Содержание

Лекция 1 - Введение в Data Mining

Определение Data Mining

Data Mining - это процесс обнаружения в сырых данных ранее неизвестных, нетривиальных, практически полезных и доступных интерпретации знаний, необходимых для принятия решений в различных сферах человеческой деятельности. Суть и цель технологии Data Mining можно охарактеризовать так: это технология, которая предназначена для поиска в больших объемах данных неочевидных, объективных и полезных на практике закономерностей. Неочевидных - это значит, что найденные закономерности не обнаруживаются стандартными методами обработки информации или экспертным путем. Объективных - это значит, что обнаруженные закономерности будут полностью соответствовать действительности, в отличие от экспертного мнения, которое всегда является субъективным. Практически полезных - это значит, что выводы имеют конкретное значение, которому можно найти практическое применение. (Григорий Пиатецкий-Шапиро)

Традиционные методы анализа данных (статистические методы) и OLAP в основном ориентированы на проверку заранее сформулированных гипотез (verification-driven data mining) и на "грубый" разведочный анализ, составляющий основу оперативной аналитической обработки данных (OnLine Analytical Processing, OLAP), в то время как одно из основных положений Data Mining - поиск неочевидных закономерностей. Инструменты Data Mining могут находить такие закономерности самостоятельно и также самостоятельно строить гипотезы о взаимосвязях. Поскольку именно формулировка гипотезы относительно зависимостей является самой сложной задачей, преимущество Data Mining по сравнению с другими методами анализа является очевидным.

Основные понятия

Родовое и видовое понятия - делимое понятие - это *родовое*, а его члены деления - это *виды* данного рода, несовместимые между собой, т.е. не пересекающиеся по своему объему (не имеющие общих элементов). **Приведем примеры деления понятий:** В зависимости от источника

энергии электростанции(*род*) делят на(*виды*) гидроэлектростанции, гелиоэлектростанции, геотермальные, ветровые и тепловые (к разновидностям тепловых относят АЭС).

Данные - это необработанный материал, предоставляемый поставщиками данных и используемый потребителями для формирования информации на основе данных.

Объект описывается как набор атрибутов. Объект также известен как запись, случай, пример, строка таблицы и т.д.

Атрибут - свойство, характеризующее объект. Например: цвет глаз человека, температура воды и т.д.Атрибут также называют переменной, полем таблицы, измерением, характеристикой.

Генеральная совокупность (population) - вся совокупность изучаемых объектов, интересующая исследователя.

Выборка (sample) - часть генеральной совокупности, определенным способом отобранная с целью исследования и получения выводов о свойствах и характеристиках генеральной совокупности.

Параметры - числовые характеристики генеральной совокупности.

Статистики - числовые характеристики выборки.

Гипотеза - частично обоснованная закономерность знаний, служащая либо для связи между различными эмпирическими фактами, либо для объяснения факта или группы фактов. Пример гипотезы: между показателями продолжительности жизни и качеством питания есть связь. В этом случае целью исследования может быть объяснение изменений конкретной переменной, в данном случае - продолжительности жизни. Допустим, существует гипотеза, что **зависимая переменная** (продолжительность жизни) изменяется в зависимости от некоторых причин (качество питания, образ жизни, место проживания и т.д.), которые и являются **независимыми переменными**.

Однако переменная изначально не является зависимой или независимой. Она становится таковой после формулировки

конкретной гипотезы. Зависимая переменная в одной гипотезе может быть независимой в другой.

Измерение - процесс присвоения чисел характеристикам изучаемых объектов согласно определенному правилу.

В процессе подготовки данных измеряется не сам объект, а его характеристики.

Шкала - правило, в соответствии с которым объектам присваиваются числа. Существует пять типов шкал измерений: номинальная, порядковая, интервальная, относительная и дихотомическая.

- **Номинальная шкала** (nominal scale) - шкала, содержащая только категории; данные в ней не могут упорядочиваться, с ними не могут быть произведены никакие арифметические действия. Номинальная шкала состоит из названий, категорий, имен для классификации и сортировки объектов или наблюдений по некоторому признаку. Пример такой шкалы: профессии, город проживания, семейное положение. Для этой шкалы применимы только такие операции: равно (=), не равно ().

- **Порядковая шкала** (ordinal scale) - шкала, в которой числа присваивают объектам для обозначения относительной позиции объектов, но не величины различий между ними.Шкала измерений дает возможность ранжировать значения переменных. Измерения же в порядковой шкале содержат информацию только о порядке следования величин, но не позволяют сказать "насколько одна величина больше другой", или "насколько она меньше другой".Пример такой шкалы: место (1, 2, 3-е), которое команда получила на соревнованиях, номер студента в рейтинге успеваемости (1-й, 23-й, и т.д.), при этом неизвестно, насколько один студент успешней другого, известен лишь его номер в рейтинге. Для этой шкалы **применимы** только такие операции: равно (=), не равно (), больше (>), меньше (<).

- **Интервальная шкала** (interval scale) - шкала, разности между значениями которой могут быть вычислены, однако их отношения не имеют смысла.Эта шкала позволяет

находить разницу между двумя величинами, обладает свойствами номинальной и порядковой шкал, а также позволяет определить количественное изменение признака.Пример такой шкалы: температура воды в море утром - 19 градусов, вечером - 24, т.е. вечерняя на 5 градусов выше, но нельзя сказать, что она в 1,26 раз выше. Номинальная и порядковая шкалы являются дискретными, а интервальная шкала - непрерывной, она позволяет осуществлять точные измерения признака и производить арифметические операции сложения, вычитания. Для этой шкалы применимы только такие операции: равно (=), не равно (), больше (>), меньше (<), операции сложения (+) и вычитания (-).

- **Относительная шкала** (ratio scale) - шкала, в которой есть определенная точка отсчета и возможны отношения между значениями шкалы. Пример такой шкалы: вес новорожденного ребенка (4 кг и 3 кг). Первый в 1,33 раза тяжелее. Цена на картофель в супермаркете выше в 1,2 раза, чем цена на базаре. Относительные и интервальные шкалы являются числовыми. Для этой шкалы применимы только такие операции: равно (=), не равно (), больше (>), меньше (<), операции сложения (+) и вычитания (-), умножения (*) и деления (/).

- **Дихотомическая шкала** (dichotomous scale) - шкала, содержащая только две категории. Пример такой шкалы: пол (мужской и женский).

- **Переменные данные** - это такие данные, которые изменяют свои значения в процессе решения задачи.

- **Постоянные данные** - это такие данные, которые сохраняют свои значения в процессе решения задачи (математические константы, координаты неподвижных объектов) и не зависят от внешних факторов.

- **Условно-постоянные данные** - это такие данные, которые могут иногда изменять свои значения, но эти изменения не зависят от процесса решения задачи, а определяются внешними факторами.

- **Данные за период** характеризуют некоторый период времени. Примером данных за период могут быть: прибыль предприятия за месяц, средняя температура за месяц.

- **Точечные данные** представляют значение некоторой переменной в конкретный момент времени. Пример точечных данных: остаток на счете на первое число месяца, температура в восемь часов утра.

Задачи анализа данных

Классификация (Classification) Наиболее простая и распространенная задача Data Mining. В результате решения задачи классификации обнаруживаются признаки, которые характеризуют группы объектов исследуемого набора данных - классы; по этим признакам новый объект можно отнести к тому или иному классу. Методы решения. Для решения задачи классификации могут использоваться методы: ближайшего соседа (Nearest Neighbor); k-ближайшего соседа (k-Nearest Neighbor); байесовские сети (Bayesian Networks); индукция деревьев решений; нейронные сети (neural networks).

Кластеризация (Clustering) Кластеризация является логическим продолжением идеи классификации. Это задача более сложная, особенность кластеризации заключается в том, что классы объектов изначально не предопределены. Результатом кластеризации является разбиение объектов на группы. Пример метода решения задачи кластеризации: обучение "без учителя" особого вида нейронных сетей - самоорганизующихся карт Кохонена.

Ассоциация (Associations) В ходе решения задачи поиска ассоциативных правил отыскиваются закономерности между связанными событиями в наборе данных. Отличие ассоциации от двух предыдущих задач Data Mining: поиск закономерностей осуществляется не на основе свойств анализируемого объекта, а между несколькими событиями, которые происходят одновременно. Наиболее известный алгоритм решения задачи поиска ассоциативных правил - алгоритм Apriori.

Последовательность (Sequence), или последовательная ассоциация (sequential association) Последовательность позволяет найти временные закономерности между

транзакциями. Задача последовательности подобна ассоциации, но ее целью является установление закономерностей не между одновременно наступающими событиями, а между событиями, связанными во времени (т.е. происходящими с некоторым определенным интервалом во времени). Другими словами, последовательность определяется высокой вероятностью цепочки связанных во времени событий. Фактически, ассоциация является частным случаем последовательности с временным лагом, равным нулю. Эту задачу Data Mining также называют задачей нахождения последовательных шаблонов (sequential pattern). Правило последовательности: после события X через определенное время произойдет событие Y. Пример. После покупки квартиры жильцы в 60% случаев в течение двух недель приобретают холодильник, а в течение двух месяцев в 50% случаев приобретается телевизор. Решение данной задачи широко применяется в маркетинге и менеджменте, например, при управлении циклом работы с клиентом (Customer Lifecycle Management).

Прогнозирование (Forecasting) В результате решения задачи прогнозирования на основе особенностей исторических данных оцениваются пропущенные или же будущие значения целевых численных показателей. Для решения таких задач широко применяются методы математической статистики, нейронные сети и др.

Определение отклонений или выбросов (Deviation Detection), анализ отклонений или выбросов Цель решения данной задачи - обнаружение и анализ данных, наиболее отличающихся от общего множества данных, выявление так называемых нехарактерных шаблонов.

Оценивание (Estimation) Задача оценивания сводится к предсказанию непрерывных значений признака.

Анализ связей (Link Analysis) - задача нахождения зависимостей в наборе данных.

Визуализация (Visualization, Graph Mining) В результате визуализации создается графический образ анализируемых данных. Для решения задачи визуализации используются графические методы, показывающие наличие

закономерностей в данных. Пример методов визуализации - представление данных в 2-D и 3-D измерениях.

Подведение итогов (Summarization) - задача, цель которой - описание конкретных групп объектов из анализируемого набора данных.

Категория обучение с учителем представлена следующими задачами Data Mining: классификация, оценка, прогнозирование.

Категория обучение без учителя представлена задачей кластеризации.

В категорию другие входят задачи, не включенные в предыдущие две стратегии.

- методы и модели Data Mining;
- практическое применение Data Mining;
- Средства Data Mining. Weka.

Лекция 2 - Классификация

Постановка задачи и представление результатов

Классификация - системное распределение изучаемых предметов, явлений, процессов по родам, видам, типам, по каким-либо существенным признакам для удобства их исследования; группировка исходных понятий и расположение их в определенном порядке, отражающем степень этого сходства. Под **классификацией** будем понимать отнесение объектов (наблюдений, событий) к одному из заранее известных классов.

Формально: I={i1,...,in}, ii={x1..xn, y} (xi - атрибуты-независимые переменные, y - зависимая).

Классификация требует соблюдения следующих правил:

- в каждом акте деления необходимо применять только одно основание;
- деление должно быть соразмерным, т.е. общий объем видовых понятий должен равняться объему делимого родового понятия;
- члены деления должны взаимно исключать друг друга, их объемы не должны перекрещиваться;
- деление должно быть последовательным.

Различают:

- вспомогательную (искусственную) классификацию, которая производится по *внешнему признаку* и служит для упорядочивания множества предметов (процессов, явлений);
- естественную классификацию, которая производится по существенным признакам, характеризующим внутреннюю общность предметов и явлений. Она является результатом и важным средством научного исследования, так как предполагает и закрепляет результаты изучения закономерностей классифицируемых объектов, а значит .

В зависимости от выбранных признаков, их сочетания и процедуры деления понятий классификация может быть:

- простой - деление родового понятия только по признаку и только один раз до раскрытия всех видов. Примером такой классификации является дихотомия, при которой членами деления бывают только два понятия, каждое из которых является противоречащим другому (т.е. соблюдается принцип: "А и не А");
- сложной - применяется для деления одного понятия по разным основаниям и синтеза этих простых делений в единое целое. Примером такой классификации является периодическая система химических элементов.

Классификация относится к задачам, требующим обучения с учителем. При обучении с учителем набор исходных данных (или выборку данных) разбивают на два множества: обучающее и тестовое. Обучающее множество (training set) - множество, которое включает данные, использующиеся для обучения (конструирования) модели. Тестовое (test set) множество также содержит входные и выходные значения примеров. Здесь выходные значения используются для проверки работоспособности модели.

Процесс классификации состоит из двух этапов: конструирования модели и ее использования.

1. Конструирование модели: описание множества предопределенных классов.

- Каждый пример набора данных относится к одному предопределенному классу.
- На этом этапе используется обучающее множество, на нем происходит конструирование модели.
- Полученная модель может быть представлена *классификационными правилами, деревом решений* или *математической формулой*.

2. Использование модели: классификация новых или неизвестных значений.

- Оценка правильности (точности) модели.

2.1. Известные значения из тестового примера сравниваются с результатами использования полученной модели.

2.2. Уровень точности - процент правильно классифицированных примеров в тестовом множестве.

2.3. Тестовое множество, т.е. множество, на котором тестируется построенная модель, не должно зависеть от обучающего множества.

- Если точность модели допустима, возможно использование модели для классификации новых примеров, класс которых неизвестен.

Для классификации используются различные методы. Основные из них:

- классификация с помощью деревьев решений;
- байесовская (наивная) классификация;
- классификация при помощи искусственных нейронных сетей;
- классификация методом опорных векторов;
- статистические методы, в частности, линейная регрессия;
- классификация при помощи метода ближайшего соседа;
- классификация CBR-методом;
- классификация при помощи генетических алгоритмов.

Оценка точности классификации может проводиться при помощи кросс-проверки. Кросс-проверка (Cross-validation) - это процедура оценки точности классификации на данных из тестового множества, которое также называют кросс-проверочным множеством. Точность классификации тестового множества сравнивается с точностью классификации обучающего множества. Если классификация тестового множества дает приблизительно такие же результаты по точности, как и классификация обучающего множества, считается, что данная модель прошла кросс-проверку. Разделение на обучающее и тестовое множества осуществляется путем деления выборки в определенной пропорции, например обучающее множество - две трети данных и тестовое - одна треть данных.

Метод деревьев решений (decision trees) является одним из наиболее популярных методов решения задач классификации и прогнозирования. Иногда этот метод Data Mining также называют деревьями решающих правил, деревьями

классификации и регрессии. Если зависимая, т.е. целевая переменная принимает дискретные значения, при помощи метода дерева решений решается задача классификации. Если же зависимая переменная принимает непрерывные значения, то дерево решений устанавливает зависимость этой переменной от независимых переменных, т.е. решает задачу численного прогнозирования.

В наиболее простом виде дерево решений - это способ представления правил в иерархической, последовательной структуре. Основа такой структуры - ответы "Да" или "Нет" на ряд вопросов. Корень - исходный вопрос, внутренний узел дерева является узлом проверки определенного условия. Далее идет следующий вопрос и т.д., пока не будет достигнут конечный узел дерева, являющийся узлом решения. Бинарные деревья являются самым простым, частным случаем деревьев решений. В остальных случаях, ответов и, соответственно, ветвей дерева, выходящих из его внутреннего узла, может быть больше двух. На этапе построения модели, собственно, и строится дерево классификации или создается набор неких правил. На этапе использования модели построенное дерево, или путь от его корня к одной из вершин, являющийся набором правил для конкретного клиента, используется для ответа на поставленный вопрос. **Правилом** является логическая конструкция, представленная в виде "если : то :".

Внутренние узлы дерева являются атрибутами базы данных. Эти атрибуты называют прогнозирующими, или атрибутами расщепления (splitting attribute). Конечные узлы дерева, или листы, именуются метками класса, являющимися значениями зависимой категориальной переменной. Каждая ветвь дерева, идущая от внутреннего узла, отмечена предикатом расщепления. Последний может относиться лишь к одному атрибуту расщепления данного узла. Характерная особенность предикатов расщепления: каждая запись использует уникальный путь от корня дерева только к одному узлу-решению. Объединенная информация об атрибутах расщепления и предикатах расщепления в узле называется критерием расщепления (splitting criterion). Качество построенного дерева решения весьма зависит от правильного выбора критерия расщепления.

Классификационная модель, представленная в виде дерева решений, является интуитивной и упрощает понимание решаемой задачи. Деревья решений дают возможность извлекать правила из базы данных на естественном языке. Алгоритм конструирования дерева решений не требует от пользователя выбора входных атрибутов (независимых переменных). На вход алгоритма можно подавать все существующие атрибуты, алгоритм сам выберет наиболее значимые среди них, и только они будут использованы для построения дерева.

В виде формулы: y = a0 + a1*x1 + ... + an*xn, логические и категориальные переменные кодируют числами.

Методы построения правил классификации

Алгоритм построения 1-правил

Пусть у нас есть независимые переменные $A^1...A^j...A^k$, принимающие значения $< x_1^1...x_n^1 >, ... < x_1^j...x_n^j >, ... < x_1^k...x_n^k >$ соответственно, и зависимая переменная C, принимающая значения $c_1...c_r$. Для любого возможного значения каждой независимой переменной формируется правило, которое классифицирует объект из обучающей выборки. В если-части правила указывают значение независимой переменной (Если $A^j = x_i^j$). В то-части правила указывается наиболее часто встречающееся значение зависимой переменной у данного значения независимой переменной(то $C = c_r$). Ошибкой правила является количество объектов, имеющих данное значение рассматриваемой независимой переменной ($A^j = x_i^j$), но не имеющих наиболее часто встречающееся значение зависимой переменной у данного значения независимой переменной($C \neq c_r$). Оценив ошибки, выбирается переменная, для которой ошибка набора минимальна.

В случае непрерывных значений манипулируют промежутками. В случае пропущенных значений - достраивают. Наиболее серьезный недостаток - сверхчувствительность, алгоритм выбирает переменные, стремящиеся к ключу (т.е. с максимальным количеством значений, у ключа ошибка вообще 0, но он не несет

информации). Эффективен, если объекты классифицируются по одному атрибуту.

Метод Naive Bayes

"Наивная" классификация - достаточно прозрачный и понятный метод классификации. "Наивной" она называется потому, что исходит из предположения о взаимной независимости признаков.

Свойства наивной классификации:

1. Использование всех переменных и определение всех зависимостей между ними.

2. Наличие двух предположений относительно переменных:

- все переменные являются одинаково важными;
- все переменные являются статистически независимыми, т.е. значение одной переменной ничего не говорит о значении другой.

Вероятность того, что некий объект i_i, относится к классу $c_r(y = c_r)$ обозначим как $P(y = c_r)$. Событие, соответствующее равенству независимых переменных определенному значению, обозначим как E, а его вероятность - P(E). Идея алгоритма в расчете условной вероятности принадлежности объекта к cr при равенстве его независимых переменных определенным значениям. Из тервера:

$$P(y = c_r|E) = \frac{P(E|y = c_r) * P(y = c_r)}{P(E)}$$

Таким образом формулируются правила, в условных частях которых сравниваются все независимые переменные с соответсввующими возможными значениями. В заключительной части - все возможные значения зависимой переменной: $x_1 = c_1^k, ..., x_n = c_n^k, y = c_r$....{и так для все наборов} Для каждого из этих правил по формуле Байеса определяется его вероятность. Так как независимые переменные независимы друг от друга, то :

$P(E|y = c_r) = P(x_1 = c_1^k|y = c_r) * ... * P(x_n = c_n^k|y = c_r),$ что подставляем в верхную формулу и получаем вероятность всего правила.

Вероятность принадлежности объекта к классу c_r при равенстве его переменной x_n определенному значению c_n^k :

$$P(x_n = c_n^k | y = c_r) = \frac{P(x_n = c_n^k \ \& \ y = c_r)}{P(y = c_r)}$$

Нормализованная вероятность вычисляется по формуле:

$$P'(y = cr|E) = \frac{P(y = c_r|E)}{\sum_{c_r} P(y = c_r|E)}$$

и является вероятностью наступления данного исхода вообще, а не только при E. P(E) просто сокращается.

Проблема: в обучающей выборке может не быть объекта с $x_n = c_n^k$ и при этом принадлежащему к классу c_r. Тогда вероятность равна нулю и соответственно вероятность правила равна нулю. Чтобы этого избежать, к каждой вероятности прибавляют значение, отличное от нуля. Это называется оценочной функцией Лапласа. При подсчете вероятностей тогда эти вероятности пропускаются.

Лекция 3 - Методы построения деревьев решений

Деревья решений - это способ представления классификационных правил в иерархической, последовательной структуре.
Обычно каждый узел включает проверку одной независимой переменной. Иногда в узле дерева две независимые переменные сравниваются друг с другом или определяется некоторая функция от одной или нескольких переменных. Если переменная, которая проверяется в узле, принимает категориальные значения, то каждому возможному значению соответствует ветвь, выходящая из узла дерева. Если значением переменной является число, то проверяется больше или меньше это значение некоторой константы. Иногда область числовых значений разбивают на интервалы. (Проверка попадания значения в один из интервалов).

Листья деревьев соответствуют значениям зависимой переменной, т.е. классам.

Методика "Разделяй и властвуй"

Методика основана на рекурсивном разбиении множества объектов из обучающей выборки на подмножества, содержащие объекты, относящиеся к одинаковым классам. Сперва выбирается независимая переменная, которая помещается в корень дерева.
Из вершины строятся ветви, соответствующие всем возможным значениям выбранной независимой переменной. Множество объектов из обучающей выборки разбивается на несколько подмножеств в соответствии со значением выбранной независимой переменной.
Таким образом, в каждом подмножестве будут находиться объекты, у которых значение выбранной независимой переменной будет одно и то же.
Относительно обучающей выборки T и множества классов C возможны три ситуации:

1. множество T содержит один или более объектов, относящихся к одному классу c_r. Тогда дерево решений для T - это лист, определяющий класс c_r;

2. множество T не содержит ни одного объекта (пустое множество). Тогда это снова лист, и класс, ассоциированный с листом, выбирается из другого множества, отличного от T, например из множества, ассоциированного с родителем;
3. Множество T содержит объекты, относящиеся к разным классам. В этом случае следует разбить множество T на некоторые подмножества. Для этого выбирается одна из независимых переменных x_h, имеющая два и более отличных друг от друга значений $c_h^1, c_h^2 ..., c_h^n$; Множество T разбивается на подмножества $T_1, T_2, ..., T_n$, где каждое подмножество T_i содержит все объекты, у которых значение выбранной зависимой переменной равно c_h^i. Далее процесс продолжается рекурсивно для каждого подмножества до тех пор, пока значение зависимой переменной во вновь образованном подмножестве не будет одинаковым (когда объекты принадлежат одному классу). В этом случае процесс для данной ветви дерева прекращается.

При использовании данной методики построение дерева решений будет происходить сверху вниз. Большинство алгоритмов, которые её используют, являются "жадными алгоритмами". Это значить, что если один раз переменная была выбрана и по ней было произведено разбиение, то алгоритм не может вернуться назад и выбрать другую переменную, которая дала бы лучшее разбиение. Вопрос в том, какую зависимую переменную выбрать для начального разбиения. От этого целиком зависит качество получившегося дерева.
Общее правило для выбора переменной для разбиения: выбранная переменная должны разбить множество так, чтобы получаемые в итоге подмножества состояли из объектов, принадлежащих к одному классу, или были максимально приближены к этому, т.е. чтобы количество объектов из других классов ("примесей") в каждом из этих множеств было минимальным.
Другой проблемой при построении дерева является проблема остановки его разбиения. Методы её решения:

1. Ранняя остановка. Использование статистических методов для оценки целесообразности дальнейшего разбиения. Экономит время обучения модели, но строит менее точные классификационные модели.
2. Ограничение глубины дерева. Нужно остановить дальнейшее построение, если разбиение ведёт к дереву с глубиной, превышающей заданное значение.
3. Разбиение должно быть нетривиальным, т.е. получившиеся в результате узлы должны содержать не менее заданного количества объектов.
4. Отсечение ветвей (снизу вверх). Построить дерево, отсечь или заменить поддеревом те ветви, которые не приведут к возрастанию ошибки. Под ошибкой понимается количество неправильно классифицированных объектов, а точностью дерева решений отношение правильно классифицированных объектов при обучении к общему количеству объектов из обучающего множества.

Построить все возможные варианты разбиения и выбрать наилучший проблематично при наличии большого числа независимых переменных или при большом числе возможных классов.

Алгоритм ID3

Рассмотрим критерий выбора независимой переменной, от которой будет строиться дерево.
Полный набор вариантов разбиения |X| - количество независимых переменных.
Рассмотрим проверку переменой x_h, которая принимает m значений $c_h1, c_h2, \ldots, c_hm$.
Тогда разбиение множества всех объектов обучающей выборке N по проверке переменной x_h даст подмножества $T_1, T_2, \ldots, T_m$.

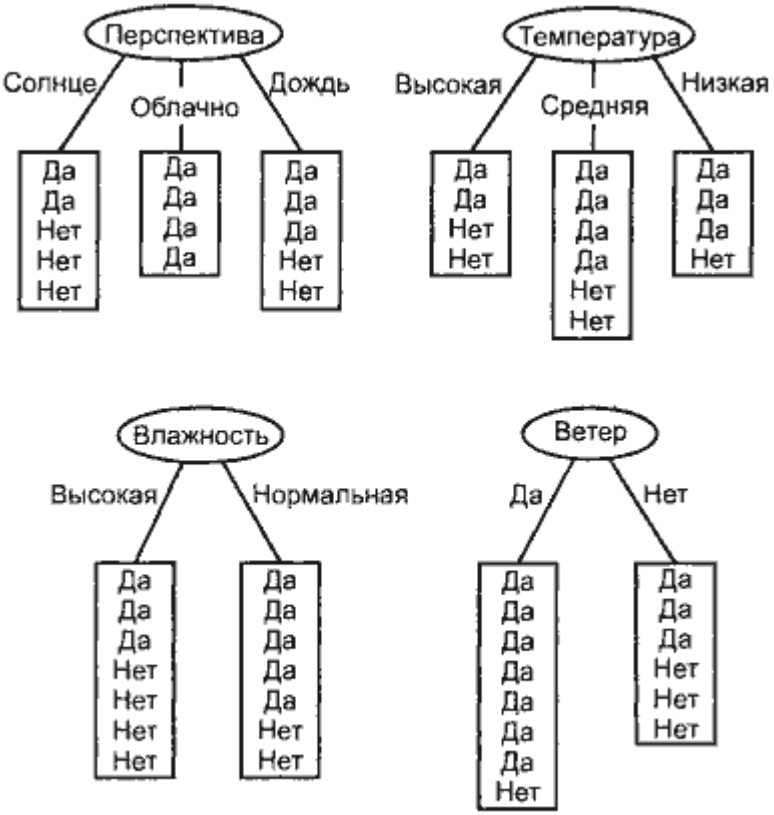
Мы ожидаем, что при разбиении исходного множества, будем получать подмножества с меньшим числом объектом, но более упорядоченные.
Так, чтобы в каждом из них были по-возможности объекты одного класса.
Эта мера упорядоченности (неопределенности)

характеризуется информацией.
В контексте рассматриваемой задачи это количество информации, необходимое для того, чтобы отнести объект к тому или иному классу.
При разделении исходного множества на более мелкие подмножества, используя в качестве критерия для разделения значения выбранной независимой переменной, неопределённость принадлежности объектов конкретным классам будет уменьшаться. Задача состоит в том, чтобы выбрать такие независимые переменные, чтобы максимально уменьшить эту неопределенность и в конечном итоге получить подмножества, содержащие объекты только одного класса.
В последнем случае неопределенность равна нулю.

Единственная доступная информация - каким образом классы распределены в множестве T и его подмножествах, получаемых при разбиении.
Именно она и используется при выборе переменной.

Рассмотрим пример, в котором требуется построить дерево решений относительно того, состоится ли игра при заданных погодных условиях.

Исходя из прошлых наблюдений (накопленных исторических данных), возможны четыре варианта разбиения дерева.



Пусть freq(c_r,I) - число объектов из обучающей выборки, относящихся к классу c_r.
Тогда вероятность того, что случайно выбранный объект из обучающего множества I будет принадлежать классу c_r равняется:
$P = \frac{freq(c_r, I)}{|I|}$.
Подсчитаем количество информации, основываясь на числе объектов того или иного класса, получившихся в узле дерева после разбиения исходного множества.
Согласно теории информации оценку среднего количества информации, необходимого для определения класса объекта из множества T, даёт выражение: $H(x) = -\sum_{i=1}^{n} p(i) \log_2 p(i)$ (информационная энтропия)
Подставляя в эту формулу полученное значение для P, получим: $Info(I) = -\sum_{r=1}^{k} \frac{freq(c_r, I)}{|I|} \log_2(\frac{freq(c_r, I)}{|I|})$.
Поскольку используется логарифм с двоичным основанием, то это выражение даёт количественную оценку в битах.
Для оценки количества информации справедливы следующие утверждения:

1. Если число объектов того или иного класса в получившемся подмножестве равно нулю, то количество информации также равно нулю.
2. Если число объектов одного класса равно числу объектов другого класса, то количество информации максимально.

Посчитаем значение информационной энтропии для исходного множества до разбиения. $Info(I) = -\frac{9}{14} * log_2(\frac{9}{14}) - \frac{5}{14} * log_2(\frac{5}{14}) = 0.94$ бит.

Ту же оценку, но уже после разбиения множества T по x_h даёт следующее выражение: $Info_{x_h}(T) = \sum_{i=1}^{m} \frac{T_i}{|T|} Info(T_i)$ или $Info_{x_h}(T) = \sum_{i=1}^{m} \frac{T_i}{|T|}(-\sum_{r=1}^{k} \frac{freq(c_r, T_i)}{|T_i|} \log_2(\frac{freq(c_r, T_i)}{|T_i|}))$.

Например, для переменной "Наблюдение", оценка будет следующей:

$Info_{sun} = -\frac{2}{5} * log_2(\frac{2}{5}) - \frac{3}{5} * log_2(\frac{3}{5}) = 0.971$ бит.

$Info_{clouds} = -\frac{4}{4} * log_2(\frac{4}{4}) - \frac{0}{4} * log_2(\frac{0}{4}) = 0$ бит.

$Info_{rain} = -\frac{3}{5} * log_2(\frac{3}{5}) - \frac{2}{5} * log_2(\frac{2}{5}) = 0.971$ бит.

$Info_{condition} = \frac{5}{14} * 0.971 + \frac{4}{14} * 0 + \frac{5}{14} * 0.971 = 0.693$ бит.

Критерием для выбора атрибута (зависимой переменной) будет являться следующая формула: $Gain(x_h) = Info(I) - Info_{x_h}(T)$. Критерий Gain рассчитывается для всех независимых переменных после чего выбирается переменная с максимальным значением Gain. Необходимо выбрать такую переменную, чтобы при разбиении по ней один из классов имел наибольшую вероятность появления. Это возможно в том случае, когда энтропия $Info_x$ имеет минимальное значение и, соответственно, критерий Gain(X) достигает своего максимума.
В нашем примере значение Gain для независимой переменной "Наблюдение" (перспектива) будет равно:

```
Gain(перспектива) = Info(I) - Info(перспектива) = 0.94
-          0.693          =          0.247          бит.
```

Аналогичные расчеты можно провести для других независимых переменных. В результате получаем:

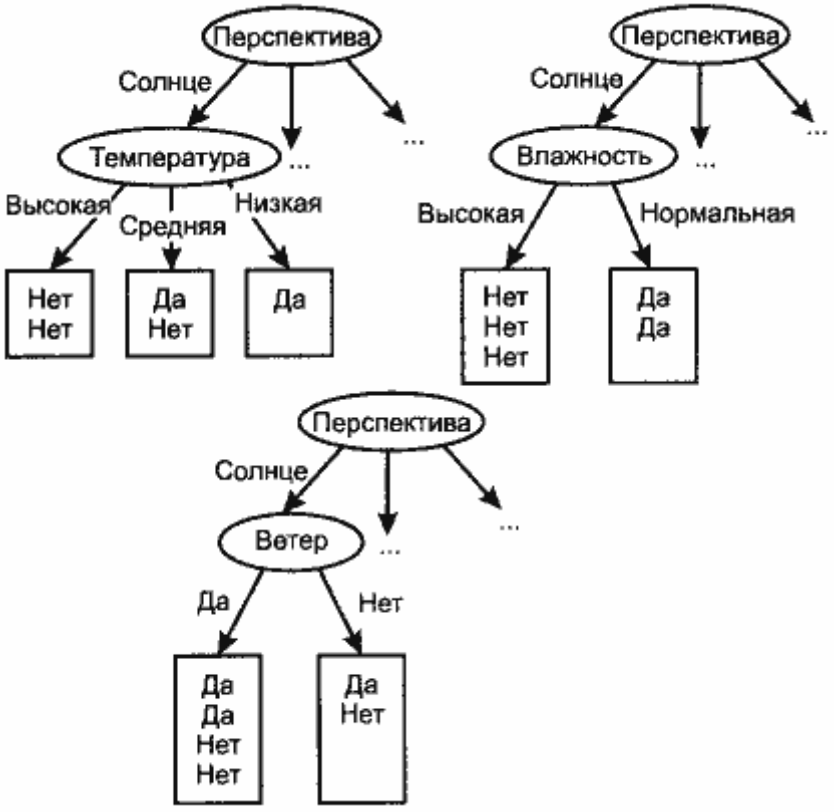
```
Gain(наблюдение)          =          0.247          бит.

Gain(температура)          =          0.029          бит.

Gain(влажность)          =          0.152          бит.

Gain(ветер)          =          0.048          бит.
```

Таким образом, для первоначального разбиения лучше всего выбрать независимую переменную "Наблюдение". Далее требуется выбрать следующую переменную для разбиения. Варианты разбиения представлены на рисунке.



Аналогичным образом можно посчитать значение Gain для каждого разбиения:

```
Gain(температура)          =          0.571          бит.

Gain(влажность)            =          0.971          бит.

Gain(ветер)                =          0.02           бит.
```

Видно, что следующей переменной, по которой будет разбиваться подмножество T (солнечно) будет "Влажность". Дальнейшее разбиение этой ветви уже не потребуется, т.к. в получившихся подмножествах все объекты относятся только к одному классу.

Если в процессе работы алгоритма получен узел, ассоциированный с пустым множеством (ни один объект не попал в данный узел), то он помечается как лист, и в качестве решения листа выбирается наиболее часто встречающийся класс у непосредственного предка данного листа.

Алгоритм C4.5

Представляет собой усовершенствованный вариант алгоритма ID3. Среди улучшений стоит отметить следующие:

- Возможность работать не только с категориальными атрибутами, но также с числовыми. Для этого алгоритм разбивает область значений независимой переменной на несколько интервалов и делит исходное множество на подмножества в соответствии с тем интервалом, в который попадает значение зависимой переменной.
- После построения дерева происходит усечение его ветвей. Если получившееся дерево слишком велико, выполняется либо группировка нескольких узлов в один лист, либо замещение узла дерева нижележащим поддеревом. Перед операцией над деревом вычисляется ошибка правила классификации, содержащегося в рассматриваемом узле. Если после замещения (или группировки) ошибка не возрастает (и не сильно увеличивается энтропия), значит замену можно произвести без ущерба для построенной модели.

Один из недостатков алгоритма ID3 является то, что он некорректно работает с атрибутами, имеющими уникальные

значения для всех объектов из обучающей выборки. Для таких объектов информационная энтропия равна нулю и никаких новых данных от построенного дерева по данной зависимой переменной получить не удасться. Поскольку получаемые после разбиения подмножества буду содержать по одному объекту.
Алгоритм C4.5 решает эту проблему путём введения нормализации.
Оценивается не количество объектов того или иного класса после разбиения, а число подмножеств и их мощность (число элементов).
Выражение $splitinfo(x_h) = -\sum_{i=1}^{m} \frac{T_i}{T} \log_2(\frac{T_i}{T})$ оценивает потенциальную информацию, получаемую при разбиении множества T на m подмножеств.
Критерием выбора переменной для разбиения будет выражение: $gainratio(x_h) = \frac{Gain(x_h)}{splitinfo(x_h)}$ или $gainratio(x_h) = \frac{Gain(x_h)}{-\sum_{i=1}^{m} \frac{T_i}{T} \log_2(\frac{T_i}{T})}$.
При условии, что имеется k классов и n - число объектов в обучающей выборке и одновременно количество значений переменных, тогда числитель максимально будет равен $log_2 k$, а знаменатель максимально равен $log_2 n$. Если предположить, что количество объектов знаведомо больше количества классов, то знаменатель растёт быстрее, чем числитель и, соответственно, значение выражения будет небольшим.
В обучающей выборке могут присутствовать объекты с пропущенными значениями атрибутов. В этом случае их либо отбрасывают (что влечёт за собой риск потерять часть данных), либо применить подход, предполагающий, что пропущенные значения по переменной вероятностно распределены пропорционально частоте появления существующих значений.

Алгоритм покрытия

Алгоритм заключается в построении деревьев решений для каждого класса по отдельности. На каждом этапе генерируется проверка узла дерева, который покрывает несколько объектов обучающей выборки.
На каждом шаге алгоритма выбирается значение переменной, которое разделяет множество на два подмножества. Разделение должно выполняться так, чтобы все объекты

класса, для которого строится дерево, принадлежали одному подмножеству. Такое разбиение производится до тех пор, пока не будет построено подмножество, содержащее только объекты одного класса.
Для выбора независимой переменной и её значения, которое разделяет множество, выполняются следующие действия:

1. Из построенного на предыдущем этапе подмножества (для первого этапа это вся обучающая выборка), включающего объекты, относящиеся к выбранному классу для каждой независимой переменной, выбираются все значения, встречающиеся в этом подмножестве.
2. Для каждого значения каждой переменное подсчитывается количество объектов, удовлетворяющих этому условию и относящихся к выбранному классу.
3. Выбираются условия, покрывающие наибольшее количество объектов выбранного класса.
4. Выбранное условие является условием разбиения подмножества на два новых.

После построения дерева для одного класса таким же образом строятся деревья для других классов.

Преимущества использования деревьев решений

- быстрый процесс обучения;
- генерация правил в областях, где эксперту трудно формализовать свои знания;
- извлечение правил на естественном языке;
- интуитивно понятная классификационная модель;
- высокая точность прогноза, сопоставимая с другими методами (статистика, нейронные сети);

Области применения деревьев решений

Деревья решений являются прекрасным инструментом в системах поддержки принятия решений, интеллектуального анализа данных (data mining). В состав многих пакетов, предназначенных для интеллектуального анализа данных, уже включены методы построения деревьев решений.

Деревья решений успешно применяются для решения практических задач в следующих областях:

- Банковское дело. Оценка кредитоспособности клиентов банка при выдаче кредитов.
- Промышленность. Контроль за качеством продукции (выявление дефектов), испытания без разрушений (например проверка качества сварки) и т.д.
- Медицина. Диагностика различных заболеваний.
- Молекулярная биология. Анализ строения аминокислот.

Лекция 4 - Методы построения математических функций

Методы, рассмотренные для правил и деревьев решений, работают наиболее естественно с категориальными переменными. Их можно адаптировать для работы с числовыми переменными, однако существуют методы, которые наиболее естественно работают с ними. Сегодня мы рассмотрим методы, которые используют математические функции в качестве правил для аналитики и прогнозирования.

Корреляционный анализ

Корреляционный анализ - это анализ переменных на предмет того, существует ли между ними какая-либо достаточно сильная зависимость. Нам требуется установить, влияют ли независимые переменные на зависимую - или это случайный набор данных. Следует отметить, тем не менее, что иногда даже корреляция не определяет причинности, а только то, что они изменяются по сходному закону. В общем, больше нам и не надо.

Очевидно, что, если мы решим использовать математические функции, то тогда зависимая переменная y будет вычисляться через какую-то функцию F (мы будем называеть ее функцией регрессии) от атрибутов $X = < X_1,...X_i,...X_n >$. Однако очевидно также, что будет какая-то погрешность θ, (иначе все слишком хорошо, у нас просто прямая зависимость - впрочем, такое тоже возможно при $\theta = 0$).

$$y = F(X) + \theta$$

На интуитивном уровне можно сказать, что независимые переменные проецируются с некоторым разбросом (погрешностью) на плоскость Oy. (Рисунок). Тогда (опустив некоторые сложные математические выкладки) можно сказать, что:

$$\sigma_y^2 = \sigma_F^2 + \sigma_\theta^2$$

Это означает, что полная вариация зависимой переменной складывается из вариации функции регрессии F(X) () и вариации остаточной случайной компоненты.

В корреляционном анализе нас интересует, насколько изменчивость зависимой переменной обуславливается

изменчивостью независимых переменных (функции от них). То есть:

$$I^2_{y,X} = \frac{\sigma^2_F}{\sigma^2_y} = 1 - \frac{\sigma^2_F}{\sigma^2_\theta}$$

I - это индекс корреляции, $0 \leq I \leq 1$, основной показатель корреляции переменных. В общем случае формулы для индекса корреляции нет, однако иногда его можно оценить.

Для двух переменных

В том случае, если рассматривается двумерная нормальная (x и y распределены нормально) система (x,y), то тогда:

$$I_{y,X} = r = \frac{cov(x,y)}{\sigma_y, \sigma_x}$$

где r - коэффициент корреляции, еще один из достаточно сильных показателей взаимосвязи переменных. Положительный - они прямо пропорциональны, отрицательный - обратно. Однако для любого отклонения от двумерности и нормальности он не является прямо определяющим корреляцию.

В том случае, если есть отклонение от нормальности, то тогда имеет смысл попробовать разбить y на интервалы (сгруппировать по оси объясняющей переменной) и посчитать среди них среднее:

$\overline{y_i} = \frac{\sum_{k=1}^{m_i} y_{ik}}{m_i}$, где k - число интервалов группировки, m_i - число точек в i-м интервале, y_ik - k-ая точка в i-м интервале.

И тогда оценкой для σ^2_F будет:

$$s^2_{\overline{y}(x)} = \frac{1}{n}\sum_{i=1}^{k} m_i(\overline{y}_i - \overline{y})^2, \overline{y} = \frac{\sum_{i=1}^{k} m_i \overline{y}_i}{n}$$

а для σ^2_y:

$$s^2_y = \frac{1}{n}\sum_{i=1}^{k}\sum_{j=1}^{m_i}(y_ij - \overline{y})^2$$

и тогда можно посчитать оценку для $I^2_{y,x}$:

$$\widehat{\rho}^2_{y,x} = \frac{s^2_{\overline{y}(x)}}{s^2_y}$$

$\widehat{\rho}_{y,x}$ - корреляционное отношение, оно похоже по свойствам на корреляционный коэффициент, но несимметрично: $\widehat{\rho}_{y,x} \neq \widehat{\rho}_{x,y}$ и не говорит о характере связи.

В том случае, если сгруппировать невозможно, то следует сначала провести регрессионный анализ и выявить коэффициенты функции регрессии$w_0,...w_p$, а затем считать

$$I^2_{y,X} = 1 - \frac{\frac{1}{n-p-1}\sum_{i=1}^{n}(y_i - F(X_i, w_0...w_p))}{\frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y})}$$

Для произвольного числа переменных

Для произdольного числа переменных $I_{y,x}$ индекс корреляции называется коэффициентом корреляции $R_{y,X}$. Для линейного случая считается через попарные коэффициенты корреляции:

$R^2_{y,X} = 1 - \frac{det(r_{ij})_{ij..pp}}{M^0_0}$, где M^0_0 - минор (определитель матрицы, получающейся из исходной матрицы r_{ij} вычеркиванием первого (нулевого) столбца и строки)

Этот коэффициент всегда больше любого коэффициента попарной корреляции.

Для нелинейного все опять же следует попробовать свести к интервалам, которые линеаризуются, или рассматривать регрессию.

Регрессионный анализ

Регрессионный анализ — метод моделирования измеряемых данных и исследования их свойств. Данные состоят из пар значений зависимой переменной (переменной отклика) и независимой переменной (объясняющей переменной). Регрессионная модель есть функция независимой переменной и параметров с добавленной случайной переменной. Параметры модели настраиваются таким образом, что модель наилучшим образом приближает данные. Критерием качества приближения (целевой функцией) обычно является среднеквадратичная ошибка: сумма квадратов разности значений модели и зависимой переменной для всех значений независимой переменной в качестве аргумента.
При построении математической функции классификации или регрессии основная задача сводится к выбору наилучшей функции из всего множества функций. Может существовать множество функций, одинаково классифицирующих одну и ту же обучающую выборку.
В результате задачу построения функции классификации и регрессии можно формально описать как задачу выбора

функции с минимальной степенью ошибки: $minR(f) = \frac{1}{m}\sum_{i=1}^{m} c(y_i, f(x_i))$

Где:

- f - функция классификации или регрессии из множества всех функций F;
- $c(y_i,f(x_i))$ - функция потерь, в которой $f(x_i)$ - значение зависимой переменной, найденное с помощью функции f для вектора x_i, а y_i - её точное (известное) значение.

В случае бинарной классификации (принадлежности объекта к одному из двух классов) простейшая функция потерь принимает значение 1 в случае неправильного предсказания и 0 в противном случае.
Но здесь не учитывается ни тип ошибки, ни её величина. Для оценки качества классификации целесообразно использовать разность f(x) - y. Разница между предсказанным и известным (по данным обучающей выборки) значением зависимой переменной.

Метод наименьших квадратов

В случае когда регрессионная функция линейна, множество всех возможных функций разбиения (F) можно представить в виде: $y = w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n = w_0 + \sum w_jx_j$,
где $w_0,w_1,...,w_n$ - коэффициенты при независимых переменных. Задача заключается в отыскании таких коэффициентов w, чтобы разница между предсказанным и известным значением зависимой переменной была минимальна. Также задачу можно сформулировать как минимизация суммы квадратов разностей рассчитанных и известных значений зависимой переменной.
$minR(f) = min\frac{1}{m}\sum_{i=1}^{m}(y_i - \sum_{j=1}^{n} w_if_i(x_i))^2$

Нелинейные методы

Нелинейные модели лучше классифицируют объекты, однако их построение более сложно. Задача также сводится к минимизации выражения $minR(f) = min\frac{1}{m}\sum_{i=1}^{m}(y_i - \sum_{j=1}^{n} w_if_i(x_i))^2$.

При этом множество F содержит нелинейные функции. В простейшем случае построение таких функций сводится к

построению линейных моделей. Для этого исходное пространство объектов преобразуется к новому. В новом пространстве строится линейная функция, которая в исходном пространстве является нелинейной. Для использования построенной функции выполняется обратное преобразование в исходное пространство. Если объекты их обучающей выборки отобразить графически (откладывая по одной оси значение одной из независимых переменных, а по другой оси значение зависимой переменной), то может получиться картина, когда объекты одного класса будут сгруппированы в центре, а объекты другого рассеяны вокруг этого центра. В этом случае применение линейных методов не поможет. Выходом из этой ситуации является перевод объектов в другое пространство. Можно возвести координату каждой точки в квадрат, тогда распределение может получиться таким, что станет возможным применение линейных методов.
Описанный подход имеет один существенный недостаток. Процесс преобразования достаточно сложен с точки зрения вычислений, причём вычислительная сложность растёт с увеличением числа данных. Если учесть, что преобразование выполняется два раза (прямое и обратное), то такая зависимость не является линейной. В связи с этим построение нелинейных моделей с таким подходом будет неэффективным. Альтернативой ему может служить метод Support Vector Machines (SVM), не выполняющий отдельных преобразований всех объектов, но учитывающий это в рассчётах.

Метод опорных векторов

Основная идея метода опорных векторов – перевод исходных векторов в пространство более высокой размерности и поиск максимальной разделяющей гиперплоскости в этом пространстве.

Каждый объект данных представлен, как вектор (точка в p-мерном пространстве (последовательность p чисел)). Рассмотрим случай бинарной классификации - когда каждая из этих точек принадлежит только одному их двух классов.

Нас интересует, можем ли мы разделить эти точки какой-либо прямой так, чтобы с одной стороны были точки одного класса, а с другой - другого. Такая прямая называется гиперплоскостью размерностью "p−1". Она описывается

уравнением $\vec{w} \cdot \vec{x} - b = 0$, где $\vec{w} \cdot \vec{x}$ - скалярное произведение векторов.

Тогда для того, чтобы разделить точки прямой, надо, чтобы выполнялось:

$\vec{w} \cdot \vec{x_i} - b > 0$ -> $y_i = 1$ (принадлежит к одному классу)

$\vec{w} \cdot \vec{x_i} - b < 0$ -> $y_i = -1$ (принадлежит к другому классу)

Для улучшения классификации попробуем найти такую разделяющую гиперплоскость, чтобы она максимально отстояла от точек каждого класса (фактически, разделим не прямой, а полосой).

$\vec{w} \cdot \vec{x_i} - b \geq 0 + \epsilon$ -> $y_i = 1$ (принадлежит к одному классу)

$\vec{w} \cdot \vec{x_i} - b \leq 0 - \epsilon$ -> $y_i = -1$ для некоторого $\varepsilon > 0$

Домножим эти уравнения на константу так, чтобы для граничных точек - ближайших к гиперплоскости - выполнялось:

$\vec{w} \cdot \vec{x_i} - b = y_i$

Это можно сделать, так как для оптимальной разделяющей гиперплоскости все граничные точки лежат на одинаковом от нее расстоянии. Разделим тогда неравенства на ε и выберем $\varepsilon = 1$. Тогда:

$\vec{w} \cdot \vec{x_i} - b \geq 1$, если $y_i = 1$ $\vec{w} \cdot \vec{x_i} - b \leq 1$, если $y_i = -1$

(Рисунок полосы)

Ширина разделительной полосы тогда равна $\frac{2}{\|w\|}$. Нам нужно, чтобы ширина полосы была как можно большей ($\|w\|^2 \to min$)(чтобы классификация была точнее) и при этом $y_i(\vec{w} \cdot \vec{x_i} - b) \geq 1$ (так как $y_i = \pm 1$). Это задача квадратичной оптимизации.

Это в том случае, если классы линейно разделимы (разделимы гиперплоскостью). В том случае, если есть точки, которые лежат по "неправильные" стороны гиперплоскости, то мы считаем эти точки ошибками. Но их надо учитывать:

$y_i(\vec{w} \cdot \vec{x_i} - b) \geq 1 - \xi_i$, ξ_i - ошибка x_i, если $\xi_i = 0$ - ошибки нет, если $0 < \xi_i < 1$, то объект внутри разделительной полосы, но классифицируется правильно, если $\xi_i > 1$ - это ошибка.

Тогда нам надо минимизировать сумму $\|w\|^2 + C\sum_i \xi_i \to min$, C - параметр, позволяющий регулировать, что нам важнее - отсутствие ошибок или выразительность результата (ширина разделительной полосы), мы его выбираем сами .

Как известно, чтобы найти минимум, надо исследовать производную. Однако у нас заданы еще и границы, в которых этот минимум искать! Это решается при помощи метода Лагранжа: Лагранж доказал (для уравнений в общем и неравенств в частности), что в той точке, где у функции F(x) условный (ограниченный условиями) минимум, в той же точке у функции

$$F(x) - \sum_i \lambda_i G_i$$

(для фиксированного набора λ_i, G_i - условия ограничений) тоже минимум, но уже безусловный, по всем x. При этом либо $\lambda_i = 0$ и соответственно ограничение G_i не активно, либо $\lambda_i \neq 0$, тогда неравенство G_i превращается в уравнение.

Тогда алгоритм поиска минимума следующий:

1. Взять все производные

$$F(x) - \sum_i \lambda_i G_i$$

по x и приравнять их к нулю.

1. С помощью полученных уравнений выразить x через $\lambda_1..\lambda_n$.
2. Подставить выраженные x в неравенства ограничений, сделав их уравнениями. Получим систему из n уравнений с n неизвестными λ. Она решается.

Тогда можно нашу задачу (минимизировать $\frac{1}{2}\|w\|^2 + C\sum_i \xi_i$ при условиях $\xi_i \geq 0$, $y_i(\overrightarrow{w} \cdot \overrightarrow{x_i} - b) \geq 1 - \xi_i$) сформулировать при помощи метода лагранжа следующим образом:

Необходимо найти минимум по w, b и ξ_i и максимум по λ_i функции $\frac{1}{2}w \cdot w + C\sum_i \xi_i - \sum_i \lambda_i(\xi_i + y_i(w \cdot x_i - b - 1)$ при условиях $\xi_i \geq 0, \lambda_i \geq 0$.

Возьмем производную лагранжианы по w, приравняем ее к нулю и из полученного выражения выразим w:

$$w = \sum_i \lambda_i y_i x_i$$

Следовательно, вектор w - линейная комбинация учебных векторов x_i, причем только таких, для которых $\lambda_i \neq 0$. Именно эти вектора называются опорными и дали название метода.

Взяв производную по b, получим

$$\sum_i \lambda_i y_i = 0$$

.

Подставим w в лагранжиану и получим новую (двойственную) задачу:

$\sum_i \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j (x_i \cdot x_j) \rightarrow max$ при

$$\sum_i \lambda_i y_i = 0$$

, $0 \leq \lambda_i \leq C$

Это финальная задача, которую нам надо решить. Обратите внимание, что в итоге все свелось к скалярному произведению двух векторов - x_iи x_j. Это произведение называется **Ядром**.

Эта задача решается методом последовательных оптимизации (благодаря тому, что коэффициенты при $\lambda_i\lambda_j$ положительны -> функция выпуклая -> локальный максимум яявляется глобальным):

1. Задаем набор λ_i, удовлетворяющий условиям;
2. Выбираем из набора первую пару улучшаемых $\lambda_i\lambda_j$;
3. При фиксированных остальных λ и имеющихся ограничениях $y_i\lambda_i + y_j\lambda_j = y_i\lambda_i^{old} + y_j\lambda_j^{old}, 0 \leq \lambda_i, \lambda_j \leq C$ находим пару таких λ_i,λ_j, при которых $\sum_i \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j (x_i \cdot x_j) \rightarrow max$ (мини-оптимизация);
4. Переходим на шаг 2 до тех пор, пока прирост функции будет меньше некого достаточно малого ε. Если функция расти перестала, мы нашли нужные нам λ.

Ядра

Это все хорошо, если набор хотя бы более-менее линеен (когда ошибка достаточно мала). В том случае, если набор не линеен, тогда ошибка велика. В 1992 году Бернхард Босер, Изабель Гуйон и Вапник предложили способ создания нелинейного классификатора, в основе которого лежит переход от скалярных произведений к произвольным ядрам, так называемый kernel trick , позволяющий строить нелинейные разделители.

Результирующий алгоритм крайне похож на алгоритм линейной классификации, с той лишь разницей, что каждое скалярное произведение в приведенных выше формулах заменяется нелинейной функцией ядра (скалярным произведением в пространстве с большей размерностью). В этом пространстве уже может существовать оптимальная разделяющая гиперплоскость. Так как размерность получаемого пространства может быть больше размерности исходного, то преобразование, сопоставляющее скалярные произведения, будет нелинейным, а значит функция, соответсвующая в исходном пространстве оптимальной разделяющей гиперплоскости будет также нелинейной.

Стоит отметить, что если исходное пространство имеет достаточно высокую размерность, то можно надеяться, что в нём выборка окажется линейно разделимой.

Наиболее распространенные ядра:

- Полиномиальное (однородное): $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}')^d$
- Полиномиальное (неоднородное): $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + 1)^d$
- Радиальная базисная функция: $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$, для $\gamma > 0$
- Радиальная базисная функция Гаусса: $k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$
- Сигмоид: $k(\mathbf{x}, \mathbf{x}') = \tanh(\kappa \mathbf{x} \cdot \mathbf{x}' + c)$, для почти всех $\kappa > 0$ и $c < 0$

Лекция 5 - Поиск ассоциативных правил

Одной из наиболее распространённых задач анализа данных является определение часто встречающихся наборов объектов в большом множестве наборов. Впервые это задача была предложена поиска ассоциативных правил для нахождения типичных шаблонов покупок, совершаемых в супермаркетах, поэтому иногда ее еще называют анализом рыночной корзины (market basket analysis).

Формальная постановка задачи

Пусть имеется база данных, состоящая из покупательских транзакций.
Каждая транзакция – это набор товаров, купленных покупателем за один визит. Такую транзакцию еще называют рыночной корзиной.

Пусть $I = \{i_1,i_2,...,i_j,...,i_n\}$ – множество (набор) товаров (объектов) общим числом n.
Пусть D – множество транзакций $D = \{T_1,T_2,T_r,...,T_m\}$, где каждая транзакция T – это набор элементов из I. $T = \{i_j | i_j \in I\}$.

В сфере торговли, например, такими объектами являются товары, представленные в прайс-листе:

Идентификатор	Наименование товара	Цена
0	Шоколад	30
1	Хлеб	12
2	Масло	10
3	Вода	4
4	Молоко	14
5	Орехи	15

Они соответствуют следующему множеству объектов: I={шоколад, хлеб, масло, вода, молоко, орехи}. Примерами транзакций могут быть T_1 = { хлеб, масло, молоко }, T_2 = { шоколад, вода, орехи }. Множество транзакций, в которые входит объект i_j, обозначим следующим образом:

$D_{i_j} = \{T_r | i_j \in T_r; j = 1..n; r = 1..m\} \subseteq D$.

Множество D может быть представлено следующим образом:

Номер транзакции	Номер товара	Наименование товара	Цена
0	1	Хлеб	12
0	3	Вода	4
0	4	Молоко	14
1	2	Масло	10
1	3	Вода	4
1	5	Орехи	15
2	5	Орехи	15
2	2	Масло	10
2	1	Хлеб	12
2	2	Масло	10
2	3	Вода	4
3	2	Масло	10
3	5	Орехи	15
3	2	Масло	10

В данном примере, множеством транзакций, содержащим объект "Вода", является следующее множество:

```
D(вода) = {{ хлеб, вода, молоко },
           { масло, вода, орехи },
{ орехи, масло, хлеб, масло, вода }}.
```

Некоторый произвольный набор объектов (itemset) обозначим следующим образом: $F = \{i_j | i_j \in I; j = 1..n\}$, например F = {хлеб, масло}. Набор, состоящий из k элементов, называется k-элементным набором.

Множество транзакций, в которые входит набор F, обозначим следующим образом: $D_F = \{T_r | F \subseteq T_r; r = 1..m\} \subseteq D$.

В данном примере:

D(масло, вода) = {{ масло, вода, орехи },
{ орехи, масло, хлеб, масло, вода }}.

Отношение количества транзакций, в которое входит набор F, к общему количеству транзакций называется поддержкой (support) набора F и обозначается Supp(F):

$Supp(F) = \frac{|D_F|}{|D|}$.

Для набора { масло, вода } поддержка будет равна 0,5, т.к. данный набор входит в две транзакции с номерами 1 и 2, а всего транзакций четыре.
При поиске аналитик может указать минимальное значение поддержки интересующих его наборов $Supp_{min}$.
Набор называется частым (large itemset), если значение его поддержки больше минимального значения поддержки, заданного пользователем: $Supp(F) > Supp_{min}$.
Таким образом, при поиске ассоциативных правил требуется найти множество всех частых наборов: $L = \{F \mid Supp(F) > Supp_{min}\}$.
В данном примере частыми наборами при $Supp_{min} = 0{,}5$ являются следующие:

{хлеб} $Supp_{min} = 0{,}5$;

{хлеб, вода} $Supp_{min} = 0{,}5$;

{масло} $Supp_{min} = 0{,}75$;

{масло, вода} $Supp_{min} = 0{,}5$;

{масло, вода, орехи} $Supp_{min} = 0{,}5$;

{масло, орехи} $Supp_{min} = 0{,}75$;

{вода} $Supp_{min} = 0{,}75$;

{вода, орехи} $Supp_{min} = 0{,}5$;

$$\{\text{орехи}\}Supp_{min} = 0{,}75;$$

Представление результатов

Решение задачи поиска ассоциативных правил, как и любой задачи, сводится к обработке исходных данных и получению результатов. Результаты, получаемые при решении данной задачи принято представлять в виде ассоциативных правил. В связи с этим в их поиске выделяю два этапа:

- нахождение всех частых наборов объектов:
- генерация ассоциативных правил из найденных частых наборов объектов.

Ассоциативные правила имеют следующий вид:

```
если (условие), то (результат),
```

где *условие* - обычно не логическое выражение (как в классификационных правилах), а набор объектов из множества I, с которым связаны (ассоциированы) объекты, включенные в *результат* данного правила. Например, ассоциативное правило: "если (молоко, масло), то (хлеб)" означает, что если потребитель покупает молоко и масло, то он покупает и хлеб. Основным достоинством ассоциативных правил является их лёгкое восприятие человеком и простая интерпретация языками программирования. Однако, они не всегда полезны. Выделяют три вида правил:

- *полезные правила* - содержат действительную информацию, которая ранее была неизвестна, но имеет логическое объяснение. Такие правила могут быть использованы для принятия решений, приносящих выгоду;
- *тривиальные правила* - содержат действительную и легко объяснимую информацию, которая уже известна. Такие правила не могут принести пользу, т.к. отражают или

известные законы в исследуемой области, или результаты прошлой деятельности. Иногда такие правила могут использоваться для проверки выполнения решений, принятых на основании предыдущего анализа;

- *непонятные правила* - содержат информацию, которая не может быть объяснена. Такие правила могут быть получены на основе аномальных значений, или сугубо скрытых знаний. Напрямую такие правила нельзя использовать для принятия решений, т.к. их необъяснимость может привести к непредсказуемым результатам. Для лучшего понимания требуется дополнительный анализ.

Ассоциативные правила строятся на основе частых наборов. Так правила, построенные на основании набора F, являются возможными комбинациями объектов, входящих в него.
Например, для набора {масло, вода, орехи}, могут быть построены следующие правила:

```
если (масло), то (вода); если (масло), то (орехи); если
(масло), то (вода, орехи);
если (вода), то (масло); если (вода), то (орехи); если
(вода), то (масло, орехи);
если (орехи), то (масло); если (орехи), то (вода); если
(орехи), то (масло, вода);
если (масло, вода), то (орехи); если (масло, орехи), то
(вода); если (вода, орехи), то (масло);
```

Таким образом, количество ассоциативных правил может быть очень большим и трудновоспринимаемым для человека. К тому же, не все из построенных правил несут в себе полезную информацию.
Для оценки их полезности вводятся следующие величины:
Поддержка(support) - показывает, какой процент транзакций поддерживает данное правило.
Так как правило строится на основании набора, то, значит, правило X=>Y имеет поддержку, равную поддержке набора F, который составляют X и Y: $Supp_{X=>Y} = Supp_F = \frac{|D_{F=X \cup Y}|}{|D|}$.
Очевидно, что правила, построенные на основании одного и того же набора, имеют одинаковую поддержку,

например, поддержка Supp(если (вода, масло), то (орехи) = Supp(вода, масло, орехи) = 0,5.

Достоверность(confidence) - показывает вероятность того, что из наличия в транзакции набора X следует наличие в ней набора Y.
Достоверностью правила X=>Y является отношение числа транзакций, содержащих X и Y, к числу транзакций, содержащих набор X:
$Conf_{X=>Y} = \frac{|D_{F=X\cup Y}|}{|D_X|} = \frac{Supp_{X\cup Y}}{Supp_X}$.
Очевидно, что чем больше достоверность, тем правило лучше, причем у правил, построенных на основании одного и того же набора,
достоверность будет разная, например:

```
Conf(если (вода), то (орехи)) = 2/3;
Conf(если (орехи), то (вода)) = 2/3;
Conf(если (вода, масло), то (орехи)) = 1;
Conf(если (вода), то (орехи, масло)) = 2/3.
```

К сожалению, достоверность не позволяет определить полезность правила. Если процент наличия в транзакциях набора Y
при условии наличия в нем набора X меньше, чем процент безусловного наличия набора Y, т.е.:
$Conf_{X=>Y} = \frac{Supp_{X\cup Y}}{Supp_X} < Supp_Y$.
Это значит, что вероятность случайно угадать наличие в транзакции набора Y больше, чем предсказать это с помощью правила X=>Y.
Для исправления такой ситуации вводится мера - *улучшение*.
Улучшение(improvement) - показывает, полезнее ли правило случайного угадывания. Улучшение правила является отношением
числа транзакций, содержащих наборы X и Y, к произведению количества транзакций, содержащих набор X, и количества транзакций,
содержащих набор Y:
$impr_{X=>Y} = \frac{|D_{F=X\cup Y}|}{|D_X||D_Y|} = \frac{Supp_{X\cup Y}}{Supp_X * Supp_Y}$.

Например, impr(если (вода, масло), то (орехи) = 0,5/(0,5*0,5) = 2.

Если улучшение больше единицы, то это значит, что с помощью правила предсказать наличие набора Y вероятнее, чем случайное угадывание, если меньше единицы, то наоборот.
В последнем случае можно использовать отрицательное правило, т.е. правило, которое предсказывает отсутствие набор Y:

X => не Y.

Правда, на практике такие правила мало применимы. Например, правило: "если (вода, масло), то не (молоко)" мало полезно,
т.к. слабо выражает поведение покупателя.
Данные оценки используются при генерации правил. Аналитик при поиске ассоциативных правил задает минимальные значения перечисленных величин. В результате те правила, которые не удовлетворяют этим условиям,
отбрасываются и не включаются в решение задачи. С этой точки зрения нельзя объединять разные правила, хотя и имеющие
общую смысловую нагрузку.
Например, следующие правила:

$$X = i_1, i_2 => Y = i_3,$$

$$X = i_1, i_2 => Y = i_4,$$

нельзя объединить в одно:

$$X = i_1, i_2 => Y = i_3, i_4,$$

т.к. достоверности их будут разные, следовательно, некоторые из них могут быть исключены, а некоторые - нет.

Алгоритм Apriori

Свойство анти-монотонности

Выявление часто встречающихся наборов элементов – операция, требующая много вычислительных ресурсов и, соответственно, времени.
Примитивный подход к решению данной задачи – простой перебор всех возможных наборов элементов.
Это потребует $O(2^{|I|})$ операций, где |I| – количество элементов.
Apriori использует одно из свойств поддержки, гласящее: поддержка любого набора элементов не может превышать минимальной поддержки любого из его подмножеств.
Например, поддержка 3-элементного набора {Хлеб, Масло, Молоко}
будет всегда меньше или равна поддержке 2-элементных наборов {Хлеб, Масло}, {Хлеб, Молоко}, {Масло, Молоко}.
Дело в том, что любая транзакция, содержащая {Хлеб, Масло, Молоко}, также должна содержать {Хлеб, Масло}, {Хлеб, Молоко}, {Масло, Молоко},
причем обратное не верно.

Это свойство носит название анти-монотонности и служит для снижения размерности пространства поиска.
Не имей мы в наличии такого свойства, нахождение многоэлементных наборов было бы практически невыполнимой задачей
в связи с экспоненциальным ростом вычислений.

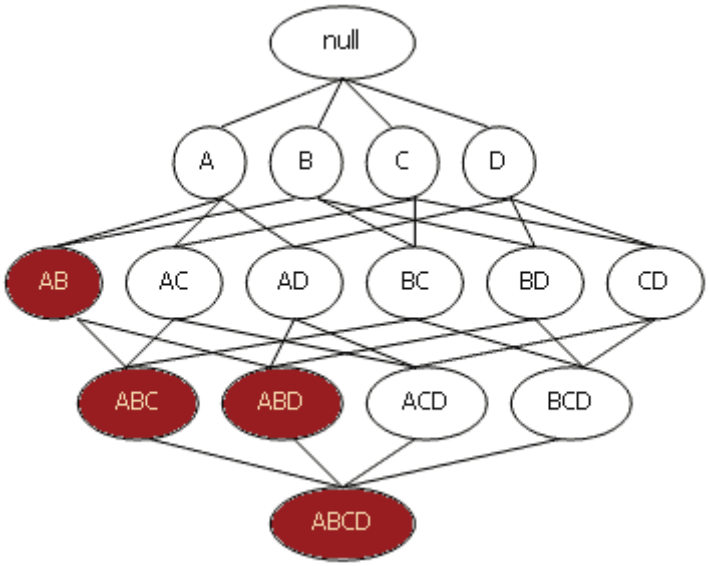
Свойству анти-монотонности можно дать и другую формулировку: с ростом размера набора элементов поддержка уменьшается,
либо остается такой же. Из всего вышесказанного следует, что любой k-элементный набор будет часто встречающимся тогда и только тогда, когда все его (k-1)-элементные подмножества будут часто встречающимися.

Все возможные наборы элементов из I можно представить в виде решетки, начинающейся с пустого множества, затем на 1 уровне 1-элементные наборы, на 2-м – 2-элементные и т.д. На k уровне представлены k-элементные наборы, связанные со всеми своими (k-1)-элементными

подмножествами.

Рассмотрим рисунок, иллюстрирующий набор элементов I – {A, B, C, D}.

Предположим, что набор из элементов {A, B} имеет поддержку ниже заданного порога и, соответственно, не является часто встречающимся.
Тогда, согласно свойству анти-монотонности, все его супермножества также не являются часто встречающимися и отбрасываются.
Вся эта ветвь, начиная с {A, B}, выделена фоном.
Использование этой эвристики позволяет существенно сократить пространство поиска.



Описание алгоритма

Алгоритм Apriori определяет часто встречающиеся наборы за несколько этапов.
На i-ом этапе определяются все часто встречающиеся i-элементные наборы. Каждый этап состоит из двух шагов:

1. формирование кандидатов (candidate generation);
2. подсчет поддержки кандидатов (candidate counting).

Рассмотрим i-ый этап. На шаге формирования кандидатов алгоритм создает множество кандидатов из i-элементных наборов,
чья поддержка пока не вычисляется. На шаге подсчета

кандидатов алгоритм сканирует множество транзакций, вычисляя поддержку наборов-кандидатов. После сканирования отбрасываются кандидаты, поддержка которых меньше
определенного пользователем минимума, и сохраняются только часто встречающиеся i-элементные наборы.
Во время первого этапа выбранное множество наборов-кандидатов содержит все одно-элементные частые наборы. Алгоритм вычисляет их поддержку во время шага подсчёта поддержки кандидатов.
Описанный алгоритм можно записать в виде следующего псевдокода:

```
  1. L1 = {часто встречающиеся 1-элементные наборы}
  2. для (k=2; Lk-1 <> ; k++) {
  3.   Ck = Apriorigen(Lk-1) // генерация кандидатов
  4.   для всех транзакций t T {
  5.     Ct = subset(Ck, t) // удаление избыточных
правил
  6.     для всех кандидатов c Ct
  7.        c.count ++
  8.     }
  9.     Lk = { c ∈ Ck | c.count >= minsupport} //
отбор кандидатов
 10.   }
 11. Результат ⋃k Lk
```

Обозначения, используемые в алгоритме:

- Lk - множество k-элементных наборов, чья поддержка не меньше заданной пользователем. Каждый член множества имеет набор упорядоченных ($i_j < i_p$ если j < p) элементов F и значение поддержки набора $Supp_F > Supp_{min}$:

$L_k = \{(F_1,Supp_1),(F_2,Supp_2),...,(F_q,Supp_q)\}$,
где $F_j = \{i_1,i_2,...,i_k\}$;

- Ck - множество кандидатов k-элементных наборов потенциально частых. Каждый член множества имеет набор упорядоченных ($i_j < i_p$ если j < p) элементов F и значение поддержки набора Supp.

Опишем данный алгоритм по шагам.
Шаг 1. Присвоить k = 1 и выполнить отбор всех 1-элементных наборов, у которых поддержка больше минимально заданной пользователем$Supp_{min}$.
Шаг 2. k = k + 1.
Шаг 3. Если не удается создавать k-элементные наборы, то завершить алгоритм, иначе выполнить следующий шаг.
Шаг 4. Создать множество k-элементных наборов кандидатов из частых наборов. Для этого необходимо объединить в k-элементные кандидаты (k-1)-элементные частые наборы. Каждый кандидат $c \in C_k$ будет формироваться путём добавления к (k-1)-элементному частому набору - p элемента из другого (k-1)-элементного частого набора - q. Причем добавляется последний элемент набора q, который по порядку выше, чем последний элемент набора p ($p.item_{k-1} < q.item_{k-1}$). При этом все k-2 элемента обоих наборов одинаковы ($p.item_1 = q.item_1, p.item_2 = q.item_2, ..., p.item_{k-2} = q.item_{k-2}$). Это может быть записано в виде SQL-подобного запроса:

```
insert into Ck
select p.item1,p.item2,...,p.itemk-1,q.itemk-1
from Lk-1p,Lk-1q
where p.item1 = q.item1,p.item2 = q.item2,...,p.itemk-2 = q.itemk-2,p.itemk-1 < q.itemk-1
```

Шаг 5. Для каждой транзакции T из множества D выбрать кандидатов C_t из множества C_k, присутствующих в транзакции T. Для каждого набора из построенного множества C_k удалить набор, если хотя бы одно из его (k-1) подмножеств не является часто встречающимся т.е. отсутствует во множестве L_{k-1}. Это можно записать в виде следующего псевдокода:

```
Для всех наборов c ∈ Ck выполнить
  для всех (k-1)-поднаборов s из c выполнить
    если (s ∉ Lk-1), то
      удалить его из Ck
```

Шаг 6. Для каждого кандидата из C_k увеличить значение поддержки на единицу.
Шаг 7. Выбрать только кандидатов L_k из множества C_k, у

которых значение поддержки больше заданной пользователем $Supp_{min}$. Вернуться к шагу 2. Результатом работы алгоритма является объединение всех множеств L_k для всех k.

Association rule learning http://en.wikipedia.org/wiki/Association_rules

Apriori - масштабируемый алгоритм поиска ассоциативных правил http://basegroup.ru/library/analysis/association_rules/apriori/

Применение ассоциативных правил для стимулирования продаж http://www.basegroup.ru/library/practice/salepromotion/

Поиск ассоциативных правил в Data Mining http://ami.nstu.ru/~vms/lecture/data_mining/rules.htm

Лекция 6 - Секвенциальный анализ

Постановка задачи

Задача поиска ассоциативных правил предполагает отыскание частых наборов в большом числе наборов данных.
В контексте анализы рыночной корзины это поиск наборов товаров, которые наиболее часто покупаются вместе.
В задаче не учитывался такой атрибут транзакции как время. Тем не менее, взаимосвязь событий во времени также представляет большой интерес.
Основываясь на том, какие события чаще всего следуют за другими, можно заранее предсказывать их появление, что позволит принимать более правильные решения.

Отличие поиска ассоциативных правил от секвенциального анализа (анализа последовательностей) в том, что в первом случае ищется набор объектов в рамках одной транзакции, т.е. такие товары, которые чаще всего покупаются ВМЕСТЕ. В одно время, за одну транзакцию.
Во втором же случае ищутся не часто встречающиеся наборы, а часто встречающиеся последовательности.
Т.е. в какой последовательности покупаются товары или через какой промежуток времени после покупки товара "А", человек наиболее склонен купить товар "Б". Т.е. данные по одному и тому же клиенту, но взятые из разных транзакций.

Получаемые закономерности в действиях покупателей можно использовать для формирования более выгодного предложения, стимулирования продаж определённых товаров, управления запасами и т.п.
Секвенциальный анализ актуален и для телекоммуникационных компаний. Основная проблема, для решения которой он используется, - это анализ данных об авариях на различных узлах телекоммуникационной сети. Информация о последовательности совершения аварий может помочь в обнаружении неполадок и предупреждении новых аварий.

Введём некоторые обозначения и определения.

- D - множество всех транзакций T, где каждая транзакция характеризуется уникальным идентификатором покупателя, временем транзакции и идентификатором объекта (id товара);
- I - множество всех объектов (товаров) общим числом m;
- s_i - набор, состоящий из элементов множества I;
- S - последовательность, состоящая из различных наборов s_i;

Дальнейшие рассуждения строятся на том, что в любой случайно выбранный момент времени у покупателя не может быть более одной транзакции.
Шаблон последовательности - это последовательность наборов, которая часто встречается в транзакциях (в определённом порядке).
Последовательность $< a_1,a_2,...,a_n >$ **является входящей** в последовательность $< b_1,b_2,...,b_n >$, если существуют такие $i_1 < i_2 < ... < i_n$, при которых $a_1 \subseteq b_{i1}, a_2 \subseteq b_{i2}, ...a_n \subseteq b_{in}$.
Например, последовательность <(3)(6,7,9)(7,9)> входит в <(2)(3)(6,7,8,9)(7)(7,9)>,
поскольку $(3) \subseteq (3), (6,7,9) \subseteq (6,7,8,9), (7,9) \subseteq (7,9)$. Хотя последовательность <(2)(3)> не входит в <(2,3)>, так как исходная последовательность говорит о том, что "3" был куплен после "2", а вторая, что товары "2" и "3" были куплены вместе.
В приведённом примере цифрами обозначены идентификаторы товаров.
В ходе анализа последовательностей нас будут интересовать такие последовательности, которые не входят в более длинные последовательности.
Подержка последовательности - это отношение числа покупателей, в чьих транзакциях присутствует указанная последовательность к общему числу покупателей.
Также как и в задаче поиска ассоциативных правил применяется минимальная и максимальная поддержка. Минимальная поддержка позволяет исключить из рассмотрения последовательности, которые не являются частыми. Максимальная поддержка исключает очевидные закономерности в появлении последовательностей. Оба параметра задаются пользователем до начала работы алгоритма.

Алгоритм AprioriALL

Существует большое число разновидностей алгоритма Apriori, который изначально не учитывал временную составляющую в наборах данных.
Первым алгоритмом на основе Apriori, позволившим находить закономерности в последовательностях событий, стал предложенный в 1995 году (Argwal и Srikant) алгоритм AprioriALL.
Данный алгоритм, также как другие усовершенствования Apriori основывается на утверждении, что последовательность, входящая в часто встречающуюся последовательность, также является часто встречающейся.
Формат данных, с которыми работает алгоритм уже был описан выше. Это таблица транзакций с тремя атрибутами (id клиента, время транзакции, id товаров в наборе).
Работа алгоритма состоит из нескольких фаз.
Фаза сортировки заключается в перегруппировке записей в таблице транзакций. Сперва записи сортируются по уникальному ключу покупателя, а затем по времени внутри каждой группы. Пример отсортированной таблицы приведён ниже.

Идентификатор покупателя	Время транзакции	Идентификаторы купленных товаров
1	Окт 23 08	30
1	Окт 28 08	90
2	Окт 18 08	10, 20
2	Окт 21 08	30
2	Окт 27 08	40, 60, 70
3	Окт 15 08	30, 50, 70
4	Окт 8 08	30
4	Окт 16 08	40, 70
4	Окт 25 08	90
5	Окт 20 08	90

Фаза отбора кандидатов - в исходном наборе данных производится поиск одно-элементных последовательностей в

соответствии со значением минимальной поддержки. Предположим, что значение минимальной поддержки 40%. Обратим внимание, что поддержка рассчитывается не из числа транзакций, в которые входит одно-элементная последовательность (в данном случае это есть набор), но из числа покупателей у которых во всех их транзакциях встречается данная последовательность.
В результате получим следующие одно-элементные последовательности.

Последовательности	**Идентификатор последовательности**
(30)	1
(40)	2
(70)	3
(40, 70)	4
(90)	5

Фаза трансформации. В ходе работы алгоритма нам многократно придётся вычислять, присутствует ли последовательность в транзакциях покупателя. Последовательность может быть достаточно велика, поэтому, для ускорения процесса вычислений, преобразуем последовательности, содержащиеся в транзакциях пользователей в иную форму.
Заменим каждую транзакцию набором одно-элементных последовательностей, которые в ней содержатся. Если в транзакции отсутствуют последовательности, отобранные на предыдущем шаге, то данная транзакция не учитывается и в результирующую таблицу не попадает.
Например, для покупателя с идентификатором 2, транзакция (10, 20) не будет преобразована, поскольку не содержит одно-элементных последовательностей с нужным значением минимальной поддержки (данный набор встречается только у одного покупателя).
А транзакция (40, 60, 70) будет заменена набором одно-элементных последовательностей {(40), (70), (40, 70)}
Процесс преобразованная будет иметь следующий вид.

Идентификатор покупателя	Последовательности в покупках	Отобранные последовательности	Преобразованные последовательности
1	<(30)(90)>	<{(30)}{(90)}>	<{1}{5}>
2	<(10, 20)(30)(40, 60, 70)>	<{(30)}{(40)(70)(40, 70)}>	<{1}{2, 3, 4}>
3	<(30, 50, 70)>	<{(30)(70)}>	<{1, 3}>
4	<(30)(40, 70)(90)>	<{(30)}{(40)(70)(40, 70)}{(90)}>	<{1}{2, 3, 4}{5}>
5	<(90)>	<{(90)}>	<{5}>

Фаза генерации последовательностей - из полученных на предыдущих шагах последовательностей строятся более длинные шаблоны последовательностей.
Фаза максимизации - среди имеющихся последовательностей находим такие, которые не входят в более длинные последовательности.
Проиллюстрируем две последние фазы на примере. Пусть после фазы трансформации имеется таблица с последовательностями покупок для пяти покупателей.

<{1,5}{2}{3}{4}>
<{1}{3}{4}{3,5}>
<{1}{2}{3}{4}>
<{1}{3}{5}>
<{4}{5}>

Значение минимальной поддержки выберем 40% (последовательность должна наблюдаться как минимум у двоих покупателей из пяти).
После фазы отбора кандидатов мы получили таблицу с одно-элементными последовательностями.

1-Последовательность L_1	**Поддержка**
<1>	4
<2>	2
<3>	4
<4>	4
<5>	5

В фазе генерации последовательностей из исходных одно-элементных последовательностей сгенерируем двух-элементные и посчитаем для них поддержку. Оставим только те, поддержка которых больше минимальной. После этого сгенерируем трёх, четырёх и т.д. элементные последовательности, пока это будет возможно.

2-Последовательность L_2	**Поддержка**
<1 2>	2
<1 3>	4
<1 4>	3
<1 5>	3
<2 3>	2
<2 4>	2
<3 4>	3
<3 5>	2
<4 5>	2

3-Последовательность L_3	**Поддержка**
<1 2 3>	2
<1 2 4>	2
<1 3 4>	3
<1 3 5>	2
<2 3 4>	2

4-Кандидаты	4-Последовательность L_4	Поддержка
<1 2 3 4>	<1 2 3 4>	2
<1 2 4 3>		
<1 3 4 5>		
<1 3 5 4>		

Последовательность <1 2 4 3>, например, не проходит отбор, поскольку последовательность <2 4 3>, входящая в неё, не присутствует в L_3.
Так как сформировать пяти-элементные последовательности невозможно, работа алгоритма на этом завершается. Результатом его работы будут три последовательности, удовлетворяющие значению минимальной поддержки и не входящие в более длинные последовательности: <1 2 3 4>, <1 3 5> и <4 5>.

Алгоритм в превдо-коде.
L_1 = {одно-элементные последовательности, удовлетворяющие минимальному значению поддержки} //Результат фазы отбора кандидатов (L-itemset)

for (k=2; $L_{k-1} \neq 0$; k++)

begin
C_k = Генерация кандидатов из L_{k-1}
foreach последовательность покупок *c* клиента в базе данных **do**
Увеличить на единицу поддержку всех кандидатов из C_k, которые содержатся в *c*
L_k = Кандидаты из C_k с минимальной поддержкой.
end

Результат: последовательности максимальной длинны (не входящие в другие последовательности) в $\bigcup_k L_k$;
Процедура генерации кандидатов выполняется следующим образом:
insert into C_k
select $p.itemset_1,...,p.itemset_{k-1},q.itemset_{k-1}$
from $L_{k-1}p,L_{k-1}q$
where $p.itemset_1 = q.itemset_1, ..., p_itemset_{k-2} = q.itemset_{k-2}$;

После чего удалить все последовательности $c \in C_k$, такие что (k-1) последовательность из *c* не входит в L_{k-1}.

Алгоритм GSP

Ограничения AprioriAll

Рассмотренный алгоритм AprioriAll позволяет находить взаимосвязи в последовательностях данных. Это стало возможно после введения на множестве наборов данных отношения порядка (в примере с анализом покупок стало учитываться время транзакции). Тем не менее, AprioriAll не позволяет определить характер взаимосвязи, её силу.
При поиске зависимостей в данных нас могут интересовать только такие, где одни события наступают вскоре после других. Если же этот промежуток времени достаточно велик, то такая зависимость может не представлять значения.
Проиллюстрируем сказанное на примере.
Книжный клуб скорее всего не заинтересует тот факт, что человек, купивший "Основание" Азимова, спустя три года купил "Основатели и Империя". Их могут интересовать покупки, интервал между которыми составляет, например, три месяца.
Каждая совершённая покупка - это элемент последовательности. Последовательность состоит из одного и более элементов. Во многих случаях не имеет значения, если бы наборы товаров, содержащиеся в элементе последовательности, входили не одну покупку (транзакцию), а составляли бы несколько покупок. При условии, что время транзакций (покупок) укладывалось бы в определённый интервал времени (окно).
Например, если книжный клуб установит значение окна равным одной неделе, то клиент, заказавший "Основание" в понедельник, "Мир-Кольцо" в субботу, и затем "Основатели и Империя" и "Инженеры Мира-Кольцо" (последние две книги в одном заказе) в течении недели, по-прежнему будет поддерживать правило 'Если "Основание" и "Мир-Кольцо", то "Основатели и Империя" и "Инженеры Мира-Кольцо"'.
Ещё одним ограничением алгоритма AprioriAll является отсутствие группировки данных. Алгоритм не учитывает их структуру. В приведённом выше примере можно было бы находить правила, соответствующие не отдельным книгам, а

также авторам или литературным жанрам.

Описание входных данных и другие определения
Множество объектов данных обозначим как $I = i_1,i_2,...,i_m$.
Пусть *T* - ориентированный граф на множестве I, задающий таксономию.
ТАКСОНОМИЯ [taxonomy] — теория и результат классификации и систематизации сложных систем, обычно иерархической структуры. Выделенные для исследования элементы и группы объектов, подсистемы называются таксонами.
Если существует ребро из вершины *p* к вершине *c*, то мы говорим, что *p* является родителем для *c*, а *c* является потомком *p*. Таким образом, *p*является обобщением для *c*.

Для применения алгоритма нам потребуется множество последовательностей D. Каждая последовательность представляет собой список транзакций. Транзакции в последовательности упорядочены по времени согласно значению соответствующего поля в таблице транзакций. Поля таблицы транзакций: идентификатор последовательности, идентификатор транзакции, время транзакции, объекты, присутствующие в транзакции. Как и в случае с AprioriAll, мы предполагаем, что никакая последовательность не содержит транзакций с одинаковым полем "время транзакции". Также нас не интересует сколько раз каждый объект встречается в отдельно взятой транзакции. Важно лишь присутствие или отсутствие.
Поддержка последовательности - это число последовательностей, содержащих указанную последовательность, отнесённое к общему числу последовательностей. В случае с алгоритмом GSP требуется учитывать дополнительные условия, чтобы определить, содержит ли последовательность указанную последовательность (подпоследовательность).
По аналогии с алгоритмом AprioriAll, последовательность содержит подпоследовательность (назовём её *s*), если она содержит все элементы подпоследовательности *s*. Добавим к этому определению понятие таксономии. Так транзакция T содержит объект $x \in I$ если x присутствует в T,

или x является потомком какого-либо элемента из T. Транзакция T содержит последовательность (одно-элементную) $y \subseteq I$ если в транзакции T присутствуют все объекты из *y*.
Последовательность $d = < d_1 ... d_m >$ содержит последовательность $s = < s_1 ... s_m >$, если существуют такие целые числа $i_1 < i_2 < \ ... \ < i_n$ что s_1 входит в d_{i_1},s_2 входит в d_{i_2}, s_n входит в d_{i_n}.
Теперь добавим к определению понятие скользящего окна. Допускается, что элемент последовательности может состоять не из одной, а из нескольких транзакций, если разница во времени между ними меньше чем размер окна. Формально это можно выразить следующим образом.
Последовательность $d = < d_1 ... d_m >$ содержит последовательность $s = < s_1 ... s_m >$, если существуют такие целые числа $l_1 \leq u_1 < l_2 \leq u_2 < ... < l_n \leq u_n$, что:

1. s_i содержится в $\bigcup_{k=l_i}^{u_i} d_k, 1 \leq i \leq n$,
2. Разница во времени транзакций d_{u_i} и $d_{l_i} \leq$ размера окна, $1 \leq i \leq n$.

Также мы упоминали, что бывает необходимо задавать временные рамки для наборов транзакций, которые представляют собой упорядоченные элементы последовательности. В примере с книгами, это означает, что нас будут интересовать правила на основе таких транзакций, где время между покупками больше некого минимального значения и меньше выбранного максимального значения (например от одного месяца до трёх). Сформулируем условия для транзакций, удовлетворяющих приведённым критериям. Для заданных параметров размера окна, минимального и максимального значения временного интервала между транзакциями последовательность $d = < d_1 ... d_m >$ содержит последовательность $s = < s_1 ... s_m >$, если существуют такие целые числа $l_1 \leq u_1 < l_2 \leq u_2 < ... < l_n \leq u_n$, что:

1. s_i содержится в $\bigcup_{k=l_i}^{u_i} d_k, 1 \leq i \leq n$,
2. Разница во времени транзакций d_{u_i} и $d_{l_i} \leq$ размера окна, $1 \leq i \leq n$.

3. Время транзакции d_{l_i} минус время транзакции $d_{u_{i-1}} >$ min-gap, $2 \leq i \leq n$,
4. Время транзакции d_{u_i} минус время транзакции $d_{l_{i-1}} \leq$ max-gap, $2 \leq i \leq n$.

При отсутствии группировки данных (таксономии), min-gap = 0, max-gap = ∞, размер окна = 0. Где каждый элемент последовательности представляет собой набор объектов из одной транзакции. Подобный случай описывался при рассмотрении алгоритма AprioriAll.
Пример. Рассмотрим набор исходных данных и попытаемся найти правила, добавляя в условие задачи понятие окна, минимального и максимального интервала времени между элементами последовательности, таксономи.

Идентификатор последовательности	Время транзакции	Наборы
C1	1	Ringworld
C1	2	Foundation
C1	15	Ringworld Engineers, Second Foundation
C2	1	Foundation, Ringworld
C2	20	Foundation and Empire
C2	50	Ringworld Engineers

Значение минимальной поддержки установим, к примеру, равным двум.
В самом простом случае (без учёта размера окна, таксономий и проч.) на основе данных можно выделить лишь одну двух-элементную последовательность: <(Ringworld)(Ringworld Engineers)>.
Теперь добавим скользящее окно, размер которого установим в семь дней.

Получим ещё одну часто встречающуюся последовательность <(Foundation, Ringworld)(Ringworld Engineers)>.
Теперь добавим к условию задачи max-gap=30 дней. Две

найденные последовательности в этом случае не поддерживаются покупателем C2.
Если же просто добавить таксономию, но не задавать значение скользящего окна и max-gap, то получим правило <(Foundation)(Asimov)>.

Описание работы алгоритма

В общем виде работа алгоритма заключается в нескольких прходах по исходному набору данных. На первом проходе алгоритм подсчитывает поддержку для каждого элемента (число последовательностей, в которые входит элемент). В выполнения данного шага алгоритм выделяет из исходного набора элементов часто встречающиеся элементы, удовлетворяющие значению минимальной поддержки. Каждый подобный элемент на первом шаге представляет собой одно-элементную последовательность. В начале каждого последующего прохода имеется некоторое число часто встречающихся последовательностей, выявленных на предыдущем шаге алгоритма. Из них будут формироваться более длинные последовательности, называемые кандидатами. Каждый кандидат - это последовательность, которая на один элемент длиннее чем те из которых кандидат был сформирован. Таким образом, число элементов всех кандидатов одинаково. После этого рассчитывается поддержка для каждого кандидата. В конце шага станет известно, какие из последовательностей кандидатов на самом деле являются частыми. Найденные частые последовательности послужат исходными данными для следующего шага алгоритма. Работа алгоритма завершается тогда, когда не найдено ни одной новой частой последовательности в конце очередного шага, или когда невозможно сформировать новых кандидатов.
Таким образом, в работе алгоритма можно выделить две основные операции:

- генерация кандидатов;
- подсчёт поддержки кандидатов.

Рассмотрим эти операции более подробно.

Генерация кандидатов

Последовательность, состоящую из k элементов будем называть k-последовательностью.
Пусть L_k - содержит все частые k-последовательности, а C_k набор кандидатов из k-последовательностей.
В начале каждого шага мы располагаем L_{k-1} - набором частых (k-1)-последовательностей. Необходимо на их основе построить набор всех частых k-последовательностей.
Введём определение смежной подпоследовательности.
При наличии последовательности $s = < s_1 s_2 ... s_n >$ и подпоследовательности *c*, *c* будет являться смежной последовательностью *s*, если соблюдается одно из условий:

1. *c* получается из *s* при удалении элемента из *s* первого (s_1) или последнего (s_n).
2. *c* получается из *s* при удалении одного объекта из элемента s_i, если в его составе не менее двух объектов.
3. *c* - смежная подпоследовательность *c`*, если *c`* смежная подпоследовательность *s*.

Например, дана последовательность s=<(1,2)(3,4)(5)(6)>. Последовательности <(2)(3,4)(5)>, <(1,2)(3)(5)(6)> и <(3)(5)> являются смежными подпоследовательностями *s*.
А последовательности <(1,2)(3,4)(6)> и <(1)(5)(6)> таковыми не являются.
Генерация кандидатов происходит в две фазы.

- **Фаза объединения**. Мы создаём последовательности кандидаты путём объединения двух последовательностей L_{k-1}.

Последовательность s_1 объединяется с s_2 если подпоследовательность, образующаяся путём удаления объекта из первого элемента s_1 будет та же, что и в случае удаления объекта из последнего элемента s_2. Объединение последовательностей происходит путём добавления к s_1последнего элемента из s_2. При объединении L_1 с L_1 нам нужно также добавить элемент к s_2 как дополнительный объект к последнему элементу, так и в качестве отдельного элемента, так как <(x y)> и <(x)(y)> дадут одну и ту же последовательность <(y)> при удалении первого объекта. Таким образом исходные последовательности s_1 и s_2 являются

смежными подпоследовательностями для новой последовательности-кандидата.

- **Фаза очистки.** Удаляем последовательности-кандидаты которые содержат смежные (k-1)-подпоследовательности, чья поддержка меньше минимально допустимой. Если параметр max-gap не задан, то также удаляются кандидаты, содержащие последовательности (в т.ч. не смежные), поддержка которых меньше минимально допустимой.

  Пример.

Частые 3-последовательности	**Кандидаты 4-последовательности**	
	После объединения	**После очистки**
<(1,2)(3)> <(1,2)(4)> <(1)(3,4)> <(1,3)(5)> <(2)(3,4)> <(2)(3)(5)>	<(1,2)(3,4)> <(1,2)(3)(5)>	<(1,2)(3,4)>

В таблице показаны L_3 и C_4 после завершения обеих фаз. В фазе объединения последовательность <(1,2)(3)> объединяется с <(2)(3,4)> и образует <(1,2)(3,4)>, а с последовательностью <(2)(3)(5)> - <(1,2)(3)(5)>. Оставшиеся последовательности не объединяются друг с другом. Например, последовательность <(1,2)(4)> не может быть объединена ни с одной другой из L_3 так как не существует последовательности вида <(2)(4 x)> или <(2)(4)(x)>. В фазе очистки последовательность <(1,2)(3)(5)> удаляется, поскольку её смежная подпоследовательность <(1)(3)(5)> не входит в L_3, а значит не является часто встречающейся.

Подсчёт поддержки кандидатов

Делая проход по исходным данным (сканируя наборы последовательностей), мы обрабатываем последовательности по одной. Для тех кандидатов, которые содержатся в обрабатываемой последовательности, увеличиваем значение

поддержки на единицу. Другими словами, имея набор кандидатов *C* и набор исходных данных (последовательностей) *d*, мы ищем все последовательности из *C*, которые содержатся в *d*.
Рассмотрим на примерах как происходит поиск кандидатов в исходных последовательностях данных.
Пусть исходный набор данных имеет вид:

Время транзакции	Объекты
10	1, 2
25	4, 6
45	3
50	1, 2
65	3
90	2, 4
95	6

Пусть max-gap = 30, min-gap = 5, размер окна равен нулю.
Для последовательности-кандидата <(1,2)(3)(4)> мы сперва найдём вхождение элемента (1,2) в транзакции с временной отметкой 10, затем найдём элемент (3) в транзакции с отметкой 45. Разница во времени между двумя этими элементами составляет 35 дней, что больше чем значение max-gap. Данные значения нас не устраивают. Продолжаем поиск элемента (1,2) с временной отметки 15, потому что временная отметка для последнего найденного элемента (3) равна 45, а значение max-gap = 30. Если (1,2) встретился бы раньше временной отметки 15, то разница во времени между (3) и (1,2) всё равно была бы больше max-gap. Находим (1,2) в транзакции с временной отметкой 50. Так как это первый элемент последовательности-кандидата, то проводить сравнение max-gap не с чем. Двигаемся дальше. Так как (3) не встречается в ближайшие 5 дней (50 + min-gap), продолжаем поиск элемента (3) после значения 55. Находим этот элемент в момент времени 65. 65 - 50 < 30 (max-gap). Производим поиск элемента (4). Находим его в момент времени 90. 90 - 65 < 30, так что этот вариант нас устраивает. Мы установили, что последовательность-кандидат присутствует в исходной последовательности данных, значит поддержка последовательности-кандидата увеличивается на

единицу.
Теперь приведём пример поиска одно-элементной последовательности-кандидата в исходном наборе данных. В этот раз установим значение размера окна равным семи дням. Допустим, нам необходимо найти вхождение элемента (2,6) после временной отметки 20. Находим "2" в момент времени 50 и "6" в момент времени 25. Так как разница во времени между ними больше размера окна данное вхождение не считается, продолжаем поиск с момента времени 50 - 7 = 43. Помним, что объект "2" был найден в момент времени 50. Находим объект "6" в момент времени 95. Разница во времени всё ещё больше чем размер окна. Продолжаем поиск с позиции 95 - 7 = 88. Находим "2" в момент времени 90, при том, что объект "6" был найден в момент времени 95. Разница между ними меньше размера окна. На этом поиск окончен.Кандидат (2,6) присутствует в последовательности данных.

Иерархия данных. Таксономия

Таксономия отражает структуру в данных. В примере с книгами можно задать ориентированный граф, вершиной которого будет являться "Science Fiction", двумя узлами, связанными с вершиной будет "Asimov" и "Niven". Каждый из этих узлов будет связан с узлами "Foundation", "Foundation and Empire", "Second Foundation" и "Ringworld", "Ringworld Engineers" соответственно.
Для учёта этих данных последовательность вида <(Foundation, Ringworld)(Second Foundation)> будет расширена до <(Foundation, Ringworld, Asimov, Niven, Science Fiction)(Second Foundation, Asimov, Science Fiction)>. После чего можно использовать алгоритм GSP для анализа полученной последовательности данных.

- Алгоритм MSAP

Лекция 7- Кластеризация. Типы алгоритмов

Постановка задачи

Что делать, если нет обучающего материала для построения классификатора? То есть нет учителя, который покажет, как следует классифицировать тот или иной объект?

В этом случае следует прибегнуть к кластеризации (или кластерному анализу). Кластеризация - это обучение без учителя. При этот она выполняет схожие с классификацией задачи: позволяет создать определенные правила, с помощью которых в дальнейшем можно относить объекты к различным классам (группам). Однако, в отличие от классификации, кластеризация эти группы еще и выявляет в наборе объектов различными способами. Объект группируются, исходя из их сходства, или близости.

Общий алгоритм кластеризации выглядит так:

1. Приведение исходных данных к нужному виду (подготовка данных);
2. Выбор меры близости;
3. Выбор алгоритма (метаалгоритма) кластеризации;
4. Выполнение алгоритма;
5. Представление полученных результатов;
6. Интерпретация полученных результатов.

Рассмотрим каждый из этапов более подробно.

На первом этапе происходит подготовка данных к кластеризации. Данные для кластеризации чаще всего представляют в виде таблиц, где каждый столбец - это один из атрибутов, строка - объект данных.

На втором этапе выбирают, как охарактеризовать сходство объектов. Для этого используются различные меры близости, то есть, фактически, оценки близости двух объектов друг к другу. Меры близости выбирают, исходя из свойств объектов. Так, популярной мерой близости является декартово расстояние (в двумерном случае): $d^2(< x_1,y_1 > , < x_2,y_2 >) = sqrt(x_1 - x_2)^2 + (y_1 - y_2)^2$ или метрика Минковского в многомерном случае: $d^n(x,y) = | | X,Y | |$ Это достаточно хорошие меры близости для представимых на координатной плоскости значений. Для

нечисленных атрибутов подбирают такие меры близости, которые позволяют свести их к численным и сравнить. Так, основным расстоянием для строк является метрика Левенштейна, которая устанавливает расстояние между двумя строками равным количеству перестановок, которые необходимо совершить, чтобы превратить одну строку в другую. Мера близости подбирается индивидуально для конкретных типов данных. Иногда адекватной меры близости подобрать не удается, и приходится ее придумывать самим.

На третьем этапе выбирают алгоритм, по которому мы будем строить модель данных, то есть группировать объекты. Выбор алгоритма сложен, и зачастую приходится использовать несколько алгоритмов прежде, чем будет получен нужный (интерпретируемый) результат. Иногда алгоритмы кластеризации комбинируют, чтобы получить метаалгоритм, результат выполнения одного когда служит промежуточным результатом выполнения другого.

На четвертом этапе алгоритм реализуется, и его результатом является построенная модель данных, то есть группировка объектов по кластерам.

На пятом этапе полученную группировку пытаются представить в наиболее удобном для интерпретации виде. Алгоритмы кластеризации на выходе выдают только группы и объекты, к ним принадлежащие. Но для человека наиболее интересным является не это чаще всего, а то, исходя из чего - каких свойств объекта - эти объекты были отнесены к определенной группе. Представление результатов кластеризации призвано помочь наиболее точно интерпретировать результаты выполнения алгоритма.

И, наконец, на последнем этапе кластеризации результаты выполнения алгоритма интерпретируются, из них получается знание, то есть полезные правила, которые можно использовать в дальнейшем для отнесения новых объектов к той или иной группе - кластеру.

Классификация алгоритмов

- Иерархические алгоритмы;
- Агломеративные алгоритмы;

- Дивизимные алгоритмы.
- Неиерархические алгоритмы
- По методу
- Итеративные
- Плотностные
- Модельные
- Концептуальные
- Сетевые.

Принято делить все алгоритмы кластеризации на иерархические и неиерархические. Деление это происходит по выдаваемым на выходе данным. Иерархические алгоритмы на выходе выают некую иерархию кластеров, и мы вольны выбрать любой уровень этой иерархии для того, чтобы интерпретировать результаты алгоритма. Неиерархические - это, фактически, все алгоритмы, которые на выходе иерархию не выдают (или выбор интерпретации происходит не по уровню иерархии).

Иерархические алгоритмы

Иерархические алгоритмы делятся на агломеративные и дивизимные. Агломеративные алгоритмы - это алгоритмы, которые начинают свое выполнение с того, что каждый объект заносят в свой собственный кластер и по мере выполнения объединяют кластеры, до тех пор, пока в конце не получает один кластер, включающий в себя все объекты набора. Дивизимные алгоритмы, напротив, сначала относят все объекты в один кластер и затем разделяют этот кластер до тех пор, пока каждый объект не окажется в своем собственном кластере.

Представление результатов иерархического алгоритма

Представлением результата иерархического алгоритма является дендрограмма - схема, показывающая, в какой последовательности происходило слияние объектов в кластер/разделение объектов на кластеры.

Алгоритм ближайшего соседа

Достаточно ярким примером иерархического агломеративного алгоритма является алгоритм "соседей". Это алгоритмы ближнего, дальнего и среднего соседей. Он объединяет

кластеры, исходя из расстояния между ближайшими, наиболее удаленными или центральными объектами кластеров. Рассмотрим схему выполнения алгоритма ближайшего соседа:

1. Составление матрицы попарных расстояний между объектами. Каждому объекту назначается свой кластер;
2. Нахождение в матрице наименьшего элемента (то есть наименьшего расстояния между соседями);
3. Объединение кластеров, в которые входят объекты, имеющие наименьшее расстояние.
4. Проверка: сколько осталось кластеров. Если один, то завершить алгоритм. Если два и более, то перейти к шагу 1.

Результаты выполнения этого алгоритма хорошо представимы в виде дендрограммы, а кластеры на каждом из этапов его выполнения легко получимы путем проведения линии, перпедикулярной направлению распространения этой дендрограммы.

Лекция 8 - Кластеризация. Итеративные и плотностные алгоритмы

Итеративные алгоритмы

Итеративные алгоритмы называются так потому, что итеративно перераспределяют объекты между кластерами.

Алгоритм k-means

Общая идея алгоритмов *-means: Минимизация расстояний между объектами в кластерах. Останов происходит, когда минимизировать расстояния больше уже невозможно. Минимизируемая функция в случае k-means такова: $J = \sum_{k=1}^{M} \sum_{i=1}^{N} d^2(x_i, c_k),$ $x_i \in X$ - объект кластеризации (точка) $c_j \in C$ - центр кластера (центроид). $|X| = N, |C| = M$

На момент старта алгоритма должно быть известно число C (количество кластеров). Выбор числа C может базироваться на результатах предшествующих исследований, теоретических соображениях или интуиции.

Описание алгоритма 0. Первоначальное распределение объектов по кластерам. Выбираются C точек. На первом шаге эти точки считаются центрами кластеров. Выбор начальных центроидов может осуществляться путем подбора наблюдений для максимизации начального расстояния, случайным выбором наблюдений или выбором первых наблюдений.

1. Итеративное перераспределение объектов по кластерам. Объекты распределяются по кластерам путем подсчета расстояния от объекта до центров кластеров и выбора наименьшего.

2. Когда все объекты распределены по кластерам, заново считаются их центры. $c_j = \frac{\sum_{i=1}^{L} x_i}{L}, x_i \in C_j, |C_j| = L$ (можно считать по каждой координате отдельно)

3. Если $c_j = c_{j-1}$, то это означает, что кластерные центры стабилизировались и соответственно распределение закончено. Иначе переходим к шагу 1.

Сложным является выбор числа кластеров. В случае, если предположений нет, обычно делают несколько попыток,

сравнивая результаты (скажем, сначала 2, потом 3 и т.д.). Проверка качества кластеризации После получений результатов кластерного анализа методом k-средних следует проверить правильность кластеризации (т.е. оценить, насколько кластеры отличаются друг от друга). Для этого рассчитываются средние значения для каждого кластера. При хорошей кластеризации должны быть получены сильно отличающиеся средние для всех измерений или хотя бы большей их части. Достоинства алгоритма k-средних:

- простота использования;
- быстрота использования;
- понятность и прозрачность алгоритма.

Недостатки алгоритма k-средних:

- алгоритм слишком чувствителен к выбросам, которые могут искажать среднее.
- Возможным решением этой проблемы является использование модификации алгоритма - алгоритм k-медианы;
- алгоритм может медленно работать на больших базах данных. Возможным решением

данной проблемы является использование выборки данных.

Более строгой интерпретацией этого алгоритма является алгоритм **hard c-means**. Его отличия - в минимизируемой функции и строгости самого алгоритма:

$J = \sum_{i=1}^{N} \sum_{k=1}^{M} u_{ij} d^2(x_i, c_k),$ $u_{ij} = 1$, если $x_i \in C_j$, и $u_{ij} = 0$,, если нет. То есть минимизируется расстояние от точек до центроида, а не от центроида до точек.

Тогда формула центроида тоже несколько меняется:

$c_j = \frac{\sum_{i=1}^{N} u_{ij} x_i}{\sum_{i=1}^{N} u_{ij}}$

Сам же метод не меняется.

Farthest First - еще одна модификация k-means, особенностью его является изначальный выбор центроидов - от 2 и выше они выбираются по принципу удаленности от

остальных(центроидом выбирается точка, наиболее отдаленная от остальных центроидов).

Алгоритм Fuzzy C-Means

Нечеткие методы - это методы, основанные не на бинарной логике - где все четко - элемент либо принадлежит одному кластеру, либо другому - а на предположении, что каждый элемент в какой-то степени принадлежит определенному кластеру. m - мера нечеткости, она как раз определяет нечеткость алгоритма. Минимизируемая функция почти аналогична hard c-means:

$$J = \sum_{i=1}^{N}\sum_{k=1}^{M} u_{ij}^{m} d(x_i, c_k),$$

0. Выбираем число классов M, меру нечеткости $1 < m < \infty$, функцию расстояний d(c,x) (обычно $d(c,x) = ||x - c||^2$), критерий окончания поиска $0 < \varepsilon < 1$. Задаем матрицу $U^0 = \{u_{ij}(x_i, c_j) : x_i \in X, c_j \in C\}$ весов принадлежности точки $x_i \in X$к кластеру $j \in C$ с центром c_j для всех точек и кластеров (можно взять принадлежности к кластеру из k-means или просто по рандому, ограничения $0 < u_{ij} < 1, 0 < \sum_{i=1}^{N} u_{ij} < N$ для $\forall 0 \le j \le C, \sum_{j=1}^{C} u_{ij} = 1$ для $\forall 0 \le i \le N$ (вытекают в общем из определения нечеткой принадлежности)).

1. Вычисляем центроиды :

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^{m} x_i}{\sum_{i=1}^{N} u_{ij}^{m}}$$

2. Перевычисляем веса :

$$u_{ij} = \frac{1}{\sum_{k=1}^{M} \left(\frac{d(c_j, x_i)}{d(c_k, x_i)}\right)^{2/(m-1)}}.$$

3. Проверяем $||U^k - U^{k-1}|| < \varepsilon$ (чаще всего достаточно $max_{ij}||u_{ij}^{k} - u_{ij}^{k-1}|| < \epsilon$)- если да, то заканчиваем, если нет, то переходим к шагу 1.

Плотностные алгоритмы

Другой класс алгоритмов - плотностные. Они так называются потому, что определяют кластер как группу объектов,расположенных достаточно кучно. Под кучностью понимают то, что в эпсилон-окрестности точки есть некоторое

минимальное количество других объектов: $d(x_i,x_j) < \varepsilon$ для некоторого количества j > Minpts.

Алгоритм DBSCAN

Алгоритм DBSCAN обычно проводится над данными, упорядоченными в R-деревья (для удобства выборки окрестных точек). Но в общем случае этого не требует.

0. Выбираем окрестность ε, в которой мы будем требовать наличия Minpts объектов, и сам Minpts.

1. Берем произвольный объект еще не обработанный объект. Проверяем для него, что в эпсилон-окрестности точки есть некоторое минимальное количество других объектов: $d(x_i,x_j) < \varepsilon$ для некоторого количества j > Minpts. Если это не так, то очевидно, что эта точка - шум. Берем следующую.

2. Если это не так, то помечаем эту точку как принадлежащую кластеру. Это так называемая корневая точка. Заносим окружающие ее точки в отдельную категорию.

2.1. Для каждой еще не обработанной точки из этой категории сначала помечаем ее как принадлежащую кластеру, а затем проверяем то же самое: что в эпсилон-окрестности точки есть некоторое минимальное количество других объектов: $d(x_i,x_j) < \varepsilon$ для некоторого количества j > Minpts. Если это так, то заносим эти точки в эту же категорию.

2.2. После проверки выносим точку из этой временной категории. Очевидно, что рано или поздно точки в данной категории кончатся (мы достигнем границ кластера, где правило кучности не выполняется). Тогда переходим к шагу 1. Иначе возвращаемся к шагу 2.1.

Этот алгоритм имеет сложность O(NlogN). Очевидно, что основным его недостатком является неспособность связывать кластеры через узкие места, где правило плотности не выполняется.

Лекция 9 - Кластеризация. Модельные, концептуальные, сетевые алгоритмы

Модельные алгоритмы

Модельные алгоритмы подразумевают, что у нас есть некоторая модель кластера (его структуры), и мы стремимся найти и максимизировать сходства между этой моделью и имеющимися у нас данными, то есть выделить в данных такие модели, которые и будут кластерами. Часто при этом для представления моделей используется широко разработанный аппарат математической статистики.

Алгоритм EM

Предполагается, что кроме известных нам из наших данных величин существуют еще и неизвестные нам, относящиеся к распределению по кластерам. То есть фактически эти неизвестные "создают" кластер, а мы наблюдаем только результат их деятельности. И именно эти неизвестные мы и стараемся максимально точно оценить.

Алгоритм использует широко известный метод максимизации ожиданий (Expectation Maximization). В наиболее простом случае предполагается, что кластер - это результаты наблюlения, распределенные нормально. Тогда для их характеристики можно применять многомерную функцию Гаусса (многомерное распределение Гаусса) $N(\mu, \sigma^2) = P(x_i|x_i \in w_j, \lambda), w_j -$ - одно из распределений. И тогда основная задача - это определить, к какому из распределений принадлежит каждая конкретная точка, оценив параметры этих распределений исходя из реального распределения точек.

0. Инициализируем : $\lambda^0 = \{\mu_1^0, ..., \mu_K^0, \sigma_1 P_1^0(w_1), ..., P_K^0(w_K)\}$ - среднее отклонение распределений относительно начала координат (т.е. центры кластеров) и вероятности этих распределений для каждой точки. K - число кластеров - задаем.

1. E-шаг:

$$P(\omega_j|x_i, \lambda^t) = \frac{P(x_i|\omega_j, \lambda^t)P(\omega_j|\lambda^t)}{P(x_i|\lambda^t)} = \frac{P(x_i|\omega_j, \mu_j^t, (\sigma_j^t)^2))P_j^t}{\sum_{j=1}^K P(x_i|\omega_j, \mu_j^t, (\sigma_j^t)^2))P_j^t},$$

где $P(x_i|\omega_j, \mu_j^t, (\sigma_j^t)^2)$ -функция плотности распределения.

2. M-шаг:

$$\mu_j^{t+1} = \frac{\sum_i P(\omega_j|x_i,\lambda^t)x_i}{\sum_i P(w_j|x_i,\lambda^t)}$$

$$P_j^{t+1} = \frac{\sum_i P(\omega_j|x_i,\lambda^t)}{N}$$

$$\sigma_j^{t+1} = \frac{\sum_{i=1}^N P(w_j|x_i,\lambda^t)(x_i-\mu_j)(x_i-\mu_j)^T}{\sum_{i=1}^N P(w_j|x_i,\lambda^t)}$$

3. Вычисление $P(x_i|\mu_j)^{t+1} = \prod_{i=1}^{N}\sum_{j=1}^{K} P(w_j)P(x_i|w_j,\lambda^{t+1})$ и сравнение с $P(x_i \mid \mu_j)^t$, если да, то стоп - найден локальный максимум. Если нет, то переходим снова к шагу 1.

Концептуальные алгоритмы

Алгоритм Cobweb

В алгоритме COBWEB реализовано вероятностное представление категорий. Принадлежность категории определяется не набором значений каждого свойства объекта, а вероятностью появления значения. Например, $P(A_i = \upsilon_{ij} \mid C_k)$ - это условная вероятность, с которой свойство A_i, принимает значение υ_{ij}, если объект относится к категории C_k. Для каждой категории в иерархии определены вероятности вхождения всех значений каждого свойства. При предъявлении нового экземпляра система COBWEB оценивает качество отнесения этого примера к существующей категории и модификации иерархии категорий в соответствии с новым представителем. Критерием оценки качества классификации является полезность категории (category utility). Критерий полезности категории был определен при исследовании человеческой категоризации. Он учитывает влияние категорий базового уровня и другие аспекты структуры человеческих категорий.

Критерий полезности категории максимизирует вероятность того, что два объекта, отнесенные к одной категории, имеют одинаковые значения свойств и значения свойств для объектов из различных категорий отличаются. Полезность категории определяется формулой

$$CU = \sum_k \sum_i \sum_j P(A = \upsilon_{ij} \mid C_k)P(C_k \mid A = \upsilon_{ij})P(A = \upsilon_{ij})$$

Значения суммируются по всем категориям C_k, всем свойствам A_j и всем значениям свойств υ_{ij}. Значение $P(A_j = \upsilon_{ij} \mid C_k)$ называется предсказуемостью (predictability). Это вероятность того, что объект, для которого свойство A_j- принимает значение υ_{ij}, относится к категорииC_k. Чем выше это значение, тем вероятнее, что свойства двух объектов, отнесенных к одной категории, имеют одинаковые значения. Величина$P(C_k \mid A = \upsilon_{ij})$ называется предиктивностью (predictiveness). Это вероятность того, что для объектов из категории C_k свойство A_j принимает значение υ_{ij}. Чем больше эта величина, тем менее вероятно, что для объектов, не относящихся к данной категории, это свойство будет принимать указанное значение. Значение $P(A = \upsilon_{ij})$ - это весовой коэффициент, усиливающий влияние наиболее распространенных свойств. Благодаря совместному учету этих значений высокая полезность категории означает высокую вероятность того, что объекты из одной категории обладают одинаковыми свойствами, и низкую вероятность наличия этих свойств у объектов из других категорий.

После некоторых преобразований (Байеса в частности) и некоторых агрументированных изменений мы получаем более правильную формулу:

$$CU = \frac{\sum_{k=1}^{N} P(A = v_{ij}|C_k) \sum_j \sum_i (P(C_k|A = v_{ij})^2 - P(A = v_{ij})^2)}{N}$$ N -число категорий.

Алгоритм COBWEB имеет следующий вид.

```
cobweb(Node, Instance)
Begin
 if узел Node - это лист, then
   begin
    создать два дочерних узла L1 и L2 для
узла Node;
    задать для узла L1 те же вероятности,
что и для узла Node;
    инициализировать вероятности для узла
L2 соответствующими значениями объекта
Instance;
    добавить Instance к Node, обновив
вероятности для узла Node ;
   end
```

```
  else
    begin
     добавить Instance к Node, обновив
вероятности для узла Node;
     для каждого дочернего узла C узла Node
вычислить полезность категории при отнесении
экземпляра Instance к категории C;
```

```
     пусть S1 - значение полезности для
наилучшей классификации C1;
     пусть S2 - значение для второй
наилучшей классификации C2;
     пусть S3 - значение полезности для
отнесения экземпляра к новой категории;
     пусть S4 - значение для слияния C1 и C2
в одну категорию;
     пусть S5 - значение для разделения C1
(замены дочерними категориями);
    end
   if S1 - максимальное значение CU, then
     cobweb(C1, Instance) %отнести экземпляр
к C1
   else
     if S3 - максимальное значение CU, then
       инициализировать вероятности для
новой категории Cm значениями Instance
     else
       if S4 - максимальное значение CU,
then
         begin
           пусть Cm - результат слияния C1 и
C2;
           cobweb(Cm, Instance);
         end
       else
         if S5 - максимальное значение CU,
then
           begin
             разделить C1; % Cm - новая
категория
             cobweb(Cm, Instance)
           end;
```

```
  end
```

В алгоритме COBWEB реализован метод поиска экстремума в пространстве возможных кластеров с использованием критерия полезности категорий для оценки и выбора возможных способов категоризации.

Сначала вводится единственная категория, свойства которой совпадают со свойствами первого экземпляра. Для каждого последующего экземпляра алгоритм начинает свою работу с корневой категории и движется далее по дереву. На каждом уровне выполняется оценка эффективности категоризации на основе полезности категории. При этом оцениваются результаты следующих операций:

- Отнесение экземпляра к наилучшей из существующих категорий.
- Добавление новой категории, содержащей единственный экземпляр.
- Слияние двух существующих категорий в одну новую с добавлением в нее этого экземпляра.
- Разбиение существующей категории на две и отнесение экземпляра к лучшей из вновь созданных категорий.

Для слияния двух узлов создается новый узел, для которого существующие узлы становятся дочерними. На основе значений вероятностей дочерних узлов вычисляются вероятности для нового узла. При разбиении один узел заменяется двумя дочерними.

Этот алгоритм достаточно эффективен и выполняет кластеризацию на разумное число классов. Поскольку в нем используется вероятностное представление принадлежности, получаемые категории являются гибкими и робастными. Кроме того, в нем проявляется эффект категорий базового уровня, поддерживается прототипирование и учитывается степень принадлежности. Он основан не на классической логике, а, подобно методам теории нечетких множеств, учитывает "неопределенность" категоризации как необходимый

компонент обучения и рассуждений в гибкой и интеллектуальной манере.

Сетевые алгоритмы

Метод WaveCluster

Алгоритм рассматривает всю совокупность данных как сигнал в N-мерном пространстве атрибутов и пытается на основании вейвлет-преобразования выделить в этом сигнале поддиапазоны частот, в которых связанные компоненты и будут кластерами.

Для применения алгоритма надо выбрать, какой именно фильтр будет применяться (какое именно из вейвлет-преобразований будет применяться).

0. Квантификация данных. Именно за это алгоритм называется сетевым. Каждое l-ное , $1 < l \leq D$ измерение D-мерного пространства данных (то есть каждый атрибут) разделяется на m_l отрезков длиной s_l. Тогда в D-мерном пространстве существует N точек $M_j = (\mu_1,...,\mu_l,...,\mu_d)$, где μ - это некоторое количество отрезков длиной s_l. Тогда $1 \leq \mu_l \leq m_l, N = \prod_{l=1}^{D} m_l$ (если число $m_l = m$ одинаково по всем осям, то $N = m^D$) .

И тогда объекту $x_i = (x_i^1, ..., x_i^l, ..., x_i^D)$ назначается в сооветствие M_j, когда для каждого l $(\mu_l - 1)s_l \leq x_i < \mu_l s_l$. Фактически мы разбили все D-мерное пространство, в котором расположены наши объекты, на некую совокупность D-мерных прямоугольных призм (кубиков), характеризуемых наиболее дальней от начала координат точкой призмы M_j и каждому кубику назначили в соответствие все те объекты, которые попали внутрь него.

1. Применение вейвлет-преобразования. Теперь мы применяем дискретное вейвлет-преобразование к этим точкам M_j. Тогда на выходе мы получаем новую совокупность точек T_k в новом пространстве признаков. Оно в общем уже незначащее, но зато в нем точки группируются, они как бы стягиваются в области сильной связности. Это особенность вейвлет-преобразований - они увеличивают плотность кластера. Количество точек и количество кластеров в них зависит от конкретного вейвлет-преобразования и от

количества его прогонов - чем оно более сложное и чем больше прогонов, тем меньше точек и кластеров.

2. Обнаружение кластеров в измененном пространстве признаков. В измененном пространстве признаков ищутся связанные компоненты </math>T_k</math>. Они являются кластерами. Более того, в общем случае многомерного пространства можно искать в подпространствах, которые также формируются вейвлет-преобразованием и в которых поиск гораздо легче в силу меньшей размерности этих пространств. В результате поиск сводится к поиску связанных компонент в двумерной картинке. В первом проходе находятся все связанные компоненты в картинке, и во втором - им назначают метки кластеров, отбрасывая уже отмеченные. Тогда мы получаем некоторое количество кластеров C и их меток c_f.

3. Создание таблицы соответствия. На этом этапе через соответствие исходных точек M_j и полученных в результате преобразования T_kсоздается таблица соответствия. Соотвествие этих точек для каждого из фильтров разное, но всегда прослеживается, несмотря на то, что преобразование не всегда обратимо. Простейшим соответствием являются индексы. В результате мы получаем принадлежность M_j к определенному кластеру c_f.

4. А так как мы знаем, каким точкам соответствует M_j, мы получаем и соответствие этих кластеров конкретным точкам.

Лекция 10 - Визуальный анализ данных

Введение

По данным университета Беркли ежегодный прирост информации в мире составляет 1 миллион терабайт (1 экзобайт).
Причём большая часть информации представлена в цифровом виде.
Это означает, что за последующие три года прирост информации превысит объём информации, накопленный за всю историю человечества до этого момента.
Откуда же берётся такое большое число данных?
Различные электронные датчики постоянно регистрируют такие процессы как использование кредитной карты, разговор по телефону и т.п.
Причём многие данные сохраняются с большой степенью детализации.
Делается это потому, что для людей представляет ценность эта информация.
Она может содержать в себе скрытые знания, закономерности и потому, при соответствующем анализе, способна оказать влияние при принятии решений в различных областях человеческой деятельности.
Существует множество способов поиска скрытых закономерностей в данных машиной, алгоритмами, но также не стоит упускать из вида возможности человека по анализу данных.
Полезно сочетать огромные вычислительные ресурсы современных компьютеров с творческим и гибким человеческим мышлением.
Визуальный анализ данных призван вовлечь человека в процесс отыскания знаний в данных.
Основная идея заключается в том, чтобы представить большие объёмы данных в такой форме, где человек мог бы увидеть то, что трудно выделить алгоритмически.
Чтобы человек смог погрузиться в данные, работать с их визуальным представлением, понять их суть, сделать выводы и напрямую взаимодействовать с данными.
Из-за сложности информации это не всегда возможно и в простейших графических видах представления знаний, таких

как деревья решений, дейтаграммы, двумерные графики и т.п. В связи с этим возникает необходимость в более сложных средствах отображения информации и результатов анализа.

С помощью новых технологий пользователи способны оценивать: большие объекты и маленькие, далеко они находятся или близко.
Пользователь в реальном времени может двигаться вокруг объектов или кластеров объектов и рассматривать их со всех сторон.
Это позволяет использовать для анализа естественные человеческие перцепционные навыки в обнаружении неопределённых образцов в визуальном трёхмерном представлении данных.

Визуальный анализ данных особенно полезен, когда о самих данных мало что известно и цели исследования до конца не понятны.
За счёт того, что пользователь напрямую работает с данными, представленными в виде визуальных образов, которые он может рассматривать с разных сторон и под любыми углами зрения, в прямом смысле этого слова, он может получить дополнительную информацию, которая поможет ему более чётко сформулировать цели исследования.

Таким образом, визуальный анализ данных можно представить как процесс генерации гипотез. При этом сгенерированные гипотезы можно проверить или автоматическими средствами (методами статистического анализа или методами Data Mining), или средствами визуального анализа.
Кроме того, прямое вовлечение пользователя в визуальный анализ имеет два основных преимущества перед автоматическими методами:

- визуальный анализ данных позволяет легко работать с неоднородными и зашумлёнными данными, в то время как не все автоматические методы могут работать с такими данными и давать удовлетворительные результаты;

- визуальный анализ данных интуитивно понятен и не требует сложных математических или статистических алгоритмов.
-
  Визуальный анализ данных обычно выполняется в три этапа:
- беглый анализ - позволяет идентифицировать интересные шаблоны и сфокусироваться на одном или нескольких из них;
- увеличение и фильтрация - идентифицированные на предыдущем этапе шаблоны отфильтровываются и рассматриваются в большем масштабе;
- детализация по необходимости - если пользователю нужно получить дополнительную информацию, он может визуализировать более детальные данные.

Характеристики средств визуализации данных

Существует достаточно большое количество средств визуализации данных, предоставляющих различные возможности.
Для выбора таких средств рассмотрим более подробно три основные характеристики средств визуализации данных:

- характер данных, которые нужно визуализировать с помощью данного средства;
- методы визуализации и образцы, в виде которых могут быть представлены данные;
- возможности взаимодействия с визуальными образами и методами для лучшего анализа данных.

Выделяют следующие виды данных, с которыми могут работать средства визуализации:

- одномерные данные - одномерные массивы, временные ряды и т.п.;
- двумерные данные - точки двумерных графиков, географические координаты и т.п.;
- многомерные данные - финансовые показатели, результаты экспериментов и т.п.;

- тексты и гипертексты - газетные статьи, веб-документы и т.п.;
- иерархические и связанные - структура подчинённости в организации, электронная переписка людей, гиперссылки документов и т.п.;
- алгоритмы и программы - информационные потоки, отладочные операции и т.п.

Для визуализации перечисленных типов данных используются различные визуальные образы и методы их создания. Очевидно, что количество визуальных образов, которыми могут представляться данные, ограничиваются только человеческой фантазией. Основное требование к ним - это наглядность и удобство анализа данных, которые они представляют. Методы визуализации могут быть как самые простые (линейные графики, диаграммы, гистограммы и т.п.), так и более сложные, основанные на сложном математическом аппарате. Кроме того, при визуализации могут использоваться комбинации различных методов. Выделяют следующие типы методов визуализации:

- стандартные 2D/3D-образы - гистограммы, линейные графики и т.п.;
- геометрические преобразования - диаграмма разброса данных, параллельные координаты и т.п.;
- отображение иконок - линейчатые фигуры (needle icons) и звёзды (star icons);
- методы, ориентированные на пикселы - рекурсивные шаблоны, циклические сегменты и т.п.;
- иерархические образы - древовидные карты и наложение измерений.

К простейшим методам визуализации относятся графики, диаграммы, гистограммы и т.п. Основным их недостатком является невозможность приемлемой визуализации сложных данных и большого количества данных. Методы геометрических преобразований визуальных образов направлены на трансформацию многомерных наборов данных с целью отображения их в декартовом и в недекартовом геометрических пространствах. Данный класс методов

включает в себя математический аппарат статистики. Другим классом методов визуализации данных являются методы отображения иконок. Их основной идеей является отображение значений элементов многомерных данных в свойства образов. такие образы могут представлять собой: человеческие лица, стрелки, звёзды и т.п. Визуализация генерируется отображением атрибутов элементов данных в свойства образов. Такие образы можно группировать для целостного анализа данных. Результирующая визуализация представляет собой шаблоны текстур, которые имеют различия, соответствующие характеристикам данных.
Основной идеей методов, ориентированных на пикселы, является отображение каждого измерения значения в цветной пиксел и из группировка по по принадлежности к измерению. Так как один пиксел используется для отображения одного значения, то, следовательно, данный метод позволяет визуализировать большое количество данных (свыше одного миллиона значений).
Методы иерархических образов предназначены для представления данных, имеющих иерархическую структуру. В случае многомерных данных должны быть правильно выбраны измерения, которые используются для построения иерархии.

Методы геометрических преобразований

- Матрица диаграмм разброса;
- параллельные координаты;

Методы, ориентированные на пикселы

- Рекурсивные шаблоны;
- циклические сегменты;

Иерархические образы

- Наложение измерений.

В результате применения методов визуализации будут построены визуальные образы, отражающие данные. Однако этого не всегда бывает достаточно для полного анализа. Пользователь должен иметь возможность работать с образами: видеть их с разных сторон, в разном масштабе и т.п. Для этого

у него должны быть соответствующие возможности взаимодействия с образами:

- динамическое проецирование;
- интерактивная фильтрация;
- масштабирование образов;
- интерактивное искажение;
- интерактивное комбинирование.

Основная идея динамического проецирования заключается в динамическом изменении проекций при проведении исследования многомерных наборов данных. Примером может служить проецирование в двумерную плоскость всех интересующих проекций многомерных данных в виде диаграмм разброса (scatter plots). Необходимо обратить внимание, что количество возможных проекций экспоненциально увеличивается с ростом числа измерений, и, следовательно, при большом количестве измерений проекции будут тяжело воспринимаемы.
При исследовании большого количества данных важно иметь возможность разделять наборы данных и выделять интересующие поднаборы - фильтровать образы. при этом важно, чтобы данная возможность предоставлялась в режиме реального времени работы с визуальными образами (т.е. интерактивно). Выбор поднабора может осуществляться или напрямую из списка, или с помощью определения свойств интересующего поднабора.
Примером масштабирования образов является "магическая линза" (Magic Lenses). Её основная идея состоит в использовании инструмента, похожего на увеличительное стекло, чтобы выполнять фильтрацию непосредственно при визуализации. Данные, попадающие под увеличительное стекло, обрабатываются фильтром, и результат отображается отдельно от основных данных. Линза показывает модифицированное изображение выбранного региона, тогда как остальные визуализированные данные не детализируются.
Масштабирование - это хорошо известный метод взаимодействия, используемый во многих приложениях. При работе с большим объёмом данных этот метод хорош тем для представления данных в сжатом обзем виде, и, в то же время,

он предоставляет возможность отображения любой их части в более детальном виде. Масштабирование может заключаться не только в простом увеличении объектов, но в изменении их представления на разных уровнях. Так, например, на нижнем уровне объект может быть представлен пикселом, на более высоком уровне - неким визуальным образом, а на следующем - текстовой меткой.

Метод интерактивного искажения поддерживает процесс исследования данных с помощью искажения масштаба данных при частичной детализации. Основная идея этого метода заключается в том, что часть данных отображается с высокой степенью детализации, а одновременно с этим остальные данные показываются с низким уровнем детализации. Наиболее популярные методы - это гиперболическиое и сферическое искажения.
Существует достаточно много методов визуализации, но все они имеют как достоинства, так и недостатки. Основная идея комбинирования заключается в объединении различных методов визуализации для преодоления недостатков одного из них. Различные проекции рассеивания точек, например, могут быть скомбинированы с методами окрашивания и компоновки точек во всех проекциях.

Любое средство визуализации может быть классифицировано по всем трём параметрам, т.е. по виду данных, с которым оно работает, по визуальным образам, которые оно может предоставлять, и по возможностям взаимодействия с этими визуальными образами. Очевидно, что одно средство визуализации может поддерживать разные виды данных, разные визуальные образы и разные способы взаимодействия с образами.

Лекция 11 - Факторный анализ

Введение

Факторный анализ - это один из способов снижения размерности, то есть выделения во всей совокупности признаков тех, которые действительно влияют на изменение зависимой переменной. Или группировки сходно влияющих на изменение зависимой переменной признаков. Или группировки просто сходно изменяющихся признаков. Предполагается, что наблюдаемые переменные являются лишь линейной комбинацией неких ненаблюдаемых факторов. Некоторые из этих факторов являются общими для нескольких переменных, некоторые характерно проявляют себя только в одной. Те, что проявляют себя только в одной, очевидно, ортогональны друг другу и не вносят вклад к ковариацию переменных, а общие - как раз и вносят эту ковариацию. Задачей факторного анализа является как раз восстановление исходной факторной структуры исходя из наблюдаемой структуры ковариации переменных, несмотря на случайные ошибки ковариации, неизбежно возникающие в процессе снятия наблюдения.

Коэффициент взаимосвязи между некоторой переменной и общим фактором, выражающий меру влияния фактора на признак, называется**факторной нагрузкой** (Factor load) данной переменной по данному общему фактору. Значение (мера проявления) фактора у отдельного объекта называется факторным весом объекта по данному фактору.

Процесс факторного анализа состоит из трех больших этапов:

1. Подготовки ковариационной матрицы (Иногда вместо нее используется корреляционная матрица);
2. Выделения первоначальных ортогональных векторов (основной этап);
3. Вращение с целью получения окончательного решения.

Подготовка к факторному анализу

При подготовке к факторному анализу часто (некоторые методы этого не требуют, но большая часть - требует) составляют ковариационные и корреляционные матрицы. Это матрицы, составленные из ковариации и корреляций

векторов-атрибутов (строки и столбцы - атрибуты, пересечение - ковариация/корреляция).

Ковариация двух векторов:

$$\operatorname{cov}(\mathbf{X}, \mathbf{Y}) = \mathbb{M}((\mathbf{X} - \mathbb{M}(\mathbf{X}))(\mathbf{Y} - \mathbb{M}(\mathbf{Y}))),$$

$\mathbb{M}()-$ математическое ожидание $\mathbb{M}(X) = \sum_i x_i p_i, P(X = x_i) = p_i, \sum_i p_i = 1$

Корреляция двух векторов:

$$\varrho_{X,Y} = \frac{\operatorname{cov}(X, Y)}{\sqrt{\mathrm{D}[X]} \cdot \sqrt{\mathrm{D}[Y]}},$$

$D[X] = M\left[(X - M[X])^2\right]$, - дисперсия.

Обратите внимание, что в этом случае корреляция и ковариация двух векторов - числа, так как считаются через матожидание вектора, а матожидание вектора - число.

Таким образом, мы переходим от матрицы, составленной из объектов (которые могут быть и не математическими), к матрице, оперирующей уже исключительно математическими понятиями, и абстрагируемся от объектов, уделяя внимания только атрибутам.

Нахождение первичной структуры факторов

Метод главных компонент

Метод главных компонент стремится выделить оси, вдоль которых количество информации максимально, и перейти к ним от исходной системы координат. При этом некоторое количество информации может теряться, но зато сокращается размерность.

Этот метод проходит практически через весь факторный анализ, и может меняться путем подачи на вход разных матриц, но суть его остается неизменной.

Основной математический метод получения главных осей - нахождение собственных чисел и собственных векторов ковариационной матрицы таких, что:

$$RV = \lambda V, \text{ где}$$

λ - собственное число R, R - матрица ковариации, V - собственный вектор R. Тогда :

$$RV - \lambda V = 0$$
$$V(R - \lambda E) = 0$$

и решение есть когда:

$$| R - \lambda E | = 0,$$

где R - матрица ковариации, λ - собственное число R, E - единичная матрица. Затем считаем этот определитель для матрицы соответствующей размерности.

V находим, подставляя собственные числа по очереди в

$$V(R - \lambda E) = 0$$

и решая соответствующие системы уравнений.

Сумма собственных чисел равна числу переменных, произведение - детерминанту корелляционной матрицы. Собственное число представляет собой дисперсию оси, наибольшее - первой и далее по убыванию до наименьшего - количество информации вдоль последней оси. Доля дисперсии, приходящаяся на данную компоненту, считается отсюда легко: надо разделить собственное число на число переменных m.

Коэффициенты нагрузок для главных компонент получаются делением коэффициентов собственных векторов на квадратный корень соответствующих собственных чисел.

Алгоритм NIPALS вычисления главных компонент

На практике чаще всего для определения главных компонент используют итерационные методы, к примеру, NIPALS:

0. Задается $0 < \varepsilon_1 < 1$ - критерий окончания поиска главного компонента, и $0 < \varepsilon_2 < 1$ - критерий окончания поиска

главных компонентов, исходная отцентрированная матрица X, i=1 - номер главной компоненты.

1. Берется $T_k = x_j \in X$ - вектор-столбец, k - шаг алгоритма, j - любой столбец (просто чтобы было с чего начинать апроксимизацию).

2. Вектор T_k транспонируется.

3. Считается $P_k = \frac{T_k^T X}{T_k^T T_k}$.

4. P_k нормируется $P_k = P_k (P_k^T P_k)^{-0,5}$

5. Считается новый $T_k^new = \frac{X P_k}{P_k^T P_k}$

6. Если $|T_k^{new} - T_k| < \epsilon_1$, то T_k^{new} и P_k - вектора весов и нагрузок соответственно для i-ой главной компоненты. Если нет, то $T_k = T_k^{new}$ и иди на 2.

7. $X = X - T_k P_k^T$.

8. Если $|X| < \varepsilon$, то стоп - найдены все основные компоненты, нас удовлетворяющие. Иначе i++. Иди на 1.

Другие методы

Метод сингулярных компонент

Метод максимального правдоподобия

Метод альфа-факторного анализа

Вращение

Вращение - это способ превращения факторов, полученных на предыдущем этапе, в более осмысленные. Бывает графическое (проведение осей, не применяется при более чем 2мерном анализе), аналитическое (выбирается некий критерий вращения, различают ортогональное и косоугольное) и матрично-приближенное (вращение состоит в приближении к некой заданной целевой матрице).

Результатом вращения является вторичная структура факторов. Первичная факторная структура (состоящая из первичных нагрузок (полученных на предыдущем этапе)) - это, фактически, проекции точек на ортогональные оси координат. Очевидно, что если проекции будут нулевыми, то

структура будет проще. А проекции будут нулевыми, если точка лежит на какой-то оси. **Де-факто вращение есть переход от одной системы координат к другой** при известных координатах в одной системе(первичные факторы) и итеративно подбираемых координатах в другой системе(вторичных факторов).

При получении вторичной структуры стремятся перейти к такой системе координат, чтобы провести через точки (объекты) как можно больше осей, чтобы как можно больше проекции (и соответственно нагрузок) были нулевыми. При этом могут сниматься ограничения ортогональности и убывания значимости от первого к последнему факторам, характерные для первичной структуры, иВторичная структура является более простой, чем первичная, и потому более ценна.

Обозначим понятие простой структуры. Пусть r - число (общих) факторов первичной структуры , V - матрица вторичной структуры, состоящая из нагрузок (координат) вторичных факторов(строка - переменная из R, столбец - вторичный фактор).

1. В каждой строке матрицы V должен быть хотя бы 1 нулевой элемент;
2. Каждый столбец из матрицы V должен содержать не менее r нулей;
3. У одного из столбцов из любой пары из матрицы V должно быть несколько нулевых коэффициентов(нагрузок) в тех позициях, где для другого столбца они ненулевые (для различения);
4. При числе факторов больше 4 в каждой паре должно быть несколько нулевых нагрузок в одних и тех же строках;
5. Для каждой пары столбцов должно быть как можно меньше больших по величине нагрузок в одних и тех же строках.

Тогда структура будет хорошо подходить для интерпретации и будет выделяться однозначно. Наипростейшей является

структура, где каждая переменная имеет **факторную сложность**(количество факторов, которые на нее влияют и оказывают факторную нагрузку), равную 1 (все факторы будут характерными). Реально это не достижимо, и потому мы стремимся приблизится к простой структуре при помощи различных методов. Существует эмпирическое правило, что для каждого фактора по крайней мере три переменных имеют значительную на него нагрузку.

Аналитическое вращение

Наиболее интересно аналитическое вращение, так как позволяет получить вторичную структуру исходя из достаточно критериев и без априорного знания о структуре факторной матрицы.

Ортогональное вращение

Ортогональное вращение подразумевает, что мы будем вращать факторы, но не будем нарушать их ортогональности друг другу. Ортогональное вращение подразумевает умножение исходной матрицы первичных нагрузок на ортогональную матрицу R(такую матрицу, что $|R| = 1, R * R^T = E, R = r \times r$)

```
V=BR
```

Алгоритм ортагонального вращения в общем случае таков:

0. B - матрица первичных факторов.

1. Ищем ортогональную матрицу R^T размера 2*2 для двух столбцов(факторов) b_i и b_j матрицы B такую, что критерий для матрицы $[b_i b_j]R$максимален.

2. Заменяем столбцы b_i и b_j на столбцы $[b_i b_j] \times R$.

3. Проверяем, все ли столбцы перебрали. Если нет, то 1.

4. Проверяем, что критерий для всей матрицы вырос. Если да, то 1. Если нет, то конец.

Критерий квартимакс

Формализуем понятие факторной сложности q i-ой переменной черех дисперсию квадратов факторных нагрузок факторов:

$$q_i = \frac{1}{r}\sum_{j=1}^{r}(b_{ij}^2 - \overline{b_i^2})^2$$, где

r - число столбцов факторной матрицы, b_ij - факторная нагрузка j-го фактора на i-ю переменную, $\overline{b_ij}$ - среднее значение. Критерий квартимакс старается максимизировать сложность всей совокупности переменных, чтобы достичь легкости интерпретации факторов (стремится облегчить описание столбцов):

$$Q = \sum_{i=1}^{N} q_i$$

Учитывая, что $\sum_{j=1}^{r}\overline{b_i^2}$ - константа (сумма собственных чисел матрицы ковариации) и раскрыв среднее значение (а также учтя, что степенная функция растет пропорционально аргументу), мы получим:

$$Q = \sum_{i=1}^{N}\sum_{j=1}^{r} b_{ij}$$

Этот критерий и предполагается итеративно максимизировать. Этот критерий стремится к одному генеральному фактору.

Критерий варимакс

Этот критерий использует формализацию сложности фактора через дисперсию квадратов нагрузок переменной:

$$v_j = \frac{n\sum_{i=1}^{n}(b_{ij}^4 - (\sum_{i=1}^{n} b_i^2)^2}{n^2}$$

И тогда критерий в общем

$$V = \frac{\sum_{j=1}^{r}(n\sum_{i=1}^{n}(b_{ij}^4) - \sum_{j=1}^{r}(\sum_{i=1}^{n} b_i^2)^2}{n^2}$$

При этом, как легко заметить, максимизируется сложность описания Факторные нагрузки могут нормироваться для

избавления от влияния отдельных переменных. Дает лучшее разделение факторов, чем квартимакс.

Другие критерии

Можно обобщить два вышеприведенных критерия и получить новый:

$$Max = \alpha Q + \beta V$$

Запишем его в следующем виде: $Max = \sum_{j=1}^{r}\sum_{i=1}^{n}(b_{ij}^4 - \gamma\frac{\sum_{j=1}^{r}(\sum_{i=1}^{n} b_i^2)^2}{n}, \gamma = \frac{\beta}{\alpha+\beta}$ Тогда при $\gamma = 0$ это квартимакс, при $\gamma = 1$ - варимакс, а при $\gamma = \frac{r}{2}$ - эвримакс, а при $\gamma = 0{,}5$ биквартимакс.

Косоугольное вращение

Косоугольное вращение не требует ортогональности между факторами. В остальном его алгоритм похож на ортогональное вращение. За счет этого можно получать больше нулей в факторных нагрузках и получать более характерные факторы. Правда, при этом возникает корреляция между факторами, что вообще не очень хорошо и приходится объяснять факторами 2го порядка. Их, кстати, тоже можно вычислить, причем используя ортогональное вращение и косоугольные факторы как исходные данные.

Методы введения вторичных осей

На основе квартимакса создается критерий квартимин:

$$N = \sum_{i=1}^{N}\sum_{j<k=1}^{r} a_i j a_i k$$

где a_ij и a_ij - проекции на j-ю и k-ю оси. При применении ортогонального вращения этот критерий сводится к квартимаксу. При наипростейшей структуре N=0, а реально должно к нему стремится.

На основе варимакса создается критерий коваримин:

$$C = \sum_{j<k=1}^{r}(n\sum_{i=1}^{N} a_i j^2 a_i k^2 - \sum_{i=1}^{N} a_i j^2 \sum_{i=1}^{N} a_i k^2$$

Минимизируется ковариация квадратов проекции на различные оси.

Объединение их, как и в ортогональном вращении, приводит к критерию облимин:

$$Min = \sum_{j<k=1}^{r} (n \sum_{i=1}^{N} a_i j^2 a_i k^2 - \gamma \sum_{i=1}^{N} a_i j^2 \sum_{i=1}^{N} a_i k^2)$$

Прямой метод облимин

Используется критерий облимин, только при этом в качестве аргументов выступают нагрузки факторов первичной структуры:

$$D = \sum_{j<k=1}^{r} (n \sum_{i=1}^{N} b_i j^2 b_i k^2 - \frac{d \sum_{i=1}^{N} b_i j^2 \sum_{i=1}^{N} b_i k^2}{n})$$

d регулирует косоугольность решения, меньшие отрицательные d- больше ортогональность