верблюды	*87.7*	*3.5*	*3.4*	*4.8*
Овцы	*82.0*	*5.6*	*6.4*	*4.7*
Олени	*64.8*	*10.7*	*20.3*	*2.5*
Киты	*64.8*	*11.1*	*21.2*	*1.6*
Кролики	*71.3*	*12.3*	*13.1*	*1.9*
Свиньи	*82.8*	*7.1*	*5.1*	*3.7*

а) Пусть расстояние будет Манхетенским. Предположим, что начальные центры кластеров выбираются методом SelectCentroids*. Используя алгоритм k-средних определить для k= 2 и k=3 разбиение видов животных на кластеры. Для каждого из k определить суммарную ошибку SSE.*

б) Пусть расстояние будет Манхетенским и расстояние между кластерами определяется методом ближайших соседей. Используя агломеративный алгоритм иерархической кластеризации, построить дендрограмму. Определить по ней разбиение видов на 2 и на 3 кластера. Для каждого из этих случаев определить суммарную ошибку SSE.

в) Сравните результаты, полученные в пп. (а) и (б). Какой метод кластеризации оказался предпочтильнее?

Задача 4.11. *Проведите разбиение на кластеры в пп. (а) и (б) предыдущей задачи, считая расстояние Евклидовым. Насколько при этом изменились кластеры?*

Задача 4.12. *В следующей таблице представлены данные о рейтингах (качестве) разных вин по годам их производства. Рейтинги имеют следующие значения: D – ужасное, P – плохое, A – среднее, G – хорошее, E – отличное.*

Название	61	62	63	64	65	66
"Medoc and Graves"	E	G	P	G	D	G
"Saint Emilion and Pomerol"	E	A	P	G	P	G
"Sauternes"	G	G	D	D	D	A
"Graves"	G	G	D	G	D	G
"Red Burgundy"	E	G	A	G	P	G
"Cote de Beaune"	E	G	A	G	A	G
"Chablis"	E	G	A	G	P	G
"Beaujolais"	E	G	P	G	D	G
"Red Rhone North"	E	G	P	G	A	G
"Red Rhone South"	G	A	P	G	A	A
"Alsace"	G	A	P	E	P	G
"Rhine"	G	A	P	G	P	G
"Moselle"	G	A	P	E	P	G

Определите способ числового кодирования рейтингов и способ определения расстояния между объектами. Пусть расстояние между кластерами определяется методом далеких соседей. Используя агломеративный алгоритм иерархической кластеризации, построить дендрограмму. Определить по ней разбиение вин на 2 и на 3 кластера. Для каждого из этих случаев определить суммарную ошибку SSE.

Задача 4.13. *В следующей таблице представлены данные о частоте некоторых типов ремонтов различных моделей автомобилей.*

"BR "FU "EL "EX "ST "EM"и "RS"означает ремонт "тормозной системы "топливной системы "электричества "выхлопной трубы "рулевого управления "механики мотора"и "дребезжание и скрип соответственно.

Значение + означает, что число ремонтов превышает среднюю величину, а − означает, число ремонтов меньше, чем в среднем.

Модель	"BR"	"FU"	"EL"	"EX"	"ST"	"EM"	"RS"
"AMC Ambassador 8"	+	-	-	-	-	-	-
"Buick 8 Full"	-	-	-	+	-	+	+
"Buick Riviera"	-	-	+	+	-	-	-
"Chevy II"	-	+	-	-	+	-	+
"Chevelle 8"	-	+	-	+	+	-	+
"Chevrolet Full"	-	+	+	+	+	-	+
"Corvair 6"	-	+	-	-	+	+	-
"Corvette"	-	-	-	+	-	-	+
"Chrysler Newport"	+	-	-	-	-	-	-
"New Yorker"	+	-	-	-	-	-	-
"Dodge Full Size"	+	-	-	-	-	-	+
"Falcon 6"	-	-	-	-	-	-	+
"Fairlane 8"	-	-	-	+	-	-	+
"Ford Full Size"	-	-	-	+	+	-	-
"Thunderbird"	-	-	+	-	+	+	-
"Olds Full"	+	+	-	-	-	-	+
"Plymouth Full"	+	-	-	-	-	-	-
"Pontiac Full"	+	+	+	-	-	-	+
"Mercedes"	-	-	-	-	-	-	-
"MG 1100"	-	-	+	+	-	-	-
"Peugeot"	-	-	-	-	-	-	-
"Porsche"	-	-	-	-	-	-	-
"Renault"	-	-	-	-	-	-	-
"Volvo"	-	-	-	+	-	-	-
"VW bug"	+	-	+	+	+	+	-
"VW bus"	-	-	+	-	-	+	-

Пусть расстояние между векторами определяется по Жаккарду. Предположим, что начальные центры кластеров выбираются методом SelectCentroids. *Используя алгоритм k-средних определить для k= 2 и k=3 разбиение моделей автомобилей на кластеры. Для каждого из k определить суммарную ошибку SSE.*

Задача 4.14. *Для заданного на плоскости набора данных* $D = A \cup B \cup C \cup F$, *где*
$A = \{(1 + i * 0.5, 10), (1 + i * 0.5, 9) |\ i = 0, 1, \ldots, 18\}$,
$B = \{(5, 1 + i * 0.5), (6, 1 + i * 0.5) |\ i = 1, 2 \ldots, 15\}$,
$C = \{(3, i * 0.5), (3.5, i * 0.5) |\ i = 1, 2 \ldots, 15\}$,
$F = \{(0, 0), (0, 9), (4, 0), (6, 7)\}$
выбрать значения параметров ε *и* $MinPts$, *при которых алгоритм кластеризации DBSCAN обнаружит два кластера. Какие это кластеры и какие точки при этом окажутся граничными и выбросами?*

Глава 5

Совместная фильтрация и рекомендующие системы

5.1 Основные задачи совместной фильтрации

В бизнес-словарях термин *совместная фильтрация (Collaborative Filtering)* объясняется как маркетинговая стратегия компаний, собирающихся работать на одном целевом рынке, которая заключается в том, что компании делятся друг с другом собранной информацией о потребителях целевого рынка.

В информатике им называют сравнительно новый раздел, занимающийся алгоритмами и системами выдачи рекомендаций (см. [1, 25]). В частности, такие системы получили большое распространение в интернет-магазинах – всякий раз при выборе того или иного товара покупателем рекомендующая система пытается предложить ему некоторые другие товары, которые по ее мнению могут его заинтересовать.

Задача выдачи рекомендаций. Один из вариантов этой задачи состоит в том, чтобы по заданным *множеству пользователей* , *множеству продуктов (items)* и набору известных для каждого пользователя *предпочтений* некоторых продуктов найти для каждого пользователя *новые продукты, которые будут для него наиболее предпочтительны.*

Пользователи. Пусть $C = \{c_1, \ldots, c_M\}$ – множество **пользователей**.

Продукты. Пусть $S = \{s_1, \ldots, s_n\}$– множество **продуктов**.

Полезность. Оценки продуктов пользователми задаются с помощью частичной функции $u : C \times S \longrightarrow \mathcal{R}$.

$u(c, s)$ – это: *полезность*, *рейтинг* или *предпочтительность* продукта s для пользователя c.

Функцию полезности $u(c, s)$ можно представлять в виде **матрицы полезности** $u[.,.]$, где $u[i, j] = u(c_i, s_j)$. В типичных случаях матрица $u[]$ является **разреженной**.

Задача. Главные задачи, решаемые методами совместной фильтрации (рекомендующих систем) можно сформулировать разными способами.

- **Рекомендации пользователю (User-based recommendations).** Для данного пользователя c найти такие продукты $s'_1, \ldots s'_k$ для которых рейтинги $u(c, s'_i)$ не определены и которые являются для c наиболее предпочтительными.
- **Индивидуальные рейтинги.** Для данного пользователя c и продукта s предсказать $u(c, s)$.
- **Рекомендации по продуктам (Item-based recommendations).** Для данного продукта s найти пользователей $c'_1, \ldots, c'_k$, для которых рейтинги $u(c, s'_i)$ не определены, но по предсказанию будут наибольшими.

5.2 Совместная фильтрация в рекомендующих системах

5.2.1 Рекомендующие системы

Рекомендующая система – это система, которая по заданным C, S и частичной функции полезности u решает одну или несколько задач выдачи рекомендаций.

Системы, рекомендующие на основе содержания: рекомендуют продукты *похожие* на те, что нравились пользователю в прошлом.

Совместные рекомендующие системы: рекомендуют те продукты, которые *другие пользователи с похожими предпочтениями* посчитали наиболее полезными.

Гибридные рекомендующие системы: соединяют рекомендации на основе содержания с совместными рекомендациями.

Системы, рекомендующие на основе содержания. Системы, рекомендующие на основе содержания, используют методы, аналогичные тем, что применяются в системах извлечения информации Information Retrieval. Эти методы будут рассмотрены в следующем разделе.

5.2.2 Методы совместной фильтрации

Подходы. Существуют различные подходы к классификации методов совместной фильтрации. В [25] выделяются три подхода:

1. **Методы, основанные на памяти (memory-based).** Эти методы используют различные *эвристики* для построения предсказаний полезности по известной матрице полезности $u[]$.
2. **Методы, основанные на модели (model-based).** Эти методы используют функцию полезности u для того, чтобы *обучить некоторую модель* специального вида. Затем на основе этой модели генерируются предсказания. В качестве таких моделей рассматриваются Баейсовские сети, кластеризация, фактор-анализ и др.
3. **Гибридные методы (hybrid recomenders).** Эти методы объединяют подходы, основанные на памяти, с методами, использующими модели. К ним относятся, в частности, и методы предсказания, основанные на содержании предметной области.

5.2.3 Методы, основанные на памяти

Агрегация полезностей. Методы совместной фильтрации, основанные на памяти, **агрегируют** известные полезности $u(c_i, s)$ продукта s, чтобы предсказать полезность (рейтинг) s для пользователя c (агрегация по пользователям):

$$u(c, s) = \mathsf{aggregate}_{c_i \in C} u(c_i, s).$$

Аналогичную агрегацию можно определить и по продуктам:

$$u(c, s) = \mathsf{aggregate}_{s_i \in S} u(c, s_i).$$

Обозначения. Для продукта $s \in S$ через C^s обозначим множество

$$C^s = \{c \in C \mid \textit{рейтинг}\ \ u(s, c) \quad \textit{определен}\},$$

т.е. множество всех пользователей, для которых уже известен рейтинг (полезность) продукта s.

Аналогично, для пользователя $c \in C$ можно определить множество

$$S^c = \{s \in S \mid \textit{рейтинг}\ \ u(s, c) \quad \textit{определен}\}$$

продуктов, рейтинги (полезности) которых для него известны.

Пусть $c \in C$ и $S = \{s_1, \ldots, s_n\}$. Через $u[c]$ обозначим (редкий) вектор:

$$u[c] = (u(c, s_1), u(c, s_2), \ldots, u(c, s_n)).$$

Замечание. В приводимых ниже вычислениях в качестве значения величины $u(c, s)$, отсутствующего явно в наборе исходных данных, как правило, будем использовать 0.

Средняя полезность. Наиболее простой метод предсказания совместной полезности – это среднее значение:

$$u(c, s) = \frac{1}{|C^s|} \sum_{c_i \in C^s} u(c_i, s).$$

Конечно, это очень упрощенное предсказание, так как не учитываются известные предпочтения пользователя c.

Взвешенная сумма. Это один из самых расспространенных методов предсказания. Он предполагает существование некоторой **функции сходства** $sim(.,.)$, которая задает степень близости между векторами полезности двух пользователей.

$$u(c, s) = k \cdot \sum_{c' \neq c} sim(u[c], u[c']) \cdot u(c', s),$$

где k – *нормализующий множитель*. Часто полагают

$$k = \frac{1}{\sum_{c' \neq c} |sim(u[c], u[c'])|}$$

и тогда

$$u(c, s) = \frac{1}{\sum_{c' \neq c} |sim(u[c], u[c'])|} \cdot \sum_{c' \neq c} sim(u[c], u[c']) \cdot u(c', s),$$

Предсказание на основе *взвешенной суммы* обладает одним существенным недостатком:

- оно *нечувствительно* к тому, что различные пользователи при определении своих предпочтений *по разному* используют шкалу предпочтений/рейтингов.

Скорректированная взвешенная сумма. При предсказании полезностей для конкретного пользователя учитывается его подход к назначению рейтингов продуктам. Вначале для пользователя cопределяется его средний рейтинг $\hat{u_c}$:

$$\bar{u}_c = \frac{1}{|S^c|} \sum_{s' \in S^c} u(c, s').$$

Затем значение $u(c, s)$ для продукта $s \in S$ предсказывается следующим образом:

$$u(c, s) = \bar{u}_c + k \cdot \sum_{c' \neq c} sim(u[c], u[c']) \cdot (u(c', s) - \bar{u}_{c'}).$$

Здесь, k тот же нормализующий множитель, что и раньше.

Аналогично, взвешенная сумму и скорректированную взвешенную сумму можно использовать для предсказаний на основе близости продуктов.

5.2.4 Предсказания на основе N ближайших соседей

Все рассмотренные выше методы предсказания можно модифицировать так, чтобы при вычислении результата предсказания рассматривать **только N ближайших соседей пользователя** c.

Пусть $C'_c = \{c' \in C | rank(sim(u[c], u[c'])) \leq N\}$ – это множество N ближайших соседей пользователя c относительно функции сходства $sim(.,.)$.

Ранжирование по среднему N ближайших соседей.

$$u(s, c) = \frac{1}{N} \sum_{c' \in C'_c} u(c', s).$$

Взвешенная сумма по N ближайшим соседям.

$$u(c, s) = k \cdot \sum_{c' \in C'_c} sim(u[c], u[c']) \cdot u(c', s).$$

$$k = \frac{1}{\sum_{c' C'_c} |sim(c, c')|}.$$

Скорректированная взвешенная сумма по N ближайшим соседям.

$$u(c, s) = \bar{u}_c + k \cdot \sum_{c' \in C} sim(u[c], u[c']) \cdot (u(c', s) - \bar{u}_{c'}).$$

Метод предсказаний по N ближайшим соседям можно использовать также для предсказаний на основе близости продуктов, определив N продуктов, наиболее близких к заданному.

5.2.5 Меры сходства пользователей и продуктов

При совместной фильтрации обычно применяются следующие две меры сходства.

Корреляция Пирсона (Pearson Correlation). Пусть для пользователей c и c' множество $I = S^c \cap S^{c'}$ включает продукты, которые были оценены обоими пользователями. Близость c и c' определяется величиной корреляции Пирсона:

$$sim(u[c], u[c']) = \frac{\sum_{s \in I}(u(c,s) - \bar{u}_c) \cdot (u(c',s) - \bar{u}_{c'})}{\sqrt{\sum_{s \in I}(u(c,s) - \bar{u}_c)^2 \cdot \sum_{s \in I}(u(c',s) - \bar{u}_{c'})^2}}.$$

Эта мера отражает *статистическую корреляцию* между двумя (редкими) векторами данных.

Аналогично, с помощью корреляции Пирсона можно оценивать и близость двух продуктов.

Сходство направлений (косинус угла). Сходство между двумя документами часто измеряют как косинус угла между их векторными представлениями, координатами которых являются частоты вхождений (ключевых) слов. Такой способ мы будем рассматривать в следующем разделе. Этот же прием можно использовать в совместной фильтрации для оценки близости векторов, представлящих рейтинги двух пользователей или рейтинги двух продуктов. Близость пользователей c и $c\prime$ можно определить как

$$sim(u[c], u[c']) = cos(u[c], u[c']) = \frac{u[c] \cdot u[c']}{||u[c]|| \cdot ||u[c']||} =$$

$$\frac{\sum_{i=1}^{n} u(c,s_i) \cdot u(c',s_i)}{\sqrt{\sum_{i=1}^{n} u(c,s_i)^2 \cdot \sum_{i=1}^{n} u(c',s_i)^2}}.$$

Косинус определяет *степень параллельности* двух векторов (он равен 1, если векторы параллельны, и 0, если они ортогональны).

Заметим, что здесь суммирование формально ведется по всем продуктам, а не по множеству $I = S^c \cap S^{c'}$, так как нулевые значения отсутствующих рейтингов не влияют на подсчитываемые суммы.

Голосование по умолчанию. Голосование по умолчанию уточняет меру сходства через корреляцию Пирсона путем подстановки некотрого значения "по умолчанию"для рейтинга продуктов, которые не были явно оцегнены пользователем.

$$sim(u[c], u[c']) =$$

$$\frac{(n+k)(\sum_j u(c,s_j)u(c',s_j) + kd^2) - (\sum_j u(c,s_i) + kd)(\sum_j u(c',s_j) + kd)}{\sqrt{((n+k)(\sum_j u^2(c,s_j) + kd^2) - (\sum_j u(c,s_j) + kd)^2)((n+k)(\sum_j u^2(c',s_j) + kd^2) - (\sum_j u(c',s_j) + kd)^2)}}$$

Здесь в дополнение ко всем продуктам оцененным обоими пользователями c и c' вес d приписан k продуктам, которым ни один из них не приписал рейтинг. *Обычно для d выбирается* ***нейтральное*** *или* ***чуть отрицательное*** *значение.*

Обратная частота числа пользователей. *Обратная частота числа пользователей* f_j для продукта s_j определяется как

$$f_j = \log_2 \frac{|C|}{|C^{s_j}|}$$

Чем чаще оценивается данный продукт, тем меньше его его обратная частота. Идея этого подхода заключается в том, что редко оцениваемые продукты должны играть более важную роль при определении сходства пользователей.

Трансформированный голос (рейтинг) $v(c, s_j)$ определяется как:

$$v(c, s_j) = f_j u(c, s_j).$$

И в мере сходства, использующей косинус, и в корреляции Пирсона можно использовать трансформированные голоса вместо исходных рейтингов. При этом формула для *сходства направлений* выглядит так:

$$sim(u[c], u[c']) = \frac{v[c] \cdot v[c']}{||v[c]|| \cdot ||v[c']||} = \frac{\sum_{i=1}^{n} v(c, s_i) \cdot v(c', s_i)}{\sqrt{\sum_{i=1}^{n} u(v, s_i)^2 \cdot \sum_{i=1}^{n} u(v', s_i)^2}}.$$

А вот новая формула для корреляции Пирсона :

$$sim(u[c], u[c']) = \frac{\sum_j f_j \sum_j f_j u(c, s_j) u(c', s_j) - (\sum_j f_j u(c, s_j))(\sum_j f_j u(c', s_j))}{\sqrt{UV}}$$

где

$$U = \sum_j f_j \left(\sum_j f_j u^2(c, s_j) - \left(\sum_j f_j u(c, s_j) \right)^2 \right);$$

$$V = \sum_j f_j \left(\sum_j f_j u^2(c', s_j) - \left(\sum_j f_j u(c', s_j) \right)^2 \right).$$

5.2.6 Усиление рейтингов (Case Amplification)

Усиление рейтингов это метод модификации рейтингов $u(c, s)$ *до того*, как они будут использованы в алгоритмах совместной фильтрации. Обычно усиление применяется к значениям $u(c, s) \in [-1, 1]$ и увеличивает ("вознаграждает") веса близкие по модулю к 1 и уменьшает ("наказывает") малые по модулю рейтинги. Вот одна из типичных схем усиления рейтингов:

$u^{amp}(c, s) = u(c, s) * |u^{\rho-1}(c, s)|$.

Здесь ρ – это *степень усиления*, $\rho \geq 1$, а типичным выбором ρ является 2.5. Усиление рейтингов позволяет уменьшить шум в данных. Большие рейтинги остаются большими, например, если $u(c, s) = 0.9$, то после усиления $u^{amp}(c, s) = 0.9^{2.5} \approx 0.8$. А небольшие рейтинги становятся пренебрежимо малыми, например, для $u(c, s) = 0.1$ $u^{amp}(c, s) = 0.1^{2.5} \approx 0.003$. Полученные после усиления рейтинги $u^{amp}(c, s)$ далее используются при вычислении предсказаний.

5.2.7 Метрики для оценки качества рекомендаций

Качество системы предсказания можно оценить по полученным ею результатам с учетом вида решаемой задачи. Для этого выделяется тестовая выборка, в которой скрываются некоторые известные рейтинги и производится их предсказание. Затем полученные результаты сравниваются с настоящими значениями рейтингов.

Рассмотрим основные метрики, используемые при оценке таких систем.

Средняя абсолютная ошибка (MAE) и нормализованная средняя абсолютная ошибка (MAE)

Средняя абсолютная ошибка (MAE - Mean Absolute Error) вычисляется как среднее абсолютных разностей между предсказаниями и истинными рейтингами.

$$MAE = \frac{\sum_{c,s} |u(c,s) - r(c,s)|}{N},$$

где в числителе суммирование ведется по всем парам (c, s), для которых был предсказан рейтинг $u(c, s)$, N – это общее количество таких пар, а $r(c, s)$ – соответствующий настоящий рейтинг. Чем меньше MAE, тем лучше предсказание системы.

Различные рекомендующие системы могут использовать различные шкалы рейтингов. Поэтому имеет смысл рассматривать *нормализованную среднюю абсолютную ошибку (NMAE – Normalized Mean Absolute Error)*, которая выражает ошибку в процентах полной шкалы

$$NMAE = \frac{MAE}{r_{max} - r_{min}},$$

где r_{max} и r_{min} это верхняя и нижняя границы рейтингов.

Корень из средней квадратичной ошибки (RMSE)

Корень из средней квадратичной ошибки (RMSE) определяется как

$$RMSE = \sqrt{\frac{1}{N} \sum_{c,s} (u(c,s) - r(c,s))^2},$$

где, как и выше, суммирование ведется по всем парам (c, s), для которых был предсказан рейтинг $u(c, s)$, N – это общее количество таких пар, а $r(c, s)$ – соответствующий настоящий рейтинг. Метрика RMSE усиливает вклады больших абсолютных ошибок отдельных предсказаний в общий результат.

Чувствительность операционной характеристики получателя (ROC Sensitivity)

Операционная характеристика получателя (ROC – Receiver Operating Characteristic) это двумерное описание качества классификатора для бинарной классификации, основанное на матрице отличия (confusion matrix), аналогичной матрице отличия для бинарных атрибутов:

Настоящие	**Предсказанные**	
	Положительные	Отрицательные
Положительные	*TP*	*FN*
Отрицательные	*FP*	*TN*

Для определения этой матрицы выбирается некоторое *пороговое значение* рейтинга r_0 и все рейтинги больше r_0 считаются положительными, а меньше – отрицательными. Тогда TP – это число правильных положительных предсказаний, FP – это число неверных положительных предсказаний, FN – это число неверных отрицательных предсказаний и TN – это число верных отрицательных предсказаний. Качество предсказаний оценивается парой ROC =(TPR, FPR), где TPR – это доля правильных положительных рейтингов: $TPR = TP/P = TP/(TP + FN)$, а FPR – это доля неверных положительных рейтингов: $FPR = FP/N = FP/(FP + TN)$. Обычно значение TPR откладывается по оси Y, а значение FPR – по оси X. Из определений следует, что пара (TPR, FPR) лучше пары (TPR', FPR'), если $TPR > TPR'$, а $FPR < FPR'$. При вариации порога r_0 получается ROC-кривая, характеризующая качество предсказания.

ROC-кривые для двух систем могут пересекаться. В этом случае мера ROC не позволяет оценить их относительное качество. Однако, если ROC-кривая для одной системы оказывается существенно лучше ROC-кривой для другой, то это безусловно свидетельствует о лучшем качестве предсказаний первой системы.

5.3 Большой приз фирмы Нетфликс (Netflix Prize)

Внимание исследователей к задачам предсказания на основе совместной фильтрации было сильно подогрето призом в миллион долларов, объявленным фирмой Нетфлих в 2006г. Эта фирма занимается рекомендацией новых фильмов пользователям. За период с декабря 1999г. по декабрь 2005г. ею было получено более 100 миллионов рейтингов 17770 фильмов от 480189 зрителей, которые и составили основу обучающего множества. Оценки давались по пятибальной шкале. Каждый рейтинг сопровождался датой его получения, а каждый фильм – названием и годом выпуска. Целью конкурса было улучшение точности предсказания на 10% по сравнению с точностью системы Leaderboard, разработанной в самой фирме Нетфликс. Для этого требовалось добиться оценки ошибки по метрике RMSE, не превышающей 0.8563.

Из множества всех рейтингов было выбрано и скрыто около 4.2 миллиона оценок, включающих последние 9 оценок каждого пользователя. Они образовали скрытое множество Hold-out. Оставшиеся данные вошли в обучающее множество. Затем Hold-out было случайным образом разбито на 3 части: Probe, Quiz, и Test. Множество Probe было присоединено к обучающему множеству и его рейтинги были восстановлены. Множества Quiz и Test использовались для тестирования предложенных систем. После проверки очередного решения его результат (ошибка по мере RMSE) на множестве Quiz открыто объявлялся на сайте Нетфикс. Таким образом, участники соревнования могли судить об успехах друг друга и пытаться улучшить свои результаты, посылая новые варианты предсказывающих систем. Победителем становилась команда, которая имела лучший результат на множестве Test, но результаты на этом множестве не оглашались.

На участие в соревновании зарегистрировалось 41305 команд из 186 различных стран. Было получено 44014 решений от 5169 различных команд. В сентябре 2009 года победителем была объявлена команда *BellKor's Pragmatic Chaos*, в которую объединились три вначале независимые команды *BellKor, Pragmatic Theory* и *BigChaos*. Их система сумела добиться ошибки RMSE = 0.8556. Каждая из команд опубликовала статью с описанием своей части победившей системы. Она является сложной комбинацией многих методов совместной фильтрации. Интересно, что важную роль в получении точных предсказаний сыграл, кроме стандартных мер частоты, ряд неочевидных параметров, связанных со временем получения рейтингов:

- время между данным рейтингом и первым рейтингом этого пользователя;
- время между данным рейтингом и средней датой рейтингов этого пользователя;
- время между данным рейтингом и медианой дат рейтингов этого пользователя;
- время между данным рейтингом и первым рейтингом данного фильма;
- время между данным рейтингом и средней датой рейтингов данного фильма;
- время между данным рейтингом и медианой дат рейтингов данного фильма;
- абсолютная дата рейтинга.

5.4 Задачи

Задача 5.1. *В следующей таблице представлены данные оценки 5 продуктов* $p1, p2, p3, p4, p5$ *пятью пользователями* $u1, u2, u3, u4, u5$ *(? означает отсутствие оценки).*

	p1	*p2*	*p3*	*p4*	*p5*
u1	*3*	*2*	*4*	*?*	*4*
u2	*5*	*?*	*3*	*4*	*3*
u3	*4*	*4*	*5*	*3*	*5*
u4	*?*	*2*	*3*	*3*	*4*
u5	*1*	*4*	*?*	*?*	*3*

Определите предлолагаемую оценку $u(u5, p3)$ *продукта* $p3$ *пользователем* $u5$*, используя формулу для скорректированной взвешенной суммы и косинус (корреляцию Пирсона) в качестве меры сходства.*

Задача 5.2. *В условиях предыдущей задачи определите оценку* $u(u5, p3)$*, используя скорректированную взвешенную сумму для предсказания на основе близости продуктов. В качестве меры сходства разных продуктов используйте косинус угла между векторами (корреляцию Пирсона).*

Глава 6

Информационный поиск (Information Retrieval)

6.1 Определения

Информационный поиск (ИП (IR) – это процесс нахождения в заданной *совокупности документов* тех *документов*, которые являются *релевантными* для *запросу пользователя.*

Документ. ИП работает с **коллекциями (наборами, совокупностями) документов.** Каждый **документ**, как правило, является **текстом** или может рассматриваться как текст.

Коллекции документов. Основное предположение ИП состоит в том, что **коллекции документов** являются большими.

Замечание: это не всегда так. Иногда системы ИП применяются для работы с наборами, включающими десятки документов, а не сотни/тысячи/миллионы, как это бывает в типичных системах ИП.

Запросы. Пользовательский запрос – это формальное представление **информационных потребностей**. Вообще говоря, это не одно и то же.

Информационная потребность – это ясно выраженное желание получить некоторую информацию (найти документы, которые содержат некоторую информацию). **Запрос** же является представлением информационной потребности, которое может быть передано на обработку специфической системе ИП.

Виды запросов. Различные сичстемы ИП используют различные виды запросов:

запросы по ключевым словам: пользователь выражает свою потребность в информации в виде одного или нескольких **ключевых слов** (**терминов**). Система ИП ищет документы, содержащие одно или несколько слов, заданныъх пользователем.

Булевские запросы: пользователь выражает потребность в информации в виде логического выражения, связывающего ключевые слова. Система ИП ищет документы, которые удовлетворяют этому выражению.

запросы-фразы: пользователь выражает свою потребность в информации в виде последовательности слов (терминов), образующих некоторую фразу. Система ИП ищет документы, которые содержат хотя бы одно вхождение этой фразы.

запросы с ограничениями на близость (proximity queries): пользователь выражает свою потребность в информации в виде последовательности слов (терминов). Система ИП ищет документы, которые содержат все слова из запроса, расположенные в тексте документа достаточно близко друг к другу.

запросы полных документов: пользователь выражает свою потребность в информации в виде некоторого документа. Система ИП возвращает список документов, которые наиболее близки к запрашиваемому документу.

запросы на естественном языке: пользователь выражает свою потребность в информации в виде вопроса (текста) на естественном языке. Система ИП ищет документы коллекции, которые дают наилучший ответ на заданный вопрос (и заодно комбинирует полученную информацию, чтобы сформировать связный ответ на вопрос).

Ранжирование. Системы ИП обычно ранжируют ответы на запросы пользователей. Ранжирование происходит в соответствии с полученной системой оценкой релевантности каждого из документов введенному запросу.

Релевантность — это числовая мера, определяющая насколько данный документ подходит для ответа на запрос. Обычно релевантность принимает значения от 0 (полностью не подходит, совсем не релевантен) до 1 (полностью подходит, абсолютно релевантен). Разные модели ИП интерпретируют релевантность по-разному: в одних моделях это вероятность того, что пользователь посчитает данный документ подходящим (релевантным), в других — это степень сходства между данным документом и запросом)или между некоторыми их представлениями).

6.2 Архитектура систем ИП

На рис. 6.1 приведена архитектура типичной системы ИП.

Она включает два компонента.

Система индексации является статичным компонентом системы ИП. Этот компонент отвечает за предварительную обработку набора документов и создание *внутреннего представления* документов, которое может использоваться *механизмом поиска* для эффективного поиска документов, отвечающих заданному запросу.

Механизм поиска — это динамичный, интерактивный компонент системы ИП. Он получает запросы от пользователей, обрабатывает их, используя внутреннее представление набора документов, построенное *системой индексации*, и возвращает полученные результаты пользователям. Этот компонент может также содержать *механизм обратной связи с пользователями*, который позволяет улучшать качество выдаваемых системой ответов.

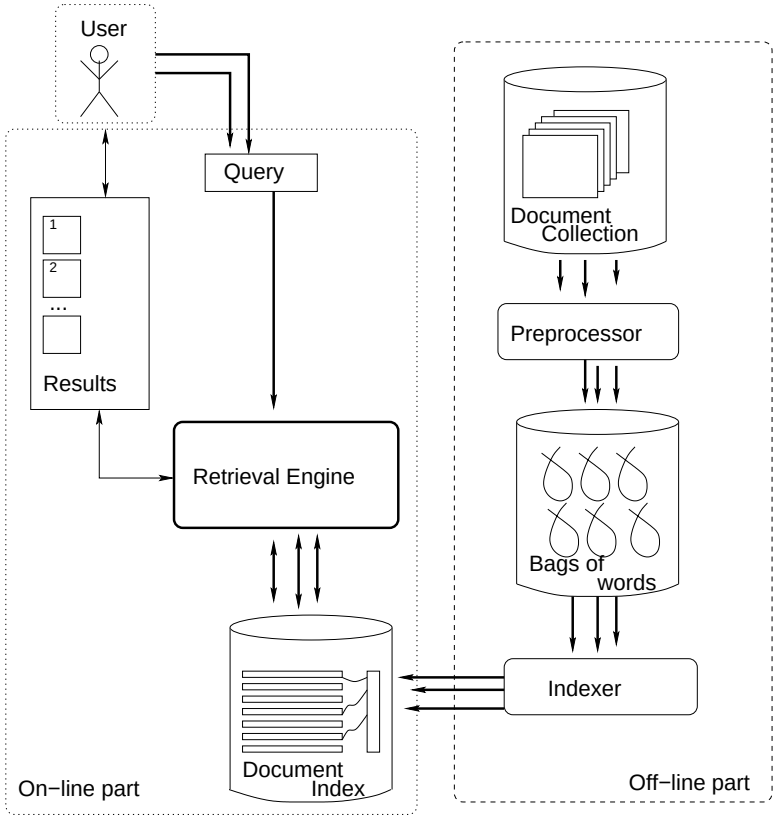

Рис. 6.1: Архитектура типичной системы информационного поиска.

В целом, процесс получения ответа проходит три этапа:

1. **Препроцессирование.** Препроцессирование включает определенный набор процедур, обычно, не зависящих от модели, которые извлекают из входных документов представления их текстов, которые далее могут использоваться **основанными на определенных моделях программами индексации** для построения **индекса коллекции документов**. Типичные шаги препроцессирования:

   (a) **Анализ и разметка** входа. На этом шаге идентифицируются отдельные ключевые слова/термины, предложения, фразы и другие конструкции. Выделяются и отфильтровываются также части входа не используемые в процессе индексирования (например, теги HTML).

   (b) **Устранение стоп-слов.** Некоторые слова встречаются очень часто и ими в качестве ключевых слов можно пренебречь. Такие слова выявляются и удаляются из представлений документов.

   (c) **Морфологический поиск и выделение основ слов (Stemming).** Это позволяет поисковой машине вести поиск слова не только в строго заданном виде, но и во всех его морфологических формах Каждое ключевое слово заменяется на его **основу** - подстроку, представляющую *неизменяемую часть ключевого слова*. Это позволяет, например, рассматривать такие слова как "компьютер", "комп", "компьютеризация", "компьютерный" как одно и то же ключевое слово. [1]

[1] Это не всегда желательно. Например, Google, не заменяет ключевые слова их основами. Однако

2. **Индексация.** Индексация создает **индекс коллекции документов**: некоторый набор структур данных, который представляет коллекцию документов в виде, (a) отражающем природу данной коллекции и (b) позволяющем осуществлять эффективный поиск ответов на запросы.

   Индексация зависит от выбранной модели: различные методы ИП используют разные подходы к построению индексов.

3. **Механизм поиска.** Механизм поиска ответа на запрос играет три роли (третья не обязательна, но как правило, обнаруживается):

   (a) **Организация приема запросов пользователя.** Пользовательские запросы принимаются и в зависимости от их типа анализируются, разбираются и индексируются.

   (b) **Обработка запроса.** Разобранные и проиндексированные запросы сравниваются с использованием *алгоритмов поиска* с индексами коллекции. Результаты *возвращаются*, *ранжируются* и передаются пользователю.

   (c) **Обработка обратной связи** (*необязательная*). Пользователь указывает релевантность полученных им документов, его оценки принимаются, анализируются и используются для уточнения списка возвращенных документов.

6.3 Модели информационного поиска

Модель ИП это способ представлять документы и запросы и вычислять релевантность первых для вторых.

Общие свойства

Большинство моделей ИП обладают следующими чертами.

Коллекция документов. Входными данными для любой модели ИП являются коллекции $D = \{d_1, \dots d_n\}$ документов.

Словарь (корпус). Совокупность терминов, не являющихся стоп-словами, входящих в документы D, называется **словарем** или **корпусом** D. Для данной D мы обозначим словарь этой коллекции через

$$V_D = \{t_1, \dots t_M\}.$$

(Если коллекция D зафиксирована, то ее словарь будем обозначать просто как V.) Каждый t_i это **отдельный термин** (ключевое слово), входящий хотя бы в один документ из D.

Бэг представлений слов. Каждый документ $d_j \in D$ представляется как **бэг слов**, т.е. как **неупорядоченная** совокупность терминов, входящих в этот документ (**бэг (bag)** означает, что в представлении учитывается число вхождений каждого термина, т.е. оно является мультимножеством).

в небольших коллекциях документов переход к основам может существенно улучшить качество поиска).

Стандартным представлением **бэга слов** являентся **вектор весов ключевых слов**: вектор, в котором для каждого термина $t_i \in V$ указан его **вес**, основанный на его вхождениях в d_j.

Таким образом, d_j рассматривается как вектор

$$d_j = (w_{1j}, w_{2j}, w_{3j}, \ldots, w_{Mj}).$$

Здесь w_{ij} – вес термина t_i в документе d_j.

В разных моделях ИП **веса терминов** представляются по-разному.

6.3.1 Булевская модель

Булевская модель информационного поиска является самой простой моделью ИП. Ее основные свойства:

Веса терминов. Документ $d_j \in D$ представляется как вектор **бинарных весов ключевых слов** $d_j = (w_{1j}, w_{2j}, w_{3j}, \ldots, w_{Mj})$, где $w_{ij} \in \{0, 1\}$ и

$$w_{ij} = \begin{cases} 1 & : \quad t_i \text{ содержится в } d_j; \\ 0 & : \quad \text{в противном случае.} \end{cases}$$

Запросы. Булевская модель допускает простые **запросы по ключевым словам** и **булевские запросы**. Запрос по ключевым словам $q = \{t_{q1}, \ldots, t_{qs}\}$ представляется булевским выражением $(t_{q1} \wedge t_{q2} \wedge \ldots \wedge t_{qs})$.

Обработка запроса. Вектор документа d_j релевантен запросу q (по ключевым словам или булевскому), если $d_j \models q$, т.е. если q **выполняется** или является **истинным на** d_j.

Это сводит задачу **обработки запроса** в булевской модели к *проверке выполнимости* или **вычислению булевского выражения** на нескольких наборах значений переменных.

Выдача результата. В качестве ответа возвращаются все документы, удовлетворяющие q. Выдаваемые документы **не ранжируются**: они либо включаются в ответ (соответствуют запросу), либо не включаются (не соответствуют запросу).

Достоинства: простота, эффективность, понятность.

Недостатки: нет ранжированя результатов, требует **точного соответствия**, что на практике всречается редко. Проверяется точное совпадение термина и не учитываются его синонимы и связанные с ним другие термины.

6.3.2 Модель векторного пространства (Vector Space Model)

Обзор. **Vector Space Model** представляет веса ключевых слов на шкале от 0 до 1 и представляет запросы тем же способом, что и документы. Для определения релевантности запроса она использует **близость косинусов**, которая и определяет ранг ответа.

Частота терминов. Для документа $d_j \in D$ и термина $t_i \in V$ **частота** f_{ij} **термина (ЧТ (TF))** t_i в d_j — это число различных вхождений t_i в d_j. Для документа d_j строится его вектор частот терминов

$$f_{d_j} = (f_{1j}, f_{2j}, \ldots, f_{Mj}).$$

Нормализованная частота терминов. Обычно для улучшения представления частоту терминов преобразуют. Используются два способа преобразования: **пороговая обработка** и **нормализация**.

Для заданного **значения порога** α определим **частоту термина** f'_{ij} как

$$f'_{ij} = \begin{cases} f_{ij} & : \quad f_{ij} < \alpha; \\ \alpha & : \quad f_{ij} \geq \alpha \end{cases}$$

(т.е., не учитываются вхождения терминов в документ сверх заданного **порога** α).

Для данного вектора f_{d_j} частот терминов (возможно, после пороговой обработки) **нормализованные частоты терминов** tf_{ij} определяются следующим образом:

$$tf_{ij} = \frac{f_{ij}}{\max(f_{1j}, f_{2j}, \ldots, f_{Mj})}.$$

Частота в документах (ЧД (DF)). Для данного термина $t_i \in V$ его **частота в документах**, df_i определяется как **число документов, в которые входит** t_i:

$$df_i = |\{d_j \in D \mid f_{ij} > 0\}|.$$

Обратная частота в документах (ОЧД (IDF)). Для данного термина $t_i \in V$ его **обратная частота в документах**, определяется как

$$idf_i = \log \frac{n}{df_i}.$$

Схема весов ключевых слов ЧД-ОЧД (TF-IDF). Для данного документа d_j и термина t_i,

$$w_{ij} = tf_{ij} \cdot idf_i = \frac{f_{ij}}{\max(f_{1j}, \ldots, f_{Mj})} \cdot \log_2 \frac{n}{df_i}.$$

Идея схемы весов tf-idf достаточно очевидна. Важность некоторого ключевого слова для документа измеряется с помощью двух правил:

1. Чем чаще встречается некоторое ключевое слово в документе, тем более оно важно для него.
2. Чем реже появляется ключевое слово в документах коллекции, тем более важным является его вхождение в каждый документ.

Частота термина (или нормализованная частота термина) соответствует первому правилу, а **обратная частота в документах** соответствует второму правилу.

Схема для весов запроса. Для запроса $q = (w_{1q}, \ldots, w_{Mq})$ веса терминов можно задать, используя ту же **схему весов TF-IDF**:

$$w_{iq} = tf_{iq} \cdot idf_i.$$

Однако, если запрос q короткий (содержит мало терминов по сравнению с документами из D), то веса можно несколько подправить:

$$w_{iq} = (0.5 + 0.5 \cdot tf_{iq}) \cdot idf_i.$$

Другие простые схемы назначения весов. Используются и некоторые другие простые схемы назначения весов:

Нормализованная частота термина (TF): $w_{ij} = tf_{ij}$.

Обратная частота термина в документах (IDF): $w_{ij} = idf_i$.

(Заметим, что когда в d_j нет повторяющихся ключевых слов, то схема назначения весов TF-IDF превращается в простую схему IDF. Это часто случается для коллекций, состоящих из небольших документов, например, списков цитат.)

Вычисление релевантности. Поиск в векторном пространстве традиционно использует **близость косинусов** в качестве меры релевантности:

$$sim(d_j, q) = cos(d_j, q) = \frac{d_j \cdot q}{||d_j|| \cdot ||q||} = \frac{\sum_{i=1}^{M} w_{ij} \cdot w_{iq}}{\sqrt{\sum_{i=1}^{M} w_{ij}^2 \cdot \sum_{i=1}^{M} w_{iq}^2}}.$$

6.3.3 Модификации поиска в модели векторного пространства

Имеются различные модификации схемы назначения весов TF-IDF и вычисления релевантности.

Okapi. **Okapi** это семейство методов поиска в векторном пространстве, которые пытаются учитывать несоответствия в размерах запросов и документов.

Эти методы можно рассматривать, как способы модификации весов или как способы модификации функций оценки релевантности.

Следующая формула показывает, как модифицируется релевантность:

$$okapi(d_j, q) = \sum_{t \in d_j, q} \ln \frac{n - df_i + 0.5}{df_i + 0.5} \times \frac{(k_1 + 1) \cdot f_{ij}}{k_1 \cdot (1 - b + b \cdot \frac{dl_j}{avdl}) + f_{ij}} \times \frac{(k_2 + 1) \cdot f_{iq}}{k_2 + f_{iq}}$$

Здесь,

dl_j:	длина документа d_j (в байтах)	
$avdl$:	средняя длина (в байтах) документов из D	
k_1:	параметр нормализации для d_j	$1.0 - 2.0$
b:	параметр нормализации для длины документа	обычно 0.75
k_2	параметр нормализации для запроса q	$1 - 1000$

Опорное нормализованнон взвешивание (Pivoted normalization weighting (pnw).

$$pnw(d_j, q) = \sum_{t_i \in d_j, q} \frac{1 + \ln(1 + \ln(f_{ij}))}{(1 - s) + s \cdot \frac{dl_j}{avdl}} \times f_{iq} \times \ln \frac{n+1}{df_i}.$$

Здесь, s – это параметр нормализации для длины документа (обычно используется значение 0.2).

6.4 Оценка систем ИП

Обозначение. Пусть $D = \{d_1, \ldots, d_n\}$ – коллекция документов. Пусть q – запрос, а $S_q = \{d'_1, \ldots, d'_k\}$ — это список документов, выданных некоторой системой ИП S. Документы ранжируются в порядке их предполагаемой релевантности запросу:

$$sim(d'_1, q) \geq sim(d'_2, q) \geq \ldots \geq sim(d'_k, q).$$

Предположим также, что если для некоторого $d_j \in D$ $sim(d_j, q) > 0$, то $d_j \in s(q)$, т.е. список S_q включает все документы, которые система считает релевантными.

Для запроса q пусть $D_q = \{d^*_1, \ldots, d^*_l\}$ — это **множество документов из D, которые на самом деле релевантны запросу q**. Это множество часто называют **"золотым стандартом"**.

Точность. Точность системы ИП S на запросе q – это процент **выданных** системой документов, которые **релевантны** этому запросу:

$$precision = \frac{|D_q \cap S_q|}{|S_q|}.$$

Полнота поиска (Recall). Полнота поиска системы ИП S на запросе q — это процент **релевантных** документов, **выданных** в ответ на этот запрос системой S:

$$recall = \frac{|D_q \cap S_q|}{|D_q|}.$$

F-мера. , **F-мера** — это гармоническое среднее **точности** и **полноты**.

$$f_1 = \frac{2 \cdot precision \cdot recall}{precision + recall}.$$

В более общем случае, для параметра $\beta \neq 0$,

$$f_\beta = \frac{(1 + \beta^2) \cdot precision \cdot recall}{\beta^2 \cdot precision + recall}.$$

Полнота и точность на уровне i. Для многих задач поиска информации все множество S_q является необозримым (чересчур много документов в ответе) либо никогда полностью не используется (например, большинство людей рассматривают только первые 1–2 страницы ссылок, возвращаемых системой Google.)

Поэтому важно определять аккуратность поиска, используя меры **полноты** и **точности** для ответов определенного ранга.

Обозначение. Пусть $S_q^i = \{d'_1, \ldots, d'_i\}$ это список первых i документов, возвращенных в ответе на запрос q.

Полнота на уровне i. Полнота на уровне i определяется как процент релевантных документов среди первых i т.е. во множестве S_q^i:

$$recall(i) = \frac{|S_q^i \cap D_q|}{|D_q|}.$$

Точность на уровне i. Точность на уровне i также определяется по множеству S_q^i:

$$precision(i) = \frac{|S_q^i \cap D_q|}{|S_q^i|}.$$

Средняя точность. **Средняя точность** это среднее значение точности по всем уровням i:

$$p_{avg} = \frac{1}{|S_q|} \cdot \sum_{i=1} |S_q| precision(i).$$

Меры для многих запросов. Средняя ожидаемая точность. Поведение системы ИП на отдельных запросах может сильно различаться. Важно уметь оценивать точность систем ИП в общем, *для многих запросов.*

Пусть $Q = \{q_1, \ldots q_L\}$ – некоторый список запросов и пусть $S_{q_1}, \ldots, S_{q_L}$ – это списки ответов, полученных на эти запросы. **Средняя точность на Q при уровне эффективности r_j** определяется как

$$\hat{precision}_Q(r_j) = \frac{1}{|Q|} \cdot \sum_{i=1}^{|Q|} precision_{q_i}(r_j).$$

Пусть $S_q(r_j)$ это множество $S_q^{s_q} = \{d_1^q, \ldots, d_{s_q}^q\} \subseteq S_q$ такое, что $recall(S_q^{s_q} \geq r_j$, но $recall(S_q^{s_q-1}) < r_j$.

Общая точность при уровне эффективности r_j определяется как

$$precision_Q(r_j) = \frac{\sum_{i=1}^{|Q|} |S_{q_i}(r_j) \cap D_{q_i}|}{\sum_{i=1}^{|Q|} |S_{q_i}(r_j)|}.$$

6.5 Препроцессирование

Препроцессирование включает ряд **независимых от модели** этапов, т.е. задач, которые необходимы для представления документов из D в виде **бэгов слов**.

Эти задания включают **анализ**, **удаление стоп-слов** и **морфологический поиск и выделение основ слов (Stemming)**.

Анализ

Анализ это единственный из этапов, который зависит от формата входа.

При **анализе** документы d_j из коллекции D читаются и обрабатываются один за другим. Как правило, выполняются следующие действия:

- Входной документ **размечается**. Анализатор определяет границы слов, знаки пунктуации и другие элементы документа.
- Если требуется, входной документы **фильтруется**. Некоторые документы поступают в систему ИП в виде `HTML` или `XML` файлов. Другие документы могут содержать команды форматизации, таблицы, рисунки и другие элементы которые либо требуют углубленного анализа, либо могут быть проигнорированы.

  (Например, в документах, поступивших в виде HTML-файлов, можно удалить многие теги.)

Устранение стоп-слов

Стоп-слова. **Стоп-слово** — это слово, которое настолько часто встречается в разговорной речи, в литературе или в данной коллекции документов, что не имеет никакого специфического значения и не может использоваться для различения документов.

Существуют различные списки стоп-слов. Типичными стоп-словами являются местоимения ("я "ты "вы "они "их "его"и др.), общие глаголы ("быть "делать "иметь "становиться"и др.), предлоги и союзы: ("в "на "и "или "но "а"и др.).

В некоторых коллекциях документов имеются специфические стоп-слова. Например вотчетах о футбольных матчах такими словами могут быть "мяч "гол "футбол" и т.п.

Устранение стоп-слов. Стоп-слова могут быть причиной *ложных позитивов*, поэтому важно от них избавиться. Традиционный способ следующий:

- Для каждого элемента типа слово из входного документа проверить входит ли оно в список известных стоп-слов.
- Если входит, то удалить его и перейти к следующему слову.
- Если не входит, то передать этот элемент на следующую стадию обработки.

Морфологический поиск и нормализация (выделение основ) слов (Stemming)

Нормализация – это процесс замены слова его **основой** или **корнем**[2].

Процедуры выделения основ свои для каждого языка. Для английского традиционным является нормализация (Stemming) на базе *алгоритма Портера (Porter Algorithm)*[21]. Подробное писание этого алгоритма и его реализацию можно найти в [21] и по ссылке
http://snowball.tartarus.org/algorithms/porter/stemmer.html
Ссылки на ряд других реализаций алгоритма приведены в
http://tartarus.org/ martin/PorterStemmer/index.html

Алгоритм Портера организован как последовательность применений некоторых правил сокращения окончаний. Для данного слова на каждом шаге алгоритма проверяется применимость к нему некоторого правила. Правило определяет окончание текущего слова и

[2]Технически основа слова это не просто его корень. Алгоритмы выделения основ оставляют нетронутыми приставки.

заменяет его более коротким окончанием (иногда – просто удаляет его). В английском языке возможных окончаний сравнительно немного – около 250. В русском языке морфология более богатая (см. [31]) и соответствующие алгоритмы гораздо сложнее.

6.6 Расширение модели векторного пространства

Модель векторного пространства может быть расширена в разных направлениях:

- Учет обратной связи.
- Тезаурус для учета синонимов или похожих слов.
- Инвертированные индексы для ускорения поиска.
- Адресные файлы запросов по сходству.

6.6.1 Учет обратной связи

Учет обратной связи для оценки релевантности основан на применении ряда методов ИП, которые используют *оценки релевантности ответов системы ИП, полученные от людей*[3] для уточнения (и, теоретически, **для улучшения**) результатов процедуры ИП.

Процесс. Пусть q – это запрос к коллекции документов D. Пусть в ответ на этот запрос выдано множество документов D_q.

Человек-аналитик (пользователь системы ИП) проверяет некоторые документы из D_q и выделяет в нем два подмножества: D_q^r – множество *релевантных документов* и D_q^{irr} – множество *нерелевантных документов*. Мы предполагаем, что

$$D_q - (D_q^r \cup D_q^{irr}) \neq \emptyset.$$

Идея. Изменить представление запроса q так, чтобы вернуть больше документов аналогичных документам из D_q^r, и исключить из ответа документы из D_q^{irr} и близкие к ним.

Метод Rocchio для учета обратной связи. Вектор, представляющий запрос q, заменяется новым вектором q_e, в котором:

- ключевым словам из документов множества D_q^r внимание усилено, а
- внимание к ключевым словам из документов множества D_q^{irr} уменьшено.

$$q_e = \alpha \cdot q + \frac{\beta}{|D_q^r|} \sum_{d_r \in D_q^r} d_r - \frac{\gamma}{|D_q^{irr}|} \sum_{d_i \in D_q^{irr}} d_i.$$

Здесь, α, β и γ, обычно выбираемые так, что $\alpha+\beta+\gamma = 1$, представляют *важность исходного запроса*, *важность позитивной информации* и *важность негативной информации*, соответственно.

[3] Обычно, тех, которые и задавали запросы.

Замечания. Формула метода Rocchio делает возможным ***отрицательный вес*** некоторых ключевых слов. Отрицательный вес некоторого ключевого слова в векторе запроса означает, что *отсутствие этого ключевого слова в документе важно для определения его релевантности*. Отметим, что обычно отрицательные значения весов игнорируются, т.е. заменяются на 0.

Вариации. Имеется много вариантов метода Rocchio.

- Не учитывать отрицательную информацию: ($\gamma = 0$)

$$q_e = \alpha \cdot q + \frac{\beta}{|D_q^r|} \sum_{d_r \in D_q^r} d_r.$$

- Уменьшить влияние отрицательной информации. Использовать лишь один вектор из D_q^{irr}:

$$q_e = \alpha \cdot q + \frac{\beta}{|D_q^r|} \sum_{d_r \in D_q^r} d_r - \gamma \cdot d_{irr}^{max},$$

  где $d_{irr}^{max} \in D_q^{irr}$ это нерелевантный документ с наибольшим рангом.

Метод учета обратной связи вслепую . Он известен также как **релевантность с помощью псевдообратной связи**. Пусть система ИП возвращает множество документов D_q в ответ на запрос q. Предположим, что первые $k << |D_q|$ документов из этого множества **релевантны**, а остальные – нет. Затем применим метод Rocchio's (с учетом или без учета негативной информации) для модификации представления запроса q.

6.6.2 Использование тезауруса

Все рассмотренные выше методы включают документ в ответ, если он содержит **хотя бы одно ключевое слово (основу)**, входящее в запрос.

Тезаурусы помогают смягчить это условие.

Простой тезаурус — это совокупность троек вида

$$(t_i, t_j, \alpha),$$

где $t_i, t_j \in V$ – это два слова (две основы) из словаря, а $\alpha \in (0, 1]$ — это *степень их близости*.

Если $\alpha = 1$, то t_i и t_j являются **точными синонимами**. Например, тройка *(person, human, 1.00)* означает, что слова *"person"* и *"human"* должны рассматриваться как полные синонимы.

Значение $\alpha < 1$ означает, что t_i и t_j похожи, но не являются полными синонимами. Например, тройку *(автомобиль, Лада, 0.5)* можно включить в тезурус, поскольку мы знаем, что *"Лада"* (в типичных случаях) является автомобилем, но не каждый автомобиль является Ладой.

Вычисление близости. В присутствии тезауруса вычисление близости между документами происходит по измененной формуле Пусть $T = \{(t_i, t_k, \alpha_{ik}\}$ — это простой тезаурус.

$$sim(d_j, q) = \frac{\sum_{i=1}^{M} d_{ij} \cdot q_i + \sum_{(t_i,t_k,\alpha_{ik})\in T} \alpha_{ik} \cdot d_{ij} \cdot q_k}{\sqrt{\sum_{i=1}^{M} d_{ij}^2 \cdot \sum_{i=1}^{M} q_i^2}}.$$

Замечание. Простые тезаурусы могу быть как *симметричными*, так и *несимметричными*. В **симметричном** случае принадлежность $(t_i, t_k, \alpha) \in T$ влечет $(t_k, t_i, \alpha) \in T$. В **несимметричном** случае возможно, что в тезаурус одновременно входят тройки $t_i, t_k, \alpha) \in T$ и $(t_k, t_i, \alpha') \in T$ для $\alpha' \neq \alpha$, или даже только одна из них. Во всех случаях приведенная выше формула работает.

6.6.3 Инвертированные индексы

Без специальной предварительной обработки, всякий раз, получив запрос q, система ИП должна вычислить все значения $sim(d_1, q), sim(d_2, q), \ldots, sim(d_n, q)$. При большом n это требует больших затрат времени.

Инвертированный индекс это структура данных, которая позволяет обрабатывать запрос более эффективно.

Коллекцию векторов документов $D = \{d_1, \ldots, d_n\}$ можно рассматривать как отображение из идентификаторов документов $(d_1, \ldots, d_n)$ во множество подмножеств 2^V ключевых слов $V = \{t_1, \ldots, t_m\}$.

Инвертированный индекс это отображение из множества V *терминов* во множество *документов*, содержащих эти термины.

Простой инвертированный индекс состоит из списков вида $\{\langle t_i, (d_1^i, \ldots, d_{k_i}^i)\rangle\}$, где $t_i \in V$, а $d_1^i, \ldots, d_{k_i}^i$ это *все* документы из D, которые содержат t_i.

Пример. Рассмотрим три следующих документа:

d_1 *When I say stop, continue.*

d_2 *When I say stop, stop and turn around.*

d_3 *Around the bend, the river continued.*

Предположим, что из них удалены стоп-слова *"the"* и *"and"*, а *"continued"* заменено на основу *"continue"*. Тогда **простой инвертированный индекс** для этой коллекции документов выглядит так:

`when`	d_1, d_2
`I`	d_1, d_2
`say`	d_1, d_2
`stop`	d_1, d_2
`continue`	d_1, d_3
`turn`	d_2
`around`	d_2, d_3
`bend`	d_3
`river`	d_3

Поиск с использованием инвертированных индексов. Пусть D — коллекция документов, V — ее словарь, и I ее инвертированный индекс. Пусть $I(t_i)$ обозначает список документов, содержащих t_i. Обработка запроса q происходит следующим образом.

- **Шаг 1: Перечисление.** Для каждого термина t_i, представленного в q, вернуть $I(t_i)$.
- **Шаг 2: Слияние.** Вычислить пересечение всех этих множеств $I(t_i)$. (При необходимости вычислить объединение этих множеств $I(t_i)$ и отсортировать документы по числу входящих в них терминов из q.
- **Шаг 3: Ранжирование.** Для каждого документа d_j из списка, полученного на шаге 2, вычислить $sim(d_j, q)$. Отсортировать все документы по степени близости к q.

6.6.4 Инвертированные индексы со списками адресов (позиций) (Postings Files)

Инвертированный индекс можно приспособить для получения ответов на запросы с условиями на близость (proximity queries).

Адреса. **Адрес** — это тройка (t_i, d_j, k), которая определяет, что термин t_i входит в документ d_j в позиции k. **Позиция** обычно определяется номером слова в документе, после устранения в нем стоп-слов.

Инвертированный индекс со списком адресов это **инвертированный индекс**, в котором для каждого документа и каждого термина заданы все позиции, в которых термин входит в документ. Более формально, **инвертированный индекс со списком адресов** это совокупность наборов вида

$$\langle t_i, \langle d^i_1, (k_{11}, \ldots, k_{1s_1})\rangle, \ldots, \langle d^i_{l_i}, (k_{l_i1}, \ldots k_{l_i s_{l_i}})\rangle\rangle.$$

Здесь $d^i_1, \ldots, d^i_{l_i}$ это **все** документы из D, содержащие термин t_i, а k_{rt} это все позиции, в которых этот термин входит в соответствующий документ.

Пример. **Инвертированный индекс со списком адресов** для указанной выше коллекции документов выглядит так:

`when`	$(d_1, 1), (d_2, 1)$
`I`	$(d_1, 2), (d_2, 2)$
`say`	$(d_1, 3), (d_2, 3)$
`stop`	$(d_1, 4), (d_2, 4, 5)$
`continue`	$(d_1, 5), (d_3, 4)$
`turn`	$(d_2, 6)$
`around`	$(d_2 7), (d_3, 1)$
`bend`	$(d_3, 2)$
`river`	$(d_3, 3)$

Запросы с условиями на близость. **Инвертированный индекс со списком адресов** можно использовать дляя эффективного поиска ответов на запросы с точным совпадением фразы и на запросы с условиями на близость ключевых слов.

Пусть D — коллекция документов, V — ее словарь и I — ее инвертированный индекс со списком адресов. Через $I(t_i)$ обозначим список документов, содержащих t_i, а через $I(t_i, d_j)$ обозначим список адресов t_i в d_j. Ответ на запрос q, в котором требуется найти документы с точным вхождением некоторой фразы (или с достаточно близким расположением нескольких елючевых слов), можно получить следующим образом:

- **Шаг 1: Перечисление.** Для каждого термина t_i из запроса q определить $I(t_i)$.
- **Шаг 2: Слияние.** Вычислить пересечение всех полученных множеств $I(t_i)$.
- **Шаг 3: Фильтрация.** Для каждого документа d_j, полученного после шага 2, определить близость ключевых слов. Если условия близости нарушены, то удалить документ из списка ответов.
- **Шаг 4: Ранжирование.** Для каждого из оставшихся документов d_j вычислить $sim(d_j, q)$. Отсортировать все документы по убыванию степени близости к q.

6.7 Информационный поиск как задача классификации

ИП и классификация. Задача ИП может быть сформулирована как задача классификации в следующей форме:

> По информации о запросе q и некоторому документу $d \in D$ классифицировать d как либо **релевантный для** q, либо **нерелевантный для** q.

Замечание: на самом деле, это задача **частичного обучения с учителем**, так как мы знаем, классы, но **не имеем** обучающей выборки.

ИП и вероятности. Можно преобразовать задачу классификации документа d как **релевантного** или **нерелевантного** в задачу оценки вероятностей

$$Pr(R|d,q) \text{ и } Pr(N|d,q) \qquad (Pr(N|d,q) = 1 - Pr(R|d,q)) .$$

Здесь,

$Pr(R\|d,q)$ это вероятность классификации документа d как **релевантного** запросу q.
$Pr(N\|d,q)$ это вероятность классификации документа d как **нерелевантного** запросу q.

6.7.1 Использование алгоритма Наивный Баейес

Мы уже рассматривали алгоритм **Наивный Баейес (Naive Bayes)** для решения задачи классификации с учителем. Напомним основные формулы и шаги этого алгоритма.

1. Использование теоремы Байеса.

$$Pr(c_i|d) = Pr(c_i|A_1 = a_1, \dots A_n = a_n) = \frac{Pr(A_1 = a_1, \dots, A_n = a_n|c_i) \cdot Pr(c_i)}{Pr(A_1 = a_1, \dots A_n = a_n)}.$$

2. Упрощение.

$$Pr(c_i|A_1 = a_1, \dots A_n = a_n) \sim Pr(A_1 = a_1, \dots, A_n = a_n|c_i) \cdot Pr(c_i).$$

3. Использование предположения о независимости.

$Pr(c_i|A_1 = a_1, \dots A_n = a_n) \sim Pr(A_1 = a_1|c_i) \cdot \dots \cdot Pr(A_n = a_n|c_i) \cdot Pr(c_i) =$
$Pr(c_i) \cdot \prod_{j=1}^{n} Pr(A_j = a_j|c_i)$.

4. Оценка вероятностей.

$$Pr(c_i) = \frac{|D_i|}{|D|}.$$

$$Pr(A_j = a_j|c_i) = \frac{|D_{ij}|}{|D_i|}.$$

$$Pr(c_i|A_1 = a_1, \dots A_n = a_n) \sim \frac{|D_i|}{|D|} \cdot \prod_{j=1}^{n} \frac{|D_{ij}|}{|D_i|} = \frac{|D_{i1}| \cdot \dots \cdot |D_{in}|}{|D| \cdot |D_i|^{n-1}}.$$

5. Предсказание.

$$c(d) = arg \max_{i=1,\dots,k} (Pr(d|c_i) \cdot Pr(c_i)) = arg \max_{i=1,\dots,k} \frac{|D_{i1}| \cdot \dots \cdot |D_{in}|}{|D| \cdot |D_i|^{n-1}}.$$

Naive Bayes для поиска информации

Модель алгоритма **Naive Bayes** для поиска информации часто называют *Вероятностным ИП*, *Бинарным независимым поиском* или *Статистической моделью языка*.

Как и ранее, *задача ИП* формулируется следующим образом:

> Найти для заданной коллекции документов D и запроса q все документы из D, которые *релевантны* q.

Рассмотрим пошаговое применение алгоритма **Наивный Баейес** к решению этой задачи.

Шаг 0: выбор модели ИП. Для сравнения документа d и запроса q будем использовать их представления в виде **бинарных векторов**: $d = (w_1, \dots, w_N)$, $q = (q_1, \dots, q_N)$, где $d_i = 1$ ($q_i = 1$), если термин t_i входит в d (q) и 0 – в противном случае.

Шаг 1: применение теоремы Байеса.

$$Pr(R|d, q) = \frac{Pr(d|R, q) \cdot Pr(R|q)}{Pr(d|q)}.$$

$$Pr(N|d, q) = \frac{Pr(d|N, q) \cdot Pr(N|q)}{Pr(d|q)}.$$

здесь,
$Pr(d|R,q)$ – это вероятность того, что документ d окажется среди документов, релевантных q (вероятность обнаружить d при условии, что в ответ на запрос q возвращен релевантный документ);
$Pr(R|q)$ – это вероятность получения релевантного документа в ответ на данный запрос q;
$Pr(d|q)$ – это вероятность получения d в ответ на q;
$Pr(d|N,q)$ – это вероятность того, что документ d окажется среди документов, нерелевантных q (вероятность обнаружить d при условии, что в ответ на запрос q возвращен нерелевантный документ);
$Pr(N|q)$ – это вероятность получения нерелевантного документа в ответ на данный запрос q.

Заметим, что мы предполагаем, что $P(d|q) \neq 0$.

Шаг 2: переход к отношению вероятностей. Алгоритм **Наивный Баейес** при классификации переходит от оценки первоначальной условной вероятности к оценке только числителя дроби.

В случае ИП у нас только два класса, поэтому можно перейти к оценке *отношения вероятности* того, что d релевантен, к вероятности того, что он нерелевантен относительно запроса q. При этом можно достичь требуемого результата без оценки $Pr(d|q)$:

$$O(R|d,q) = \frac{Pr(R|d,q)}{Pr(N|d,q)} = \frac{\frac{Pr(d|R,q)\cdot Pr(R|q)}{Pr(d|q)}}{\frac{Pr(d|N,q)\cdot Pr(N|q)}{Pr(d|q)}} = \frac{Pr(d|R,q)\cdot Pr(R|q)}{Pr(d|N,q)\cdot Pr(N|q)}.$$

Шаг 3: упрощение отношения. Заметим, что отношение

$$\frac{Pr(R|q)}{Pr(N|q)},$$

вероятностей получения релевантного и нерелевантного ответов на запрос q *не зависит от документа* d, т.е. является константой для каждого запроса q.

Таким образом, наша задача сводится к оценке отношения

$$\frac{Pr(d|R,q)}{Pr(d|N,q)} \sim O(R|d,q).$$

Шаг 4: "наивное"предположение. Мы предполагаем *условную независимость* терминов из d для данного q. Это позволяет использовать следующие подстановки:

$$Pr(d|R,q) = Pr(d[1] = w_1, \ldots, d[N] = w_N|R,q) = \prod_{i=1}^{N} Pr(d[i] = w_i|R,q).$$

$$Pr(d|N,q) = Pr(d[1] = w_1, \ldots, d[N] = w_N|N,q) = \prod_{i=1}^{N} Pr(d[i] = w_i|N,q).$$

Следовательно, получаем

$$\frac{Pr(d|R,q)}{Pr(d|N,q)} = \prod_{i=1}^{N} \frac{Pr(d[i] = w_i|R,q)}{Pr(d[i] = w_i|N,q)},$$

или

$$O(R|d,q) = O(R|q) \cdot \prod_{i=1}^{N} \frac{Pr(d[i] = w_i|R,q)}{Pr(d[i] = w_i|N,q)}.$$

Шаг 5: разделение вероятностей вхождения и отсутствия терминов. Заметим, что в наших уравнениях w_i может принимать только два значения: 1 (термин t_i входит в документ) и 0 (термин t_i не входит в документ). Тогда можно переписать последнюю формулу следующим образом:

$$O(R|d,q) = O(R|q) \cdot \prod_{i:w_i=1} \frac{Pr(d[i]=1|R,q)}{Pr(d[i]=1|N,q)} \cdot \prod_{i:w_i=0}^{N} \frac{Pr(d[i]=0|R,q)}{Pr(d[i]=0|N,q)}.$$

Шаг 6: информационный поиск (сопоставление терминов). Введем следующие обозначения.

Обозначим через p_i вероятность $Pr(d[i]=1|R,q)$ того, что документ, релевантный q, содержит термин t_i.

Обозначим через u_i вероятность $Pr(d[i]=1|R,q)$ того, что документ, нерелевантный q, содержит термин t_i.

Используя эти обозначения , можно определить следующую матрицу:

Документ:	Релевантен q (R)	Нерелевантен q (N)
Термин входит: $w_i = 1$	p_i	u_i
Термин отсутствует: $w_i = 0$	$1 - p_i$	$1 - u_i$

Используя новые обозначения, можно переписать интересующее нас отношение в виде:

$$O(R|d,q) = O(R|q) \cdot \prod_{i:w_i=1} \frac{p_i}{u_i} \cdot \prod_{i:w_i=0} \frac{1-p_i}{1-u_i}.$$

Сейчас сделаем следующее *упрощающее предположение*:

Если термин t_i **не входит** в запрос q, т.е., если $q_i = 0$, то считаем, что

$$p_i = u_i.$$

(Термины, не входящие в запрос, с одинаковой вероятностью могут появляться в релевантных и в нерелевантных документах).

Используя это предположение, можно устранить некоторые из дробей $\frac{p_i}{u_i}$ и $\frac{1-p_i}{1-u_i}$ из формулы для отношения:

$$O(R|d,q) = O(R|q) \cdot \prod_{i:q_i=w_i=1} \frac{p_i}{u_i} \cdot \prod_{i:q_i=1,w_i=0} \frac{1-p_i}{1-u_i}.$$

Можем переписать эту формулу как

$$O(R|d,q) = O(R|q) \cdot \prod_{i:q_i=w_i=1} \frac{p_i(1-u_i)}{u_i(1-p_i)} \cdot \prod_{i:q_i=1} \frac{1-p_i}{1-u_i}.$$

Заметим, что в этой формуле выражение

$$\prod_{i:q_i=1} \frac{1-p_i}{1-u_i}$$

для данного запроса является константой, не зависящей от документа.

$$O(R|d,q) = K_q \cdot \prod_{i:q_i=w_i=1} \frac{p_i(1-u_i)}{u_i(1-p_i)},$$

где $K_q = O(R|q) \cdot \prod_{i:q_i=1} \frac{1-p_i}{1-u_i}$ – некоторая константа для данного q.

Шаг 7: переход к значению статуса возврата (Retrieval Status Value). Наша цель – оценка

$$\prod_{i:q_i=w_i=1} \frac{p_i(1-u_i)}{u_i(1-p_i)}.$$

Вместо этой величины мы можем оценивать **значение статуса возврата** для документа d по отношению к запросу q, обозначаемое через RSV_d и определяемое следующим образом:

$$RSV_d = \log\left(\prod_{i:q_i=w_i=1} \frac{p_i(1-u_i)}{u_i(1-p_i)}\right) = \sum_{i:q_i=w_i=1} \log\frac{p_i(1-u_i)}{u_i(1-p_i)}.$$

Замечание: переход к логарифмам *сохраняет монотонность* наших оценок ($O(R|d,q) > O(R|d',q) \iff RSV_d > RSV_{d'}$). При этом произведение заменилось на сумму, которая менее чувствительна к малым значениям вероятностей.

Определим c_i как **логарифм отношения вероятностей для термина t_i из запроса** q:

$$c_i = \log\frac{p_i(1-u_i)}{u_i(1-p_i)} = \log\frac{p_i}{1-p_i} + \log\frac{1-u_i}{u_i}.$$

Вычисление близости. Близость между документом и запросом определяется как соответствующее значение статуса возврата:

$$sim(d,q) = RSV_d = \sum_{i:q_i=w_i=1} c_i.$$

Шаг 8: оценка параметров. Нам нужен способ оценки значений c_i или их компонент p_i и u_i.

Теория. Предположим, что нам доступно множество D_q всех документов, которые **релевантны** запросу q. Пусть $|D_q| = S$. Пусть также $D_{qi} = \{d \in D_q | d[i] = 1\}$ и пусть $|D_{qi}| = s$ (т.е., s из S релевантных документов содержат термин t_i).

Тогда можно оценить p_i и u_i следующим образом:

$$p_i = \frac{s}{S-s}$$

$$u_i = \frac{df_i - s}{|D| - df_i - s + S}$$

$$c_i = \log\frac{s\cdot(|D| - df_i - s + S)}{(df_i - s)\cdot(S-s)}.$$

Сглаживание. Оценку c_i можно *сгладить*, чтобы устранить влияние нулей:

$$c_i = \log\frac{(s+0.5)\cdot(|D| - df_i - s + S + 0.5)}{(df_i - s + 0.5)\cdot(S - s + 0.5)}.$$

Практика. На практике *множество ответов* (т.е., список релевантных документов), как правило, недоступен (в частности, потому что запросы поступают динамически).

Поэтому оценки *огрубляются (kludged)* следующим образлом:

$$\log \frac{1-u_i}{u_i} = \log \frac{|D| - df_i}{df_i} \approx \log |D| df_i = idf_i.$$

Это предполагает, что число релевантных документов **намного меньше** общего размера коллекции (так что, $|D| - df_i$ почти совпадает с $|D|$.)

Оценка p_i. Оценки p_i-ых получить *труднее*. На практике эти оценки выбираются достаточно наобум следующим способом:
a) Положим $p_i = c$ для некоторого $c \in (0, 1)$. Типичным значением этого вида являются $p_i = 0.5$.
b) Свяжем p_i с df_i:

$$p_i = \frac{1}{3} + \frac{2}{3} \cdot \frac{df_i}{N}.$$

6.8 Задачи

Задача 6.1. *Пусть коллекция документов содержит следующие тексты:*
1: В лесу растут сосны, ели, дубы.
2: Лес – это граф, состоящий из нескольких деревьев.
3: В лесах вокруг Твери много ягод.
4: В хвойных лесах вокруг Твери растут ели и сосны.
5: Вблизи Твери в лесах не растут дубы.
6: Сосны – деревья, ели – деревья, дубы – деревья, графы – не деревья.

а) Определите словарь (набор основ терминов) этой коллекции.
б) Постройте представления документов в булевской модели.
в) Постройте представления документов в модели векторного пространства.
г) Найдите ответы на следующий запрос в булевской модели:
q1 = сосны $\wedge$ ели $\wedge$ дубы $\wedge\neg$ графы.
д)Постройте представления следующих запросов:
q2 = "Какие деревья растут в лесах вокруг Твери?"
q3 = "Сосны – это деревья или графы?"
и найдите 3 лучших ответа на каждый из этих запросов, используя при вычислении релевантности близость косинусов.

Задача 6.2. *Точность (precision) и полнота (эффективность) поиска (recall) являются важными мерами качества систем ИП.*
(a) Объясните, почему обычно рассматривают обе эти меры. Почему для этой цели хорошо подходит F-мера.
(b) Предложите способы, которые могут эффективно улучшить F-меру в системах ИП.

Задача 6.3. *Схема весов ЧД-ОЧД (TF-IDF) используется как эффективная мера для классификации документов.*
(a) Приведите пример, показывающий что эта мера не всегда подходит для сравнения документов.
(b) Определите другую меру, которая может преодолеть возникшую в вашем примере трудность.

Задача 6.4. *Можно предположить, что ряд терминов зависит друг от друга. То есть они с большой вероятностью попадут в один документ. Объединение таких терминов может уменьшить размерность векторного пространства. Предложите способ кластеризации, позволяющий выявить классы "похожих"терминов и сократить размерность векторного пространства.*

Задача 6.5. *При каких условиях модифицированный методом Rocchio запрос q_r совпадает с исходным запросом q? Можно ли утверждать, что во всех других вариантах запрос q_r ближе к центроиду релевантных документов, чем запрос q?*

Задача 6.6. *Почему положительная обратная связь в системах ИП считается более полезной, чем отрицательная? Почему использование только одного нерелевантного документа из D_q^{irr} может быть более эффективным, чем использование нескольких?*

Задача 6.7. *В следующей таблице представлены данные о частоте вхождения 4-х терминов в три документа и обратная частота idf этих терминов в некоторой реальной коллекции документов.*

Термин	*Doc1*	*Doc2*	*Doc3*	idf_t
car	*27*	*4*	*24*	*1.65*
auto	*3*	*33*	*0*	*2.08*
insurance	*0*	*33*	*29*	*1.62*
best	*14*	*0*	*17*	*1.5*

Вычислите представления трех документов по схеме TDF-IDF (считая, что других терминов в документах нет) и определите наиболее релевантный из них запросу "What is the best insurance scheme for my auto?"

Задача 6.8. *Для заданного запроса q можно ранжировать документы $d_1, d_2, ..., d_n$ по возрастанию Евклидового расстояния от q. Покажите, что если q и все d_i нормализованы к единичным векторам, то порядок ранжирования документов с использованием Евкидового расстояния совпадает с порядком их ранжирования относительно q по косинусам.*

Глава 7

Основы анализа социальных сетей

7.1 Социальные сети

Социальные сети начали изучаться в социологии в 50-е годы XX-го века. В учебниках социологии *социальную сеть* определяют как совокупность сетевых акторов (точек, вершин, агентов), вступающих во взаимодействие друг с другом и связи между которыми преимущественно социальные, такие как дружественные отношения, совместная работа, обмен информацией и т.п.

Таким образом, социальная сеть это **граф**, вершины которого акторы, а ребра представляют связи, взаимодейсвия отношения между различными акторами.

Среди них могут быть связи, отражающие подобие акторов, их социальные отношения, взаимодействия, местоположение, атрибуты, родственные связи, роли, эмоциии, когнитивные отношения и т.п.

Примерами таких взаимодействий и отношений являются:

- Посылать электронное письмо кому-то (асимметрично);
- Быть начальником кого-то (асимметрично);
- Быть другом кого-то (симметрично);
- Работать над одним проектом с кем-то (симметрично);
- Написать совместную статью с кем-то (симметрично);
- Сослаться на работу кого-то (асимметрично);
- Состоять в одних клубах с кем-то (симметрично);
- Участвовать в событии (асимметрично);
- Быть матерью кого-то (асимметрично);
- Иметь общего родителя с кем-то (симметрично);
- Любить кого-то (асимметрично);
- Знать о чем-то (асимметрично);

- Обсуждать с кем-то (симметрично);
- Советовать кому-то (асимметрично);
- Помогать кому-то (асимметрично);
- ...

Размеры социальных сетей варьируются от нескольких акторов (в сетях отношений одной семьи, сотрудников малого предприятия и т.п.) до миллионов и десятков миллионов акторов в интернет-сетях таких как Facebook, MySpace, Одноклассники, В контакте и др. Да и вся паутина Web-страниц может рассматриваться (и на самом деле рассматривается поисковыми системами) как одна социальная сеть с отношением "ссылается на".

Главные задачи анализа социальных сетей состоят в определении их теоретико графовых свойств, которые характеризуют
а) структуру сети (анализ на уровне сети);
б) положение в сети (анализ на уровне вершин);
в) попарные свойства (анализ на уровне пар).

Анализ на уровне сети связан с двумя видами свойств: связанностью и формой. Связанность характеризуется такими свойствами как плотность, длина путей, фрагментация. Меры связанности позволяют также выделять в сетях подгруппы ("сообщества","клики") – области, обладающие таким специфическими свойствами связанности как высокая плотность, небольшая длина путей между акторами и др. Форма сети связана с распределением ее соединений (ребер). Сюда относятся такие свойства как ядерность / переферийность, массивность и т.п. Анализ на уровне вершин занимается свойствами центральности вершин, связанными с важностью вершин, их доминирующим положением в сети. Например, одним из свойств центальности является *промежуточность по Фримену (Freeman)*, которая отражает свойство вершины лежать на на кратчайших путях, соединяющих две другие вершины. Это можно интерпретировать как потенциальную силу актора, который может замедлить идущие сквозь него потоки или исказить их в свою пользу. Анализ на уровне пар, как правило, связан с двумя видами свойств: парной связанностью и эквивалентностью. Связанность пары вершин означает как их близость в сети, так и наличие многих типов свзей между ними. Эквивалентность оценивает степень, с которой вершины пары играют в структуре сети одинаковые роли, например, изоморфность их окрестностей.

7.2 Социальные сети как графы

Мы будем рассматривать социальные сети как ориентированные или неориентированные графы в зависимости от видов представляемых ими связей, взаимодействий или отношений ([23, 24, 28].

Ребра в графах социальных сетей называются связями (links) или соединениями (ties).

Обозначение. Пусть $I = \{i_1, \dots i_n\}$ – это некоторое множество акторов. Социальная сеть – это граф без петель $G = \langle I, E \rangle$, где E – множество (ориентированных или неориентированных) связей между парами акторов. Для конкретных сетей вершины и ребра представлящих их графов могут иметь специальные метки, содержащие дополнительную информацию. Например, имена вершин, типы и интенсивность связей, веса ребер и т.п.

Введем следующие обозначения для сети $G =< I, E >$ и актора $i \in I$ пусть:

$d(i) = |\{j \mid (i,j) \in E\}|$ обозначает степень i в неориентированном графе сети G;
$d_i(i) = |\{j \mid (j,i) \in E\}|$ обозначает (полу)степень захода в i в G (для ориентированного G);
$d_o(i) = |\{j \mid (i,j) \in E\}|$ обозначает (полу)степень исхода из i в G (для ориентированного G).

Очевидно, максимальная степень актора в сети G с n акторами равна $n-1$.

Для акторов i и j обозначим через $d(i,j)$ длину кратчайшего пути из i to j (число ребер на этом пути).

Важную роль в анализе социальных сетей играют следующие формализации в терминах теории графов понятий, содержательно описанных выше.

7.2.1 Центральность (Centrality)

Центральный актор – это актор, у которого много связей с другими акторами. Такие акторы играют важную роль в анализе социальных сетей. Имеется несколько формализаций центральности в терминах графов.

Степень центральности оценивает относительную степень вершины актора в сети.

Для *неориентированного графа* G степень центральности актора i, обозначаемая $C_D(i)$, определяется как

$$C_D(i) = \frac{d(i)}{n-1}.$$

Для *неориентированного графа* G степень центральности актора i, также обозначаемая $C_D(i)$, определяется как

$$C_D(i) = \frac{d_o(i)}{n-1}.$$

(т.е., учитываются только выходящие из i связи).

Близкая центральность (Closeness Centrality)

Эта величина оценивает насколько близки к данному актору остальные акторы сети. Как для ориентированных, так и для неориентированных сетей близкая центральность актора i обозначается $C_C(i)$ и определяется как:

$$C_C(i) = \frac{n-1}{\sum_{j \in I} d(i,j)}.$$

Из этого определения следует, что $0 < C_C(i) \leq 1$, так как минимальное расстояние $d(i,j)$ для всех $j \neq i$ равно 1, то минимальное значение суммы $\sum_{j \in I} d(i,j)$ равно $n-1$.

Чтобы избежать бесконечности в знаменателе из i должны быть достижимы все вершины. Поэтому для определения близкой центральности для всех акторов неориентированный граф должен быть связным, а ориентированный – сильно связным.

Срединная центральность (Betweenness Centrality)

Эта величина определяет центральность, учитывая, как часто данный актор появляется на путях взаимодействия других акторов. Если актор i появляется на всех или почти всех путях, связывающих акторы j и k, то он может "контролировать" их взаимодействие. Срединная центральность позволяет выделять акторы, без прохождения через которые нельзя связать многие пары других акторов.

Неориентированные графы. Для акторов j и k пусть p_{jk} обозначает число кратчайших путей между j и k. Пусть $i \neq j$ и $i \neq k$ – это третий актор. Обозначим через $p_{jk}(i)$ число кратчайших путей между j и k, проходящих через i.

Тогда срединная центральность актора i, обозначаемая $C_B(i)$, определяется как

$$C_B(i) = \sum_{j \in I, j \neq i} \sum_{k \in I, k \neq j, k \neq i} \frac{p_{jk}(i)}{p_{jk}}.$$

$C_B(i)$ может принимать значения от 0 до $\frac{(n-1)(n-2)}{2}$ (это число пар акторов, не содержащих i).

$C_B(i)$ можно нормализовать:

$$C_B(i) = \frac{2 \sum_{j \in I, j \neq i} \sum_{k \in I, k \neq j, k \neq i} \frac{p_{jk}(i)}{p_{jk}}}{(n-1)(n-2)}.$$

Ориентированные графы. В этом случае срединную центральность определяют как

$$C_B(i) = \frac{\sum_{j \in I, j \neq i} \sum_{k \in I, k \neq j, k \neq i} \frac{p_{jk}(i)}{p_{jk}}}{(n-1)(n-2)}.$$

(для ориентированного графа (x, y) и (y, x) это разные ребра).

7.2.2 Престижность

Содержательно, некоторый актор имеет высокий престиж, если на него направлены связи от многих других акторов. Основное отличие престижности от центральности заключается в том, что в определении центральности участвуют *выходящие из вершины* ребра и расстояния от данной вершины до других, а престижность зависит от *входящих в вершину* ребер и расстояниях от других вершин до нее.

Престижность определяется только для ориентированных графов.

Степень престижности $P_D(i)$ актора i в социальной сети определяется как: $P_D(i) = \frac{d_i(i)}{n-1}$.

Эта величина принимает значения от 0 до 1. $P_D(i) = 1$ означает, что все акторы сети непосредственно связаны с i.

Средняя престижность (Proximity Prestige)

Эта мера обобщает степень престижности, т.к. учитывает не только соседей актора i, но и всех других акторов, из которых есть пути в i. .

Область влияния актора i, обозначаемая I_i, это множество акторов, из которых можно достичь i в социальной сети G.

Среднее расстояние от акторов из I_i до i определяется как

$$dm(I_i, i) = \frac{\sum_{j \in I_i} d(j, i)}{|I_i|}.$$

Средняя престижность актора i, обозначаемая $P_P(i)$, определяется как

$$P_P(i) = \frac{|I_i|}{(n-1) dm(I_i, i)} = \frac{|I_i|^2}{(n-1) \sum_{j \in I_i} d(j, i)}.$$

Здесь, $\frac{|I_i|}{n-1}$ это доля акторов из I, которые могут достичь i.

$P_P(i)$ находится в интервале от 0 до 1.

Ранжированный престиж (Rank Prestige)

Во всех предыдущих мерах считалось, что на престиж данного актора одинаково влияют все ссылающиеся на него акторы. В реальных сетях ссылки более авторитетных акторов должны иметь больший вес, чем менее авторитетных.

Ранжированный престиж позволяет оценить престиж актора через престиж его соседей. Ранжированный престиж $P_R(i)$ актора i определяется как

$$P_R(i) = \sum_{j \in I} A_{ji} \cdot P_R(j).$$

здесь A_{ij} – это элементы матрицы смежности графа G: $A_{ij} = 1$, если $(i, j) \in E$, и $A_{ij} = 0$ иначе.

Заметим, что значение $P_R(i)$ определено **рекурсивно**.

Пусть $\mathbf{P} = (P_R(i_1), \ldots, P_R(i_n))$ – вектор-столбец ранжированных престижей всех акторов. Тогда

$$\mathbf{P} = A^T \mathbf{P}.$$

Решениями этой системы уравнений являются **собственные векторы P** матрицы A^T.

Ранжированный престиж используется в алгоритмах **PageRank** и **HITS**, которые будут рассмотрены ниже.

7.3 Ранжирование интернет-страниц. Алгоритм PageRank

Интернет можно рассматривать как очень большую социальную сеть. *Хороший метод поиска в интернете* должен соединять поиск страниц, которые содержат все или почти все термины из запроса с **устойчивым ранжированием**, которое *передвигает важные высококачественные и надежные страницы* к началу списка ответов. Выше мы определили **престижность** как меру важности интернет-страниц.

PageRank. PageRank это процедура вычисления **престижа** каждой страницы в связанном множестве страниц. Она была впервые предложена основателями компании Google Л. Пейджем (L. Page) и С. Брином (S. Brin) в 1998 г. [20]. Алгоритм PageRank основан на двух содержательных соображениях:
1) ссылка на некоторую страницу является признаком авторитетности или престижности этой страницы. Поэтому страница является важной, если у нее много входящих ребер;
2) ссылка из более авторитетной страницы важнее ссылки из менее авторитетной страницы. Поэтому чем более престижные страницы ссылаются на данную страницу, тем выше ее собственный престиж.

Существует достаточно много определений и вариантов вычисления PageRank-рейтингов страниц. Здесь мы приведем самое простое определение.

Интернет как граф. Мы рассматриваем World Wide Web как граф, вершинами которого являются **отдельные web-страницы** (urls) , а **гипертекстовые ссылки** между страницами играют роль ребер.

Более формально, рассмотрим ориентированный граф $G_{WWW} = (V, E)$. множество V его вершин — это список web-страниц (urls). Ребро $(v, w) \in E$ означает, что в теле страницы v имеется *тег* `<a href="URL>`, где `URL` это URL страницы w.

Для страницы $i \in V$ через $I(i)$ обозначим множество всех входящих в нее ребер, т.е. таких $e \in E$, что $e = (v, i)$ для некоторой $v \in V$, а через $O(i)$ – множество всех выходящих из i ребер, т.е. таких $e \in E$, что $e = (i, v)$ для некоторой $v \in V$.

Замечание: часто в I(i) и O(i) учитываются только входящие и выходящие ребра из страниц на других сайтах.

Интернет-сёрфинг. PageRank моделирует поведение пользователя интернет, в одном окне браузера. В частности, PageRank моделирует следующие действия:

Обход PageRank:

1. Пользователь начинает обход web-страниц с некоторой *случайно выбранной страницы* из V[a].
2. На каждом шаге пользователь обозревает некоторую страницу i. *С вероятностью* $d \in (0, 1)$ он выбирает переход по ссылкам, расположенным на этой странице (в предположении, что хоть одна такая ссылка имеется).
3. Любая из ссылок на странице i *может быть выбрана с равной вероятностью.*
4. *С вероятностью* $1-d$ пользователь, устав от последовательного обхода страниц, перепрыгивает сразу на *случайно выбранную страницу из* V.
5. Если из текущей страницы нет ссылок на другие страницы, то пользователь просто переходит на *случайно выбранную страницу из* V.

[a] на самом деле, PageRank позволяет ослабить это условие и стартовать с некоторой страницы случайно выбранной из некоторого небольшого множества страниц.

Что вычисляет PageRank. Рейтинг, который **PageRank** присваивает странице $i \in V$ это *стационарная вероятность посещения этой страницы* в описанном выше бесконечном процессе обхода [20] или, иными словами, доля времени, которую пользователь будет проводить на этой странице.

Вывод PageRank

Пусть $p(i)$ это *вероятность посещения web-страницы* i (т.е., рейтинг PageRank для страницы i). Пусть $I(i) = \{j_1, \dots j_s\}$ – множество всех страниц, которые ссылаются **на** i. Пусть $p(j_1), \dots, p(j_s)$ это вероятности достижения каждой из этих страниц.

Предположение: Из каждой страницы в V выходит хоть одно ребро.

- Предположим, что мы достигли страницу j_1. Тогда с вероятностью d мы выберем

движение по ссылкам. Так как число разных ссылок с j_1 равно $|O(j_1)|$, то *с вероятностью*

$$p(i|j_1, \text{следуем по ссылкам}) = \frac{1}{|O(j_1)|}$$

мы можем достичь страницу i. Так как $p(\text{следуем по ссылкам}) = d$, то:

$$p(i|j_1) = d \cdot \frac{1}{|O(j_1)|}.$$

- Аналогичные рассуждения справедливы и для остальных страниц $j \in I(i)$. Тогда для $k = 1, \ldots, s$ имеем

$$p(i|j_k) = d \cdot \frac{1}{|O(j_k)|}.$$

- Страницу i можно достичь одним из двух способов:

  1. следуя по некоторой ссылке на нее со страниц $j_1, \ldots, j_s$;
  2. случайно выбирая прыжок на i с любой текущей страницы.

- Отсюда получаем следующую формулу для вычисления вероятности $p(i)$:

$$p(i) = (1 - d) \cdot \frac{1}{|V|} + (p(i|j_1) \cdot p(j_1) + \ldots + p(i|j_s) \cdot p(j_s)).$$

  На рис. 7.1 проиллюстрировано вычисление этих вероятностей. Подставляя сюда $p(i|j_k)$, получаем:

$$p(i) = (1 - d) \cdot \frac{1}{|V|} + d \cdot \sum_{k=1}^{s} \frac{1}{|O_{j_k}|} \cdot p(j_k) \qquad (*)$$

Отметим, что это *рекурсивное определение.*

В матричном виде выведеное соотношение можно записать как

$$\mathbf{P} = ((1 - d)\frac{\mathbf{ONES}}{|V|} + d\mathbf{A}^T)\mathbf{P},$$

где $\mathbf{P}$ – вектор-столбец искомых вероятностей $p(i)$, $\mathbf{ONES}$ – квадратная матрица размера $|V| \times |V|$, все элементы которой равны 1, а $\mathbf{A}^T$ – это транспонированный вариант матрицы $\mathbf{A}$ цепи Маркова с состояниями V и вероятностями переходов из i в j, равными $A_{ij} = \frac{1}{|O_i|}$, при $(i, j) \in E$, и $A_{ij} = 0$, если ребра (i, j) нет в графе. Таким образом, $\mathbf{P}$ – это собственный вектор матрицы

$$\mathbf{M} = (1 - d)\frac{\mathbf{ONES}}{|V|} + d\mathbf{A}^T,$$

соответствующий ее главному собственному значению 1.

Параметр d называется *коэффициентом затухания (demping factor)* и может принимать значения между 0 и 1. В [20] использовалось $d = 0.85$.

Вычисление рейтинга PageRank

Из формулы (*) следует, что для вычисления PageRank для некоторой страницы нам требуется знать рейтинги PageRank для всех ее "предков". Стандартный способ организовать такое вычисление состоит в последовательном итерировании.

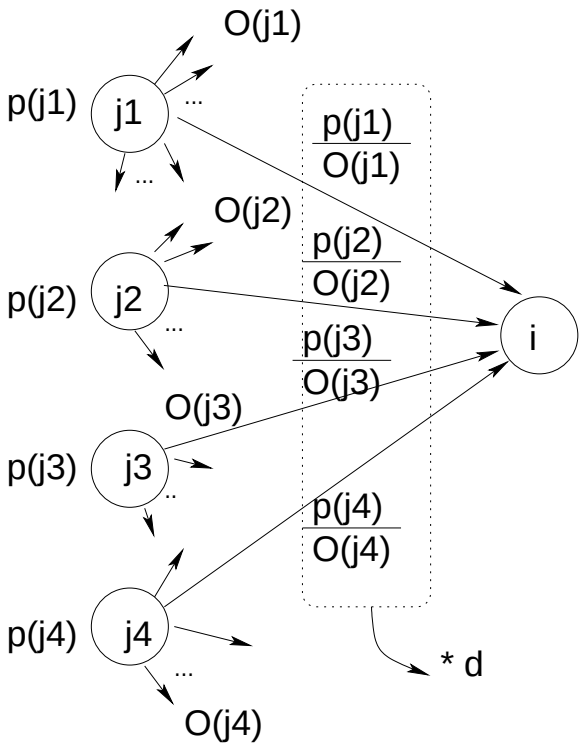

Рис. 7.1: Вычисление вероятности достичь web-страницу.

Итеративное вычисление PageRank. Традиционный итеративный алгоритм для вычисления PageRank использует следующую процедуру:

$$pageRank^0(i) = \frac{1}{|V|} \qquad \text{для всех } i \in V \tag{7.1}$$

$$pageRank^r(i) = (1-d) \cdot \frac{1}{|V|} + d \cdot \sum_{k=1}^{s} \frac{1}{|O_{j_k}|} \cdot pageRank^{r-1}(j_k) \tag{7.2}$$

$$\textbf{Остановиться, когда } \left(\sum_{i \in V} (pageRank^r(i) - pageRank^{r-1}(i))^2 \right) < \varepsilon \tag{7.3}$$

Доказано, что при некоторых условиях на граф (сильная связность и апериодичность – эти условия обеспечиваются введением прыжков с любой страницы на любую страницу) приведенная процедура сходится к некоторым стационарным значениям рейтингов $pageRank(i)$ (главному собственному вектору соответствующей стохастической матрицы). На самом деле, поскольку нас интересуют относительные рейтинги страниц, для определения их порядка сходимость не требуется и процедуру можно прерывать после небольшого числа итераций. В экспериментах авторов алгоритма ([20]) для графа с 322 миллионами ребер алгоритм PageRank сошелся за 52 итерации.

Рассмотрим сеть G_1, показанную на рис. 7.2. В качестве d зафиксируем значение 0.8.

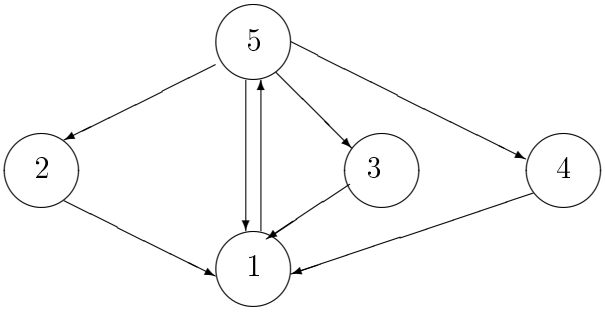

Рис. 7.2: Граф сети G_1

Тогда матрица

$$\mathbf{M} = (1-d)\frac{\mathbf{ONES}}{|V|} + d\mathbf{A}^T = \begin{pmatrix} 0,04 & 0,84 & 0,84 & 0,84 & 0,24 \\ 0,04 & 0,04 & 0,04 & 0,04 & 0,24 \\ 0,04 & 0,04 & 0,04 & 0,04 & 0,24 \\ 0,04 & 0,04 & 0,04 & 0,04 & 0,24 \\ 0,84 & 0,04 & 0,04 & 0,04 & 0,04 \end{pmatrix}$$

Начав вычисление рейтингов с равномерного распределения $\mathbf{P}^0 = (0.2, 0.2,\ 0.2, 0.2, 0.2)^T$, получаем следующую последовательность значений $\mathbf{P}^r$:

$\mathbf{P}^0$	$\mathbf{P}^1$	$\mathbf{P}^2$	$\mathbf{P}^3$	$\mathbf{P}^4$	$\mathbf{P}^5$	$\mathbf{P}^6$	$\mathbf{P}^7$
0,2	0,56	0,272	0,3296	0,42176	0,320384	0,357248	0,37641728
0,2	0,08	0,08	0,1376	0,09152	0,100736	0,1154816	0,09926144
0,2	0,08	0,08	0,1376	0,09152	0,100736	0,1154816	0,09926144
0,2	0,08	0,08	0,1376	0,09152	0,100736	0,1154816	0,09926144
0,2	0,2	0,488	0,2576	0,30368	0,377408	0,2963072	0,3257984

Квадрат расстояния между $\mathbf{P}^6$ и $\mathbf{P}^7$ меньше 0,0021. Поэтому можно использовать для ранжирования страниц ранги, полученные округлением $\mathbf{P}^7$, т.е. $pageRank(1) = 0,38; pageRank(2) = pageRank(3) = pageRank(4) = 0,1; pageRank(5) = 0,32$.

7.4 Поиск по теме, индуцированный гипертекстом (HITS)

Алгоритм **HITS**, (Hypertext Induced Topic Search) это алгоритм информационного поиска, который по заданному запросу возвращает и ранжирует интернет страницы, которые предположительно содержат информацию, относящуюся к заданному запросу. Этот алгоритм был предложен Клейнбергом (Kleinberg) в 1998г [13]. Он заметил, что в научной литературе некоторые публикации (часто это доклады на конференциях) *инициируют новые идеи*, а другие – *консолидируют и обозревают результаты проведенных исследований* (обычно это публикации в журналах или монографии). По аналогии он выделил два вида популярных интернет-страниц: *авторитеты (authorities)*, которые содержат высококачественную информацию, и *хабы (hubs)* или *концентраторы*, которые содержат обширные наборы ссылок на авторитетные страницы. На самом деле каждая страница до некоторой степени

авторитетна, а до некоторой степени является хабом. Алгоритм HITS как раз и пытается определить эти степени.

Он включает три этапа:

1. **Информационный поиск.** Задачу информационного поиска HITS передоверяет другим агоритмам. Предложенный ему запрос q он передает одной известных машин интернет-поиска и получает от нее набор найденных страниц.

2. **Расширение.** HITS добавляет ряд других страниц, собирая и обрабатывая связи страниц, полученных на предыдущем этапе.

3. **Ранжирование.** HITS ранжирует полученное на этапе 2 множество страниц, вычисляя для каждой из них две числовые оценки.

   - Оценка авторитетности является мерой *престижа* страницы;
   - Оценка центральности (Hub score) является мерой *центральности* страницы.

   После этого алгоритм возвращает список страниц, имеющих наибольшие оценки авторитетности и центральности.

Более формально:

1. **Этап 1. Информационный поиск.**

   (a) Получив на вход запрос q, передать его одной известных машин интернет-поиска.

   (b) Получить от нее список W из N (обычно, $N \approx 200$) наиболее релевантных запросу страниц.

   Назовем W **корневым множеством** для запроса q.

2. **Этап 2. Расширение.**

   (a) Для каждой $w \in W$ добавить до k (обычно, $k \approx 50$) страниц, на которые ссылается w.

   (b) Полученное таким образом множество страниц S, состоящее из W и множества всех добавленных страниц, назовем **базовым множеством** для запроса q.

3. **Этап 3. Ранжирование.**

   (a) Построить ориентированный граф $G = (S, E)$, вершинами которого являются страницы базового множества, а множество ребер E содержит все связи между страницами из S. Пусть L – матрица смежности графа G.

   (b) Вычислить для каждой страницы $i \in S$ оценки ее **авторитетности** $a(i)$ и **центральности** $h(i)$, используя следующие рекурсивные соотношения:

$$a(i) = \sum_{(j,i) \in E} h(j)$$

$$h(i) = \sum_{(i,j) \in E} a(j)$$

```
Алгоритм HITS-Iterate(G)
begin
  a_0 = (1, 1, ..., 1);
  h_0 = (1, 1, ..., 1);
  k := 1;
  repeat
    a_k := L^T L a_{k-1};
    h_k := L L^T h_{k-1};
    a_k := a_k / ||a_k||;
    h_k := h_k / ||h_k||;
  until ||a_k - a_{k-1}|| < ε_1  и  ||h_k - h_{k-1}|| < ε_2;
  return a_k, h_k;
end
```

Рис. 7.3: Итеративный вариант этапа ранжирования алгоритма HITS

От рекурсии к итерации.

Оценки авторитетности и центральности взаимно рекурсивны. Пусть $S = \{i_1, \dots i_n\}$. Пусть $\mathbf{a} = (a(i_1), \dots, a(i_n))$ и $\mathbf{h} = (h(i_1), \dots h(i_n)$.

Эти два вектора связаны следующим образом:

$$\mathbf{a} = L^T \mathbf{h}$$

$$\mathbf{h} = L\mathbf{a}$$

Подставляя каждое из этих соотношений в другое, получим следующие простые рекурсивные соотношения:

$$\mathbf{a} = L^T L\mathbf{a}$$

$$\mathbf{h} = LL^T \mathbf{h}$$

Они, в свою очередь, могут быть преобразованы в итеративные процедуры:

$$\mathbf{a_k} = L^T L\mathbf{a_{k-1}}$$

$$\mathbf{h_k} = LL^T \mathbf{h_{k-1}}$$

В качестве начальных значений обычно выбираются $\mathbf{a_0} = \mathbf{h_0} = (1, 1, \dots 1)$.

На рис. 7.3 представлен псевдокод итеративного варианта этапа ранжирования алгоритма HITS, который для обеспечения сходимости на каждом шаге использует нормализацию. Эта нормализация гарантирует, что на каждом шаге

$$\sum_{i \in S} a(i) = 1 \quad и \quad \sum_{i \in S} h(i) = 1.$$

Алгоритм HITS всегда сходится, но для некоторых видов графов различные инициализации могут приводить к различгым авторитетам и хабам, поскольку матрицы $L^T L$ и LL^T могут оказаться приводимыми (иметь кратные главные собственные значения). PageRank для преодоления аналогичной трудности ввел прыжки с с любой страницы на любую страницу. Аналогичный трюк можно применить и в алгоритме HITS.

7.5 Анализ цитируемости

При **анализе цитируемости** рассматриваются графы социальных сетей, в которых вершинами являются *научные работы: доклады, статьи, книги* и *другие источники научной информации*, а связи представляют *ссылки* из одних работ на другие.

7.5.1 Ко-цитируемость и (Co-citation) и связь библиографий (Bibliographic Coupling)

Ко-цитируемость как мера сходства.

Определение. Статья k **ко-цитирует** статьи i и j, если в графе цитирования G имеются ребра (k, i) и (k, j).

Можно предположить, что если две статьи ко-цитируются во многих статьях, то они похожи. Степень сходства статей i и j можно определить с помощью **коэффициента ко-цитируемости** C_{ij}:

$$C_{ij} = \sum_{k \in I} A_{ki} A_{kj}.$$

Связь библиографий (Bibliographic Coupling) Связь библиографий дуальна понятию ко-цитируемости.

Определение. Статьи i и j **библиографически связаны**, если имеется третья статья k, на которую есть ссылки как из i, так и из j, т.е. (i, k) и (j, k) являются связями в G.

Связь библиографий как мера сходства. Чем больше работ статьи i и j совместно цитируют, тем более вероятно, что они похожи друг на друга. Оценить эту схожесть можно с помощью коэффициента библиографической близости B_{ij}:

$$B_{ij} = \sum_{k \in I} A_{ik} A_{jk}.$$

7.6 Обнаружение сообществ

Сообщество. Пусть $S = \{s_1, \ldots, s_n\}$ это множество однотипных объектов. **Сообщество** это пара $C = \langle T, G \rangle$, в которой T это объединяющая данное сообщество **тема**, а $G \subseteq S$ это множество **членов сообщества**.

Свойства. Это *весьма общее определение* **сообществ**. Алгоритмы для решения разных задач используют разные уточнения этого определения.

- Темы. Темы определяют соответствующие сообщества. Можно ожидать, что если для двух сообществ $C_1 = \langle T_1, G_1 \rangle$ и $C_2 = \langle T_2, G_2 \rangle$, $T_1 = T_2$, то также $G_1 = G_2$ и, следовательно, $C_1 = C_2$.

  (Заметим, что обратное неверно. Вполне возможно, что у два разных сообщества образованы из одного и того же множества членов.)

- Темы сообществ могут быть самыми различными. Например, какие-то события, хобби, профессиональные интересы и т.д.
- Каждый объект может быть членом многих сообществ.
- Для некоторых приложений важен также временной аспект.

Задача обнаружения сообществ: для заданного множества данных, содержащего информацию об объектах, выявить (скрытые) сообщества этих объектов. Для каждого сообщества определить его тему и его членов.

Обычно тема представляется набором некоторых ключевых слов.

7.6.1 Двудольное ядро сообществ

(i, j) **двудольное ядро** это **полный двудольный граф** $G = (F, C, E)$, такой что:

- $F, C \subseteq S$
- $F \cap C = \emptyset$,
- $E = \{(f, c) | f \in F, c \in C\}$,
- $|F| = i$,
- $|C| = j$.

Элементы множества F называются фанами (fans), элементы множества C – центрами.

На рис. 7.4 показано (4,5)-двудольное ядро.

Двудольное ядро представляет группу объектов (фанов), которые ссылаются на одно и то же множество центров. Поэтому двудольное ядро можно рассматривать как ядро некоторого сообщества, состоящего из фанов ядра, тема которого находится среди центров ядра.

7.6.2 Выявление двудольных ядер

В общем случае задача определения по графу G и параметрам i и j наличия в G (i, j) двудольного ядра является NP-полной. Поэтому для поиска двудольных ядер используют эвристические процедуры.

Процедура поиска двудольного ядра

Вход: Граф $G_S = (S, E_S)$, $S = \{s_1, \ldots, s_n\}$, $E \subseteq S \times S$. i, j - размер двудольного ядра.

Шаг 1. Сокращение. Множество S уменьшают дважды:

Шаг 1.1. Сокращение по степени захода. Удалить все вершины (pages) со степенью захода, превосходящей некоторую большую константу K. (например, $K = 50$). Причина этого сокращения в том, что страницы, на которые чересчур много ссылок, как правило, принадлежат "универсальным" сайтам типа Yahoo, Rambler и т.п. и не связаны с определенной темой или сообществом.

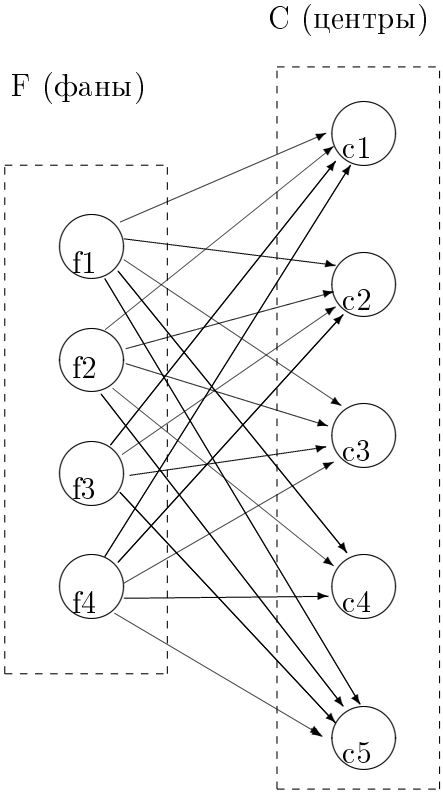

Рис. 7.4: (4,5)-двудольное ядро.

Шаг 1.2. Итеративное сокращение фанов и центров.
$S_0 = S$. Пока $S_i \neq S_{i-1}$:

- Удалить из S_{i-1} все вершины s со степенью исхода $d_o(s) < i$.
- Удалить из S_{i-1} все вершины s со степенью захода $d_i(s) < j$.

Шаг 2. Порождение двудольного ядра.
For $i = 1 \ldots k$ **do**:

// Получение (i, j) двудольные сообществ.

Базис: $(1, j)$ сообщество состоит из одной вершины s со степенью исхода $d_o(s) = j$.

Индукционный шаг: Для каждого центра $(i-1, j)$ сообщества, найти ссылающуюся на него вершину f', не входящую в сообщество. Если f' связана со всеми j центрами сообщества, то добавить f' в двудольное ядро в качестве нового фана.

Комментарий. Вообще говоря, двудольные ядра не определяют полностью сообщества. Скорее они выявляют их "центральную"часть и дают некоторое направление для поиска темы.

7.7 Сообщества максимального потока

Сообщества максимального потока. Пусть $G_S = (S, E_S)$ – граф связей над множеством объектов S. Одно из определений сообщества можно сформулировать следующим образом:
Сообщество это подмножество объектов $C \subset S$ такое, что каждый объект $u \in C$ имеет больше ребер (входящих и исходящих), связывающих его с другими членами C, чем ребер, связывающих его с объектами из $S \setminus C$.

В общем случае, задача выделения таких сообществ является NP-полной. Поэтому в работе [8] было предложено приближенное решение этой задачи, основанное на нахождении максимальных потоков. Обнаруженные таким образом сообщества называются *сообществами максимальных потоков.*

Напомним, что транспортная сеть $N =< G = (V, E), c, s, t >$, состоит из ориентированного графа G, ребра которого имеют пропускные способности $c(e) \geq 0$, источника $s \in V$, в который не входят ребра, и стока $t \in V$, из которого не выходят ребра. *Поток* f *в сети* N - это функция $f : E \to R$ такая, что
1) $0 \leq f(e) \leq c(e)$ для каждого ребра $e \in E$;
2) для каждой вершины $v \in V \setminus \{s, t\}$
$\sum\{f(e) |$ *e входит в* $v\} = \sum\{f(e) |$ *e выходит из* $v\}$.
Величина потока f в сети N $val(f) = \sum_{u \in V} f(s, u)$.

Поток с наибольшим значением величины называется *максимальным.*

Задача определения максимального потока хорошо изучена в теории графов. Имеется несколько алгоритмов, решающих эту задачу за время $O(|V|^3)$ (см., например,[32, 33]). Применение этих алгоритмов для обнаружения сообществ в социальных сетях основано на следующей теореме Форда-Фалкерсона: *величина максимального потока в сети равна величине минимального разреза в ней.* Здесь *разрез* A *в сети* N - это подмножество вершин $A \subset V$ такое, что источник $s \in A$, а сток $t \in V \setminus A$. Разрез A однозначно задает множество ребер $P(A) = \{(u, v) \mid u \in A, v \in V \setminus A\}$, выходящих из A.

Если пропускные способности ребер равны 1, то для максимального потока $fmax$ его величина $val(fmax)$ равна минимальному числу ребер $|P(A)|$, соединяющих некоторый (минимальный) разрез A с остальной частью сети.

Рис. 7.5 иллюстрирует такие разрезы.

Поток между вершинами s и t ограничен двумя ребрами "бутылочного горлышка" $(u3, u4)$ и $(u2, u5)$. Аналогично, поток между вершинами s и w ограничен ребрами $(u1, u6)$ $(u2, u6)$. Если эти ребра отсечь (удалить из сети), то мы получим три различных сообщества.

Алгоритм Find-Community представлен на рис. 7.6. Он работает следующим образом.

- Вход. Социальная сеть G_S и множество исходных объектов (страниц) S^*. Предполагается, что пользователю известно, что исходные объекты принадлежат одному сообществу. Алгоритм будет определять границы этого сообщества.
- Выход. $C \supset S^*$ – список объектов сообщества.
- Процесс. Алгоритм работает в два этапа:
  - Этап 1. Расширение исходных страниц. Алгоритм обходит социальную сеть, начиная с исходных страниц, и собирает некоторую окрестность множества $C = S^*$.

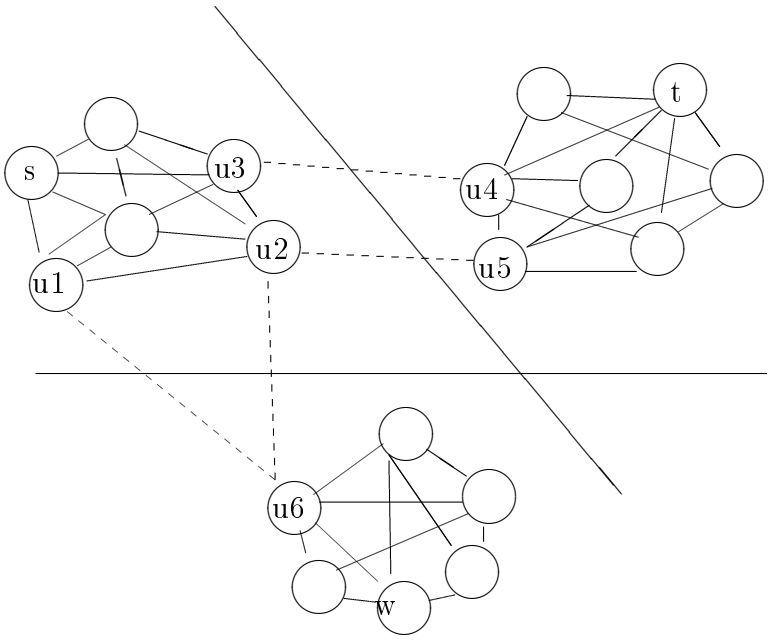

Рис. 7.5: Максимальный поток в сети. Сечения разделяют обнаруженные сообщества

Для этого используется процедура crawlGraph(C), которая добавляет к C все вершины, находящиеся на ограниченном расстоянии от вершин C (при этом обход в ширину происходит без учета направления ребер).

– Этап 2. Максимальный поток.

Для отделения сообщества $C \supset S^*$ от остальной сети применяется алгоритм поиска максимального потока Max-Flow(G, s, t). Предполагается, что этот алгоритм удалит ребра, соединяющие минимальный разрез с остальным графом.

Для нахождения требуемого сообщества эти этапы могут повторяться несколько раз.

Значения параметров K и k выбираются эмпирически. Часто выбирается $k = |S|$

7.8 Задачи

Задача 7.1. *Пусть $G = (V, E)$ – ориентированный граф. Предложите алгоритм для вычисления для вершины $i \in V$ значения ее близкой центральности (Closeness Centrality) $C_C(i)$ по формуле*

$$C_C(i) = \frac{n-1}{\sum_{j \in I} d(i,j)}.$$

Оцените сложность предложенного алгоритма.

Задача 7.2. *Пусть $G = (V, E)$ – неориентированный граф. Предложите алгоритм для вычисления для вершины $i \in V$ значения ее срединной центральности (Betweenness Centrality) $C_B(i)$, определяемой по формуле*

$$C_B(i) = \sum_{j \in I, j \neq i} \sum_{k \in I, k \neq j, k \neq i} \frac{p_{jk}(i)}{p_{jk}},$$

Алгоритм Find-Community(S^*, K - число итераций)

begin
  $C := S^*$;
  for $it = 1$ **to** K **do**
    $G :=$ crawlGraph(C);
    $n := |S^*|$;
    $C^* :=$ Max-Flow-Community(G,C,n);
    ранжировать все $v \in C^*$ по числу ребер внутри C^*.
    $C := C \cup \{v \in C^* | v$ *имеет высокий ранг в* $C^*\}$
  end while
  return C;
end

Function Max-Flow-Community(G= (V,E), C, k)

begin
  добавить новые вершины s и t в V;
  for all $v \in V$ **do** включить (s, v) в E с $c(s, v) = \infty$;
  for all $(u, v) \in E$, $u \neq s$ **do**
    $c(u, v) = k$;
    if $(v, u) \notin E$ **then**
      включить (v, u) в E с $c(v, u) = k$;
  end for
  for all $v \in V$, $v \notin C \cup \{s, t\}$ **do** включить (v,t) в E с $c(v, t) = 1$
  Max-Flow(G,s,t);
  return $\{v \in V \mid v$ *достижима из* $s\}$
end

Рис. 7.6: Алгоритм поиска сообщества максимального потока

где p_{jk} обозначает число кратчайших путей между j и k, а $p_{jk}(i)$ это число кратчайших путей между j и k, проходящих через i. Оцените сложность предложенного алгоритма.

Задача 7.3. *Пусть $G = (V, E)$ – ориентированный граф. Предложите алгоритм для вычисления для вершины $i \in V$ значения ее срединной центральности (Betweenness Centrality) $C_B(i)$, определяемой по формуле*

$$C_B(i) = \frac{\sum_{j \in I, j \neq i} \sum_{k \in I, k \neq j, k \neq i} \frac{p_{jk}(i)}{p_{jk}}}{(n-1)(n-2)},$$

где $n = |V|$, а p_{jk} обозначает число кратчайших путей между j и k, а $p_{jk}(i)$ это число кратчайших путей между j и k, проходящих через i. Оцените сложность предложенного алгоритма.

Задача 7.4. *Пусть $G = (V, E)$ – ориентированный граф. Предложите алгоритм для вычисления для вершины $i \in V$ значения ее средней престижности (Proximity Prestige) $P_P(i)$,*

определяемой по формуле

$$P_P(i) = \frac{|I_i|^2}{(n-1)\sum_{j \in I_i} d(j,i)},$$

где I_i *это множество вершин, из которых можно достичь* i *в* G, $n = |V|$. *Оцените сложность предложенного алгоритма.*

Задача 7.5. *Какие ранги присвоит алгоритм PageRank вершинам графа* $G = (V, E)$*:*
$V = \{a, b, c, d\}$
$E = \{(a, b), (b, a), (b, c), (c, a), (c, d), (d, a), (d, c)\}$.
Поскольку из каждой вершины выходят ребра, положим параметр $d = 1$.

Задача 7.6. *Какие ранги присвоит алгоритм PageRank вершинам графа* $G = (V, E)$*:*
$V = \{a, b, c, d, e\}$
$E = \{(a, b)(b, a), (b, c), (c, a), (c, d), (d, a), (d, e), (e, a), (e, b)\}$.
Поскольку из каждой вершины выходят ребра, положим параметр $d = 1$.

Задача 7.7. *Какие ранги присвоит алгоритм PageRank вершинам графа* $G = (V, E)$*:*
$V = \{a, b, c, d, e\}$
$E = \{(a, b)(b, a), (c, a), (c, b), (d, a), (d, b), (e, a), (e, b)\}$.
Поскольку из каждой вершины выходят ребра, положим параметр $d = 1$.

Литература

[1] G. Adomavicius, A. Tuzhilin. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions, *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, No. 6, 2005.

[2] Agrawal R., Gehrke J., Gunopulos D., and Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications. *In Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, Seattle, WA, 1998, 94–105.

[3] Agrawal R, Sricant R. Fast algorithms for mining association rules. *Proc. 1994 Int. Conf. Very Large Databases (VLDB'94)* Santiago, Chile, 1994, 487-499.

[4] Breiman, L., Friedman, J.H., Olshen, R., and Stone, C.J., Classification and Regression Tree. Wadsworth & Brooks/Cole Advanced Books & Software, Pacific California, 1984.

[5] Ceglar A., Roddick J.F. Association mining // ACM Comp. Surveys, Vol. 38, No. 2, 2006.

[6] El-Hajj M., Zaiane O. R. Non recursive generation of frequent k-itemsets from frequent pattern tree representations. *In Proc. of 5th International Conference on Data Warehousing and Knowledge Discovery (DaWak'2003)*, 2003, 371–380.

[7] Ester M., Kreigel H.-P-, Sander J., Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise. *In Proc. of the 2-nd Int.Conf. KDDM*, Portland: AAAI Press, 1996, 226-231.

[8] FlakeG. W., Lawrence S., Giles C. L., Coetzee F. Self-Organization of the Web and Identification of Communities. IEEE Computer 35(3),2002, 66–71.

[9] E. Forgey. Cluster Analysis of Multivariate Data: Efficiency vs. Interpretability of Calssification. *Biometrics*, vol. 21, 1965.

[10] Y. Freund, R.E. Shapire. Experiments with a New Boosting Algorithm. In *Proceedings, 13th International Conference on Machine Learning (ICML'96)*,1996, 148–156.

[11] Han, J., Kamber, M. Data Mining: Concepts and Techniques. 2-nd edition. Elsevier Inc., 2006.

[12] Han, J., Pei, J., and Yin, Y. Mining frequent patterns without candidate generation. In Proc. 2000 ACMSIGMOD Int. Conf. Management of Data (SIGMOD'00), Dallas, TX, 2000, 1–12.

[13] Kleinberg J. M. Authoritative sources in a hyperlinked environment. In Proc. of ACM-SIAM Symposium on Discrete Algorithms, 1998.

[14] W. Li, J. Han, and J. Pei. CMAR: Accurate and efficient classification based on multiple class-association rules. In Proc. 2001 Int. Conf. Data Mining (ICDM'01), San Jose, CA, Nov. 2001. 369–376.

[15] R. Lienhart, A. Kuranov, V. Pisarevsky. Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection. MRL Technical Report, 2002.

[16] B. Liu, W. Hsu, and Y.Ma. Integrating classification and association rule mining. In Proc. 1998 Int. Conf. Knowledge Discovery and DataMining (KDD'98), New York, 1998, 80–86.

[17] S.P. Lloyd. Least squares quantization in PCM. *Unpublished Bell Labs Tech. Note.* (1957). *IEEE Trans. on Information Theory*, vol. IT-28, 1982, pp. 129—137.

[18] H. Mannila, H.Toivonen, A.I. Verkamo. Efficient algorithm for discovering association rules. *Proc. AAAI'94 Workshop Knowledge Discovery in Databases(KDD'94)*, Seatle, WA, 1994, 181-192.

[19] Manning C. D., Raghavan P., Schutze H. Introduction to information retrieval, Cambridge University Press,N.Y., 2008.

[20] Page, Lawrence; Brin, Sergey; Motwani, Rajeev and Winograd, Terry (1998). The PageRank citation ranking: Bringing order to the Web. *Technical Report, Department of Computer Science, Stanford University.* http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf

[21] M.F. Porter, 1980, An algorithm for suffix stripping, Program, 14(3) pp 130–137.

[22] J.R. Quinlan. *C4.5: Program for Machine Learning*, Morgan Kaufman, 1992.

[23] Scott, J. Social network analysis: A handbook (2nd ed.). SAGE publications, 2000.

[24] Social Network Data Analytics. Ed. C.C.Aggarwal. Springer, 2011.

[25] Su X., Khoshgoftaar T.M. A survey of collaborative filtering techniques. Advances in Artificial Intelligence, Volume 2009, Article ID 421425.

[26] The top 10 algorithms in data mining. (Eds. Wu X., Kumar V.), – N.Y.: CRC Press, 2009.

[27] Wu X., Kumar V., et. al. Top 10 algorithms in data mining. Knowl. Inf. Syst., 14, 2008, 1-37.

[28] Xu G., Zhang Y., Li L. Web Mining and Social Networking. Techniques and Applications/ Springer Science+Business Media, 2011.

[29] Барсегян А.А., Купрянов М.С., Степаненко В.В., Холод И.И. Методы и модели анализа данных: OLAP и Data Mining, СПб: "БХВ-Петербург 2004.

[30] Дюк В., Самойленко А., Data Mining: учебный курс. СПб:Питер, 2001.

[31] Зализняк. А.А. Грамматический словарь русского языка. Словоизменение. М.: Изд. "Русский язык 1977.

[32] Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы(построение и анализ).М.: МЦНО, 1999.

[33] Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация. Алгоритмы и сложность. М.:Мир, 1985.

[34] Чубукова И.А. Data Mining. БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий - ИНТУИТ.ру, 2008.

УДК 681.3.06 (075)
ББК 32.973.26 - 018

Учебное издание

Дехтярь Александр Михайлович, Дехтярь Михаил Иосифович

Алгоритмы извлечения знаний из данных

Учебное пособие

Подписано в печать 01.10.2012. Уч.-изд.л.8,75. Электронное изд. Заказ 487.
Тверской государственный университет
Факультет прикладной математики и кибернетики
Адрес: 170000, г.Тверь, пер.Садовый, 35.