



СПбГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ



Н.А. Жукова

Машинное обучение

Конспект лекций

СПбГЭТУ «ЛЭТИ», 2022 г.





ВВЕДЕНИЕ

Термин машинное обучение охватывает большой круг технологий, которые преследуют одну цель: извлечь имеющую смысл информацию из данных. Под данными здесь подразумевается все, что может быть записано и измерено (курс акций в удачные дни, или масса планет в Солнечной системе, рост и вес людей, различные звуки (слова, произносимые кем-нибудь в его мобильный телефон), изображения (к примеру, фотографии кошек или других животных), тексты газетной статьи или романа. или все, что мы хотели бы изучить или исследовать. «Имеющая смысл информация» - это все, что мы можем извлечь из данных и что будет полезным для нас в некотором смысле. Мы решаем, что имеет смысл для нас, и затем создаем алгоритм, чтобы найти столь много из наших данных, насколько это возможно.

Машинное обучение заключается в извлечении знаний из данных. Это научная область, находящаяся на пересечении статистики, искусственного интеллекта и компьютерных наук и также известная как прогнозная аналитика или статистическое обучение. В последние годы применение методов машинного обучения в повседневной жизни стало обыденным явлением. Многие современные веб-сайты и устройства используют алгоритмы машинного обучения, начиная с автоматических рекомендаций по просмотру фильмов, заказа еды или покупки продуктов, и заканчивая персонализированными онлайн-радиотрансляциями и распознаванием друзей на фотографиях. Когда вы видите сложный сайт типа Facebook, Amazon или Netflix, то весьма вероятно, что каждый раздел сайта содержит несколько моделей машинного обучения.

На заре появления «интеллектуальных» приложений многие системы использовали жесткие правила «if» и «else» для обработки данных или корректировки информации, введенной пользователем. Вспомните о спам-филт্রে, чья работа состоит в том, чтобы переместить соответствующие входящие сообщения электронной почты в папку «Спам». Вы можете составить черный список слов, которые будут идентифицировать письмо как спам. Это пример использования системы экспертных правил для разработки





«интеллектуального» приложения. Разработка правил принятия решений в ручном режиме допустимо в некоторых задачах, особенно в тех, где люди четко понимают процесс моделирования.

Однако, использование жестких решающих правил имеет два основных недостатка:

- *логика, необходимая для принятия решения, относится исключительно к одной конкретной области и задачи. Даже незначительное изменение задачи может повлечь за собой переписывание всей системы.*
- *разработка правил требует глубокого понимания процесса принятия решения.*

Один из примеров, где этот жесткий подход потерпит неудачу - это распознавание лиц на изображениях. На сегодняшний день каждый смартфон может распознать лицо на изображении. Тем не менее, распознавание лиц была нерешенной проблемой, по крайней мере, до 2001 года. Основная проблема заключается в том, что способ, с помощью которого компьютер «воспринимает» пиксели, формирующие изображение на компьютере, очень сильно отличается от человеческого восприятия лица. Эта разница в принципе не позволяет человеку сформулировать подходящий набор правил, описывающих лицо с точки зрения цифрового изображения.

Главная идея, лежащая в основе машинного обучения, — можно представить действительность, используя математические функции, которые алгоритму заранее неизвестны, но которые он может предположить на основании наблюдения некоторых данных. Вы можете выразить действительность и всю ее комплексную сложность в терминах неизвестных математических функций, которые алгоритмы машинного обучения способны выявить и сделать доступными. Эта концепция лежит в основе идей всех видов алгоритмов машинного обучения.

Обучение при машинном обучении является просто математической функцией, и она заканчивается ассоциацией определенных входных данных с определенными выходными. Она не имеет отношения к пониманию того,





что алгоритм изучил (анализ данных вырабатывает понимание до некоторой степени), поэтому процесс обучения зачастую называют тренировкой, поскольку алгоритм учится находить правильный ответ (вывод) на каждый предоставленный вопрос (ввод).

Многие люди привыкли к идее, что приложения начинают работу с получения данных на входе, а затем предоставляют некий результат. Например, программист мог бы создать функцию `add( )`, которая получает как ввод два значения, например, 1 и 2, а затем возвращает результат 3. Выводом этого процесса является значение. В прошлом написание программы означало понимание того, как функция должна манипулировать данными, чтобы получить заданный результат при определенных входных данных. Машинное обучение осуществляет переворот в этом процессе. В данном случае вы знаете входные данные, это 1 и 2. Вы также знаете, что желаемый результат — 3. Однако вы не знаете, какую функцию применить, чтобы получить желаемый результат. Обучение предоставляет алгоритм со всякого рода примерами входных данных и ожидаемыми результатами для этих входных данных. Затем алгоритм использует заданный ввод, чтобы создать функцию. Другими словами, обучение — это процесс, в ходе которого обучаемый алгоритм сопоставляет с данными гибкую функцию. Выводом обычно является вероятность определенного класса или числового значения. Задачи определения функции при заданном входе и выходе, а также определение входа выходным данным и функции являются *некорректными задачами*.

Корректно поставленная задача в математике — прикладная задача, математическое решение которой существует, единственно и устойчиво. (Задача или алгоритм решения задачи называются вычислительно неустойчивыми, если малые изменения входных данных приводят к заметным изменениям решения).

Важно понимать, что задача машинного обучения изначально не является корректно поставленной задачей.

Успешное решение задачи машинного обучения на 50% состоит из знания «стандартных» методов и умения их применить и на 50% из смекалки,





умения правильно поставить задачу, внимательного анализа исходных данных и выделения признаков.

Выйдя за пределы коммерческих приложений, машинное обучение уже оказало огромное влияние на научные исследования, управляемые данными. Инструменты, представленные в этой книге, использовались для решения различных научных задач (исследование звезд, поиск далеких планет, открытие новых частиц, анализ последовательностей ДНК, а также разработка персонализированных методов лечения рака).

Дадим некоторые определения машинного обучения (единственное и всеобъемлющее определение практически дать невозможно, так как в зависимости от предметной области где оно используется возникает масса нюансов и дополнительных факторов, которые могут отсутствовать в других областях). Если использовать аналогию, то в теории систем существует до 300 определений понятия система, но ни одно из них не дает достаточно полного и не противоречивого определения «система». Также и с машинным обучением.

Для анализа огромных наборов данных машинное обучение полагается на алгоритмы. В настоящее время машинное обучение не способно обеспечить такой вид искусственного интеллекта, как это показывают в кино. Даже наилучшие алгоритмы не могут думать, чувствовать, обладать любой формой самосознания, морали, этики или иметь свободу выбора. На что машинное обучение способно, так это выполнять прогнозирующую аналитику намного быстрее, чем любой человек. В результате машинное обучение может помочь людям работать эффективнее. Да, искусственный интеллект в текущем состоянии способен выполнить выдающийся анализ, но смысл его результатов все еще должны осознавать люди, они же принимают необходимые моральные и этические решения на его основании. По существу, машинное обучение обеспечивает лишь часть обучения искусственного интеллекта, и эта часть ничуть не готова создать искусственный интеллект того вида, который вы видите в фильмах.

Машинное обучение (англ. machine learning, ML) — класс методов искусственного интеллекта, характерной чертой которых является не





прямое решение задачи, а обучение в процессе применения решений множества сходных задач.

Машинное обучение - это обширный подраздел искусственного интеллекта, изучающий методы построения алгоритмов, способных обучаться.

Машинное обучение - это подобласть искусственного интеллекта. Data-driven подход, полагающийся на выучивание закономерностей из имеющихся размеченных или неразмеченных данных.

Можно дать еще одно определение машинного обучения.

Машинное обучение — это наука, изучающая алгоритмы, автоматически улучшающиеся благодаря опыту.

С теоретической стороны машинное обучение - дисциплина, находящаяся на пересечении математической статистики, численных методов оптимизации, теории вероятностей, теории систем и системный анализ, а также дискретного анализа. С помощью ее методов происходит решение задачи извлечения знаний из данных, которой занимается еще только формирующаяся область «Интеллектуальный анализ данных» (DataMining).

Data Mining (добыча данных, интеллектуальный анализ данных, глубинный анализ данных) — совокупность методов обнаружения в данных ранее неизвестных, нетривиальных, практически полезных и доступных интерпретации знаний, необходимых для принятия решений в различных сферах человеческой деятельности.

В теории искусственного интеллекта и экспертных систем, знание - это совокупность утверждений о мире, свойствах объектов, закономерностях процессов и явлений, а также правил логического вывода одних утверждений из других и правил использования их для принятия решений. Главное отличие знаний от данных состоит в их структурности и активности: появление в базе знаний новых фактов или установление новых связей между ними может стать источником изменений в принятии решений.





Если, опираясь на это определение знания, рассмотреть DataMining как процесс, то его сутью будет нахождение в «сырых» данных знаний, обладающих свойствами нетривиальности, интерпретируемости и практической полезности для принятия решений в различных сферах деятельности человека. Причем, эти знания ранее неизвестны.

С практической же стороны машинное обучение нацелено на создание систем, способных адаптироваться к решению различных задач без явного кодирования алгоритма, то есть систем, способных обучаться.

Машинное обучение (ML) содержательно включает в себя 3 компоненты:

1. Представление (Representation).
2. Оценка (Evaluation).
3. Оптимизация (Optimization).

Представление (Representation). Что собой представляет модель, какие классы задач она способна (и не способна) решать, как именно представляются данные (включая этап выделения нужных признаков). Примеры: разделяющая гиперплоскость, деревья решений, нейросети.

Оценка (Evaluation). Как оценивать качество модели в контексте решения задачи, как выбирать лучшую модель из нескольких.

Оптимизация (Optimization). Как именно проводить обучение модели, каким именно образом осуществлять перебор пространства возможных моделей, чтобы найти лучшую.

Цель машинного обучения — предсказать результат по имеющимся данным. Чем качественнее и полнее входные данные, тем легче машине найти какие-либо закономерности и тем более точным является результат.

Машинное обучение нацелено на создание систем, способных получать знания из данных способных с помощью обучения улучшать показатели своей работы. Таким образом, можно утверждать, что машинное обучение является одной из областей науки о данных (DataScience). На рис.1 представлены ее составляющие и показано место машинного обучения среди них.



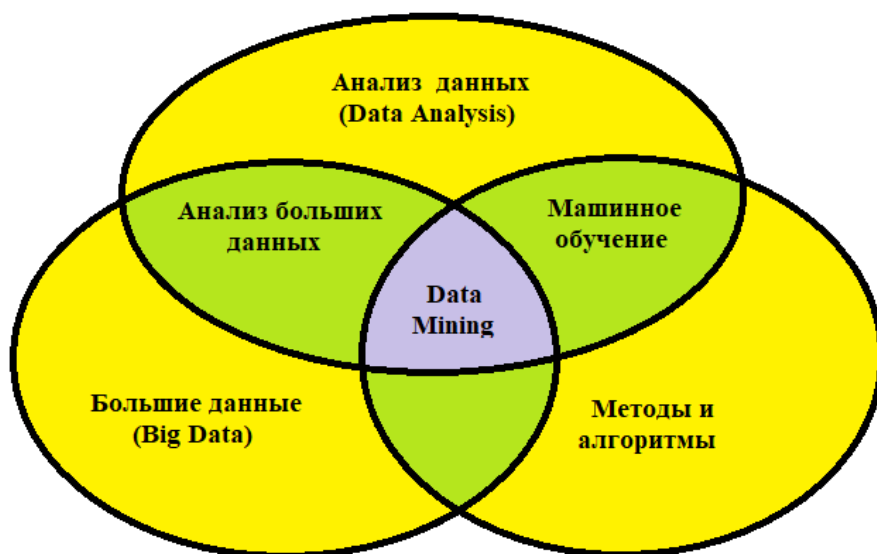


Рис.1. Области науки о данных

Сложность работы в сфере DataMining обуславливается неточностью, противоречивостью, разнородность данных, которые при этом могут иметь гигантские объемы. обработки требуются специальные программные средства: инструменты преобразования

«сырых» данных в информацию, а информация в знания. Кроме того, алгоритмы обработки данных должны иметь возможность обучаться по *прецедентам*. Машинное обучение (МО) как раз и занимается разрешением подобных сложностей. Причем в настоящее время усилия сконцентрированы на повышении не столько качества решения, сколько скорости и оптимальности его получения.

Как видно из рис.1, машинное обучение пересекается с анализом данных, то есть имеет общие черты, но также и свою специфику. Анализ данных - это огромная и уже относительно не молодая область математики и информатики. Под нее попадает всяческая обработка данных. Например, обработка данных физического эксперимента статистическими методами для получения обобщенной информации (усредненных значений, точных значений, доверительных интервалов, оценок и тому подобное), обработка сигнала методом Фурье, даже подсчет минимального и максимального



значений или отношения каких-то величин может быть анализом данных, если эти операции влекут за собой важные выводы, знания о данных и позволяют решить поставленную задачу. Но получение дисперсии некой величины хотя и является анализом данных, но не является в чистом виде машинным обучением. При этом необходимо отметить, что методы статистики имеют большое значение и в машинном обучении.

Отличительной особенностью методов машинного обучения является то, что они способны решать широкий спектр задач без явного кодирования алгоритма решения.

Например, метод регрессии входит в машинное обучение. Регрессия давно применяется в различных областях человеческой деятельности: экономика, прикладная математика, технические и гуманитарные науки и т.д. Она используется для прогнозирования и оценки разнообразных параметров, потому что данная математическая модель в некотором роде универсальна. Однако стоит заметить, что регрессия появилась в математике гораздо раньше, чем ее стали использовать в прикладных областях.

Машинным обучением регрессия стала только тогда, когда все ее методы начали рассматриваться не как обособленные механизмы решения задач из специальных областей, а как методы преобразования информации вообще.

Для разработки решений на основе МО необходимы оцифрованные данные. Они являются ключевым аспектом, от которого зависит качество и полнота решения. Таким образом, очень важно понимать, что для применения методов машинного обучения требуются собранные и специально подготовленные данные.

Как правило, процесс машинного обучения проходит несколько этапов. На рис.3 представлен обобщенный алгоритм, выполняющийся при решении задач методами рассматриваемой области.

Для реализации машинного обучения необходимы следующие вещи:





Данные. Хотим фильтровать спам — нужны экземпляры спам-писем, предугадывать курс валюты — нужна ретроспектива цен, определять предпочтения клиента — нужны персональные лайки или посты в социальных сетях. Необходимо собирать максимально возможное количество данных. Десятки тысяч кейсов — это минимум, не гарантирующий качественного обучения системы.

Сбор данных осуществляется разными способами. Часть данных собирается вручную — способ долгий, но снижается количество ошибок. Часть — автоматически — можно загрузить все, что есть в систему и ждать какого-то, возможно отрицательного результата. Хорошим способом, используемым, например, Google, является совмещение автоматического сбора данных с ручной разметкой.

Хорошие и проработанные наборы данных (датасетами) очень ценны в мире машинного обучения. Иногда компании, даже раскрывают алгоритмы своих систем, но датасеты практически никогда не публикуются в открытую.

Необходимо отметить, что данная иллюстрация отражает идеальный случай, когда после обучения мы сразу получили работающую модель. Однако в реальности, как правило, между вторым и третьим этапом есть промежуточное, но очень важное действие — оценка качества модели. И именно по его результатам принимается решение о переходе на следующий этап или о возврате к одному из предыдущих. Этот вопрос мы рассмотрим подробнее далее, здесь же введем определение пространства признаков, для лучшего понимания схемы на рис.2.

Пространство признаков.

Пространство признаков — это N -мерное пространство, где N — число измеряемых характеристик объектов, выделенное для конкретной задачи. При этом объекты в пространстве признаков задаются N -мерными векторами, каждая компонента которых представляет собой значение определенной характеристики.

Признаки иногда называют фичами (features). Фичи — это свойства, характеристики, признаки, которые описывают тот или иной объект или





явление. К примеру, фичами являются пробег автомобиля, массо-габариты человека, его пол, стоимость акций, частота появления букв в тексте и т.д. При обучении машина должна знать, что она должна учитывать. При структурированном хранении данных, когда информация хранится в таблицах названия столбцом можно считать признаком. Однако, чаще всего данные имеют совершенно другой формат. Если признаков слишком много, модель становится медленной и неэффективной. Может случиться так, что отбор корректных свойств занимает больше времени, чем всё остальное обучение. С другой стороны, возможны и противоположные ситуации, когда человек самостоятельно решает использовать «правильные» на его взгляд признаки. Это вносит в модель субъективность, которая приводит к тому, что возрастает процент ошибок.

Алгоритм. Как правило, любая задача может быть решена большим количеством способов. От выбранного способа решения зависит размерность готовой математической модели, точность, вычислительные затраты, скорость работы, емкость. Однако следует учитывать, что если данные изначально некачественные, то, даже самый лучший и точный алгоритм не поможет. Из лжи может следовать все, что угодно, и это правило Булевой алгебры никто не отменял. Поэтому еще раз стоит отметить, что необходимо собрать как можно больше данных.





Рис.2. Этапы машинного обучения

Очень часто в различных изданиях в силу моды «искусственным интеллектом» называют любую линейную регрессию, которая применяется даже в средней школе. На самом деле, все немного сложнее. Место машинного обучения в системе искусственного интеллекта представлено на рис.3.



Рис.3. Место машинного обучения в системе искусственного интеллекта





Искусственный интеллект — это область науки и инжиниринга, занимающаяся созданием машин и компьютерных программ, обладающих интеллектом. Это название всей области, как физика или математика.

Машинное обучение — это один из разделов искусственного интеллекта, класс методов искусственного интеллекта, характерной чертой которых является не прямое решение задачи, а обучение в процессе применения решений множества сходных задач.

Нейросети — всего лишь вид машинного обучения. Это математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей. Есть и другие виды обучения, используемые для разных задач.

Глубокое обучение — архитектура нейросетей, один из подходов к их построению и обучению. По этим термином понимают совокупность методов машинного обучения (с учителем, с частичным привлечением учителя, без учителя, с подкреплением), основанных на обучении представлениям (фичам), а не специализированным алгоритмам под конкретные задачи. В реальности определять глубину обучения сейчас не стараются, просто используют имеющиеся движки.

МАШИНА МОЖЕТ	МАШИНА НЕ МОЖЕТ
Предсказывать	Создавать новое
Запоминать	Поумнеть самостоятельно
Воспроизводить	Выйти за рамки задачи
Выбирать лучшее	Брать ответственность

Классифицировать алгоритмы можно десятком способов. Ментальные карты сейчас - самый простой и наглядный способ. Однако не бывает так, чтобы какую задачу решал только один метод. Как правило, задачу можно решить несколькими способами, и явно есть «специалист», который захочет все решить нейросетями: лучше, быстрее, дешевле.

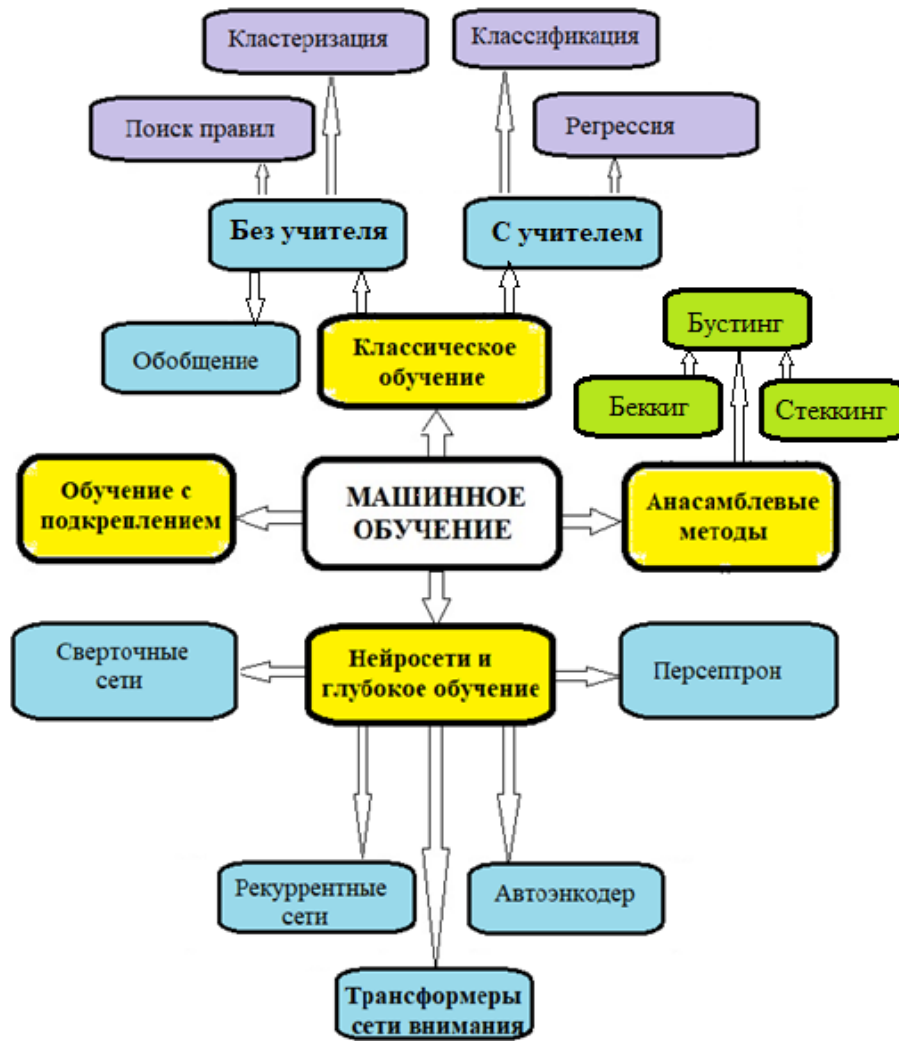


Рис.4. Задачи машинного обучения

На данный момент в машинном обучении есть 4-ре крупных основных направления.

1. Классическое обучение (простые данные, понятные признаки).
2. Обучение с подкреплением (данных нет, но есть среда, с которой можно взаимодействовать).
3. Ансамбли (сложные данные).
4. Нейросети и глубокое обучение (сложные данные, размытые признаки).

Задачи машинного обучения представлены на рис.4.

Процесс решения задачи с помощью машинного обучения включает в себя:



1. Определиться со способом решения вашей задачи средствами *ML*.
2. Выбрать метрику качества.
3. Получить данные и выделить признаки.
4. Определиться с классами моделей.
5. Подготовить данные для обучения и оценки.
6. Провести обучение моделей, оценить результат, повторить шаги, если требуется.

Классическое обучение

Первые алгоритмы классического машинного обучения пришли из чистой статистики еще в 1950-х. Задачи, которые они решали, носили формальный характер — поиск закономерностей в числах, оценка близости точек в пространстве и вычисление различных направлений.

Но и по сей день на классических алгоритмах основана практически половина ресурсов интернета. Яндекс. Дзен и «Рекомендованные статьи» на сайте - один из наиболее часто встречаемых примеров. Блокировка мобильной связи после первого звонка за границей — это тоже решение, принятое одним из классических алгоритмов.

Однако, крупные компании предпочитают решать свои задачи с использованием нейросетей. Это связано с высокой ценой риска, связанной со снижением точности. Для них даже 1% точности превращается в дополнительный миллиард прибыли или убытка. Но в реальности необходимо обоснованно подходить к выбору метода решения. Если задача может быть решена классическими методами, дешевле и проще реализовать любую систему для бизнеса на них, а уже потом думать об усложнении одели и улучшениях. В данном случае, важнее решить задачу в принципе, чем решить ее с точностью 99 процентов, а не 98.

Бывают ситуации, когда постоянные улучшения системы не имеют смысла, т.к. не улучшают бизнес-процесс компании. Например, мы можем сколь угодно улучшать систему рекомендаций на сайте интернет-магазина,



однако при изучении конверсии понять, что клиент приходит к товару просто из поисковой строки браузера.

При всей своей популярности, классические алгоритмы довольно просты.

Для специалистов в области данных они как основы арифметики – используются постоянно, но часто забываются.

Классическое обучение включает в себя:

1. Обучение с учителем (данные заранее категоризированы или численные))
 - классификация (предсказывает категорию);
 - регрессия (предсказывает значения).
2. Обучение без учителя (данные никак не размечены):
 - кластеризация (разделение по схожести);
 - уменьшение размерности (обобщение) (найти зависимости);
 - ассоциация (выявить последовательность).

Обучение с учителем

Как было сказано выше, основная характеристика систем, разрабатываемых с помощью методов машинного обучения, - способность к обучению. В зависимости от видов решаемых задач применяют различные алгоритмы реализации этой ключевой особенности. В рамках данного пособия рассмотрим три основных вида обучения, а также определим классы задач, подходящие для каждого из этих видов.

Классическое обучение как правило делят на две категории:

- с учителем (Supervised);
- без учителя (Unsupervised Learning).

В обучении с учителем случае машине предоставлен некоторый «учитель», который указывает ей, что в данном случае является правильным.

К примеру, показывает, что на одной картинке изображен светофор, а на другой пальма. «Учитель» в данном случае изначально определяет, что





данные делятся по какому-то признаку, размечает их (где светофор, где пальма), а машина учится на предоставленных примерах.

В обучении без учителя, машина сама устанавливает связи и ассоциации.

Машине просто предоставляют набор изображений животных и ставят задачу определить, кто на кого похож. Данные в этом случае не размечаются, и машина пытается самостоятельно найти любые имеющиеся закономерности (примерно, как «если есть крылья - это птица»).

Разумеется, с «учителем» машина обучается быстрее и точнее, потому в реальных задачах он используется гораздо чаще. Такие задачи делятся на два типа: классификация — предсказание категории объекта, и регрессия — предсказание места на числовой прямой.

Классификация

Классификация объекта — номер или наименование класса, выдаваемый алгоритмом классификации в результате его применения к данному конкретному объекту. Другими словами, это присвоение объекту какого-либо заранее известного признака. Одежду по цветам, документы по языкам, книги по жанрам.

Наиболее часто классификация используется для:

- спам-фильтров (определение признаков нежелательной рассылки);
- определение языка (сопоставление слов словарям);
- поиск похожих документов (по ключевым словам);
- анализ тональности (соотнесение с образцами);
 - распознавание рукописных букв и цифр (сопоставление с эталоном);
 - определение подозрительных транзакций (выявление аномалий);
 - определение объектов при решении задач освещения обстановки и т.д.

Популярные алгоритмы:





- наивный Байеса;
- деревья решений;
- логистическая регрессия;
- К-ближайших соседей;
- машины опорных векторов;
- нейросети и т.д.

Классификация вещей — это наиболее распространенная задача во всём машинном обучении. Машина в ней практически учится раскладывать игрушки в разные ящики: роботов в один ящик, танки в другой. Но не стоит забывать, что может попасться весьма неоднозначный объект. Например, трансформер. И тут уже необходимо вмешательство «учителя».

Чтобы осуществить правильную классификацию всегда нужен человек для разметки данных по признакам и категориям, которые машина будет учиться определять и сопоставлять с признаками (вспомним пример со столом). После этого классифицировать можно что угодно: пользователей по интересам — так делают ленты рекомендаций, книги по языкам и тематикам — важно для поисковиков, музыку по жанрам, и даже письма в электронном почтовом ящике.

Раньше большинство спам-фильтров работало на алгоритме Наивного Байеса. Машина считала, сколько раз слово «казино» встречается в спаме, а сколько раз в обычных письмах. Определялась вероятность появления в спаме и не спаме, эти две вероятности перемножались по формуле Байеса, складывались результаты всех слов и это считалось машинным обучением.

Но системы спама тоже «учились», тем более, что такой фильтр легко обойти, просто вставляя в конец письма или в невидимую область много слов с положительными рейтингами. Метод получил ироничное название Отравление Байеса, а фильтры пришлось существенно модернизировать. Этот метод навсегда остался в учебниках как исторически первый, самый простой, красивый и практически полезный.

Есть другой пример востребованной классификации. Это оценка кредитоспособности заемщика. Каким образом банк может оценить





вероятность возврата кредита? Точно никак, но у банка есть тысячи кредитных историй с профилями других людей, которые уже брали кредит.

Там указан их возраст, образование, должность, уровень зарплаты и главное возврат или невозврат кредита с указанием причин. Несложно понять, каким образом получить желаемый результат. Машина получит данные, найдет закономерности, выдаст ответ. Однако проблема в другом: банк не может просто довериться варианту машины, без каких-либо уточнений. Может быть технический сбой, аномалия или еще что-либо.

Деревья решений. Для решения этой проблемы используются «Деревья Решений». Машина бинарно разделяет все данные по вопросам, ответы на которые «да» или «нет». Пусть даже эти вопросы и не всегда адекватны с точки зрения обычного человека, например, «зарплата заёмщика больше, чем 12345 рублей?», но они придуманы так, чтобы на каждом вопросе разбиение было максимально точным.

Дерево вопросов. Построенная совокупность вопросов формирует *дерево вопросов*. В зависимости от уровня определяется глубина вопроса: чем выше уровень, тем более общий вопрос. Созданные деревья можно отдать аналитикам, и они определяют логику составления запросов. Деревья решений применяются в областях с высокой ответственностью: *диагностике, медицине, финансах*. Два самых популярных алгоритма построения деревьев — CART и C4.5. В чистом виде деревья сегодня используют редко, но вот их ансамбли (леса) лежат в основе довольно крупных систем и зачастую показывает результат лучше, чем нейросети. Например, при запросе Яндексу, именно ансамбль странных деревьев определяет ранжирование результатов. Самым популярным методом классической классификации заслуженно является.

Метод Опорных Векторов (SVM). Он испытывался на максимально разнообразных вещах: видах растений, улыбки на фотографиях, документы по тематикам. Логика SVM довольно проста — он пытается найти способ, как провести две прямые между определенными категориями, чтобы между ними образовался наибольший зазор.





Обратная сторона классификации - *поиск аномалий* - тоже может быть весьма полезной. Если какой-то объекта по различным параметрам не вписывается в установленные классы, он подсвечивается на экране. Сейчас это наиболее распространено в медицине: компьютер подсвечивает врачу все подозрительные области МРТ или выделяет отклонения в анализах. На фондовых биржах аналитические системы определяют нестандартных игроков, которые могут являться инсайдерами. Это следует из того, что, научив машину что есть правильно, сразу получается научить что есть неправильно. Но с течением времени для задач классификации всё чаще используются нейросети, ведь это их основная и первая задача.

Разумное правило: чем сложнее данные — тем сложнее алгоритм.

Для стандартизированного текста, цифр, табличек хорошо работает классическое обучение. Модели менее масштабные, обучаются быстрее и работают понятнее. Для «больших данных» - картинок, видео и юзертрекеров — лучше применять нейросети.

Регрессия

Регрессия - это набор статистических методов исследования влияния одной или нескольких независимых переменных на зависимую переменную.

Сегодня используют для:

- прогноз стоимости ценных бумаг (экстраполяция);
- анализ спроса, объема продаж (выявления зависимостей);
- медицинские диагнозы (прогнозы лечения);
- любые зависимости числа от времени (временные ряды).

Популярные алгоритмы: Линейная или Полиномиальная Регрессия.

Регрессия — вид классификации, только вместо категории тут предсказывается число. Например, стоимость автомобиля в зависимости от пробега, прогноз пробок по времени суток, объем спроса на товар от сезонности и т.д. На регрессию идеально ложатся любые задачи, где есть зависимость от времени.



Регрессию чаще всего применяют финансисты и аналитики, реализуется она даже стандартным функционалом Excel. Внутри всё устроено тоже весьма понятно: машина рисует линию, которая в среднем отражает зависимость. Механизм построения математически точный — считается среднее расстояние до каждой точки и находится оптимальное положение линии.

Если регрессия рисует прямую, её называют линейной, кривую — полиномиальной. Это два основных вида регрессии, но есть еще различные модификации. Так же есть еще *логистическая регрессия*, которая на самом деле регрессией не является, а является одним из методов классификации. Это не надо путать. Регрессия и классификация довольно похожи, это подтверждается тем, что зачастую классификаторы, после до настройки, превращаются в регрессоры. Самый простой способ - это когда не просто определяется к какому классу принадлежит объект, а фиксируется, насколько он близок к нему.

Обучение без учителя

Обучение без учителя (Unsupervised Learning) это более позднее изобретение, но все же 90-х, и в практических задачах используется редко. Чаще всего оно применяется, когда просто нет другого выбора. Размеченные данные, как уже говорилось, довольно редкие и дорогие.

Обучение без учителя используется для решения следующих типов задач:

1. Задача кластеризации.
2. Поиск правил ассоциации.
3. Сокращение размерности данных.
4. Визуализация данных.

В определенной степени каждая из последних трех задач является производной от первой или ее частным случаем.





Каким образом их получать? Нереально все собирать руками. Для этого надо, например, идти на улицу руками фотографировать миллион котиков и подписывать где кто.

Когда изначально разметки нет, возможно аутсорсинговое ее получение, с помощью сервисов типа Яндекс.Толока (Яндекс.Толока — это сервис с оплачиваемыми заданиями по анализу и оценке контента в интернете, которые легко сделать человеку, но не по силам компьютеру). Много людей, которые готовы делать для вас что угодно за небольшую плату. Так обычно и поступают на практике. Большинство серьезных датасетов так и получаются. Так же в таком случае, можно использовать обучение без учителя. Однако примеры реально хорошей работы этого метода найти довольно сложно.

Обучение без учителя чаще всего используется как метод анализа данных, а не как отдельный алгоритм. Человек загружает в машину огромный набор данных и ждет ответов. Возможно будут определены кластеры или выявлены зависимости. Возможно, даже те, которые требовались. Но довольно часто может получиться весьма странная классификация и не трактуемые зависимости. Поэтому, эти методы подходят скорее для проверки имеющихся гипотез, а не для их генерации.

Как следует из описанного выше, обучение без учителя в какой-то мере так или иначе сводится к кластеризации. Поэтому для оценки качества обучения данным способом, как правило, используют метрики качества кластеризации. Причем при их выборе учитывается, что эти метрики не должны зависеть от исходных данных, а только от результатов разбиения. Все оценки качества можно разделить на внешние и внутренние. Первые используют внешнюю информацию об истинном разбиении объектов на кластеры, вторые опираются только на набор исходных данных, то есть данные метрики могут работать с неразмеченной выборкой, когда заранее не известно истинное разбиение объектов на группы. И именно с их помощью определяют оптимальное число кластеров.





Кластеризация

Кластеризация (или кластерный анализ) — это задача разбиения множества объектов на группы, называемые кластерами. При этом признак заранее неизвестен. Машина сама определяет, как определять кластеры.

Сегодня кластеризация используется для:

- сегментация рынка (типов покупателей, лояльности);
- объединение близких точек на карте (зуммирование);
- сжатие изображений (снижение объема);
- анализ и разметка новых данных (из уже имеющихся);
- детекторы аномального поведения (всплески активности).

Популярные алгоритмы: Метод К-средних, Mean-Shift, DBSCAN

Кластеризация фактически является классификацией, только заранее неизвестно ни количество классов, ни их признаки. Машина в данном случае сама определяет схожие объекты и собирает их в кластеры. На самом деле количество кластеров можно задать или оставить на выбор алгоритма. Схожесть объектов машина определяет по тем заданным признакам, которые ей определили: если у объектов наблюдается большое количество совпадений, то они объединяются в один класс.

Задача кластеризации сложнее аппроксимации и классификации. Это обусловлено следующими причинами:

1. Нет однозначного критерия качества кластеризации. Существует ряд эвристических критериев, а также ряд бескритериальных алгоритмов, выполняющих вполне осмысленную кластеризацию, но дающих на одних и тех же данных разные результаты.

2. Число кластеров, как правило, заранее неизвестно и задается субъективно.

3. На результат кластеризации существенное влияние оказывает выбранная метрика расстояния, которая, как правило, также выбирается субъективно.





Однако, несмотря на сложности, кластеризация помогает достичь следующие цели:

1. Улучшить понимание данных за счет выявления их кластерной структуры: разбиение объемной выборки на группы схожих объектов может упростить дальнейшую обработку данных за счет применения к каждому кластеру своих методов анализа.

2. Осуществить сжатие данных. В данном случае подразумевается сокращение объемной выборки за счет работы с наиболее яркими представителями кластеров (групп схожих объектов).

3. Выявить новизну в массиве данных: обнаружить нетипичные объекты, которые не удастся отнести ни к одному кластеру.

4. Решить задачу таксономии: построить древообразную иерархическую структуру, упорядочивающую исходные данные. Ее построение достигается за счет дробления крупных кластеров на более мелкие, которые в свою очередь также дробятся на еще более мелкие. Визуально таксономия отображается в виде графика - *дендрограммы*.

Самый явный пример работы алгоритмов — маркеры на интернет-картах. При поиске круглосуточных магазинов в крупном городе, сайт группирует их исходя из масштаба, чтобы браузер не пытался отобразить миллионы поставленных меток.

Более сложным примером кластеризации являются современные приложения для создания фотогалерей, типа iPhoto или Google Photos. Они определяют лица людей или пейзажи на фотографиях и распределяют их по альбомам. Приложение разделяет фото на портреты, природу, животных, еду и т.д. При этом, конечно, необходимо сначала обучить алгоритм, но этим занимаются сами разработчики приложений.

Сжатие изображений - тоже довольно актуальная задача. Хотя ее тоже можно решать нейросетью. Сохраняя картинку в PNG, можно установить палитру в 16 или 32 цвета. Тогда алгоритм найдет все «более-менее синие» пиксели изображения, определит усредненный оттенок - «условно синий» и



заменит все найденные на него. Меньше цветов в палитре меньше памяти занимает файл.

Но тут может возникнуть проблема с цветами типа «бирюзовый» ■ — система может определить его как синий или как зеленый. Для этого предназначен другой алгоритм кластеризации — Метод К-средних (K-Means). Этот метод называют также метод центра тяжести. На ри.5 представлена иллюстрация метода средних.

На палитру случайным образом наносятся 32 точки, которые называются центроидами. Остальные точки сегментируются к ближайшему центроиду от них — получаются созвездия из самых близких по тону цветов. Далее центроид перемещается в центр своего созвездия до тех пор, пока это возможно. Как только перемещение прекратилось, значит кластеры найдены и стабильны.

Иллюстрация метода К-средних

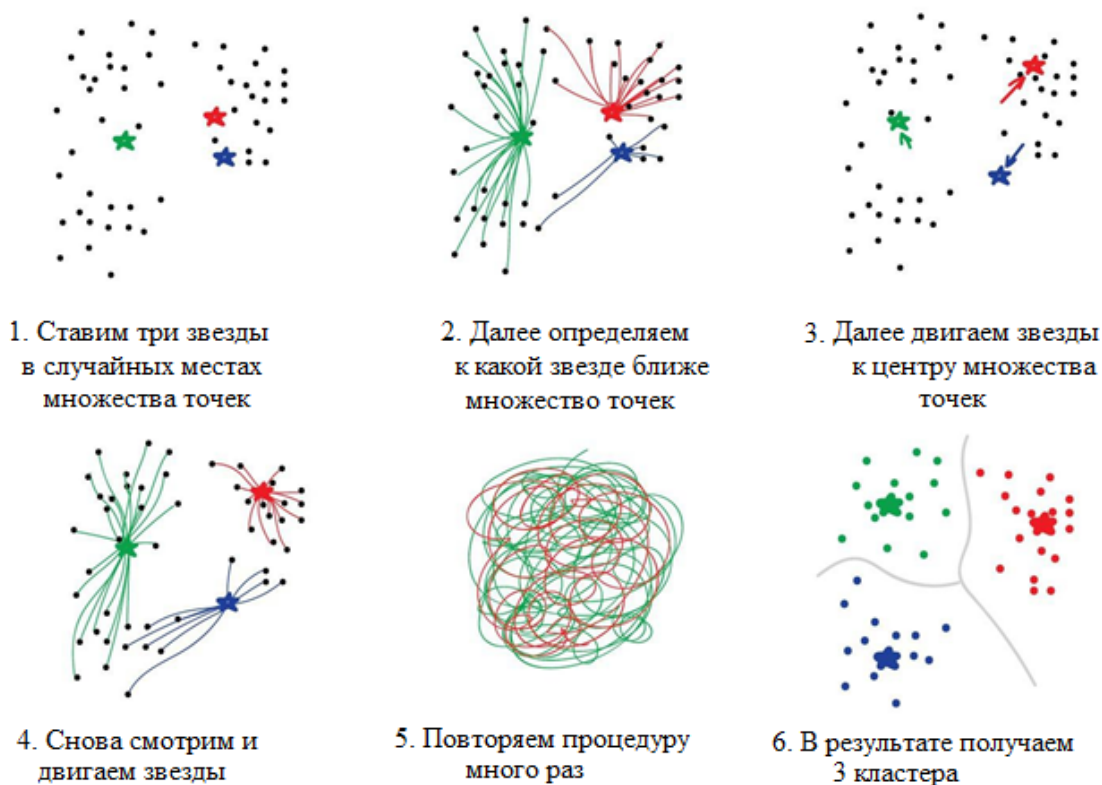


Рис.5. Иллюстрация метода средних



Поиск центроидой довольно простая задача, но в реальной жизни потенциальные кластеры могут быть абсолютно произвольной формы. Например, в сельском хозяйстве агроному может быть нужно определить на карте схожие по структуре типы почв. В этом случае изначально может быть неясно, сколько кластеров получится, да и они скорее всего будут вложены друг в друга.

Для решения более сложных задач существуют более продвинутые методы, такие как DBSCAN. Тут алгоритм сам находит вероятные скопления точек и вокруг строит кластеры. Самый простой пример его работы - если взять за точки людей, стоящих на площади. Любые три близко стоящих человека берутся за руки и образуют кластер. Затем подхватывают тех, до кого могут дотянуться, пока никто больше не сможет взять кого-то за руку. Таким

образом, кластер сформирован. Повторяем до тех пор, пока все не взялись за руки с кем-то. О такие тоже останутся, и это называется аномалиями, или выбросами:

Ровно, как и классификация, алгоритмы кластеризации тоже являются детектором аномалий. Например, трекинг пользователя сайта после регистрации не соответствует большинству. Система может принять решение временно заблокировать его запросить проверку бот это или нет.

При этом нет необходимости точно знать, что такое «нормальное поведение» пользователя. Вся история действий загружается в алгоритм, и машина сама определяет, что считать нормальным. Что, точность тут существенно страдает. И использовать этот подход в задачах с повышенной ответственностью не стоит. Например, доверить машине лечить пациента методом усреднения лечения нескольких предыдущих никто не позволит. Однако, как рекомендация такой способ имеет право на жизнь.





Снижение размерности (Обобщение)

Снижение размерности – это преобразование данных, состоящее в уменьшении числа переменных путём получения главных переменных.

В данном случае алгоритм объединяет конкретные признаки в абстракции более высокого уровня.

Задача сокращения размерности данных состоит в следующем. Существует большой (значительно большой) объем признаковых описаний объектов. Причем этот объем обуславливается внушительным количеством измерений признакового пространства. Необходимо представить те же данные в пространстве меньшей размерности, при этом минимизировав потери информации. Группировка по кластерам как раз и будет одним из вариантов решения проблемы.

Сегодня снижение размерности используют в:

- рекомендательных системах (чаще всего);
- визуализации (для адаптации изображений);
- определении тематики и поиска похожих документов (в многоуровневых классификаторах);
- анализе фейковых изображений (не вписывающихся в общие признаки);
- риск-менеджменте (оценка последствий действий).

Популярные алгоритмы:

- метод главных компонент (*principal component analysis*, PCA);
- метод независимых компонент (ICA);
- Многомерное шкалирование (*Multidimensional scaling*, MDS);
- сингулярное разложение (SVD);
- латентное размещение Дирихле (LDA);
- латентно-семантический анализ (LSA, LSA, GLSA), t-SNE (для визуализации).

В начале эти методы применяли поклонники Data masters, которые пытались в огромном количестве цифр найти какие-то новые закономерности или аномалии. Когда графических способов уже не хватало,





пришла идея научить машины решать такие задачи самостоятельно. Это дало начало разработке методов Dimension Reduction или Feature Learnin.

Практическая польза этих методов состоит в том, что они помогают объединить несколько признаков в один и получить некоторый уровень абстракции. При этом теряется информация о конкретных представителях выбранного вида, но эта информация на самом деле может быть и несущественной. При этом надо отметить, что, обучение на меньшем количестве признаков проходит существенно быстрее. Такой метод очень хорошо подходит для определения тематик текстов (Topic Modelling).

При это получается поднять уровень абстракции от конкретных слов до уровня смыслов или значений даже без привлечения учителя, который бы распределял их по спискам категорий. Этот алгоритм называется латентно-семантический анализ (LSA), и логика его заключается в статистическом принципе: частота появления слова в тексте зависит от его тематики.

На рис.6 представлена проекция двумерных данных на прямую (PCA).

Например, в научных статьях больше различных терминов, в политических новостях — фамилий известных политиков. Конечно, на статистических зависимостях лучше работают классические методы. И можно было бы просто взять все слова и кластеризовать их. Но при этом теряются полезные связи между словами и смыслами, например, что батарейка и аккумулятор имеют одно и то же значение в разном контексте, а какие-то слова могут его менять.



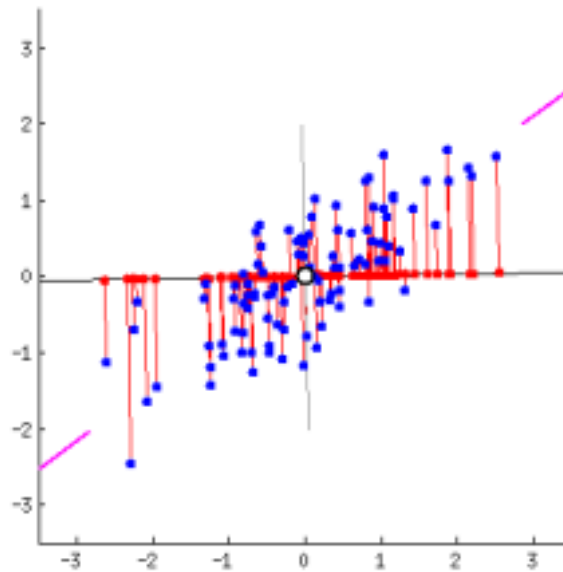


Рис.6. Проекция двумерных данных на прямую (РСА)

Сингулярное разложение. Точность латентно-семантический анализа оставляет желать лучшего. Для начала нужно каким-то образом объединить слова и документы в один признак, и при этом не потерять эти латентные (скрытые) связи. В принципе, для этого используется, сингулярное разложение (SVD), которое может выявлять полезные тематические кластеры из слов, часто встречающихся вместе.

В настоящее время Сейчас, наверное, самым популярным применением метода уменьшения размерности являются рекомендательные системы и коллаборативная фильтрация. Поднимая уровень абстракции в оценках пользователями контента, может получиться неплохая система рекомендаций кино, музыки, игр и чего угодно вообще. Полученная абстракция практически не воспринимается человеком напрямую, но при детальном исследовании новых признаков выяснилось, что какие-то из них явно коррелируют с возрастом пользователя (дети чаще играли в покемонов и смотрели мультфильмы), другие с определёнными жанрами кино или литературы, а третьи вообще с синдромом поиска глубокого смысла. При этом алгоритм, знавший только оценки пользователей, смог добраться до таких сложных и высоких взаимосвязей, даже не воспринимая их. А дальше



уже открывается простор для аналитиков, которые будут выяснять, почему такие закономерности имеют место быть.

Ассоциация (Поиск правил)

Метод ассоциаций заключается в поиске закономерностей в наборах объектов.

В настоящее время метод ассоциаций используют для:

- прогноза акций и распродаж (каким образом спроектировать предложение);
- анализа товаров, покупаемых вместе (для комплексных предложений);
- расстановки товаров на полках (чтобы связанные товары стояли рядом, или наоборот);
- анализ паттернов поведения на веб-сайтах (оценка восприятия контента);

Популярные алгоритмы: Apriori, Euclat, FP-growth.

Эти алгоритмы объединяют все методы анализа продуктовых корзин, стратегий маркетинга и других последовательностей. Классический пример применения: покупатель берёт в дальнем углу магазина колбасу и следует к кассе. Должен ли на его пути быть хлеб? Масло? Насколько часто люди покупают их вместе? Какие еще товары обычно добавляются к этому набору? Для владельцев продуктовой сети очень часто правильная расстановка приносит огромную прибыль. А если мы рассматриваем интернет-магазин, то задача становится еще интереснее — необходимо определить, за каким товаром покупатель вернётся в следующий раз?

При всей его практической ценности, поиск правил — самая слабо проработанная категория среди всех методов обучения. Классические способы заключаются в банальном переборе пар всех купленных товаров с построением деревьев или наполнением множеств. Сами алгоритмы работают максимум наполовину — они вполне четко находят





закономерности, но не умеют обобщать или воспроизводить их на новых примерах.

В реальности каждая крупная торговая сеть пытается выстроить свои системы. Наиболее успешным примером принято считать Амазон, но это тоже весьма условный лидер. Максимальный уровень используемых технологий — сформировать систему рекомендаций, как предыдущем пункте. Но однако, зачастую после покупки утюга вы будете несколько месяцев получать предложения на новые утюги, причем, что обидно, может быть даже более выгодные, чем было.

Обучение с подкреплением

Обучение с подкреплением этот метод машинного обучения, при котором происходит обучение модели, которая не имеет сведений о системе, но имеет возможность производить какие-либо действия в ней.

Постановка задачи обучения с подкреплением:

1. Есть среда, в которой действует агент. У среды есть состояние.
2. Агент может совершать действия.
3. Действия приносят некий результат (reward) (не обязательно мгновенно).
4. Надо научиться такой модели поведения, которая максимизирует результат.

Наиболее часто используется для:

- самоуправляемых автомобилей (навигация и предотвращение ДТП);
- управление роботами;
- игр (виртуальные персонажи);
- автоматической торговли (в частности, автотрейдинг ценных бумаг);
- управления ресурсами предприятий (системы JIT).

Популярные алгоритмы: Q-Learning, SARSA, DQN, A3C, генетический алгоритм.





Эти алгоритмы и методы, представляют собой так называемый «настоящий искусственный интеллект». Часто в источниках (особенно при рассмотрении нейросетей) его описывают как нечто среднее между обучением с учителем и без него, однако этот метод становится все более самостоятельным и представляет уже принципиально другой раздел машинного обучения.

Задачи обучения с подкреплением возникают тогда, когда задачей является не анализ данных, а деятельность в реальной среде. Средой, к примеру, может выступать видеоигра. Роботы, играющие в Марио, были популярны еще несколько лет назад. Часто средой становится реальный мир. Всем известен автопилот Тесла Моторс, задача которого - довезти пассажира до места назначения и не попасть в ДТП. Популярность набирают роботы-пылесосы, которые обеспечивают качественную уборку, при этом стараясь не разбиться о мебель и не травмировать домашних животных.

Познания о внешней среде такому алгоритму могут быть полезны, но исключительно в справочных целях. Вне зависимости от объема собранных данных возможности предусмотреть все ситуации просто не будет. Потому в данном случае важно минимизировать ошибки, а не рассчитать все ходы. Робот должен научиться выполнять задачи в пространстве с максимальной выгодой: собранными монетками в Марио, временем поездки в Тесле. При этом необходимо соблюдать ограничения - не попасть на врагов или не спровоцировать ДТП.

Основная идея обучения с подкреплением - обеспечить выполнение реальных задач в реальной среде.

Раз на людях работает методика штрафов за ошибки и премирования за правильное поведение, значит и машина сможет этому научиться. Умные модели роботов-пылесосов и самоуправляемые автомобили обучаются именно так: создается модель города (зачастую на основе карт реальных городов), устанавливаются случайные пешеходы и запускает обучение со штрафами за сбитого человека или клумбу. Когда робот уверенно ориентируется в виртуальном мире, его выпускают на реальные улицы. Причем запоминать сам город роботу не нужно — это называется



независимым от моделей подходом (Model-Free). Конечно, есть и основанный на моделировании классический подход (Model-Based), но в этом случае необходимо запоминать все карты мира и все возможные дорожные ситуации. Это конечно не сработает. Попробуйте создать карту для робота-пылесоса, а потом поставить посередине комнаты вазу с цветами, и посмотрим, как он будет ее обходить. В обучении с подкреплением машина не запоминает каждое движение, а пытается обобщить ситуации, чтобы выходить из них с максимальной выгодой

Ансамбли

Ансамбли - использование нескольких связанных между собой деревьев решений, которые направлены на повышение точности друг друга.

Сегодня ансамбли используют для:

- всего, где подходят классические алгоритмы (но с многократно большей точностью);
- поисковые системы (классическая задача для деревьев);
- компьютерное зрение (определение расстояний, форм и т.д.);
- распознавание объектов (в том числе и живых).

Популярные алгоритмы: Random Forest, Gradient Boosting.

Ансамбли — это класс методов, который так же является настоящим искусственным интеллектом. Ансамбли, нейросети — это основные инструменты обучения машин решению реальных задач. Сегодня они дают самые точные результаты и используются большинством крупных.

И тут работает *золотое правило Бритвы Оккама*. Задача должна решаться самым простым способом. На самом деле, если взять несколько не очень эффективных методов обучения и научить исправлять ошибки друг друга, то качество такой системы будет намного выше, чем каждого из методов по отдельности.



И что самое интересно, даже лучше, когда взятые алгоритмы нестабильны, сильно расходятся и зависят от входных данных. Чаще всего используют Регрессию и Деревья Решений, в которых хотя бы одна аномалия приводит к расхождению всей модели. При этом методы Байеса и K-NN практически не используют, т.к. они достаточно стабильны. Ансамбль можно собрать абсолютно любым способом и в любом порядке. Можно даже называть этот способ «Салатом». Правда, точность при этом не будет гарантирована. Потому есть три проверенных способа делать ансамбли.

Стекинг. Обучаем несколько разных алгоритмов и передаём их результаты на вход последнему, который принимает итоговое решение. Каким-то образом это напоминает экспертные системы, которые сначала оценивают варианты, а потом выбирают наилучший.

На рис.7 представлена процедура стекинга.

Самое главное здесь, чтобы задача решалась разными алгоритмами, ведь учить один и тот же алгоритм на одних и тех же данных не имеет смысла. Какие алгоритмы выбирать в качестве начальных - дело вкуса, но в качестве решающего алгоритма лучше брать регрессию

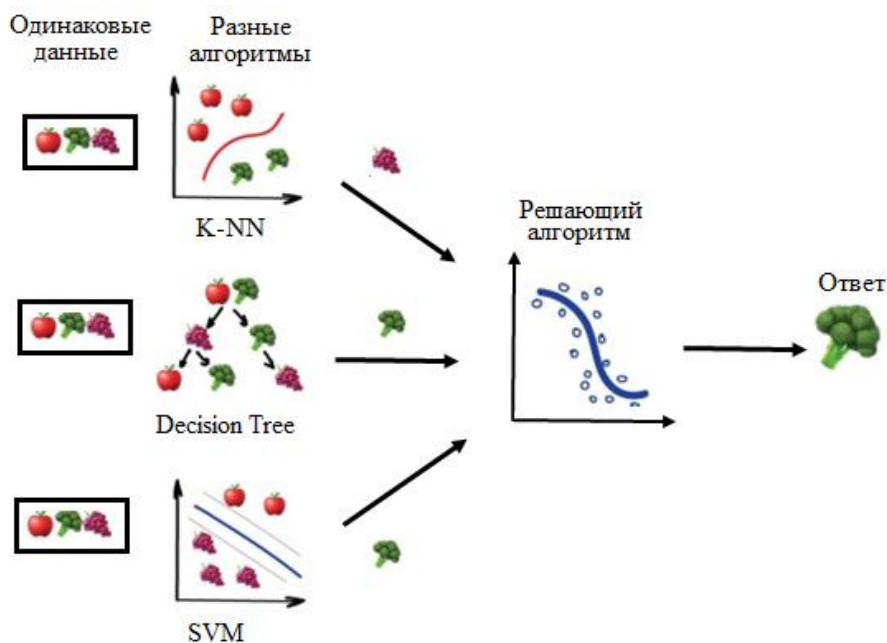


Рис.7. Стекинг.

Беггинг он же Bootstrap AGGREGatING. Есть еще один интересный метод.

Сущность беггинга заключается в следующем.

1. Выбираем базовый алгоритм.
2. Обучаем алгоритм на случайных выборках из исходных данных много раз.
3. На финальном этапе результаты просто усредняются.

Самое интересное, что данные в этих случайных выборках могут повторяться. Из маленького набора 1-2-3 можно собрать выборки 2-2-3, 1-2-2, 3-1-2 и далее по теории вероятностей. Подаем полученные выборки на алгоритм и включаем оценочное голосование.

Самый популярный пример беггинга – алгоритм Random Forest. Именно он применяется в распознавании образов. Нейросеть слишком медленно работает в реальном времени на обычных мощностях, а беггинг может считать свои деревья параллельно на всех шейдерах видеокарты. На рис.8 представлена процедура беггинга.

Именно эта способность распараллеливаться даёт беггингу преимущество даже над другим методом (бустинга), который работает точнее, но по конвейерной схеме. В этом случае можно конечно применить искусственный параллелизм, но это приведет к значительному уже усложнению задачи.

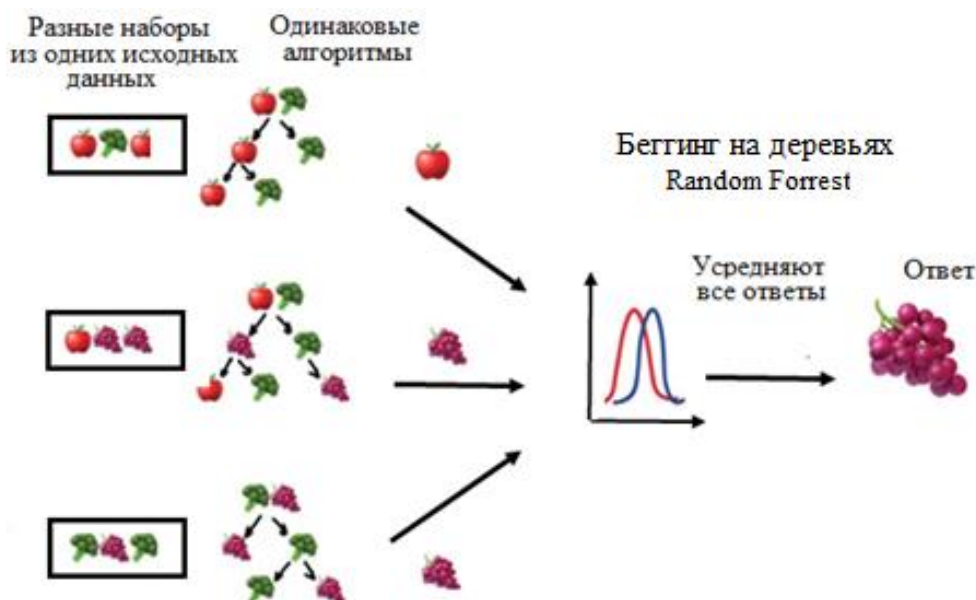


Рис.8. Беггинг

Бустинг. Идея состоит в том, алгоритмы обучаются последовательно, при этом каждый следующий уделяет особое внимание ситуациям, в которых ошибся предыдущий. Так же, как и в предыдущем случае, делаются выборки из исходных данных, но уже не совсем случайно. В каждую следующую выборку обязательно включается часть тех данных, на которых ошибся предыдущий алгоритм. Получается, что мы доучиваем новый алгоритм, реализуем так называемый «опыт поколений» (рис.9).

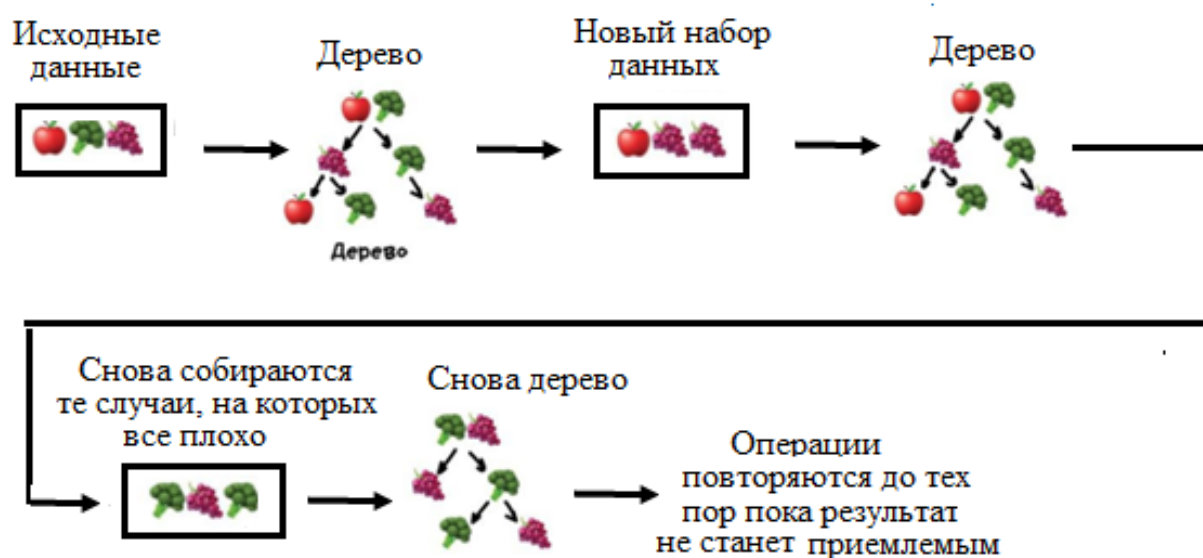


Рис.9. Бустинг

Плюсом является запредельная точность классификации, может быть даже большая, чем у экспертных систем. Минусы - последовательная обработка. Однако, все равно при этом работает все равно быстрее нейросетей. Реальный пример работы бустинга - ранжирование результатов работы Яндекс.

Глубокое обучение и нейросети

Нейронная сеть (также искусственная нейронная сеть, ИНС) — математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования



биологических нейронных сетей — сетей нервных клеток живого организма.

Это понятие возникло при изучении процессов, протекающих в мозге, и при попытке смоделировать эти процессы.

«Есть большое количество ресурсов, подготовленная нейронная сеть, условная эффективность и нечеткие задачи. Решает задачи очень хорошо, но непонятно какие».

Сегодня используют для:

- вместо всех вышеперечисленных алгоритмов;
- распознавание объектов на фото и видео (определение облаков, номерных знаков);
- распознавание и синтез речи;
- обработка изображений, перенос стиля (сжатие, расшифровка);
- машинный перевод (обработка текстов).

Наиболее часто используемые архитектуры:

- перцептрон;
- сверточные сети (CNN);
- рекуррентные Сети (RNN);
- автоэнкодеры;
- трансформеры;
- сети внимания.

Любая нейросеть представляет набор нейронов и связей между ними (синапсы). Нейрон представляет собой функцию с большим количеством входов (аксонов) и одним выходом (сوما). Задача нейрона - преобразовать численные значения с входов выполнением функции и передать результат на выход. Простой пример самого простого нейрона: определить сумму цифр на входах, и, если их сумма больше заданного числа — выдать на выход единицу, иначе — ноль. Связи - это каналы, по которым передаются внутри данные. У каждой связи есть вес - единственный параметр, который можно условно представить как степень значимости связи. Когда через связь с весом 0.3 проходит число





100, оно превращается в 30. Собственно нейрону не важно, что поступило на вход, он выполняет свою заложенную функцию, например, суммирует всё подряд. Веса каналов и нужны, чтобы определять, на какие входы нейрон должен реагировать, а на какие нет.

Для упорядочивания связей, нейроны распределяются по слоям. Внутри одного слоя нейроны никак не связаны, но соединены с нейронами следующего и предыдущего слоя. Эта организация называется многослойной сетью. Данные в ней идут только в одном направлении – от входов первого слоя к выходам последнего. При правильной организации связей, балансировке весов, реализуется следующий алгоритм работы: на вход подается изображение, например, написанной от руки буквы, чёрные пиксели активируют связанные с их сектором нейроны, активируются следующие слои, и так далее, пока в итоге не загорится выход, отвечающий за нужную букву. Примерно, как конструктор, который соберет букву по имеющимся секторам. В зависимости от количества начальных признаков будет увеличиваться точность.

В реальных задачах, разумеется, всё представляется матрицами и матричными произведениями, потому что необходимо осуществлять вычисления в режиме реального времени

вычисления в режиме реального времени.

Многосвязная сеть, где имеется несколько слоёв, а все нейроны полносвязные, называется многослойным персептроном (MLP) и считается самой простой архитектурой.

Предназначена она чаще всего для учебных задач и понимания сути организации сетей. После выбора архитектуры следующей задачей является правильная расстановка весов, чтобы обеспечить реакцию нейронов на нужные сигналы. Для этого используется обучающая выборка: данные, которые уже были соотнесены – примеры «входов» и правильных «выходов». На вход подается изображение буквы и осуществляется подбор весов так, чтобы значение на выходе ей соответствовало. Сначала веса расставляются случайно, показывается изображение, и сеть выдаёт какой-то случайный ответ (из-за отсутствия весов). Наша задача в данном случае



определить погрешность. После этого осуществляется проход в обратном направлении, от выходов ко входам, и определяются ложные срабатывания. Исходя из этого происходит корректировка весов. И так продолжается до тех пор, пока погрешность не станет более-менее приемлемой.

Такой подход называется «Метод обратного распространения ошибки» (Backpropagation). Метод предложили около 20 лет назад. В ряде задач правильно обученная нейросеть может копировать действия любого вышеупомянутого алгоритма, при этом реально показывая более высокую точность. Когда-то даже считалось, что это универсальная архитектура, подобная человеческому мозгу, способная с увеличением количества слоев обучаться на любых данных. Но на самом деле прирост количества слоев сети требует такого же прироста ресурсов. А тогда это было довольно сложно для слабеньких компьютеров. И решение об их развитии подвисло. Однако, в 2010-х годах активно начало развиваться глубокое обучение. В 2012 году сверточная нейросеть победила другие алгоритмы в ImageNet, и после этого *методы глубокого обучения* получили второе дыхание. В настоящее время вычислительные возможности современных ЭВМ на порядки превосходят по мощности ЭВМ 2000-2012 года и справляются с тривиальными для нынешних сетей задачами.

Принципиальное отличие глубокого обучения от классических нейросетей состоит в использовании новых методов обучения, которые рассчитаны на большие размеры сетей. В настоящее время наиболее часто в практическом применении нейросетей используется язык Python и его библиотеки KMeans, Keras, TensorFlow и PyTorch. Эти методики одни из самых удобных и простых в применении.

1 РЕГРЕССИЯ: ЛИНЕЙНАЯ, ЛОГИСТИЧЕСКАЯ

Регрессия — зависимость математического ожидания случайной величины от одной или нескольких других случайных величин (свободных переменных), то есть $E(y|x) = f(x)$. Регрессионным анализом называется поиск такой функции f , которая описывает эту зависимость. Регрессия





может быть представлена в виде суммы неслучайной и случайной составляющих.

$$y = f(x) + \nu,$$

где f — функция регрессионной зависимости, а ν — аддитивная случайная величина с нулевым матожиданием. Предположение о характере распределения этой величины называется гипотезой порождения данных. Обычно предполагается, что величина ν имеет гауссово распределение с нулевым средним и дисперсией σ_ν^2 .

Задача нахождения регрессионной модели нескольких свободных переменных ставится следующим образом. Задана выборка — множество $\{x_1, \dots, x_N | x \in \mathbb{R}^M\}$ значений свободных переменных и множество $\{y_1, \dots, y_N | y \in \mathbb{R}\}$ соответствующих им значений зависимой переменной. Эти множества обозначаются как D , множество исходных данных $\{(x, y)_i\}$. Задана регрессионная модель — параметрическое семейство функций $f(w, x)$ зависящая от параметров $w \in \mathbb{R}$ и свободных переменных x . Требуется найти наиболее вероятные параметры $\bar{w}$:

$$\bar{w} = \underset{w \in \mathbb{R}^W}{\operatorname{argmax}} p(y|x, w, f) = p(D|w, f).$$

Функция вероятности p зависит от гипотезы порождения данных и задается Байесовским выводом или методом наибольшего правдоподобия.

1.1 Обучение с учителем

Обучение с учителем (Supervised learning) — один из разделов машинного обучения, посвященный решению следующей задачи. Имеется множество объектов (ситуаций) и множество возможных ответов (откликов, реакций). Существует некоторая зависимость между ответами и объектами, но она неизвестна. Известна только конечная совокупность прецедентов — пар «объект, ответ», называемая обучающей выборкой. На основе этих данных требуется восстановить зависимость, то есть построить алгоритм, способный для любого объекта выдать достаточно точный ответ. Для измерения точности ответов определённым образом вводится функционал





качества. Под учителем понимается либо сама обучающая выборка, либо тот, кто указал на заданных объектах правильные ответы.

1.2 Стандартный/стохастический метод градиентного спуска

Рассмотрим задачу поиска минимума функции $f(x): \mathbb{R}^n \rightarrow \mathbb{R}$, записываемая в виде:

$$f(x) \rightarrow \min_{x \in \mathbb{R}^n}$$

Пусть функция $f(x)$ такова, что можно вычислить ее градиент. Тогда можно применить метод градиентного спуска, описанный ниже.

Также будут приведены теоремы сходимости метода градиентного спуска, а также рассмотрены его варианты:

- Метод градиентного спуска с постоянным шагом
- Метод градиентного спуска с дроблением шага
- Метод наискорейшего спуска

Идея метода

Основная идея метода заключается в том, чтобы осуществлять оптимизацию в направлении наискорейшего спуска, а это направление задаётся антиградиентом $-\nabla f$:

$$x^{[k+1]} = x^{[k]} - \lambda^{[k]} \nabla f(x^{[k]})$$

где $\lambda^{[k]}$ выбирается:

- постоянной, в этом случае метод может расходиться;
- дробным шагом, т.е. длина шага в процессе спуска делится на некое число;
- наискорейшим спуском: $\lambda^{[k]} = \operatorname{argmin}_{\lambda} f(x^{[k]} - \lambda \nabla f(x^{[k]}))$

Алгоритм

Вход: функция $f: \mathbb{R}^n \rightarrow \mathbb{R}$

Выход: найденная точка оптимума x

1. Повторять:
2. $x^{[k+1]} = x^{[k]} - \lambda^{[k]} \nabla f(x^{[k]})$, где $\lambda^{[k]}$ выбирается одним из описанных выше способов





3. если выполнен критерий останова, то возвращаем текущее значение $x^{[k+1]}$

Критерий останова

Критерии остановки процесса приближенного нахождения минимума могут быть основаны на различных соображениях. Некоторые из них:

1. $\|x^{[k+1]} - x^{[k]}\| \leq \epsilon$
2. $\|f(x^{[k+1]}) - f(x^{[k]})\| \leq \epsilon$

Здесь $x^{[k]} \in \mathbb{R}^n$ - значение, полученное после k -го шага оптимизации. ϵ - наперед заданное положительное число.

Сходимость градиентного спуска с постоянным шагом

Теорема 1 о сходимости метода градиентного спуска с постоянным шагом.

Пусть $\lambda^{[k]} = \lambda = \text{const}$, функция f дифференцируема, ограничена снизу. Пусть выполняется условие Липшица для градиента $f'(x)$: $\|f'(x) - f'(y)\| \leq L\|x - y\|$. Пусть $0 < \lambda < 2/L$.

Тогда $\lim_{k \rightarrow \infty} f'(x^{[k]}) = 0$, $f(x^{[k+1]}) < f(x^{[k]})$ при любом выборе начального приближения.

В условиях теоремы градиентный метод обеспечивает сходимость $\{x^{[k]}\}$ либо к точной нижней грани $\inf_x f(x)$ (если функция $f(x)$ не имеет минимума) либо к

значению $f(x^*)$: $\lim_{k \rightarrow \infty} x^{[k]} = x^*$, $f'(x^*) = 0$. Существуют примеры, когда в точке x^* реализуется седло, а не минимум. Тем не менее, на практике методы градиентного спуска обычно обходят седловые точки и находят локальные минимумы целевой функции.

Определение. Дифференцируемая функция f называется сильно выпуклой (с константой $\Lambda > 0$), если для любых x и y из $\mathbb{R}^n$ справедливо

$$f(x+y) \geq f(x) + \langle f'(x), y \rangle + \Lambda \|y\|^2 / 2.$$

Теорема 2 о сходимости метода градиентного спуска с постоянным шагом.



Пусть функция f дифференцируема, сильно выпукла с константой Λ . Пусть выполняется условие Липшица для градиента $f'(x)$: $\|f'(x) - f'(y)\| \leq L\|x - y\|$. Пусть $0 < \lambda < 2/L$.

Тогда $\lim_{k \rightarrow \infty} x^{[k]} = x^*$, $\|x^{[k]} - x^*\| \leq q^k \|x^{[0]} - x^*\|$, $q = \max\{|1 - \lambda\Lambda|, |1 - \lambda L|\}$ при любом выборе начального приближения.

Выбор оптимального шага

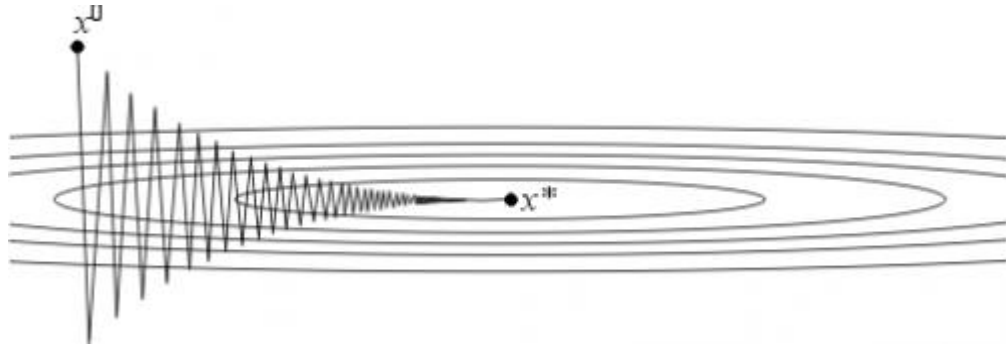


Рис.11 Ситуация, когда метод градиентного спуска сходится плохо.

Константа q , фигурирующая в теореме 2 и характеризующая скорость сходимости метода, зависит от шага λ . Нетрудно видеть, что величина $q = q(\lambda) = \max\{|1 - \lambda\Lambda|, |1 - \lambda L|\}$ минимальна, если шаг λ выбирается из условия $|1 - \lambda\Lambda| = |1 - \lambda L|$, т. е. если $\lambda = \lambda^* = 2/(\Lambda + L)$.

При таком выборе шага оценка сходимости будет наилучшей и будет характеризоваться величиной:

$$q = q^* = \frac{L - \Lambda}{L + \Lambda}.$$

В качестве Λ и L могут выступать равномерные по x оценки сверху и снизу собственных значений оператора $f''(x)$. Если $\lambda \ll \Lambda$, то $q^* \approx 1$ и метод сходится очень медленно. Геометрически случай $\lambda \ll \Lambda$ соответствует функциям с сильно вытянутыми линиями уровня. Простейшим примером такой функции может служить функция на $\mathbb{R}^2$, задаваемая формулой:

$$f(x_1, x_2) = \Lambda x_1^2 + L x_2^2, \quad \Lambda \ll L.$$

Поведение итераций градиентного метода для этой функции изображено на рис. 11 — они, быстро спустившись на "дно оврага", затем медленно "зигзагообразно" приближаются к точке минимума. Число $\mu = L/\Lambda$ (характеризующее, грубо говоря, разброс собственных



значений оператора $f''(x)$ называют числом обусловленности функции f . Если $\mu \gg 1$, то функции называют плохо обусловленными или овражными. Для таких функций градиентный метод сходится медленно.

Но даже для хорошо обусловленных функций проблема выбора шага нетривиальна в силу отсутствия априорной информации о минимизируемой функции. Если шаг выбирается малым (чтобы гарантировать сходимость), то метод сходится медленно. Увеличение же шага (с целью ускорения сходимости) может привести к расходимости метода. Далее будут описаны два алгоритма автоматического выбора шага, позволяющие частично обойти указанные трудности.

Градиентный метод с дроблением шага

В этом варианте градиентного метода величина шага $\lambda^{[k]}$ на каждой итерации выбирается из условия выполнения неравенства

$$f(x^{[k+1]}) = f(x^{[k]} - \lambda^{[k]} f'(x^{[k]})) \leq f(x^{[k]}) - \epsilon \lambda^{[k]} \|f'(x^{[k]})\|^2, \quad (1)$$

где $\epsilon \in (0,1)$ - некоторая заранее выбранная константа.

Процедуру нахождения такого $\lambda^{[k]}$ обычно оформляют так. Выбирается число $\delta \in (0,1)$ и некоторый начальный шаг $\lambda^{[0]}$. Теперь для каждого k полагают $\lambda^{[k]} = \lambda^{[0]}$ и делают шаг градиентного метода. Если с таким $\lambda^{[k]}$ условие (1) выполняется, то переходят к следующему k . Если же (1) не выполняется, то умножают $\lambda^{[k]}$ на δ ("дробят шаг") и повторяют эту процедуру до тех пор пока неравенство (1) не будет выполняться. В условиях теоремы 1 эта процедура для каждого k за конечное число шагов приводит к нужному $\lambda^{[k]}$.

Можно показать, что в условиях теоремы 2 градиентный метод с дроблением шага линейно сходится. Описанный алгоритм избавляет нас от проблемы выбора $\lambda^{[k]}$ на каждом шаге, заменяя ее на проблему выбора параметров ϵ , δ и $\lambda^{[0]}$, к которым градиентный метод менее чувствителен. При этом, разумеется, объем вычислений возрастает (в связи с необходимостью процедуры дробления шага), впрочем, не очень сильно, поскольку в большинстве задач основные вычислительные затраты ложатся на вычисление градиента.



Метод наискорейшего спуска

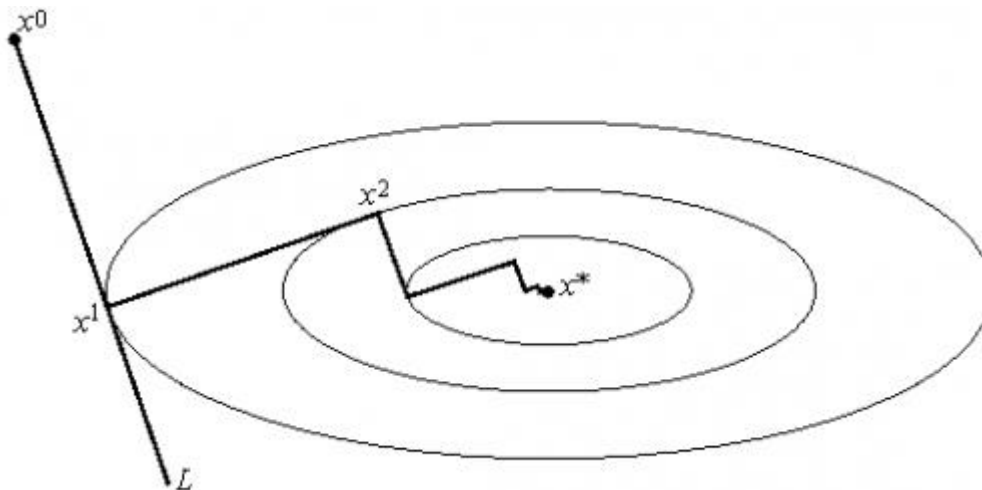


Рис.12 Геометрическая интерпретация метода наискорейшего спуска.

На каждом шаге $\lambda^{[k]}$ выбирается так, чтобы следующая итерация была точкой минимума функции f на луче L .

Этот вариант градиентного метода основывается на выборе шага из следующего соображения. Из точки $x^{[k]}$ будем двигаться в направлении антиградиента до тех пор пока не достигнем минимума функции f на этом направлении, т. е. на луче $L = \{x = x^{[k]} - \lambda f'(x^{[k]}); \lambda \geq 0\}$:

$$\lambda^{[k]} = \arg \min_{\lambda \in [0, \infty)} f(x^{[k]} - \lambda f'(x^{[k]}))$$

Другими словами, $\lambda^{[k]}$ выбирается так, чтобы следующая итерация была точкой минимума функции f на луче L (см. рис. 12). Такой вариант градиентного метода называется методом наискорейшего спуска. Заметим, кстати, что в этом методе направления соседних шагов ортогональны.

Метод наискорейшего спуска требует решения на каждом шаге задачи одномерной оптимизации. Практика показывает, что этот метод часто требует меньшего числа операций, чем градиентный метод с постоянным шагом.

В общей ситуации, тем не менее, теоретическая скорость сходимости метода наискорейшего спуска не выше скорости сходимости градиентного метода с постоянным (оптимальным) шагом.



1.3 Метод наименьших квадратов

Метод наименьших квадратов – метод нахождения оптимальных параметров линейной регрессии, таких, что сумма квадратов ошибок (регрессионных остатков) минимальна. Метод заключается в минимизации евклидова расстояния $|Aw - y|$ между двумя векторами – вектором восстановленных значений зависимой переменной и вектором фактических значений зависимой переменной.

Задача метода наименьших квадратов состоит в выборе вектора w , минимизирующего ошибку $S = |Aw - y|^2$. Эта ошибка есть расстояние от вектора y до вектора Aw . Вектор Aw лежит в пространстве столбцов матрицы A , так как Aw есть линейная комбинация столбцов этой матрицы с коэффициентами $w_1, \dots, w_N$. Отыскание решения w по методу наименьших квадратов эквивалентно задаче отыскания такой точки $p = Aw$, которая лежит ближе всего к y и находится при этом в пространстве столбцов матрицы A . Таким образом, вектор p должен быть проекцией y на пространство столбцов и вектор невязки $Aw - y$ должен быть ортогонален этому пространству. Ортогональность состоит в том, что каждый вектор в пространстве столбцов есть линейная комбинация столбцов с некоторыми коэффициентами $v_1, \dots, v_N$, то есть это вектор Av . Для всех v в пространстве Av , эти векторы должны быть перпендикулярны невязке $Aw - y$:

$$(Av)^T(Aw - y) = v^T(A^T Aw - A^T y) = 0.$$

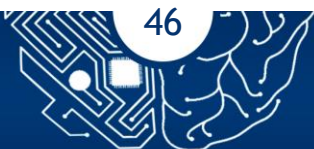
Так как это равенство должно быть справедливо для произвольного вектора v , то

$$A^T Aw - A^T y = 0.$$

Решение по методу наименьших квадратов несовместной системы $Aw = y$, состоящей из M уравнений с N неизвестными, есть уравнение

$$A^T Aw = A^T y,$$

которое называется *нормальным уравнением*. Если столбцы матрицы A линейно независимы, то матрица $A^T A$ обратима и единственное решение





$$w = (A^T A)^{-1} A^T y.$$

Проекция вектора y на пространство столбцов матрицы имеет вид

$$p = Aw = A(A^T A)^{-1} A^T y = Py.$$

Матрица $P = A(A^T A)^{-1} A^T$ называется матрицей проектирования вектора y на пространство столбцов матрицы A . Эта матрица имеет два основных свойства: она идемпотентна, $P^2 = P$, и симметрична, $P^T = P$. Обратное также верно: матрица, обладающая этими двумя свойствами есть матрица проектирования на свое пространство столбцов.

Метод взвешенных наименьших квадратов (WLS), также известный как взвешенная линейная регрессия, представляет собой обобщение обычного метода наименьших квадратов и линейной регрессии, в котором знание дисперсии наблюдений включается в регрессию. WLS также является специализацией обобщенных наименьших квадратов.

1.4 Метод Ньютона

Постановка задачи одномерной оптимизации.

Задача одномерной оптимизации определяется следующим образом:

1. Допустимое множество – множество $X \subseteq \mathbb{R}$;
2. Целевую функцию – отображение $f: X \rightarrow \mathbb{R}$;
3. Критерий поиска (max или min).

Тогда решить задачу $f(x) \rightarrow \min_{x \in X}$ означает одно из:

1. Показать, что $X = \emptyset$.
2. Показать, что целевая функция $f(x)$ не ограничена.
3. Найти $x^* \in X: f(x^*) = \min_{x \in X} f(x)$.
4. Если $\nexists x^*$, то найти $\inf_{x \in X} f(x)$.

Если минимизируемая функция не является выпуклой, то часто ограничиваются поиском локальных минимумов и максимумов: точек x_0 таких, что всюду в некоторой их окрестности $f(x) \geq f(x_0)$ для минимума и $f(x) \leq f(x_0)$ для максимума.





Если допустимое множество $X = \mathbb{R}$, то такая задача называется *задачей безусловной оптимизации*, в противном случае – *задачей условной оптимизации*.

Метод Ньютона

Это итерационный численный метод нахождения корня (нуля) заданной функции. Метод был впервые предложен английским физиком, математиком и астрономом Исааком Ньютоном (1643–1727), под именем которого и обрёл свою известность. Поиск решения осуществляется путём построения последовательных приближений и основан на принципах простой итерации. Метод обладает квадратичной сходимостью. В случае решения задач оптимизации предполагается, что функция $f(x)$ дважды непрерывно дифференцируема. Отыскание минимума функции $f(x)$ производится при помощи отыскания стационарной точки, т.е. точки x^* , удовлетворяющей уравнению $f'(x) = 0$, которое решается методом Ньютона.

Если x^k - точка, полученная на k -м шаге, то функция $f'(x)$ аппроксимируется своим уравнением касательной:

$$y = f'(x^k) + (x - x^k)f''(x^k),$$

а точка x^{k+1} выбирается как пересечение этой прямой с осью Ox , т.е.

$$x^{k+1} = x^k - \frac{f'(x^k)}{f''(x^k)}.$$

Неудобство этого метода состоит в необходимости вычисления в каждой точке первой и второй производных. Значит, он применим лишь тогда, когда функция $f(x)$ имеет достаточно простую аналитическую форму, чтобы производные могли быть вычислены в явном виде вручную. Действительно, всякий раз, когда решается новая задача, необходимо выбрать две специфические подпрограммы (функции) вычисления производных $f'(x)$ и $f''(x)$, что не позволяет построить общие алгоритмы, т.е. применимые к функции любого типа.

Когда начальная точка итераций x_0 достаточно близка к искомому минимуму, скорость сходимости метода Ньютона в общем случае





квадратическая. Однако, глобальная сходимость метода Ньютона, вообще говоря, не гарантируется.

Хороший способ гарантировать глобальную сходимость этого метода состоит в комбинировании его с другим методом для быстрого получения хорошей аппроксимации искомого оптимума. Тогда несколько итераций метода Ньютона, с этой точкой в качестве исходной, достаточны для получения превосходной точности.

Ограничения.

Пусть задано уравнение $f(x)=0$, где $f(x): \mathbb{X} \rightarrow \mathbb{R}$ и надо найти его решение.

Ниже приведена формулировка основной теоремы, которая позволяет дать чёткие условия применимости. **Теорема Канторовича.**

Если существуют такие константы A, B, C , что:

1. $\frac{1}{|f'(x)|} < A$ на $[a, b]$, то есть $f'(x)$ существует и не равна нулю;
2. $\left| \frac{f(x)}{f'(x)} \right| < B$ на $[a, b]$, то есть $f(x)$ ограничена;
3. $\exists f''(x)$ на $[a, b]$, и $|f''(x)| \leq C \leq \frac{1}{2AB}$;

Причём длина рассматриваемого отрезка $|a-b| < \frac{1}{AB}(1-\sqrt{1-2ABC})$. Тогда справедливы следующие утверждения:

1. на $[a, b]$ существует корень x^* уравнения $f(x)=0$: $\exists x^* \in [a, b]: f(x^*)=0$;
2. если $x_0 = \frac{a+b}{2}$, то итерационная последовательность сходится к этому корню: $\left\{ x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \right\} \rightarrow x^*$;
3. погрешность может быть оценена по формуле $|x^* - x_n| \leq \frac{B}{2^{n-1}}(2ABC)^{2^{n-1}-1}$

Из последнего из утверждений теоремы в частности следует квадратичная сходимость метода:

$$|x^* - x_n| \leq \frac{B}{2^{n-1}}(2ABC)^{2^{n-1}-1} = \frac{1}{2} \frac{B}{2^{n-2}} \left((2ABC)^{2^{n-2}} \right)^2 = \alpha |x^* - x_{n-1}|^2$$

Тогда ограничения на исходную функцию $f(x)$ будут выглядеть так:

1. функция должна быть ограничена;
2. функция должна быть гладкой, дважды дифференцируемой;





3. её первая производная $f'(x)$ равномерно отделена от нуля;
4. её вторая производная $f''(x)$ должна быть равномерно ограничена.

В случае решения задачи оптимизации под функцией понимаем ее производную.

Проблема области сходимости.

Запишем итерационный процесс:

$$x^{k+1} = x^k - \frac{f'(x^k)}{f''(x^k)}.$$

Известно, что условием сходимости этого процесса будет неравенство $|S'(x)| \leq q < 1$,

где $S(x) = x^k - \frac{f'(x^k)}{f''(x^k)}$, отсюда получаем условие сходимости:

$$\left| \frac{f'(x)f'''(x)}{(f''(x))^2} \right| \leq q < 1.$$

В силу того что мы ищем корень уравнения $f'(x) = 0$, существует такая окрестность, где $|S'(x)| \leq q < 1$, но в общем случае эта область будет мала, то есть нужно подбирать начальное приближение достаточно близко расположенным к корню.

Теорема о сходимости метода Ньютона.

Пусть x^* - простой вещественный корень уравнения $f(x) = 0$, а функция $f(x)$ - дважды дифференцируема в некоторой окрестности $U_r(x^*)$, причем первая производная нигде не обращается в нуль.

Тогда, следуя обозначениям

$$0 < m_1 = \inf_{x \in U_r(x^*)} |f'(x)|, M_2 = \sup_{x \in U_r(x^*)} |f''(x)|,$$

При выборе начального приближения x^0 из той же окрестности $U_r(x^*)$ такого, что

$$\frac{M_2|x^0 - x^*|}{2m_1} = q < 1,$$

итерационная последовательность

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}, k = 0, 1, \dots$$





будет сходиться к x^* , причем для погрешности на k -м шаге будет справедлива оценка:

$$|x^k - x^*| \leq q^{2^k - 1} |x^0 - x^*|$$

1.5 Семейство экспоненциальных распределений

Некоторые распределения из экспоненциального семейства:

- нормальное (гауссовское)
- распределение Пуассона
- биномиальное и мультиномиальное
- геометрическое
- χ^2 -распределение
- бета-распределение
- гамма-распределение
- распределение Дирихле
- распределение Лапласа с фиксированным матожиданием

1.6 Централизованные линейные модели

Обобщённые (централизованные) линейные модели (Generalized Linear Model, GLM), называемые также обобщёнными аддитивными моделями (Generalized Additive Model, GAM), можно рассматривать как дальнейшее

обобщение нелинейной регрессии $y = g^{-1}\left(\sum_{i=1}^n a_i g_i(x)\right)$, где $g(y)$ называется функцией связи (link function). Обычно в GLM дополнительно предполагается, что зависимая переменная подчиняется экспонентному распределению.

Логистическая регрессия – частный случай обобщённой линейной модели, если взять логит-функцию связи $g(y) = \ln\left(\frac{p}{1-p}\right)$, где p – зависимая переменная, имеющая смысл вероятности. Логистическая регрессия применяется для решения задач классификации и позволяет оценивать вероятности принадлежности объекта каждому из классов.



2 ГЕНЕРАЦИОННЫЕ АЛГОРИТМЫ ОБУЧЕНИЯ

2.1 Дискриминантный анализ Гаусса

Многомерное нормальное распределение

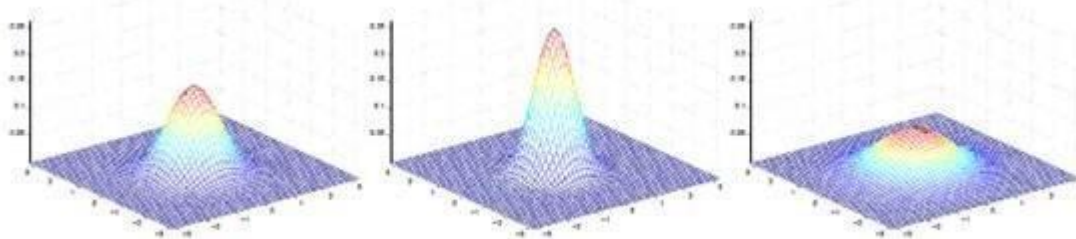
Многомерное нормальное распределение описывает распределение n -мерных случайных величин, здесь μ становится вектором, σ становится матрицей Σ . Предположим, что существует n случайных величин $X_1, X_2, \dots,$

X_n . i -тым компонентом является $E(X_i)$, и $\Sigma_{ii} = \text{Var}(X_i)$, $\Sigma_{ij} = \text{Cov}(X_i, X_j)$.

Функция плотности вероятности выглядит следующим образом:

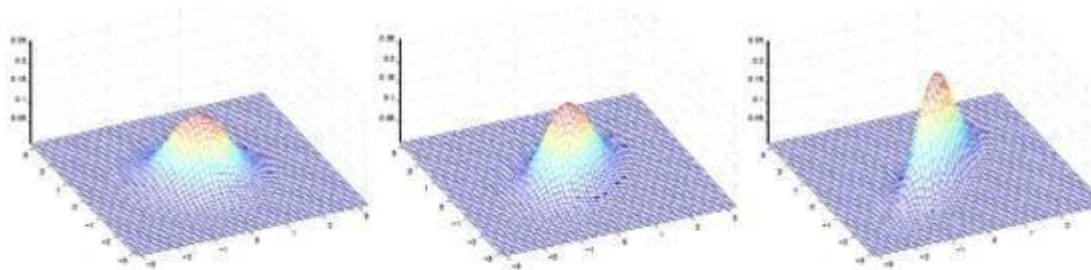
$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp \left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right).$$

Где определитель $|\Sigma|$ это ковариационная матрица, и она симметрична и положительно определена. Когда μ - двумерная, функцию плотности вероятности можно представить в виде следующего рисунка:



μ определяет центральное положение, Σ определяет ориентацию и размер проецируемого эллипса.

Как показано ниже:



The figures above show Gaussians with mean 0, and with covariance matrices respectively

$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}.$$



Анализ модели и применение.

Если входная функция x является непрерывной случайной величиной, то возможно определить **Модель гауссовского дискриминантного анализа для определения $p(x | y)$** .

Модель выглядит следующим образом:

$$\begin{aligned}y &\sim \text{Bernoulli}(\phi) \\x|y=0 &\sim \mathcal{N}(\mu_0, \Sigma) \\x|y=1 &\sim \mathcal{N}(\mu_1, \Sigma)\end{aligned}$$

Выходные результаты подчиняются распределению Бернулли, а характеристики соответствуют многомерному распределению Гаусса согласно данной модели.

Таким образом, функция плотности вероятности может быть задана:

$$\begin{aligned}p(y) &= \phi^y(1-\phi)^{1-y} \\p(x|y=0) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right) \\p(x|y=1) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)\end{aligned}$$

Оценка максимального правдоподобия выглядит следующим образом:

$$\begin{aligned}\ell(\phi, \mu_0, \mu_1, \Sigma) &= \log \prod_{i=1}^m p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) \\&= \log \prod_{i=1}^m p(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma)p(y^{(i)}; \phi).\end{aligned}$$

Обратите внимание, что здесь есть два параметра μ_0, μ_1 . Это означает, что при разных моделях результата среднее значение функции отличается, но мы предполагаем одну и ту же ковариацию. На рисунке отражено, что разные модели имеют разные центральные позиции, но одинаковую форму. Таким образом, прямая линия может использоваться для разделения и оценки.

После вывода получается формула оценки параметра:

$$\begin{aligned}\phi &= \frac{1}{m} \sum_{i=1}^m 1\{y^{(i)} = 1\} \\ \mu_0 &= \frac{\sum_{i=1}^m 1\{y^{(i)} = 0\} x^{(i)}}{\sum_{i=1}^m 1\{y^{(i)} = 0\}} \\ \mu_1 &= \frac{\sum_{i=1}^m 1\{y^{(i)} = 1\} x^{(i)}}{\sum_{i=1}^m 1\{y^{(i)} = 1\}} \\ \Sigma &= \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T.\end{aligned}$$

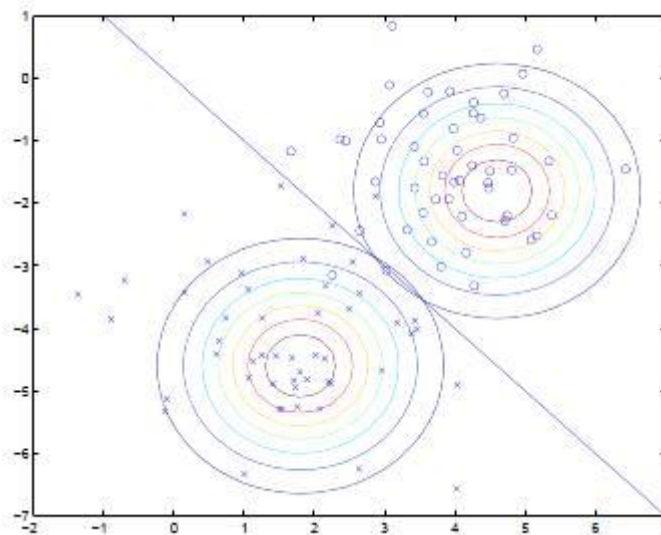
ϕ - соотношение результатов при $y = 1$ в обучающей выборке.

μ_0 - среднее значение признаков в выборке при $y = 0$.

μ_1 - среднее значение признаков в выборке при $y = 1$.

Σ - средняя дисперсия выборочных характеристик.

Как упоминалось ранее, модель представлена на графике ниже:



Значения y по обе стороны линии различны, но ковариационная матрица одинакова, поэтому форма отображения одинакова. А так как μ различны, то и положение на графике разное.

Связь между гауссовским дискриминантным анализом (GDA) и логистической регрессией

Чтобы выразить GDA в условной вероятности, следующим образом:

$$p(y = 1|x; \phi, \mu_0, \mu_1, \Sigma),$$

Где y является функцией x , и $\phi, \mu_0, \mu_1, \Sigma$ - параметры.



Дальнейший вывод

$$p(y = 1|x; \phi, \Sigma, \mu_0, \mu_1) = \frac{1}{1 + \exp(-\theta^T x)},$$

Где θ - функция от $\phi, \Sigma, \mu_0, \mu_1$.

Эта форма является формой логистической регрессии.

Другими словами, если $p(x|y)$ соответствует многомерному распределению Гаусса, то $p(y|x)$ соответствует модели логистической регрессии. В противном случае это не так. Почему не наоборот? Потому что GDA имеет более сильные предположения и ограничения.

Если определено, что данные обучения удовлетворяют многомерному распределению Гаусса, то GDA может быть лучшей моделью на обучающем множестве. Тем не менее, мы часто не знаем заранее, каким распределением удовлетворяют данные обучения и не можем делать сильные предположения. Условное допущение логистической регрессии слабее, чем GDA, поэтому логистическая регрессия используется чаще.

Например, данные обучения удовлетворяют распределению Пуассона,

$$\begin{aligned} x|y = 0 &\sim \text{Poisson}(\lambda_0) \\ x|y = 1 &\sim \text{Poisson}(\lambda_1), \end{aligned}$$

тогда $p(y|x)$ также является логистической регрессией. Если в это время используется GDA, эффект будет относительно слабым, поскольку распределение функций обучающих данных является не многомерным распределением Гаусса, а распределением Пуассона.

Это также причина, почему логистическая регрессия используется чаще.

2.2 Простейшая Байесовская теория

В GDA мы требуем, чтобы вектор признаков x был непрерывным вещественным вектором. Если x является дискретным значением, вы можете рассмотреть возможность использования наивного байесовского метода классификации.



Например, предполагается классифицировать спам и обычную почту. Классификация почты является приложением текстовой классификации.

Предполагая самый простой способ описания функции, сначала следует найти словарь и перечислить все слова в нем. Затем каждое сообщение представляется в виде вектора, и каждое измерение в векторе представляет собой значение слова в словаре 0/1, 1 указывает, что слово появляется в письме, а 0 указывает, что оно не появляется.

Например, «a» и «покупка» появляются в электронном письме, но «aardvark», «aardwolf» и «zygmurgy» не отображаются, поэтому их можно выразить как

$$x = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} a \\ \text{aardvark} \\ \text{aardwolf} \\ \vdots \\ \text{buy} \\ \vdots \\ \text{zygmurgy} \end{matrix}$$

Предположим, что в словаре 5000 слов, тогда x 5000-мерный. В это время, требуется построить модель полиномиального распределения (расширение биномиального распределения).

Полиномиальное распределение

Если случайный эксперимент имеет k возможных результатов $A_1, A_2, \dots, A_k$, и их вероятностные распределения равны $p_1, p_2, \dots, p_k$, то в общем результате из N выборок A_1 появляется n_1 раз, A_2 появляется n_2 , ..., A_k появляется n_k раз. Вероятность появления P этого события, которое появляется n_k раз, имеет следующую формулу: (X_i представляет n_i раз)

$$P(X_1 = x_1, \dots, X_k = x_k) = \begin{cases} \frac{n!}{x_1! \dots x_k!} p_1^{x_1} \dots p_k^{x_k} & \text{when } \sum_{i=1}^k x_i = n \\ 0 & \text{otherwise.} \end{cases}$$

В соответствии с вышеприведенным вопросом, рассмотрим каждое письмо как случайный тест, а затем возможность результатов как 2^{5000}



вариантов. Значит мы имеем 2^{5000} параметров, что слишком много и не может быть использовано для моделирования.

Другими словами, нам требуется $p(y|x)$, и мы можем найти $p(x|y)$ и $p(y)$ в соответствии с определением порождающей модели. Предположим, что особенности в x условно независимы. Это называется наивной байесовской гипотезой. Если сообщение является спамом ($y = 1$), и слово «покупка» появляется в этом сообщении, это не имеет никакого отношения к тому, появляется ли сообщение «цена», тогда условия между «купить» и «цена» независимы.

Формально выражается как (если задано Z , условия X и Y независимы):

$$P(X|Z) = P(X|Y, Z)$$

Это также может быть выражено как:

$$P(X, Y|Z) = P(X|Z)P(Y|Z)$$

Возвращаясь к вопросу

$$\begin{aligned} p(x_1, \dots, x_{50000}|y) \\ &= p(x_1|y)p(x_2|y, x_1)p(x_3|y, x_1, x_2) \cdots p(x_{50000}|y, x_1, \dots, x_{49999}) \\ &= p(x_1|y)p(x_2|y)p(x_3|y) \cdots p(x_{50000}|y) \\ &= \prod_{i=1}^n p(x_i|y) \end{aligned}$$

Это несколько похоже на модель синтаксиса n -граммы в НЛП, которая эквивалентна униграмме.

Здесь мы находим, что наивная байесовская гипотеза является очень ограничительной гипотезой, «покупка» обычно связана с «ценой», и мы предполагаем, что это условная независимость. (Обратите внимание, что условная независимость и независимость разные понятия)

Установим формальное представление модели:

$$\phi_{i|y=1} = p(x_i = 1|y = 1)$$

$$\phi_{i|y=0} = p(x_i = 1|y = 0)$$

$$\phi_y = p(y = 1)$$





Тогда мы хотим, чтобы модель имела наибольший вероятностный продукт по обучающим данным, то есть максимальная оценка правдоподобия выглядит следующим образом:

$$\mathcal{L}(\phi_y, \phi_{i|y=0}, \phi_{i|y=1}) = \prod_{i=1}^m p(x^{(i)}, y^{(i)}).$$

Обратите внимание, что здесь представлен самый большой продукт совместного распределения вероятностей, указывающий на то, что Наивный Байес является порождающей моделью.

Решение:

$$\begin{aligned}\phi_{j|y=1} &= \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^m 1\{y^{(i)} = 1\}} \\ \phi_{j|y=0} &= \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^m 1\{y^{(i)} = 0\}} \\ \phi_y &= \frac{\sum_{i=1}^m 1\{y^{(i)} = 1\}}{m}\end{aligned}$$

Последнее выражение представляет собой отношение количества выборок с $y = 1$ к общему количеству выборок, а первые два - это отношение признака $X_j = 1$ в выборках с $y = 1$ или 0.

Однако то, что мы хотим, это

$$\begin{aligned}p(y = 1|x) &= \frac{p(x|y = 1)p(y = 1)}{p(x)} \\ &= \frac{(\prod_{i=1}^n p(x_i|y = 1)) p(y = 1)}{(\prod_{i=1}^n p(x_i|y = 1)) p(y = 1) + (\prod_{i=1}^n p(x_i|y = 0)) p(y = 0)},\end{aligned}$$

Фактически, числитель может быть получен, и пара знаменателей $y = 1$ и $y = 0$ одинаковы.

Конечно, метод Наивного Байеса можно распространить на случай, когда и x , и y имеют несколько дискретных значений. Для случая, когда свойство является непрерывным значением, мы также можем использовать сегментированный метод для преобразования непрерывного значения в дискретное значение. Конкретное преобразование может быть оптимизировано, мы можем использовать метод измерения прироста информации.





Например, размер дома можно разделить на дискретные значения следующим образом:

Living area (sq. feet)	< 400	400-800	800-1200	1200-1600	>1600
x_i	1	2	3	4	5

2.3 Сглаживание Лапласа

Наивный байесовский метод имеет фатальный недостаток, который слишком чувствителен к разреженности данных.

Например, вышеупомянутая классификация почты, теперь есть новая почта, заголовок почты - «вызов NIPS для бумаг». Для классификации мы используем большой онлайн-словарь (количество слов изменилось с 5000 до 35000), предполагая, что позиция слова NIPS в словаре составляет 35000. Однако слово NIPS не появилось в данных обучения, и NIPS впервые появился в этом письме. Затем мы рассчитываем вероятность следующим образом:

$$\phi_{35000|y=1} = \frac{\sum_{i=1}^m 1\{x_{35000}^{(i)} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^m 1\{y^{(i)} = 1\}} = 0$$

$$\phi_{35000|y=0} = \frac{\sum_{i=1}^m 1\{x_{35000}^{(i)} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^m 1\{y^{(i)} = 0\}} = 0$$

Поскольку NIPS никогда не появлялся в спаме или обычной почте, результат может быть только 0.

Очевидно, что конечная условная вероятность также равна нулю.

$$p(y=1|x) = \frac{\prod_{i=1}^n p(x_i|y=1)p(y=1)}{\prod_{i=1}^n p(x_i|y=1)p(y=1) + \prod_{i=1}^n p(x_i|y=0)p(y=0)}$$

$$= \frac{0}{0}.$$

Причина в том, что наши вероятностные условия признака независимы, и мы используем умножение, чтобы получить результат.

Чтобы решить эту проблему, мы намерены присвоить «маленькому» значению вместо 0 значение функции, которое не отображается.

Конкретный метод сглаживания заключается в следующем:

Предполагая, что дискретная случайная величина z имеет значения $\{1, 2, \dots, k\}$, мы используем $\Phi_i = p(z=i)$ для представления вероятности каждого





значения. Предположим, что есть m обучающих образцов, значение наблюдения $z \{z^{(1)}, \dots, z^{(m)}\}$. Каждое значение наблюдения соответствует одному из значений k . Тогда по оригинальному методу оценки можно получить

$$\phi_j = \frac{\sum_{i=1}^m 1\{z^{(i)} = j\}}{m}.$$

Говоря прямо, это пропорция $z = j$.

Метод сглаживания Лапласа заранее добавляет 1 к числу вхождений каждого значения k . Предполагается, что все они встречаются один раз.

Тогда модифицированное выражение:

$$\phi_j = \frac{\sum_{i=1}^m 1\{z^{(i)} = j\} + 1}{m + k}.$$

Каждый числитель в $z = j$ добавляет 1, а знаменатель добавляет k .

Это похоже на метод сглаживания «один плюс» в NLPбно описаны.

Возвращаясь к вопросу о классификации почты, модифицированная формула:

$$\begin{aligned}\phi_{j|y=1} &= \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\} + 1}{\sum_{i=1}^m 1\{y^{(i)} = 1\} + 2} \\ \phi_{j|y=0} &= \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\} + 1}{\sum_{i=1}^m 1\{y^{(i)} = 0\} + 2}\end{aligned}$$

2.4 Метод опорных векторов

Опорные вектора и расстояния (зазоры)

Пусть $D = \{(x_i, y_i)\}_{i=1}^n$ будет набором классифицируемых данных с n точками и размерностью d . Далее, предположим, что есть только две метки класса, то есть $y_i \in \{+1, -1\}$, обозначающих положительный и отрицательный классы.

Гиперплоскости

Гиперплоскость в d измерениях задается как множество всех точек $x \in \mathbb{R}^d$, которые удовлетворяют уравнению $h(x) = 0$, где $h(x)$ - функция гиперплоскости, определяемая следующим образом:





$$\begin{aligned} h(\mathbf{x}) &= \mathbf{w}^T \mathbf{x} + b \\ &= w_1 x_1 + w_2 x_2 + \dots + w_d x_d + b \end{aligned} \quad (3)$$

Здесь $\mathbf{w}$ - d -мерный вектор весов, а b - скаляр, называемый смещением. Для точек, лежащих на гиперплоскости, имеем

$$h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0 \quad (4)$$

Таким образом, гиперплоскость определяется как множество всех таких точек, что $\mathbf{w}^T \mathbf{x} = -b$. Чтобы увидеть роль, которую играет b , предполагая,

что $w_1 \neq 0$, и устанавливая $x_i = 0$ для всех $i > 1$, мы можем получить смещение, где гиперплоскость пересекает первую ось, так как по формуле. (4) имеем

$$w_1 x_1 = -b \text{ или } x_1 = \frac{-b}{w_1}$$

Другими словами, точка $\left(\frac{-b}{w_1}, 0, \dots, 0\right)$ лежит на гиперплоскости. Аналогичным образом мы можем получить смещение в месте пересечения гиперплоскости с каждой из осей, которое задается как $\frac{-b}{w_i}$ (при $w_i \neq 0$).

Разделяющая гиперплоскость

Гиперплоскость разбивает исходное d -мерное пространство на два полупространства. Набор данных называется линейно разделимым, если каждое полупространство имеет точки только из одного класса. Если входной набор данных является линейно разделимым, то мы можем найти разделяющую гиперплоскость $h(\mathbf{x}) = 0$, такую, что для всех точек с меткой $y_i = -1$ мы имеем $h(\mathbf{x}_i) < 0$, а для всех точек с меткой $y_i = +1$, мы имеем $h(\mathbf{x}_i) > 0$. Фактически, функция гиперплоскости $h(\mathbf{x})$ служит линейным классификатором или линейным дискриминантом, который предсказывает класс y для любой заданной точки $\mathbf{x}$ в соответствии с решающим правилом:

$$y = \begin{cases} +1 & \text{if } h(\mathbf{x}) > 0 \\ -1 & \text{if } h(\mathbf{x}) < 0 \end{cases} \quad (5)$$

Пусть $\mathbf{a}_1$ и $\mathbf{a}_2$ - две произвольные точки, лежащие на гиперплоскости. Из уравнения. (4) имеем

$$\begin{aligned} h(\mathbf{a}_1) &= \mathbf{w}^T \mathbf{a}_1 + b = 0 \\ h(\mathbf{a}_2) &= \mathbf{w}^T \mathbf{a}_2 + b = 0 \end{aligned}$$

Вычитая одно из другого, получаем





$$\mathbf{w}^T(\mathbf{a}_1 - \mathbf{a}_2) = 0$$

Это означает, что вектор весов $\mathbf{w}$ ортогонален гиперплоскости, потому что он ортогонален любому произвольному вектору $(\mathbf{a}_1 - \mathbf{a}_2)$ на гиперплоскости. Другими словами, вектор весов $\mathbf{w}$ определяет направление, нормальное к гиперплоскости, которое фиксирует ориентацию гиперплоскости, тогда как смещение b фиксирует смещение гиперплоскости в d -мерном пространстве. Поскольку и $\mathbf{w}$, и $-\mathbf{w}$ нормальны к гиперплоскости, мы устраняем эту двусмысленность, требуя, чтобы $h(\mathbf{x}_i) > 0$, когда $y_i = 1$, и $h(\mathbf{x}_i) < 0$, когда $y_i = -1$.

Расстояние от точки до гиперплоскости

Рассмотрим точку $\mathbf{x} \in \mathbb{R}^d$ такую, что $\mathbf{x}$ не лежит на гиперплоскости. Пусть $\mathbf{x}_p$ - ортогональная проекция $\mathbf{x}$ на гиперплоскость, и пусть $\mathbf{r} = \mathbf{x} - \mathbf{x}_p$, тогда, как показано на рисунке 13, мы можем записать $\mathbf{x}$ как

$$\mathbf{x} = \mathbf{x}_p + \mathbf{r}$$

$$\mathbf{x} = \mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|} \quad (6)$$

где r - направленное расстояние точки $\mathbf{x}$ от $\mathbf{x}_p$, то есть r дает смещение $\mathbf{x}$ от $\mathbf{x}_p$ в терминах единичного вектора веса $\frac{\mathbf{w}}{\|\mathbf{w}\|}$. Смещение r положительно, если r находится в том же направлении что и $\mathbf{w}$, и r отрицательно, если r находится в направлении, противоположном $\mathbf{w}$.



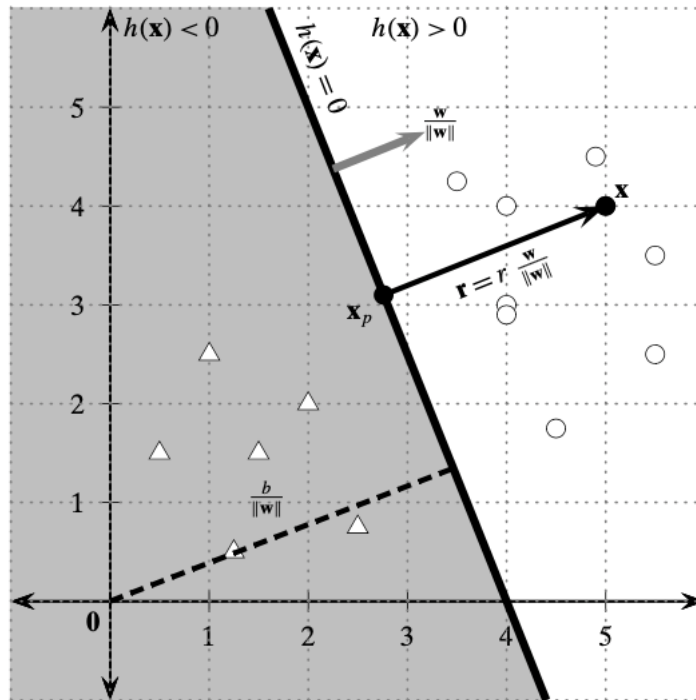


Рисунок 13 - Геометрия разделяющей гиперплоскости в 2D

Подставляя уравнение (6) в функцию гиперплоскости [Ур. (3)], получаем

$$\begin{aligned}
 h(\mathbf{x}) &= h\left(\mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|}\right) \\
 &= \mathbf{w}^T \left(\mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|}\right) + b \\
 &= \underbrace{\mathbf{w}^T \mathbf{x}_p + b}_{h(\mathbf{x}_p)} + r \frac{\mathbf{w}^T \mathbf{w}}{\|\mathbf{w}\|} \\
 &= \underbrace{h(\mathbf{x}_p)}_0 + r \|\mathbf{w}\| \\
 &= r \|\mathbf{w}\|
 \end{aligned}$$

Последний шаг следует из того, что $h(\mathbf{x}_p) = 0$, поскольку $\mathbf{x}_p$ лежит на гиперплоскости. Используя результат выше, мы получаем выражение для направленного расстояния от точки до гиперплоскости:

$$r = \frac{h(\mathbf{x})}{\|\mathbf{w}\|}$$

Чтобы получить расстояние, которое должно быть неотрицательным, мы можем умножить r на метку класса y точки, потому что, когда $h(\mathbf{x}) < 0$,



класс равен -1 , а когда $h(\mathbf{x}) > 0$, класс равен $+1$. Таким образом, расстояние от точки $\mathbf{x}$ до гиперплоскости $h(\mathbf{x}) = 0$ задается как

$$\delta = yr = \frac{yh(\mathbf{x})}{\|\mathbf{w}\|} \quad (7)$$

В частности, для начала координат $\mathbf{x} = 0$ направленное расстояние равно

$$r = \frac{h(0)}{\|\mathbf{w}\|} = \frac{\mathbf{w}^T \mathbf{0} + b}{\|\mathbf{w}\|} = \frac{b}{\|\mathbf{w}\|}$$

как показано на рисунке 15.1.

Расстояние (зазор) и опорные векторы гиперплоскости

Для обучающего набора размеченных точек, $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$ с $y_i \in \{+1, -1\}$, и для разделяющей гиперплоскости $h(\mathbf{x}) = 0$, для каждой точки $\mathbf{x}_i$ мы можем найти её расстояние до гиперплоскости по формуле (7):

$$\delta_i = \frac{y_i h(\mathbf{x}_i)}{\|\mathbf{w}\|} = \frac{y_i (\mathbf{w}^T \mathbf{x}_i + b)}{\|\mathbf{w}\|}$$

По всем n точкам мы определяем границу линейного классификатора как минимальное расстояние точки от разделяющей гиперплоскости, заданное как

$$\delta^* = \min_{\mathbf{x}_i} \left\{ \frac{y_i (\mathbf{w}^T \mathbf{x}_i + b)}{\|\mathbf{w}\|} \right\} \quad (8)$$

Заметим, что $\delta^* \neq 0$, так как $h(\mathbf{x})$ - разделяющая гиперплоскость, и должно выполняться уравнение (5).

Все точки (или векторы), которые достигают этого минимального расстояния, называются опорными векторами для гиперплоскости. Другими словами, опорный вектор $\mathbf{x}^*$ - точка, которая лежит точно на границе классификатора и, таким образом, удовлетворяет условию

$$\delta^* = \frac{y^* (\mathbf{w}^T \mathbf{x}^* + b)}{\|\mathbf{w}\|}$$

где y^* - метка класса для $\mathbf{x}^*$. Числитель $y^* (\mathbf{w}^T \mathbf{x}^* + b)$ дает абсолютное расстояние опорного вектора к гиперплоскости, а знаменатель $\|\mathbf{w}\|$ делает его относительным расстоянием в терминах $\mathbf{w}$.

Каноническая гиперплоскость





Рассмотрим уравнение гиперплоскости [Ур. (4)]. Умножение с обеих сторон на некоторый скаляр s дает эквивалентную гиперплоскость:

$$sh(\mathbf{x}) = s\mathbf{w}^T \mathbf{x} + sb = (s\mathbf{w})^T \mathbf{x} + (sb) = 0$$

Чтобы получить единственную или каноническую гиперплоскость, мы выбираем скаляр s так, чтобы абсолютное расстояние вектора поддержки от гиперплоскости равнялось 1. То есть,

$$sy^*(\mathbf{w}^T \mathbf{x}^* + b) = 1$$

что подразумевает

$$s = \frac{1}{y^*(\mathbf{w}^T \mathbf{x}^* + b)} = \frac{1}{y^*h(\mathbf{x}^*)} \quad (9)$$

В дальнейшем мы будем считать любую разделяющую гиперплоскость канонической. То есть, она уже соответствующим образом масштабирована так, что $y^*h(\mathbf{x}^*) = 1$ для вектора поддержки $\mathbf{x}^*$, а расстояние (зазор) задается как

$$\delta^* = \frac{y^*h(\mathbf{x}^*)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

Для канонической гиперплоскости, для каждого вектора поддержки $\mathbf{x}_i^*$ (с меткой y_i^*), мы имеем $y_i^*h(\mathbf{x}_i^*) = 1$, и для любой точки, которая не является опорным вектором мы имеем $y_i h(\mathbf{x}_i) > 1$, потому что, по определению, он должен быть дальше от гиперплоскости, чем опорный вектор. Таким образом, по всем n точкам в наборе данных $\mathbf{D}$ мы получаем следующий набор неравенств:

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \text{ для всех точек } \mathbf{x}_i \in \mathbf{D} \quad (10)$$

Случай линейного разбиения

Имея набор данных $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$, где $\mathbf{x} \in \mathbb{R}^d, y \in \{+1, -1\}$, предположим, что точки можно линейно разбить, т.е. существует гиперплоскость, разбиение по которой безошибочно классифицирует каждую точку. Другими словами, все точки $y_i = +1$ лежат на одной стороне какой-либо гиперплоскости ($h(\mathbf{x}) > 0$), а все $y_i = -1$ лежат на другой стороне той же гиперплоскости ($h(\mathbf{x}) < 0$). Очевидно, что в этом случае количество подходящих гиперплоскостей бесконечно. Встает вопрос выбора гиперплоскости.

Гиперплоскость с максимальным зазором





Основная идея SVM - выбор канонической гиперплоскости, заданной вектором весов w и ошибкой b , которая дает наибольший зазор среди всех возможных разделяющих гиперплоскостей. Если δ_h^* - зазор гиперплоскости $h(x) = 0$, то задача найти оптимальную гиперплоскость h^* :

$$h^* = \arg \max_h \{\delta_h^*\} = \arg \max_{\{w,b\}} \left\{ \frac{1}{\|w\|} \right\}.$$

Задача SVM - поиск гиперплоскости, максимизирующей зазор $\frac{1}{\|w\|}$ при n ограничениях из (10), а именно $y_i (w^T x_i + b) \geq 1$ для всех точек $x \in D$. Заметьте, что максимизацию $\frac{1}{\|w\|}$ можно заменить минимизацией $\|w\|$. Эквивалентная формулировка задачи условной минимизации:

Целевая функция: $\min_{\{w,b\}} \left\{ \frac{\|w\|^2}{2} \right\}$

Линейные ограничения: $y_i (w^T x_i + b) \geq 1, \forall x \in D$

Поставленную задачу *прямой* выпуклой минимизации с n линейными ограничениями можно решить стандартными алгоритмами оптимизации. Однако, чаще решается *двойственная* задача, которую можно получить с помощью множителей Лагранжа. Основная идея - ввести множитель Лагранжа α_i для каждого ограничения, которое удовлетворяет условиям Каруша-Куна-Такера для оптимального решения:

$$\alpha_i (y_i (w^T x_i + b) - 1) = 0 \\ \text{и } \alpha \geq 0$$

С учетом всех n ограничений, новая целевая функция - *Лагранжиан* - становится

$$\min_L L = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i (y_i (w^T x_i + b) - 1) \quad (11)$$

L минимизируется с учетом w и b , и взяв их нулевые значения, получаем

$$\frac{\partial}{\partial w} L = w - \sum_{i=1}^n \alpha_i y_i x_i = 0 \text{ или } w = \sum_{i=1}^n \alpha_i y_i x_i \quad (12)$$

$$\frac{\partial}{\partial b} L = \sum_{i=1}^n \alpha_i y_i = 0 \quad (13)$$



Вышеприведенные уравнения помогают получить важное предположение относительно оптимального веса вектора w . В частности, (12) предполагает, что w можно выразить как линейную комбинацию точек данных x_i с коэффициентами в виде множителей Лагранжа, α, y_i . Затем, (13) предполагает, что сумма множителей Лагранжа, α_i, y_i должна быть равна 0.

Подставляя эти уравнения в (11), можно получить двойственную целевую функцию Лангранжа, заданную исключительно в терминах множителей Лагранжа.

$$\begin{aligned} L_{dual} &= \frac{1}{2} \mathbf{w}^T \mathbf{w} - \mathbf{w}^T \underbrace{\left(\sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \right)}_{\mathbf{w}} - b \underbrace{\sum_{i=1}^n \alpha_i y_i}_0 + \sum_{i=1}^n \alpha_i \\ &= -\frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_{i=1}^n \alpha_i \\ &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \end{aligned}$$

Таким образом, двойственная задача:

Целевая функция: $\max_{\alpha} L_{dual} = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ (14)

Линейные ограничения: $\alpha_i \geq 0, \forall i \in \mathbf{D}$, and $\sum_{i=1}^n \alpha_i y_i = 0$

где $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)^T$ – вектор из множителей Лагранжа. L_{dual} – задача выпуклого квадратичного программирования (обратите внимание на множители $\alpha_i \alpha_j$), которая может быть решена стандартными методами оптимизации.

Вектор весов и ошибка

Получив значения α_i для $i = 1, \dots, n$, мы можем провести решение для вектора w и ошибки b . С учетом условия Каруша-Куна-Такера

$$\alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1) = 0,$$

что дает две возможности:

(1) $\alpha_i = 0$, или

(2) $y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 = 0$, что предполагает $y_i (\mathbf{w}^T \mathbf{x}_i + b) = 1$





Это важный результат, поскольку если $\alpha_i > 0$, то $y_i(\mathbf{w}^T \mathbf{x}_i + b) = 1$, и точка $\mathbf{x}_i$ должна быть опорным вектором. С другой стороны, если $y_i(\mathbf{w}^T \mathbf{x}_i + b) > 1$, т.е. точка - не опорный вектор, то $\alpha_i = 0$.

Как только мы знаем α_i для всех точек, мы можем вычислить вектор весов $\mathbf{w}$ с помощью уравнения (15.10), но суммирую только опорные векторы:

$$\mathbf{w} = \sum_{i, \alpha_i > 0} \alpha_i y_i \mathbf{x}_i \quad (15)$$

Другими словами, $\mathbf{w}$ получен как линейная комбинация опорных векторов, где $\alpha_i y_i$ представляют веса. Остальные точки (с $\alpha = 0$) - не опорные вектора и не важны при определении $\mathbf{w}$.

Чтобы вычислить ошибку b , сначала вычисляется одно решение b_i для одного опорного вектора:

$$\begin{aligned} \alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1) &= 0 \\ y_i (\mathbf{w}^T \mathbf{x}_i + b) &= 1 \\ b_i &= \frac{1}{y_i} - \mathbf{w}^T \mathbf{x}_i = y_i - \mathbf{w}^T \mathbf{x}_i \end{aligned} \quad (16)$$

b берется как средняя ошибка по опорным векторам:

$$b = \text{avg}_{\alpha_i > 0} \{b_i\} \quad (17)$$

Классификатор SVM

Имея оптимальную гиперплоскость $h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$, для каждой новой точки $\mathbf{z}$ можно предсказать её класс как

$$\hat{y} = \text{sign}(h(\mathbf{z})) = \text{sign}(\mathbf{w}^T \mathbf{z} + b) \quad (18)$$

где sign - функция, возвращающая +1 для положительного аргумента и -1 для отрицательного.

SVM с мягким зазором: случай линейной неразделимости

Пока мы предполагали, что набор данных можно безошибочно линейно разделить. Теперь рассмотрим случай, когда классы до некоторой степени пересекаются, так что безошибочное разделение невозможно, как показано на рис. 14



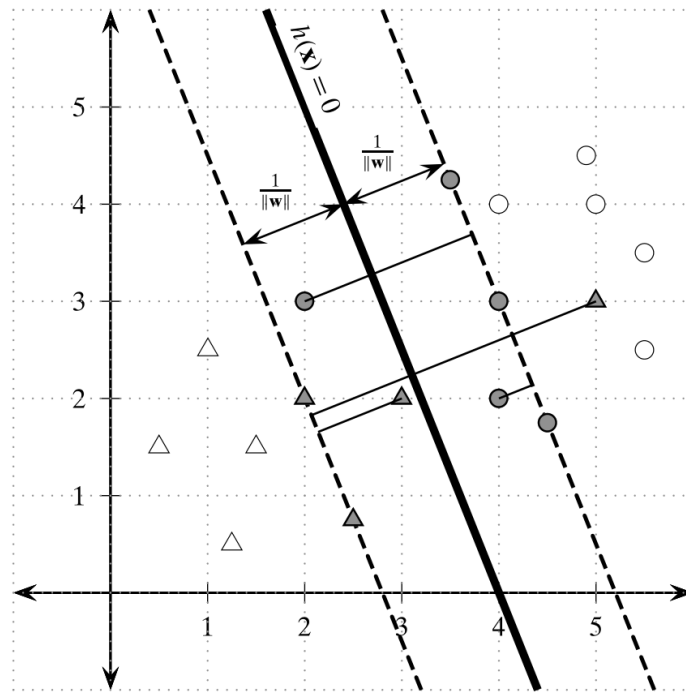


Рисунок 14. Гиперплоскость с мягким зазором: серые точки - опорные векторы. Показан зазор $\frac{1}{\|w\|}$

Вспомним, что, когда точки линейно разделимы, можно найти разделяющую гиперплоскость, такую, что для всех точек $y_i(w^T x_i + b) \geq 1$. Обработать неразделимые точки можно, введя *слабые переменные* ξ_i в уравнение (10) следующим образом:

$$y_i(w^T x_i + b) \geq 1 - \xi_i,$$

где $\xi_i \geq 0$ – слабая переменная для точки x_i , показывающая, насколько точка нарушает условие разбиения, т.е. точка не обязана быть на расстоянии как минимум $\frac{1}{\|w\|}$ от гиперплоскости. Слабые переменные обозначают 3 типа точек. Если ξ_0 , то точка на расстоянии хотя бы $\frac{1}{\|w\|}$ от гиперплоскости. Если $0 < \xi_i < 1$, то точка внутри зазора и всё ещё корректно классифицируется, т.е. на правильной стороне гиперплоскости. Если $\xi_i \geq 1$, точка классифицируется неправильно и находится на неправильной стороне гиперплоскости.

В случае линейной неразделимости (иначе - случай с *мягким зазором*), цель SVM - найти гиперплоскость с максимальным зазором и минимальными слабыми переменными. Тогда новая цель:



Целевая функция: $\min_{\mathbf{w}, b, \xi_i} \left\{ \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^n (\xi_i)^k \right\}$

Линейные ограничения:

$$\begin{aligned} y_i(\mathbf{w}^T \mathbf{x}_i + b) &\geq 1 - \xi_i, \forall \mathbf{x}_i \in \mathbf{D} \\ \xi_i &\geq 0 \forall \mathbf{x}_i \in \mathbf{D} \end{aligned} \quad (19)$$

Где C, k - ограничения, включающие в себя стоимость ошибочной классификации. Член $\sum_{i=1}^n (\xi_i)^k$ показывает *потерю*, т.е. отклонение от случая линейной разделимости. Скаляр C , выбирающийся эмпирически - *константа регуляризации*, контролирующая компромисс между максимизацией зазора (минимизацией $\frac{\|\mathbf{w}\|^2}{2}$) или минимизацией потери (минимизацией суммы слабых членов $\sum_{i=1}^n (\xi_i)^k$). Так, если $C \rightarrow 0$, то компонент потери пропадает, и цель становится исключительно максимизацией зазора. Если же $C \rightarrow \infty$, то зазор перестает оказывать существенный эффект, и целевая функция минимизирует потерю. Константа k показывает вид потери. Обычно, k ставится в 1 или 2. Когда $k = 1$, это называется *петлевая функция потери* (hinge loss), и цель - минимизация суммы слабых переменных, когда $k = 2$, это *квадратичная функция потерь*, и цель - минимизация суммы квадратов слабых переменных

Петлевая функция потерь

Предположив $k = 1$, можно вычислить Лагранжиан для задачи (19), введя множители Лагранжа α_i, β_i , которые удовлетворяют следующим условиям Каруша-Куна-Такера для оптимального решения:

$$\begin{aligned} \alpha_i(y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i) &= 0 \text{ при } \alpha_i \geq 0 \\ \beta_i(\xi_i - 0) &= 0 \text{ при } \beta_i \geq 0 \end{aligned} \quad (20)$$

И Лагранжиан задается как:

$$L = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i(y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i) - \sum_{i=1}^n \beta_i \xi_i \quad (21)$$

Взятием частных производных по $\mathbf{w}$, b, ξ , и приравниванием их к 0, получаем:





$$\begin{aligned}
\frac{\partial}{\partial \mathbf{w}} L &= \mathbf{w} - \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i = \mathbf{0} \text{ or } \mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \\
\frac{\partial}{\partial b} L &= \sum_{i=1}^n \alpha_i y_i = 0 \\
\frac{\partial}{\partial \xi_i} L &= C - \alpha_i - \beta_i = 0 \text{ or } \beta_i = C - \alpha_i
\end{aligned} \tag{22}$$

Постановка этих значений в (15.19):

$$\begin{aligned}
L_{\text{dual}} &= \frac{1}{2} \mathbf{w}^T \mathbf{w} - \underbrace{\mathbf{w}^T \left(\sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \right)}_{\mathbf{w}} - \underbrace{b \sum_{i=1}^n \alpha_i y_i}_0 + \sum_{i=1}^n \alpha_i + \sum_{i=1}^n \underbrace{(C - \alpha_i - \beta_i)}_0 \xi_i \\
&= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j
\end{aligned}$$

И двойственная задача:

Целевая функция:

$$\max_{\alpha} L_{\text{dual}} = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \tag{23}$$

Линейные ограничения:

$$0 \leq \alpha_i \leq C, \forall i \in \mathbf{D} \text{ и } \sum_{i=1}^n \alpha_i y_i = 0$$

Заметьте, что целевая функция - такая же, как и для случая линейной разделимости (14). Однако, ограничения на α_i теперь другие, т.к. мы требуем $\alpha_i + \beta_i = C$, где $\alpha_i \geq 0$ и $\beta_i \geq 0$, что предполагает $0 \leq \alpha_i \leq C$.

Вектор весов и потеря

После решения для α_i , мы оказываемся в той же ситуации, что и раньше - точки $\alpha_i = 0$ - не опорные вектора, и $\alpha_i > 0$ только для опорных векторов, составляющих все $\mathbf{x}_i$, для которых

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) = 1 - \xi \tag{24}$$

Заметим, что опорные вектора теперь включают как все точки вне зазора, для которых $\xi_i = 0$, так и точки $\xi_i > 0$.

Как и раньше, вектор весов можно получить из опорных векторов:





$$\mathbf{w} = \sum_{i, \alpha_i > 0} \alpha_i y_i \mathbf{x}_i \quad (25)$$

Можно получить решение для β_i с помощью уравнения (15.20):

$$\beta_i = C - \alpha_i$$

Заменяя β_i в условиях Каруша-Куна-Такера (20), получаем:

$$(C - \alpha_i) \xi_i = 0 \quad (26)$$

Т.е. для опорных векторов с $\alpha_i > 0$ возможны 2 случая:

$\xi_i > 0$, что предполагает $C - \alpha_i = 0$, т.е. $\alpha_i = C$, или

$C - \alpha_i > 0$, т.е. $\alpha_i < C$. В этом случае $\xi_i = 0$, и эти опорные вектора находятся вне зазора.

С использованием опорных векторов вне зазора, для которых $0 < \alpha_i < C$ и $\xi_i = 0$, можно решить для b_i :

$$\begin{aligned} \alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + b_i) - 1) &= 0 \\ y_i (\mathbf{w}^T \mathbf{x}_i + b_i) &= 1 \\ b_i &= \frac{1}{y_i} - \mathbf{w}^T \mathbf{x}_i = y_i - \mathbf{w}^T \mathbf{x}_i \end{aligned} \quad (27)$$

Чтобы получить итоговую ошибку b , нужно среднее по всем b_i . Из (25) и (27), и вектор весов $\mathbf{w}$, и ошибка b могут быть вычислены без явного вычисления членов ξ для каждой точки.

Когда оптимальная гиперплоскость определена, модель SVM предсказывает класс точки $\mathbf{z}$ следующим образом:

$$\hat{y} = \text{sign}(h(\mathbf{z})) = \text{sign}(\mathbf{w}^T \mathbf{z} + b)$$

Квадратичная потеря

Для квадратичной потери имеем $k = 2$ в целевой функции (19). В этом случае, ограничение на положительность $\xi_i \geq 0$ можно отбросить, т.к. (1) сумма слабых членов $\sum_{i=1}^n \xi_i^2$ всегда положительна, и (2) возможные отрицательные значения слабых членов исключаются во время оптимизации, потому что выбор $\xi_i = 0$ приводит к уменьшению значения основной цели, и всё ещё удовлетворяет ограничению $y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i$ когда $\xi_i < 0$. Другими словами, процесс оптимизации заменит отрицательные значения слабых членов нулями. Таким образом, задача SVM для квадратичной потери задана следующим образом:



Целевая функция: $\min_{\mathbf{w}, b, \xi_i} \left\{ \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^n (\xi_i)^2 \right\}$

Линейные ограничения: $y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \forall \mathbf{x}_i \in \mathbf{D}$

И Лагранжиан:

$$L = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i^2 - \sum_{i=1}^n \alpha_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i) \quad (28)$$

Дифференцированием по $\mathbf{w}$, b и ξ_i и приравниванием соответствующих результатов к 0, получаем:

$$\begin{aligned} \mathbf{w} &= \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \\ \sum_{i=1}^n \alpha_i y_i &= 0 \\ \xi_i &= \frac{1}{2C} \alpha_i \end{aligned}$$

Подстановка результата обратно в (28) дает двойственную задачу:

$$\begin{aligned} L_{dual} &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j - \frac{1}{4C} \sum_{i=1}^n \alpha_i^2 \\ &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i^T \mathbf{x}_j + \frac{1}{2C} \delta_{ij} \right) \end{aligned}$$

где δ – дельта-функция Кронекера, заданная как $\delta_{ij} = 1$, где $i = j$, и $\delta_{ij} = 0$ во всех остальных случаях. Таким образом, двойственная задача:

$$\begin{aligned} \max_{\alpha} L_{dual} &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i^T \mathbf{x}_j + \frac{1}{2C} \delta_{ij} \right) \\ &\text{с ограничениями } \alpha_i \geq 0, \forall i \in \mathbf{D}, \text{ и } \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned} \quad (29)$$

Решая для α_i , можно получить вектор весов и ошибку следующим образом:

$$\begin{aligned} \mathbf{w} &= \sum_{i, \alpha_i > 0} \alpha_i y_i \mathbf{x}_i \\ b &= \text{avg}_{i, \alpha_i > 0} \{y_i - \mathbf{w}^T \mathbf{x}_i\} \end{aligned}$$





2.5 Дискриминантный анализ ядра

Дискриминантный анализ ядра, как и линейный дискриминантный анализ, пытается найти направление, которое максимизирует разделение между классами. Однако он делает это в пространстве функций с помощью функций ядра.

Задан набор данных, $D = \{(x_i, y_i)\}_{i=1}^n$ где x_i точка во входном пространстве, а $y_i \in \{c_1, c_2\}$ является метка класса, пусть $D_i = \{x_j \mid y_j = c_i\}$ обозначим подмножество данных ограниченному классу c_i , и пусть $n_i = |D_i|$. Далее, пусть $\phi(x_i)$ обозначает соответствующую точку в пространстве признаков и пусть K - функция ядра.

Цель ядра линейного дискриминантного анализа в том, чтобы найти вектор направления w в пространстве признаков, который максимизирует

$$\max_w J(w) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} \quad (30)$$

где m_1 и m_2 прогнозируемые средние арифметические значения, а s_1^2 и s_2^2 прогнозируемые значения рассеяния в пространстве функций. Сначала мы покажем, что w можно выразить как линейную комбинацию точек в пространстве признаков, а затем преобразуем задачу линейного дискриминантного анализа в терминах матрицы ядра.

Оптимальный линейный дискриминант: линейная комбинация характерных точек.

Среднее арифметическое значение для класса c_i в пространстве признаков задается как

$$\mu_i^\phi = \frac{1}{n_i} \sum_{x_j \in D_i} \phi(x_j) \quad (31)$$

а ковариационная матрица для класса c_i в пространстве признаков

$$\Sigma_i^\phi = \frac{1}{n_i} \sum_{x_j \in D_i} (\phi(x_j) - \mu_i^\phi)(\phi(x_j) - \mu_i^\phi)^T$$

Возможно получить выражение для матрицы рассеяния между классами в пространстве признаков





$$\mathbf{B}_\phi = (\boldsymbol{\mu}_1^\phi - \boldsymbol{\mu}_2^\phi)(\boldsymbol{\mu}_1^\phi - \boldsymbol{\mu}_2^\phi)^T = \mathbf{d}_\phi \mathbf{d}_\phi^T \quad (32)$$

где $\mathbf{d}_\phi = \boldsymbol{\mu}_1^\phi - \boldsymbol{\mu}_2^\phi$ разность между средними векторами двух классов. Матрица рассеяния внутри класса в пространстве признаков задается следующим образом

$$\mathbf{S}_\phi = n_1 \boldsymbol{\Sigma}_1^\phi + n_2 \boldsymbol{\Sigma}_2^\phi$$

$\mathbf{S}_\phi$ -симметричная положительная полуопределенная матрица, размером $d \times d$, где d -размерность пространства признаков. Возможно доказать, что наилучшим линейным дискриминантным вектором $\mathbf{w}$ в пространстве признаков является доминирующий собственный вектор, удовлетворяющий выражению

$$(\mathbf{S}_\phi^{-1} \mathbf{B}_\phi) \mathbf{w} = \lambda \mathbf{w} \quad (33)$$

Где мы предполагаем, что $\mathbf{S}_\phi$ не вырожден. Пусть δ_i обозначает i -е собственное значение, а $\mathbf{u}_i$ i -ый собственный вектор $\mathbf{S}_\phi$, при $i = 1, \dots, d$. Собственное разложение $\mathbf{S}_\phi$ даёт $\mathbf{S}_\phi = \mathbf{U} \boldsymbol{\Delta} \mathbf{U}^T$, с обратным значением $\mathbf{S}_\phi$, заданным как $\mathbf{S}_\phi^{-1} = \mathbf{U} \boldsymbol{\Delta}^{-1} \mathbf{U}^T$. Здесь $\mathbf{U}$ матрица, столбцы которой являются собственными векторами $\mathbf{S}_\phi$ а $\boldsymbol{\Delta}$ диагональная матрица собственных значений $\mathbf{S}_\phi$. Таким образом, $\mathbf{S}_\phi^{-1}$ может быть выражена как спектральная сумма

$$\mathbf{S}_\phi^{-1} = \sum_{r=1}^d \frac{1}{\delta_r} \mathbf{u}_r \mathbf{u}_r^T \quad (34)$$

Включая уравнения (32) и (34) в уравнении (33) получаем:

$$\lambda \mathbf{w} = \left(\sum_{r=1}^d \frac{1}{\delta_r} \mathbf{u}_r \mathbf{u}_r^T \right) \mathbf{d}_\phi \mathbf{d}_\phi^T \mathbf{w} = \sum_{r=1}^d \frac{1}{\delta_r} (\mathbf{u}_r (\mathbf{u}_r^T \mathbf{d}_\phi) (\mathbf{d}_\phi^T \mathbf{w})) = \sum_{r=1}^d b_r \mathbf{u}_r$$

где $b_r = \frac{1}{\delta_r} (\mathbf{u}_r^T \mathbf{d}_\phi) (\mathbf{d}_\phi^T \mathbf{w})$ скалярное значение. Используя деривацию, r -й собственный вектор $\mathbf{S}_\phi$ может быть выражен как линейная комбинация характерных точек, скажем $\mathbf{u}_r = \sum_{j=1}^n c_{rj} \phi(\mathbf{x}_j)$, где

c_{rj} скалярный коэффициент. Таким образом мы можем переписать $\mathbf{w}$





$$\begin{aligned}
\mathbf{w} &= \frac{1}{\lambda} \sum_{r=1}^d b_r \left(\sum_{j=1}^n c_{rj} \phi(\mathbf{x}_j) \right) \\
&= \sum_{j=1}^n \phi(\mathbf{x}_j) \left(\sum_{r=1}^d \frac{b_r c_{rj}}{\lambda} \right) \\
&= \sum_{j=1}^n a_j \phi(\mathbf{x}_j)
\end{aligned}$$

Где $a_j = \sum_{r=1}^d b_r c_{rj} / \lambda$ скалярное значение для характерной точки $\phi(\mathbf{x}_j)$. Поэтому вектор направления $\mathbf{w}$ может быть выражен как линейная комбинация точек в пространстве признаков.

Задача линейного дискриминантного анализа через матрицу ядра

Теперь мы перепишем задачу ядра линейного дискриминантного анализа [Ур. (30)] в терминах матрицы ядра. Проецируя среднее значение для класса c_i приведенное в уравнении (31) на направление линейного дискриминанта $\mathbf{w}$, мы имеем

$$\begin{aligned}
m_i = \mathbf{w}^T \boldsymbol{\mu}_i^\phi &= \left(\sum_{j=1}^n a_j \phi(\mathbf{x}_j) \right)^T \left(\frac{1}{n_i} \sum_{\mathbf{x}_k \in \mathbf{D}_i} \phi(\mathbf{x}_k) \right) \\
&= \frac{1}{n_i} \sum_{j=1}^n \sum_{\mathbf{x}_k \in \mathbf{D}_i} a_j \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_k) \\
&= \frac{1}{n_i} \sum_{j=1}^n \sum_{\mathbf{x}_k \in \mathbf{D}_i} a_j K(\mathbf{x}_j, \mathbf{x}_k) \\
&= \mathbf{a}^T \mathbf{m}_i
\end{aligned} \tag{35}$$

Где $\mathbf{a} = (a_1, a_2, \dots, a_n)^T$ весовой вектор, и



$$\mathbf{m}_i = \frac{1}{n_i} \begin{pmatrix} \sum_{\mathbf{x}_k \in \mathbf{D}_i} K(\mathbf{x}_1, \mathbf{x}_k) \\ \sum_{\mathbf{x}_k \in \mathbf{D}_i} K(\mathbf{x}_2, \mathbf{x}_k) \\ \vdots \\ \sum_{\mathbf{x}_k \in \mathbf{D}_i} K(\mathbf{x}_n, \mathbf{x}_k) \end{pmatrix} = \frac{1}{n_i} \mathbf{K}^{c_i} \mathbf{1}_{n_i} \quad (36)$$

Где $\mathbf{K}^{c_i}$ является $n \times n_i$ подмножеством ядра матрицы, ограниченного столбцами, принадлежащими точкам только в $\mathbf{D}_i$, и $\mathbf{1}_{n_i}$ является n_i -мерным вектором вес записи которого равны единице. Таким образом, вектор n -длины $\mathbf{m}_i$ сохраняет для каждой точки $\mathbf{D}$ свое среднее значение ядра по отношению к указанным точкам $\mathbf{D}_i$.

Мы можем переписать разделение между проецируемыми средними в пространстве признаков следующим образом:

$$\begin{aligned} (m_1 - m_2)^2 &= (\mathbf{w}^T \boldsymbol{\mu}_1^\phi - \mathbf{w}^T \boldsymbol{\mu}_2^\phi)^2 \\ &= (\mathbf{a}^T \mathbf{m}_1 - \mathbf{a}^T \mathbf{m}_2)^2 \\ &= \mathbf{a}^T (\mathbf{m}_1 - \mathbf{m}_2) (\mathbf{m}_1 - \mathbf{m}_2)^T \mathbf{a} \\ &= \mathbf{a}^T \mathbf{M} \mathbf{a} \end{aligned} \quad (37)$$

где $\mathbf{M} = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T$ матрица рассеяния между классами.

Мы также можем вычислить прогнозируемый разброс для каждого класса, s_1^2 и s_2^2 , исключительно в терминах функции ядра, как



$$\begin{aligned}
s_1^2 &= \sum_{\mathbf{x}_i \in \mathbf{D}_1} \|\mathbf{w}^T \phi(\mathbf{x}_i) - \mathbf{w}^T \boldsymbol{\mu}_1^\phi\|^2 \\
&= \sum_{\mathbf{x}_i \in \mathbf{D}_1} \|\mathbf{w}^T \phi(\mathbf{x}_i)\|^2 - 2 \sum_{\mathbf{x}_i \in \mathbf{D}_1} \mathbf{w}^T \phi(\mathbf{x}_i) \cdot \mathbf{w}^T \boldsymbol{\mu}_1^\phi + \sum_{\mathbf{x}_i \in \mathbf{D}_1} \|\mathbf{w}^T \boldsymbol{\mu}_1^\phi\|^2 \\
&= \left(\sum_{\mathbf{x}_i \in \mathbf{D}_1} \left\| \sum_{j=1}^n a_j \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_i) \right\|^2 \right) - 2 \cdot n_1 \cdot \|\mathbf{w}^T \boldsymbol{\mu}_1^\phi\|^2 + n_1 \cdot \|\mathbf{w}^T \boldsymbol{\mu}_1^\phi\|^2 \\
&= \left(\sum_{\mathbf{x}_i \in \mathbf{D}_1} \mathbf{a}^T \mathbf{K}_i \mathbf{K}_i^T \mathbf{a} \right) - n_1 \cdot \mathbf{a}^T \mathbf{m}_1 \mathbf{m}_1^T \mathbf{a} \\
&\quad \text{используя ур.(35)} \\
&= \mathbf{a}^T \left(\left(\sum_{\mathbf{x}_i \in \mathbf{D}_1} \mathbf{K}_i \mathbf{K}_i^T \right) - n_1 \mathbf{m}_1 \mathbf{m}_1^T \right) \mathbf{a} \\
&= \mathbf{a}^T \mathbf{N}_1 \mathbf{a}
\end{aligned}$$

где $\mathbf{K}_i$ i -ый столбец ядра матрицы, а $\mathbf{N}_1$ матрица рассеяния класса для c_1 . Пусть $K(x_i, x_j) = K_{ij}$. Мы можем выразить $\mathbf{N}_1$ более компактно в матричных обозначениях следующим образом:

$$\begin{aligned}
\mathbf{N}_1 &= \left(\sum_{\mathbf{x}_i \in \mathbf{D}_1} \mathbf{K}_i \mathbf{K}_i^T \right) - n_1 \mathbf{m}_1 \mathbf{m}_1^T \\
&= (\mathbf{K}^{c_1}) \left(\mathbf{I}_{n_1} - \frac{1}{n_1} \mathbf{1}_{n_1 \times n_1} \right) (\mathbf{K}^{c_1})^T \quad (38)
\end{aligned}$$

где $\mathbf{I}_{n_1}$ это тождественная матрица $n_1 \times n_1$, а $\mathbf{1}_{n_1 \times n_1}$ матрица $n_1 \times n_1$, все записи которой равны 1.

Аналогично получаем $s_2^2 = \mathbf{a}^T \mathbf{N}_2 \mathbf{a}$, где

$\mathbf{N}_2 = (\mathbf{K}^{c_2}) \left(\mathbf{I}_{n_2} - \frac{1}{n_2} \mathbf{1}_{n_2 \times n_2} \right) (\mathbf{K}^{c_2})^T$ где $\mathbf{I}_{n_2}$ это тождественная матрица $n_2 \times n_2$, а $\mathbf{1}_{n_2 \times n_2}$ матрица $n_2 \times n_2$, все записи которой равны 1.

Затем сумма прогнозируемых значений рассеяния задаётся в виде:

$$s_1^2 + s_2^2 = \mathbf{a}^T (\mathbf{N}_1 + \mathbf{N}_2) \mathbf{a} = \mathbf{a}^T \mathbf{N} \mathbf{a} \quad (39)$$

где $\mathbf{N}$ - матрица рассеяния $n \times n$ внутри класса



При замене уравнений (36) и (39) в уравнении (30) получим условие максимизации ядра линейного дискриминантного анализа

$$\max_{\mathbf{w}} J(\mathbf{w}) = \max_{\mathbf{a}} J(\mathbf{a}) = \frac{\mathbf{a}^T \mathbf{M} \mathbf{a}}{\mathbf{a}^T \mathbf{N} \mathbf{a}}$$

Обратите внимание, что все термины в приведенном выше выражении включают только функции ядра. Весовой вектор $\mathbf{a}$ - это собственный вектор, соответствующий наибольшему собственному значению обобщенной задачи на собственные значения:

$$\mathbf{M} \mathbf{a} = \lambda_1 \mathbf{N} \mathbf{a} \quad (40)$$

Если $\mathbf{N}$ невырожденный, то $\mathbf{a}$ -доминирующий собственный вектор, соответствующий наибольшему собственному значению для системы

$$(\mathbf{N}^{-1} \mathbf{M}) \mathbf{a} = \lambda_1 \mathbf{a}$$

Как и в случае линейного дискриминантного анализа, когда есть только два класса, нам не нужно решать для собственного вектора, потому что $\mathbf{a}$ может быть получено непосредственно:

$$\mathbf{a} = \mathbf{N}^{-1}(\mathbf{m}_1 - \mathbf{m}_2)$$

Как только $\mathbf{a}$ получено, мы можем нормализовать $\mathbf{w}$ как единичный вектор, гарантируя, что

$\mathbf{w}^T \mathbf{w} = 1$, из чего следует, что

$$\sum_{i=1}^n \sum_{j=1}^n a_i a_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = 1, \text{ или}$$

$$\mathbf{a}^T \mathbf{K} \mathbf{a} = 1$$

Иными словами, мы можем гарантировать, что $\mathbf{w}$ является единичным вектором, если масштабируем $\mathbf{a}$ на $\frac{1}{\sqrt{\mathbf{a}^T \mathbf{K} \mathbf{a}}}$.

В конечном итоге, мы можем спроецировать любую точку $\mathbf{x}$ на дискриминантное направление следующим образом:

$$\mathbf{w}^T \phi(\mathbf{x}) = \sum_{j=1}^n a_j \phi(\mathbf{x}_j)^T \phi(\mathbf{x}) = \sum_{j=1}^n a_j K(\mathbf{x}_j, \mathbf{x}) \quad (14)$$





В алгоритме 1 приводится псевдокод для дискриминантного анализа ядра. Суть метода заключается в вычислении ядерной матрицы K размера $n \times n$ и ядерных матриц K^{c_i} размером $n_i \times n_i$, для каждого класса c_i . После вычисления матриц межклассовой и внутриклассовой инвариантности M и N , вектор весов a получается как доминирующий собственный вектор $N^{-1}M$. Последний шаг масштабирует a так, чтобы w был нормализован до единичной длины. Сложность дискриминантного анализа ядра $O(n^3)$, при этом основными шагами являются вычисление N и нахождение доминирующего собственного вектора $N^{-1}M$, оба шага занимают время $O(n^3)$.

Алгоритм 1: Дискриминантный анализ ядра

KERNELDISCRIMINANT ($D = \{(x_i, y_i)\}_{i=1}^n, K$):

- 1 $K \leftarrow \{K(x_i, x_j)\}_{i,j=1,\dots,n}$ // compute $n \times n$ kernel matrix
- 2 $K^{c_i} \leftarrow \{K(j, k) \mid y_k = c_i, 1 \leq j, k \leq n\}, i = 1, 2$ // class kernel matrix
- 3 $m_i \leftarrow \frac{1}{n_i} K^{c_i} \mathbf{1}_{n_i}, i = 1, 2$ // class means
- 4 $M \leftarrow (m_1 - m_2)(m_1 - m_2)^T$ // between-class scatter matrix
- 5 $N_i \leftarrow K^{c_i} (\mathbf{I}_{n_i} - \frac{1}{n_i} \mathbf{1}_{n_i} \mathbf{1}_{n_i}^T) (K^{c_i})^T, i = 1, 2$ // class scatter matrices
- 6 $N \leftarrow N_1 + N_2$ // within-class scatter matrix
- 7 $\lambda_1, a \leftarrow \text{eigen}(N^{-1}M)$ // compute weight vector
- 8 $a \leftarrow \frac{a}{\sqrt{a^T K a}}$ // normalize w to be unit vector

3 НЕЙРОННЫЕ СЕТИ

3.1 Обучение нелинейных моделей с учителем

Обучение с учителем (Supervised learning) — один из разделов машинного обучения, посвященный решению следующей задачи. Имеется множество объектов (ситуаций) и множество возможных ответов (откликов, реакций). Существует некоторая зависимость между ответами и объектами, но она неизвестна. Известна только конечная совокупность прецедентов — пар «объект, ответ», называемая обучающей выборкой. На основе этих данных требуется восстановить зависимость, то есть построить алгоритм, способный для любого объекта выдать достаточно точный ответ. Для измерения точности ответов определённым образом вводится функционал качества.

Под учителем понимается либо сама обучающая выборка, либо тот, кто указал на заданных объектах правильные ответы.





Пусть X — множество описаний объектов, Y — множество допустимых ответов. Существует неизвестная целевая зависимость — отображение $y^*: X \rightarrow Y$, значения которой известны только на объектах конечной обучающей выборки $X^m = \{(x_1, y_1), \dots, (x_m, y_m)\}$. Требуется построить алгоритм $a: X \rightarrow Y$, который приближал бы неизвестную целевую зависимость как на элементах выборки, так и на всём множестве X .

Говорят также, что алгоритм должен обладать способностью к обобщению эмпирических фактов, или выводить общее знание (закономерность, зависимость) из частных фактов (наблюдений, прецедентов).

Данная постановка является обобщением классических задач аппроксимации функций. В классической аппроксимации объектами являются действительные числа или векторы. В реальных прикладных задачах входные данные об объектах могут быть неполными, неточными, неоднородными, нечисловыми. Эти особенности приводят к большому разнообразию методов обучения с учителем.

Функции потерь и функционалы качества

Вводится функция потерь $\mathcal{L}(y, y')$, характеризующая величину отклонения ответа $y = a(x)$ от правильного ответа $y' = y^*(x)$ на произвольном объекте $x \in X$.

Типичный выбор функции потерь:

- В задачах классификации $\mathcal{L}(y, y') = [y' \neq y]$;
- В задачах регрессии $\mathcal{L}(y, y') = (y' - y)^2$.

Вводится функционал качества, характеризующий среднюю ошибку (эмпирический риск) алгоритма a на произвольной выборке X^m

$$Q(a, X^m) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(a(x_i), y^*(x_i)).$$

Метод минимизации эмпирического риска — один из наиболее распространённых подходов к обучению алгоритмов по прецедентам. Он заключается в том, чтобы в заданной модели





алгоритмов $A = \{a: X \rightarrow Y\}$ найти алгоритм, минимизирующий среднюю ошибку на обучающей выборке:

$$a = \operatorname{argmin}_{a \in A} Q(a, X^m).$$

Тем самым задача обучения сводится к оптимизации и может быть решена численными методами оптимизации.

Обобщающая способность и проблема переобучения

К сожалению, малое значение функционала качества на обучающей выборке не гарантирует, что построенный алгоритм будет хорошо восстанавливать целевую зависимость на всём пространстве X . Существует опасность *переподгонки* или *переобучения*, когда делается попытка описать конкретные данные точнее, чем в принципе позволяет уровень шума в данных и погрешность самой модели.

Легко указать пример алгоритма, который минимизирует эмпирический риск до нуля, но при этом не обладает способностью к обобщению. Получив обучающую выборку X^m , он запоминает её, и потом сравнивает предъявляемый объект x с обучающими объектами x_i из X^m . В случае совпадения $x = x_i$ алгоритм выдаёт правильный ответ y_i . Иначе выдаётся произвольный ответ. Эмпирический риск принимает наименьшее возможное значение, равное нулю. Однако этот алгоритм не способен восстановить зависимость вне объектов обучения. Этот пример убедительно показывает, что для успешного обучения необходимо не только запоминать, но и обобщать.

Практически в каждом методе предпринимаются специальные усилия, чтобы избежать переобучения. границы применимости метода минимизации эмпирического риска и проблему переобучения изучает *теория вычислительного обучения*.

3.2 Нейронные сети

Искусственная нейронная сеть (artificial neural network, ANN), или просто нейронная сеть — это математическая модель, а также ее программные или аппаратные реализации, построенная в некотором смысле по образу и подобию сетей нервных клеток живого организма.





Нейронные сети — один из наиболее известных методов машинного обучения.

Суть метода

Как и линейные методы классификации и регрессии, по сути

нейронные сети выдают ответ вида: $y(x, w) = f\left(\sum_{j=1}^N w_j \phi_j(x)\right)$, где f — нелинейная функция активации, w — вектор весов, ϕ — нелинейные базисные функции. Обучение нейронных сетей состоит в настройке весов а также базисных функций.

Перцептрон (Персептрон, англ. perceptron от лат. perceptio — восприятие) — устройство МАРК-1, а также соответствующая ему математическая модель, созданная Фрэнком Розенблаттом с целью построения модели мозга. Под «моделью мозга» понимается любая теоретическая система, которая стремится объяснить физиологические функции мозга с помощью известных законов физики и математики, а также известных фактов нейроанатомии и нейрофизиологии. Перцептрон (строгое определение которого будет дано ниже) представляет собой передающую сеть, состоящую из генераторов сигнала трёх типов: сенсорных элементов, ассоциативных элементов и реагирующих элементов. Производящие функции этих элементов зависят от сигналов, возникающих либо где-то внутри передающей сети, либо, для внешних элементов, от сигналов, поступающих из внешней среды. Но, как правило, когда говорится "перцептрон Розенблатта", имеется в виду частный случай — т. н. элементарный перцептрон, который упрощён по сравнению с общим видом перцептрона по ряду параметров.

Определения, данные Ф. Розенблаттом

Серьезное ознакомление с теорией перцептронов требует знания базовых определений и теорем, совокупность которых и представляет собой базовую основу для всех последующих видов искусственных нейронных сетей

Виды элементов (нейронов) в перцептроне



- Простым **S-элементом** (Сенсорным элементом) является чувствительный элемент, который от воздействия какого-либо из видов энергии (например, света, звука, давления, тепла и т. п.) вырабатывает сигнал. Если входной сигнал c_{wi}^* превышает некоторый порог θ_i , то элемент выдаёт выходной сигнал $S_i^* = +1$, в противном случае выходной сигнал равен нулю.

- Простым **A-элементом** (Ассоциативным элементом) называется логический решающий элемент, который выдаёт выходной сигнал, когда алгебраическая сумма его входных сигналов a_i равна или превышает некоторую пороговую величину $\theta_i > 0$. Выходной сигнал a_i^* равен +1, если была превышена пороговая величина θ_i ; в противном случае он равен нулю. Если $a_i^* = +1$, то говорят, что A-элемент является активным.

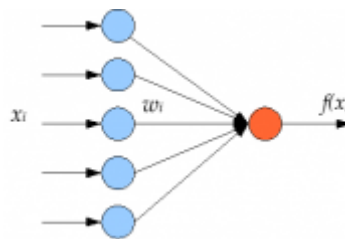


Рис. 15 Схема порогового A-элемента с пятью входами

- Простым **R-элементом** (Реагирующим элементом) называется элемент, который выдаёт выходной сигнал $r^* = +1$, если сумма его входных сигналов является строго положительной, и сигнал $r^* = -1$, если сумма его входных сигналов является строго отрицательной. Если сумма входных сигналов равна нулю, выход можно считать либо равным нулю, либо неопределённым.

Определения и классификация перцептронов

- **Перцептрон** представляет собой сеть, состоящую из S, A, R — элементов, с переменной матрицей взаимодействия V (элементы которой v_{ij} — весовые коэффициенты), определяемой последовательностью прошлых состояний активности сети.

- Логическое расстояние от элемента u_i к элементу u_j равно числу слоёв между ними.



- **Перцептроном с последовательными связями** называется система, в которой все связи, начинающиеся от элементов с логическим расстоянием d от ближайшего S-элемента, оканчиваются на элементах с логическим расстоянием $d+1$ от ближайшего S-элемента.

- **Перцептроном с перекрестными связями** называется система, в которой некоторые связи соединяют друг с другом элементы одного типа (S, A или R), находящиеся на одинаковом логическом расстоянии от S-элементов, причем все остальные связи — последовательного типа.

- **Перцептроном с обратной связью** называется система, в которой по крайней мере один элемент A или R, находящийся на расстоянии d_1 от ближайшего S-элемента, является исходным в цепи обратной связи к S-элементу или A-элементу, расстояние которого до ближайшего S-элемента $d_2 < d_1$; другими словами, это система с обратными связями от элементов, расположенных ближе к выходу, к элементам, расположенным ближе к входу системы.

- **Простым перцептроном** называется любая система, удовлетворяющая следующим пяти условиям:

- В системе имеется только один R-элемент, который связан со всеми A-элементами.
- Система представляет собой перцептрон с последовательными связями, идущими только от S-элементов к A-элементам и от A-элементов к R-элементам.
- Веса всех связей от S-элементов к A-элементам являются фиксированными (не изменяются во времени).
- Время передачи каждой связи равно либо нулю, либо фиксированной постоянной t .
- Все активирующие функции S, A, R - элементов имеют вид :

$$u_i^*(t) = f(a_i(t)),$$

где $a_i(t)$ - алгебраическая сумма всех сигналов, поступающих одновременно на вход элемента u_i .

- **Элементарным перцептроном** называется простой перцептрон с простыми R и A элементами, активирующая функция которого имеет вид:





$$c_{ij}(t) = u_i(t - t_0) * v_{ij}(t)$$

Определения, данные М. Минским

Минский изучал свойства параллельных вычислений, частным случаем которых на то время был перцептрон. Для анализа его свойств ему пришлось переизложить теорию перцептронов на язык предикатов. Хотя такой математический аппарат позволил применить анализ только к элементарному перцептрону Розенблатта, зато он вскрыл много принципиальных ограничений для параллельных вычислений, от которых не свободен не один вид современных искусственных нейронных сетей.

Определения и классификация перцептронов на языке предикатов

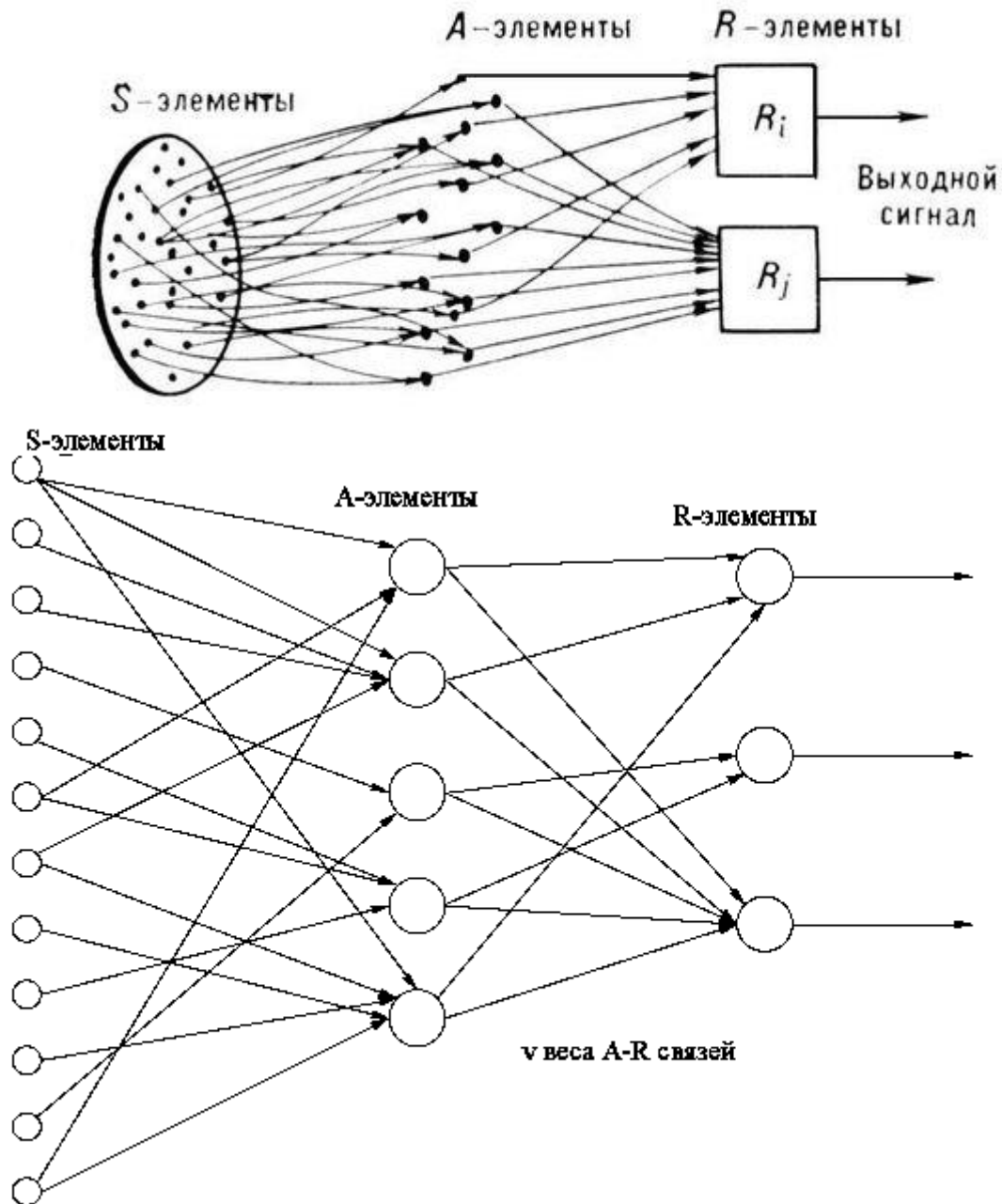
Определение. Перцептроном называется устройство, способное вычислять все предикаты, линейные относительно некоторого заданного множества Φ частных предикатов.

С этой позиции перцептроны можно классифицировать следующим образом:

- 1. **Перцептроны, ограниченные по диаметру.** Для каждого предиката F из Φ диаметр множества точек плоскости, от которых зависит F , не превосходит некоторой фиксированной величины.
- 2. **Перцептроны ограниченного порядка.** Мы будем говорить, что перцептрон имеет порядок $< n$, если каждый элемент из Φ зависит не более, чем от n точек.
- 3. **Перцептроны Гамбы.** Каждый элемент из Φ может зависеть от всех точек, но должен быть линейной пороговой функцией, т.е. сам должен вычисляться перцептроном первого порядка, определенным в предыдущем пункте.
- 4. **Случайные перцептроны.** Именно эта модель наиболее подробно изучалась группой Розенблатта. Предикаты F представляют собой случайно выбранные булевы функции. Иначе говоря, случайные перцептроны являются перцептронами ограниченного порядка, а множество Φ порождается случайным процессом с определенной функцией распределения.

- 5. Ограниченные перцептроны. Множество Φ предикатов F бесконечно, а множество значений принимаемых коэффициентами k , конечно.

Описание элементарного перцептрона



Перцептрон состоит из элементов 3-х типов: S — элементов, A — элементов и R — элемента. S — элементы это — слой рецепторов. Эти рецепторы соединены с A — элементами с помощью тормозных или возбуждающих связей. Каждый рецептор может находиться в одном из двух



состояний — покоя или возбуждения. А — элементы представляют собой сумматоры с порогом (то есть формальные нейроны). Это означает, что А — элемент возбуждается, если алгебраическая сумма возбуждений, приходящих к нему от рецепторов, превышает определенную величину — его порог. При этом сигнал от рецептора, приходящий по возбуждающей связи, считается положительным, а приходящий по тормозной связи — отрицательным. Сигналы от возбуждившихся А — элементов передаются в сумматор R, причем сигнал от i -го ассоциативного элемента передается с коэффициентом k_i .

А- или R- элементы (который является пороговыми) подсчитывают некоторую линейную форму (как правило сумму весовых коэффициентов) от своих входов и сравнивает ее с заданным значением — *порогом*. Если у А-элемента n входов, то в нем должны быть заданы n весов $w_1, w_2, \dots, w_n$ и *порог* θ . Перцептрон выдает 1, если линейная форма от входов с коэффициентами w_i превышает θ и -1 иначе.

$$f(x) = \text{sign}(\theta + \sum_{i=1}^n w_i x_i)$$

С помощью одного порогового элемента можно реализовать 14 булевых функций, кроме эквиваленции (знак равно) и исключающего или (XOR). Любая булева функция представима в виде построенной из пороговых элементов нейронной сети глубины 2.

Система связей между рецепторами S и А — элементами, так же как и пороги А — элементов выбираются некоторым случайным, но фиксированным образом, а обучение состоит лишь в изменении коэффициентов w_i . Считаем, что мы хотим научить перцептрон разделять два класса объектов, и потребуем, чтобы при предъявлении объектов первого класса выход перцептрона был положителен, а при предъявлении объектов второго класса — отрицательным. Начальные коэффициенты w_i полагаем равными нулю. Далее предъявляем обучающую выборку: объекты (например, круги либо квадраты) с указанием класса, к которому они принадлежат. Показываем перцептрону объект первого класса. При этом некоторые А — элементы возбуждятся. Коэффициенты w_i ,



соответствующие этим возбужденным элементам, увеличиваем на 1. Затем предъявляем объект второго класса и коэффициенты w_i тех A — элементов, которые возбуждятся при этом показе, уменьшаем на 1. Этот процесс продолжим для всей обучающей выборки. В результате обучения сформируются значения весов связей w_i .

После обучения перцептрон готов работать в режиме распознавания или обобщения. В этом режиме перцептрону предъявляются «не знакомые» перцептрону объекты, и перцептрон должен установить, к какому классу они принадлежат. Работа перцептрона состоит в следующем: при предъявлении объекта возбуждившиеся A — элементы передают сигнал R — элементу, равный сумме соответствующих коэффициентов w_i . Если эта сумма положительна, то принимается решение, что данный объект принадлежит к первому классу, а если она отрицательна — то второму.

Алгоритмы обучения

Классический метод обучения перцептрона — это обучение с коррекцией ошибки. Представляет собой такой метод обучения, при котором вес связи не изменяется до тех пор, пока текущая реакция перцептрона остается правильной. При появлении неправильной реакции вес изменяется на единицу, а знак (+/-) определяется противоположным от знака ошибки.

Кроме того, Розенблатт пытался классифицировать различные алгоритмы обучения, называя их системами подкрепления. В результате он различал следующие виды:

- **Положительное подкрепление** — представляет собой такой процесс подкрепления, при котором вес связи, начинающийся на активном элементе u_i и оканчивающийся на элементе u_j , изменяется на величину r , знак которой совпадает со знаком сигнала u_j^* .
- **Отрицательное подкрепление** — представляет собой такой процесс подкрепления, при котором вес связи, начинающийся на активном элементе u_i и оканчивающийся на элементе u_j , изменяется на величину r , знак которой противоположен знаку сигнала u_j^* .





Кроме, классического метода обучения перцептрона Розенблатт также ввел понятие об обучении без учителя, предложив следующий способ обучения:

- **Альфа – системой подкрепления** называется система подкрепления, при которой веса всех активных связей c_{ij} , которые оканчиваются на некотором элементе u_j , изменяются на одинаковую величину α , а веса не активных связей за это время не изменяются.

Описывая эти системы подкрепления Розенблатт основывался на идеях Хебба об обучении, уточняя различные возможные виды. Затем с появлением многослойного перцептрона оно было модифицировано и его стали называть дельта-правило. Модификация была проведена с целью сделать функцию обучения дифференцированной, что в свою очередь нужно для применения метода градиентного спуска, благодаря которому и возможно обучение более одного слоя. При этом пришлось отказаться от бинарного сигнала, и пользоваться на входе вещественными числами.

Возможности и ограничения модели

Исследования перцептронов показали, что перцептроны способны обучаться. Справедлива теорема о сходимости перцептрона, согласно которой независимо от начальных значений коэффициентов и порядка показа образцов при обучении перцептрон за конечное число шагов научится различать два класса объектов, если только существует такая классификация.

Первые успехи исследований перцептронов и других нейросетей вызвал взрыв активности и энтузиазма. М. Минский, Ф.Розенблат, Б. Уидроу и другие разработали ряд искусственных нейронных сетей. В течение некоторого времени казалось, что ключ к интеллекту найден, и воспроизведение человеческого мозга является лишь вопросом конструирования достаточно большой сети.

Но эта иллюзия вскоре рассеялась. Возможности перцептронов оказались довольно ограниченными. Серьезный математический анализ перцептронов был проведен М.Минским и С. Пейпертом (подробнее см.





ниже). В последствии работа Вассермана, вызвала новый всплеск активности в области искусственных нейронных сетей, и применение идей теории перцептронов на новый лад с образованием собственной новой терминологии и становлением науки о нейросетях, но с точки зрения технического приложения в противовес построению моделей мозга. Но к сожалению некоторые неточности в его работе привели к ряду недоразумений. Так, например, Вассерманом была предложена классификация искусственных нейронных сетей на основе подсчета числа обучаемых слоев связей, а не по числу структурных элементов сети. Но такая классификация проблематична, так как не позволяет говорить об особенностях определенного вида нейросетей. Это вызвало ряд недоразумений в последующие годы при определении перцептрона, так как сам автор всегда говорил о нем, как о трехслойном, а классификация по числу обучаемых слоев предполагала называть его однослойным. Но к сожалению, это сказалось не только на терминологии, но и не верном представлении о перцептроне как простейшем пороговом элементе. Так как была не учтена роль первого необучаемого слоя.

Структура, состоящая из нескольких соединенных слоев пороговых элементов называется многослойным перцептроном. Такой термин появился когда был предложен алгоритм обратного распространения, с помощью которого стали обучать (настраивать коэффициенты) более одного слоя. Но такую структуру не стоит прямо сравнивать с перцептроном Розенблатта. Это достаточно разные модели нейросетей, но которые во многом эквивалентны друг другу в общем случае, и специфичны для разного рода задач.

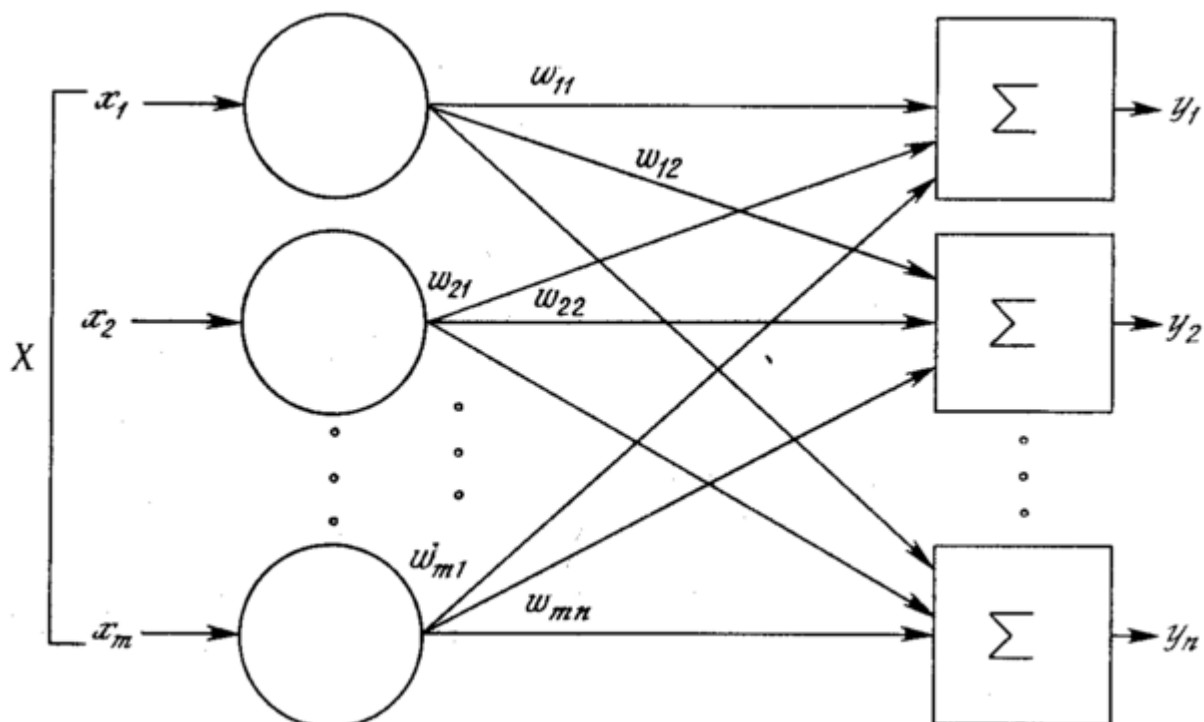
Виды нейронных сетей.

Однослойная нейронная сеть (англ. *Single-layer neural network*) – сеть, в которой сигналы от входного слоя сразу подаются на выходной слой, который и преобразует сигнал и сразу же выдает ответ.

Как видно из схемы однослойной нейронной сети, представленной справа, сигналы $x_1, x_2, \dots, x_n$ поступают на входной слой (который не считается за слой нейронной сети), а затем сигналы распределяются на выходной слой



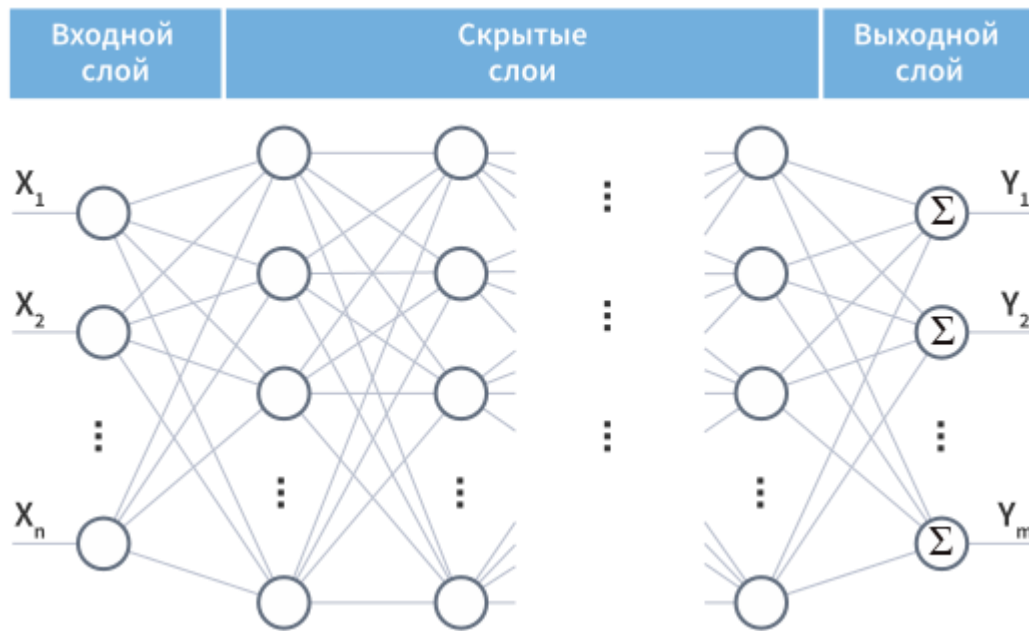
обычных нейронов. На каждом ребре от нейрона входного слоя к нейрону выходного слоя написано число – вес соответствующей связи.



Многослойная нейронная сеть (англ. *Multilayer neural network*) – нейронная сеть, состоящая из входного, выходного и расположенного(ых) между ними одного (нескольких) скрытых слоев нейронов.

Помимо входного и выходного слоев эти нейронные сети содержат промежуточные, *скрытые слои*. Такие сети обладают гораздо большими возможностями, чем однослойные нейронные сети, однако методы обучения нейронов скрытого слоя были разработаны относительно недавно.

Работу скрытых слоев нейронов можно сравнить с работой большого завода. Продукт (выходной сигнал) на заводе собирается по стадиям на станках. После каждого станка получается какой-то промежуточный результат. Скрытые слои тоже преобразуют входные сигналы в некоторые промежуточные результаты.



Сети прямого распространения (англ. *Feedforward neural network*) (feedforward сети) — искусственные нейронные сети, в которых сигнал распространяется строго от входного слоя к выходному. В обратном направлении сигнал не распространяется.

Все сети, описанные выше, являлись сетями прямого распространения, как следует из определения. Такие сети широко используются и вполне успешно решают определенный класс задач: прогнозирование, кластеризация и распознавание.

Однако сигнал в нейронных сетях может идти и в обратную сторону.

Сети с обратными связями (англ. *Recurrent neural network*) — искусственные нейронные сети, в которых выход нейрона может вновь подаваться на его вход. В более общем случае это означает возможность распространения сигнала от выходов к входам.

В сетях прямого распространения выход сети определяется входным сигналом и весовыми коэффициентами при искусственных нейронах. В сетях с обратными связями выходы нейронов могут возвращаться на входы. Это означает, что выход какого-нибудь нейрона определяется не только его весами и входным сигналом, но еще и предыдущими выходами (так как они снова вернулись на входы).



3.3 Обратное распространение ошибки

Метод обратного распространения ошибки (англ. *backpropagation*) – метод вычисления градиента, который используется при обновлении весов многослойного перцептрона. Впервые метод был описан в 1974 г. А. И. Галушкиным, а также независимо и одновременно Полом Дж. Вербосом. Далее существенно развит в 1986 г. Дэвидом И. Румельхартом, Дж. Е. Хинтоном и Рональдом Дж. Вильямсом и независимо и одновременно С.И. Барцевым и В.А. Охониным. Это итеративный градиентный алгоритм, который используется с целью минимизации ошибки работы многослойного перцептрона и получения желаемого выхода.

Основная идея этого метода состоит в распространении сигналов ошибки от выходов сети к её входам, в направлении обратном прямому распространению сигналов в обычном режиме работы. Барцев и Охонин предложили обобщающий метод («принцип двойственности»), применимый к более широкому классу систем, включая системы с запаздыванием, распределённые системы, и т. п.

Для возможности применения метода обратного распространения ошибки передаточная функция нейронов должна быть дифференцируема. Метод является изменением классического метода градиентного спуска.

Обучение как задача оптимизации

Рассмотрим простую нейронную сеть без скрытых слоев, с двумя входными вершинами и одной выходной, в которых каждый нейрон использует линейную функцию активации, (обычно, многослойные нейронные сети используют нелинейные функции активации, линейные функции используются для упрощения понимания) которая является взвешенной суммой входных данных.



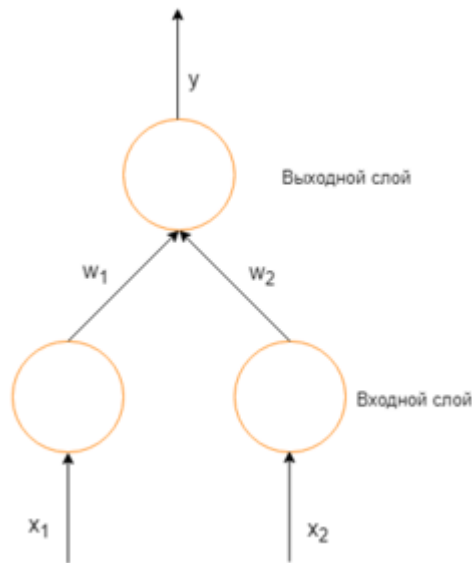


Рис. 16 Простая нейронная сеть с двумя входными вершинами и одной выходной.

Изначально веса задаются случайно. Затем, нейрон обучается с помощью тренировочного множества, которое в этом случае состоит из множества троек (x_1, x_2, t) где x_1 , x_2 — это входные данные сети и t — правильный ответ. Начальная сеть, приняв на вход x_1 и x_2 , вычислит ответ y , который вероятно отличается от t . Общепринятый метод вычисления несоответствия между ожидаемым t и получившимся y ответом — квадратичная функция потерь:

$$E = (t - y)^2, \text{ где } E - \text{ошибка.}$$

В качестве примера, обучим сеть на объекте $(1, 1, 0)$, таким образом, значения x_1 и x_2 равны 1, а t равно 0. Построим график зависимости ошибки E от действительного ответа y , его результатом будет парабола. Минимум параболы соответствует ответу y , минимизирующему E . Если тренировочный объект один, минимум касается горизонтальной оси, следовательно ошибка будет нулевая и сеть может выдать ответ y равный ожидаемому ответу t . Следовательно, задача преобразования входных значений в выходные может быть сведена к задаче оптимизации, заключающейся в поиске функции, которая даст минимальную ошибку.



Рис. 17 График ошибки для нейрона с линейной функцией активации и одним тренировочным объектом

В таком случае, выходное значение нейрона — взвешенная сумма всех его входных значений:

$$y = x_1 w_1 + x_2 w_2,$$

где $x_1 w_1$ и $x_2 w_2$ — веса на ребрах, соединяющих входные вершины с выходной. Следовательно, ошибка зависит от весов ребер, входящих в нейрон. И именно это нужно менять в процессе обучения. Распространенный алгоритм для поиска набора весов, минимизирующего ошибку — градиентный спуск. Метод обратного распространения ошибки используется для вычисления самого "крутого" направления для спуска.

Дифференцирование для однослойной сети

Метод градиентного спуска включает в себя вычисление дифференциала квадратичной функции ошибки относительно весов сети. Обычно это делается с помощью метода обратного распространения ошибки. Предположим, что выходной нейрон один, (их может быть несколько, тогда ошибка — это квадратичная норма вектора разницы) тогда квадратичная функция ошибки:

$E = \frac{1}{2}(t - y)^2$, где E — квадратичная ошибка, t — требуемый ответ для обучающего образца, y — действительный ответ сети.

Множитель $\frac{1}{2}$ добавлен чтобы предотвратить возникновение экспоненты во время дифференцирования. На результат это не повлияет, потому что позже выражение будет умножено на произвольную величину скорости обучения (англ. *learning rate*).

Для каждого нейрона j его выходное значение o_j определено как



$$o_j = \varphi(\text{net}_j) = \varphi\left(\sum_{k=1}^n w_{kj} o_k\right).$$

Входные значения net_j нейрона — это взвешенная сумма выходных значений o_k предыдущих нейронов. Если нейрон в первом слое после входного, то o_k входного слоя — это просто входные значения x_k сети. Количество входных значений нейрона n . Переменная w_{kj} обозначает вес на ребре между нейроном k предыдущего слоя и нейроном j текущего слоя.

Функция активации φ нелинейна и дифференцируема. Одна из распространенных функций активации — сигмоида:

$$\varphi(z) = \frac{1}{1+e^{-z}}$$

у нее удобная производная:

$$\frac{d\varphi(z)}{dz} = \varphi(z)(1 - \varphi(z))$$

Находим производную ошибки

Вычисление частной производной ошибки по весам w_{ij} выполняется с помощью цепного правила:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial w_{ij}}$$

Только одно слагаемое в net_j зависит от w_{ij} , так что

$$\frac{\partial \text{net}_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{k=1}^n w_{kj} o_k \right) = \frac{\partial}{\partial w_{ij}} w_{ij} o_i = o_i.$$

Если нейрон в первом слое после входного, то o_i — это просто x_i .

Производная выходного значения нейрона j по его входному значению — это просто частная производная функции активации (предполагается что в качестве функции активации используется сигмоида):

$$\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial}{\partial \text{net}_j} \varphi(\text{net}_j) = \varphi(\text{net}_j)(1 - \varphi(\text{net}_j))$$

По этой причине данный метод требует дифференцируемой функции активации. (Тем не менее, функция ReLU стала достаточно популярной в последнее время, хоть и не дифференцируема в 0)

Первый множитель легко вычислим, если нейрон находится в выходном слое, ведь в таком случае $o_j = y_j$ и





$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y} = \frac{\partial}{\partial y} \frac{1}{2} (t - y)^2 = y - t$$

Тем не менее, если j произвольный внутренний слой сети, нахождение производной E по o_j менее очевидно.

Если рассмотреть E как функцию, берущую на вход все нейроны $L=u, v, \dots, w$ получающие на вход значение нейрона j ,

$$\frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(\text{net}_u, \text{net}_v, \dots, \text{net}_w)}{\partial o_j}$$

и взять полную производную по o_j , то получим рекурсивное выражение для производной:

$$\frac{\partial E}{\partial o_j} = \sum_{\ell \in L} \left(\frac{\partial E}{\partial \text{net}_\ell} \frac{\partial \text{net}_\ell}{\partial o_j} \right) = \sum_{\ell \in L} \left(\frac{\partial E}{\partial o_\ell} \frac{\partial o_\ell}{\partial \text{net}_\ell} w_{j\ell} \right)$$

Следовательно, производная по o_j может быть вычислена если все производные по выходным значениям o_ℓ следующего слоя известны.

Если собрать все вместе:

$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i$$

и

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j} = \begin{cases} (o_j - t_j) o_j (1 - o_j) & \text{если } j \text{ является нейроном выходного слоя,} \\ (\sum_{\ell \in L} w_{j\ell} \delta_\ell) o_j (1 - o_j) & \text{если } j \text{ является нейроном внутреннего слоя.} \end{cases}$$

Чтобы обновить вес w_{ij} используя градиентный спуск, нужно выбрать скорость обучения, $\eta > 0$. Изменение в весах должно отражать влияние E на

увеличение или уменьшение в w_{ij} . Если $\frac{\partial E}{\partial w_{ij}} > 0$,

увеличение w_{ij} увеличивает E ; наоборот, если $\frac{\partial E}{\partial w_{ij}} < 0$, увеличение w_{ij} уменьшает E . Новый Δw_{ij} добавлен к старым весам, и произведение скорости обучения на градиент, умноженный на -1 , гарантирует, что w_{ij} изменения будут всегда уменьшать E . Другими словами,

в следующем уравнении, $-\eta \frac{\partial E}{\partial w_{ij}}$ всегда изменяет w_{ij} в такую сторону, что E уменьшается:





$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta \delta_j o_i$$

Алгоритм

- η - скорость обучения
- α - коэффициент инерциальности для сглаживания резких скачков при перемещении по поверхности целевой функции
- $\{x_i^d, t^d\}_{i=1, d=1}^{n, m}$ — обучающее множество
- *steps* — количество повторений
- *network*(*x*) — функция, подающая *x* на вход сети и возвращающая выходные значения всех ее узлов
- *layers* — количество слоев в сети
- *layer_i* — множество нейронов в слое *i*
- *output* — множество нейронов в выходном слое

```
fun BackPropagation( $\eta, \alpha, \{x_i^d, t^d\}_{i=1, d=1}^{n, m}, steps$ ):  
  init  $\{w_{ij}\}_{i, j}$   
  repeat steps:  
    for  $d = 1$  to  $m$ :  
       $o = network(\{x_i^d\})$   
      for  $k \in output$ :  
         $\delta_k = \sigma'(o_k)(t_k - o_k)$   
      for  $i = layers - 1$  to  $1$ :  
        for  $j \in layer_i$ :  
           $\delta_j = \sigma'(o_j) \sum_{k \in layer_{i+1}} \delta_k w_{j, k}$   
      for  $\forall w_{i, j}$ :  
         $\Delta w_{i, j}^n = \alpha \Delta w_{i, j}^{n-1} + (1 - \alpha) \eta \delta_j o_i$   
         $w_{i, j}^n = w_{i, j}^{n-1} + \Delta w_{i, j}^n$   
  return  $w_{ij}$ 
```

Недостатки алгоритма

Несмотря на многочисленные успешные применения обратного распространения, оно не является универсальным решением. Больше всего неприятностей приносит неопределённо долгий процесс обучения. В сложных задачах для обучения сети могут потребоваться дни или даже недели, она может и вообще не обучиться. Причиной может быть одна из описанных ниже.





Рис. 18 Градиентный спуск может найти локальный минимум вместо глобального

Паралич сети

В процессе обучения сети значения весов могут в результате коррекции стать очень большими величинами. Это может привести к тому, что все или большинство нейронов будут функционировать при очень больших выходных значениях, а производная активирующей функции будет очень мала. Так как посылаемая обратно в процессе обучения ошибка пропорциональна этой производной, то процесс обучения может практически замереть.

Локальные минимумы

Градиентный спуск с обратным распространением ошибок гарантирует нахождение только локального минимума функции; также, возникают проблемы с пересечением плато на поверхности функции ошибки.

4 ТЕОРИЯ ОБУЧЕНИЯ

4.1 Матожидание и дисперсия

Ознакомиться с материалами по настоящему подразделу можно здесь:

Математические методы обучения по прецедентам (теория обучения машин) К. В. Воронцов <http://www.ccas.ru/voronvoron@ccas.ru>



4.2 Неравенства Хеффдинга

Неравенство Хеффдинга даёт верхнюю границу вероятности того, что сумма случайных величин отклоняется от своего математического ожидания. Неравенство Хеффдинга было доказано Василием Хеффдингом в 1963 году. Неравенство Хеффдинга является частным случаем неравенства Адзума – Хеффдинга и более общим случаем неравенства Бернштейна, доказанного Сергеем Бернштейном в 1923 году. Они также являются частными случаями неравенства МакДиармида.

Частный случай для случайных величин Бернулли

Неравенство Хеффдинга может быть применено к важному частному случаю одинаково распределённых бернуллиевских случайных величин, и, как неравенство, часто используется в комбинаторике и информатике. Рассмотрим смещённую монету, у которой орёл выпадает с вероятностью p и решка – с вероятностью $1-p$. Мы бросаем монету n раз. Математическое ожидание того, сколько раз монета упадет орлом, есть $p \cdot n$. Далее, вероятность того, что монета упадет орлом не более k раз, может быть точно оценена выражением:

$$\Pr(H(n) \leq k) = \sum_{i=0}^k \binom{n}{i} p^i (1-p)^{n-i}.$$

В случае $k = (p - \varepsilon)n$ для некоторого $\varepsilon > 0$ неравенство Хеффдинга ограничивает эту вероятность выражением, которое экспоненциально убывает от $\varepsilon^2 \cdot n$:

$$\Pr(H(n) \leq (p - \varepsilon)n) \leq \exp(-2\varepsilon^2 n).$$

Похожим образом, в случае $k = (p + \varepsilon)n$ для некоторого $\varepsilon > 0$ неравенство Хеффдинга ограничивает вероятность того, что выпадет не меньше εn орлов, чем ожидаемо, выражением:



$$\Pr \left(H(n) \geq (p + \varepsilon)n \right) \leq \exp \left(- 2\varepsilon^2 n \right).$$

Таким образом, неравенство Хёфдинга означает, что число выпадений орла, концентрируется вокруг среднего, с экспоненциально малым хвостом.

$$\Pr \left((p - \varepsilon)n \leq H(n) \leq (p + \varepsilon)n \right) \geq 1 - 2 \exp \left(- 2\varepsilon^2 n \right).$$

Общий случай

Пусть $X_1, \dots, X_n$ — независимые случайные величины.

Положим, что X_i являются почти достоверно ограниченными, то есть, положим для $1 \leq i \leq n$, что:

$$\Pr(X_i \in [a_i, b_i]) = 1.$$

Мы определяем эмпирическое среднее этих переменных:

$$\bar{X} = (X_1 + \dots + X_n)/n.$$

Теорема 2 из Hoeffding (1963), доказывает неравенства:

$$\Pr(\bar{X} - \mathbb{E}[\bar{X}] \geq t) \leq \exp \left(- \frac{2t^2 n^2}{\sum_{i=1}^n (b_i - a_i)^2} \right),$$

$$\Pr(|\bar{X} - \mathbb{E}[\bar{X}]| \geq t) \leq 2 \exp \left(- \frac{2t^2 n^2}{\sum_{i=1}^n (b_i - a_i)^2} \right),$$

которые верны для всех положительных значений t . Здесь $\mathbb{E}[\bar{X}]$ является мат.ожиданием $\bar{X}$.

Заметим, что неравенство также верно, если X_i были получены с использованием выборки без замены, в данном случае случайные переменные не являются больше независимыми. Доказательство этого утверждения можно найти в статье Хёфдинга. Для несколько лучших оценок границ в случае выборки без замены.

4.3 Регуляризация

Регуляризация (англ. *regularization*) в статистике, машинном обучении, теории обратных задач — метод добавления некоторых дополнительных ограничений к условию с целью решить некорректно поставленную задачу или предотвратить переобучение. Чаще всего эта информация имеет вид штрафа за сложность модели.

Регуляризация полезна для борьбы с переобучением. Если вы выбрали сложную модель, и при этом у вас недостаточно данных, то легко можно получить итоговую модель, которая хорошо описывает обучающую выборку, но не обобщается на тестовую.

На примере линейной регрессии

В качестве наглядного примера рассмотрим линейные регрессионные модели. Восстановить зависимость для нескольких точек можно пытаться полиномами разной степени M .

На Рис. 19 представлена зависимость, которая хорошо подходит для описания данных, а на Рис. 20 — модель, слишком сильно заточенная под обучающую выборку.

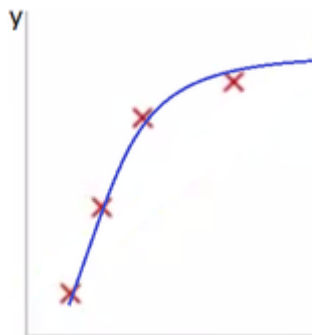


Рис. 19. Норма.
 $M = 2$

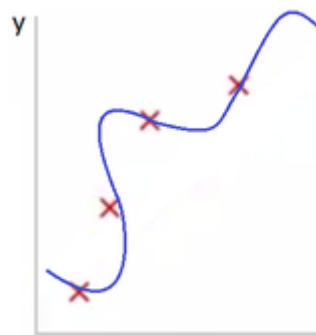


Рис. 20. Переобучение.
 $M = 4$

Одним из способов бороться с негативным эффектом излишнего подстраивания под данные — использование регуляризации, т. е. добавление некоторого штрафа за большие значения коэффициентов y



линейной модели. Тем самым запрещаются слишком "резкие" изгибы, и предотвращается переобучение.

Основные виды регуляризации

Переобучение в большинстве случаев проявляется в том, что итоговые модели имеют слишком большие значения параметров. Соответственно, необходимо добавить в целевую функцию штраф за это. Наиболее часто используемые виды регуляризации — L_1 и L_2 , а также их линейная комбинация — эластичная сеть.

В представленных ниже формулах для эмпирического риска Q : L является функцией потерь, а β — вектором параметров $g(x, \beta)$ из модели алгоритма, а λ — неотрицательный гиперпараметр, являющийся коэффициентом регуляризации.

L_2 -регуляризация

Определение:

L_2 -регуляризация, или регуляризация Тихонова (англ. *ridge regularization* или *Tikhonov regularization*):

$$Q(\beta, X^l) = \sum_{i=1}^l \mathcal{L}(y_i, g(x_i, \beta)) + \lambda \sum_{j=1}^n \beta_j^2.$$

Минимизация регуляризованного соответствующим образом эмпирического риска приводит к выбору такого вектора параметров β , которое не слишком сильно отклоняется от нуля. В линейных классификаторах это позволяет избежать проблем мультиколлинеарности и переобучения.

L_1 -регуляризация

Определение:

L_1 -регуляризация (англ. *lasso regularization*), или регуляризация через манхэттенское расстояние:

$$Q(\beta, X^l) = \sum_{i=1}^l \mathcal{L}(y_i, g(x_i, \beta)) + \lambda \sum_{j=1}^n |\beta_j|.$$





Данный вид регуляризации также позволяет ограничить значения вектора β . Однако, к тому же он обладает интересным и полезным на практике свойством – обнуляет значения некоторых параметров, что в случае с линейными моделями приводит к отбору признаков.

Запишем задачу настройки вектора параметров β :

$$Q(\beta) = \sum_{i=1}^l \mathcal{L}_i(\beta) + \lambda \sum_{j=1}^n |\beta_j|,$$

где $\mathcal{L}_i(\beta) = \mathcal{L}(y_i, g(x_i, \beta))$ – некоторая ограниченная гладкая функция потерь. Сделаем замену переменных, чтобы функционал стал гладким. Каждой переменной β_j поставим в соответствие две новые неотрицательные переменные:

$$\begin{cases} u_j = \frac{1}{2}(|\beta_j| + \beta_j) \\ v_j = \frac{1}{2}(|\beta_j| - \beta_j) \end{cases}$$

Тогда:

$$\begin{cases} \beta_j = u_j - v_j \\ |\beta_j| = u_j + v_j \end{cases}$$

В новых переменных функционал становится гладким, но добавляются ограничения-неравенства:

$$\begin{cases} Q(u, v) = \sum_{i=1}^l \mathcal{L}_i(u - v) + \lambda \sum_{j=1}^n (u_j + v_j) \rightarrow \min_{u, v} \\ u_j \geq 0, v_j \geq 0, j = 1, \dots, n \end{cases}$$

Для любого j хотя бы одно из ограничений $u_j \geq 0$ и $v_j \geq 0$ обращается в равенство, иначе второе слагаемое в $Q(u, v)$ можно было бы уменьшить, не изменив первое. Если гиперпараметр λ устремить к ∞ , в какой-то момент все $2n$ ограничений обратятся в равенство. Постепенное увеличение гиперпараметра λ приводит к увеличению числа таких j , для которых $u_j = v_j = 0$, откуда следует, что $\beta_j = 0$. Как говорилось ранее, в линейных моделях это



означает, что значения j -го признака игнорируются, и его можно исключить из модели.

Эластичная сеть

Определение:

Эластичная сеть (англ. *elastic net regularization*):

$$Q(\beta, X^l) = \sum_{i=1}^l \mathcal{L}(y_i, g(x_i, \beta)) + \lambda_1 \sum_{j=1}^n |\beta_j| + \lambda_2 \sum_{j=1}^n \beta_j^2.$$

Приведенная регуляризация использует как L_1 , так и L_2 регуляризации, учитывая эффективность обоих методов. Ее полезной особенностью является то, что она создает условия для группового эффекта при высокой корреляции переменных, а не обнуляет некоторые из них, как в случае с L_1 -регуляризацией.

4.4 Метод k -средних

Для набора данных с n точками в d -мерном пространстве, $\mathbf{D} = \{\mathbf{x}_i\}_{i=1}^n$, и количества желаемых кластеров k , цель репрезентативной кластеризации состоит в том, чтобы разделить набор данных на k групп или кластеров, обозначаемых как $C = \{C_1, C_2, \dots, C_k\}$. Далее, для каждого кластера C_i существует репрезентативная точка, которая суммирует кластер. Обычно выбирается среднее значение (также называемое центроидом) μ_i всех точек в кластере, т. е.,

$$\mu_i = \frac{1}{n_i} \sum_{x_j \in C_i} x_j,$$

где $n_i = |C_i|$ - количество точек в кластере C_i .

Полный перебор или исчерпывающий алгоритм поиска хорошей кластеризации состоит в том, чтобы просто сгенерировать все возможные разбиения n точек на k кластеров, произвести оценку оптимизации для каждого из них и сохранить кластеризацию, которая дает лучший результат. Точное количество способов разбить n точек на k непустых и непересекающихся частей можно получить с помощью чисел Стирлинга второго рода, заданных как:





$$S(n, k) = \frac{1}{k!} \sum_{t=0}^k (-1)^t \binom{k}{t} (k-t)^n$$

Неформально каждую точку можно назначить любому из k кластеров, поэтому существует не более k^n возможных кластеров. Однако любая перестановка k кластеров в рамках данной кластеризации дает эквивалентную кластеризацию; следовательно, имеется $O(k^n/k!)$ кластеризаций n точек в k групп. Понятно, что исчерпывающее перечисление и оценка всех возможных кластеров практически неосуществимы. В этой главе мы описываем два подхода к репрезентативной кластеризации, а именно алгоритм К-средних и ЕМ-алгоритм.

Алгоритм К-средних

Для кластеризации $C = \{C_1, C_2, \dots, C_k\}$, нам нужна некоторая функция оценки, которая оценивает её качество. Эта функция оценки суммы квадратов ошибок определяется как:

$$SSE(C) = \sum_{i=1}^k \sum_{x_j \in C_i} \|x_j - \mu_i\|^2 \quad (41)$$

Цель состоит в том, чтобы найти кластеризацию, которая минимизирует оценку SSE:

$$C^* = \arg \min_C \{SSE(C)\}$$

К-средних использует жадный итерационный подход для поиска кластеризации, которая минимизирует цель SSE [уравнение. (41)]. Таким образом, он может сходиться к локальным оптимумам вместо глобально оптимальной кластеризации.

К-средних инициализирует средние кластеров путем случайной генерации k точек в пространстве данных. Обычно это делается путем генерации равномерной случайной величины в пределах диапазона для каждого измерения. Каждая итерация К-средних состоит из двух шагов: (1) назначение кластера и (2) обновление центроида. Для k средних кластера на этапе назначения кластера каждой точке $x_j \in D$ присваивается ближайшее среднее значение, что вызывает кластеризацию, причем каждый кластер C_i



содержит точки, которые ближе к μ_i , чем любое другое среднее значение кластера. То есть каждая точка x_j относится к кластеру C_{j^*} , где

$$j^* = \arg \min_{i=0}^k \{ \|x_j - \mu_i\|^2 \} \quad (42)$$

Для набора кластеров $C = \{C_1, C_2, \dots, C_k\}$, на этапе обновления центроида, новые средние значения вычисляются для каждого кластера из точек в C_i . Этапы назначения кластера и обновления центроида выполняются итеративно, пока мы не достигнем фиксированной точки или локальных минимумов. С практической точки зрения можно предположить, что К-средние сошлись, если центроиды не меняются от одной итерации к другой. Например, мы можем остановиться, если $\sum_{i=1}^k \|\mu_i^t - \mu_i^{t-1}\|^2 \leq \epsilon$, где $\epsilon > 0$ - порог сходимости, t обозначает текущую итерацию, и μ_i^t - среднее для кластера C_i в итерации t .

Поскольку метод К-средних начинается со случайного предположения для начальных центроидов, К-средних обычно запускается несколько раз, и для отчета окончательной кластеризации выбирается запуск с наименьшим значением SSE. Также стоит отметить, что К-средних генерирует кластеры выпуклой формы, поскольку область в пространстве данных, соответствующем каждому кластеру, может быть получена как пересечение полупространств, возникающих из гиперплоскостей, которые делят пополам и перпендикулярны линейным сегментам, которые соединяют пары центроидов кластера.

С точки зрения вычислительной сложности К-средних, мы можем видеть, что этап назначения кластера занимает $O(nkd)$ времени, потому что для каждой из n точек мы должны вычислить расстояние до каждого из k кластеров, что требует d операций в d размерностях. Шаг повторного вычисления центроида занимает $O(nd)$ времени, потому что мы должны добавить в общей сложности n d -мерных точек. Предполагая, что существует t итераций, общее время для К-средних задается как $O(nkd)$. С точки зрения затрат ввода-вывода для этого требуется $O(t)$ полных



сканирований базы данных, потому что мы должны читать всю базу данных на каждой итерации.

Алгоритм 2: К-средних

K-MEANS ($\mathbf{D}, k, \epsilon$):

```

1  $t = 0$ 
2 Randomly initialize  $k$  centroids:  $\mu_1^t, \mu_2^t, \dots, \mu_k^t \in \mathbb{R}^d$ 
3 repeat
4    $t \leftarrow t + 1$ 
5    $C_j \leftarrow \emptyset$  for all  $j = 1, \dots, k$ 
   // Cluster Assignment Step
6   foreach  $\mathbf{x}_j \in \mathbf{D}$  do
7      $j^* \leftarrow \operatorname{argmin}_i \{ \|\mathbf{x}_j - \mu_i^{t-1}\|^2 \}$  // Assign  $\mathbf{x}_j$  to closest centroid
8      $C_{j^*} \leftarrow C_{j^*} \cup \{\mathbf{x}_j\}$ 
   // Centroid Update Step
9   foreach  $i = 1$  to  $k$  do
10     $\mu_i^t \leftarrow \frac{1}{|C_i|} \sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j$ 
11 until  $\sum_{i=1}^k \|\mu_i^t - \mu_i^{t-1}\|^2 \leq \epsilon$ 
```

Ядерный алгоритм К-средних

В К-средних разделяющая граница между кластерами линейна. Ядерный К-средних позволяет извлекать нелинейные границы между кластерами с помощью хитрости с ядром, описанной в главе 5. Таким образом, метод может быть использован для обнаружения невыпуклых кластеров.

В ядерном К-средних основная идея состоит в концептуальном отображении точки данных $\mathbf{x}_i$ во входном пространстве в точку $\phi(\mathbf{x}_i)$ в некотором многомерном пространстве признаков посредством соответствующего нелинейного отображения ϕ . Однако трюк с ядром позволяет нам выполнять кластеризацию в пространстве признаков исключительно в терминах функции ядра $K(\mathbf{x}_i, \mathbf{x}_j)$, которая может быть вычислена во входном пространстве, но соответствует скалярному (или внутреннему) произведению $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ в пространстве признаков.

Предположим пока, что все точки $\mathbf{x}_i \in \mathbf{D}$ сопоставлены с соответствующими отображениями $\phi(\mathbf{x}_i)$ в пространстве признаков. Пусть $\mathbf{K} = \{K(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j=1,\dots,n}$ обозначает симметричную ядерную матрицу $n \times n$, где





$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$. Пусть $\{C_1, \dots, C_k\}$ задает разбиение n точек на k кластеров, и пусть соответствующие средние значения кластеров в пространстве признаков задаются как $\{\boldsymbol{\mu}_1^\phi, \dots, \boldsymbol{\mu}_k^\phi\}$, где

$$\boldsymbol{\mu}_i^\phi = \frac{1}{n_i} \sum_{\mathbf{x}_j \in C_i} \phi(\mathbf{x}_j)$$

среднее кластера C_i в пространстве признаков и $n_i = |C_i|$.

В пространстве признаков ядерная K-средних сумма квадратов отклонений может быть записана как

$$\min_C SSE(C) = \sum_{i=1}^k \sum_{\mathbf{x}_j \in C_i} \|\phi(\mathbf{x}_j) - \boldsymbol{\mu}_i^\phi\|^2$$

Расширяя ядерную SSE в терминах функции ядра, мы получаем

$$\begin{aligned} SSE(C) &= \sum_{i=1}^k \sum_{\mathbf{x}_j \in C_i} \|\phi(\mathbf{x}_j) - \boldsymbol{\mu}_i^\phi\|^2 \\ &= \sum_{i=1}^k \sum_{\mathbf{x}_j \in C_i} \|\phi(\mathbf{x}_j)\|^2 - 2\phi(\mathbf{x}_j)^T \boldsymbol{\mu}_i^\phi + \|\boldsymbol{\mu}_i^\phi\|^2 \\ &= \sum_{i=1}^k \left(\left(\sum_{\mathbf{x}_j \in C_i} \|\phi(\mathbf{x}_j)\|^2 \right) - 2n_i \left(\frac{1}{n_i} \sum_{\mathbf{x}_j \in C_i} \phi(\mathbf{x}_j) \right)^T \boldsymbol{\mu}_i^\phi + n_i \|\boldsymbol{\mu}_i^\phi\|^2 \right) \\ &= \left(\sum_{i=1}^k \sum_{\mathbf{x}_j \in C_i} \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_j) \right) - \left(\sum_{i=1}^k n_i \|\boldsymbol{\mu}_i^\phi\|^2 \right) \\ &= \sum_{i=1}^k \sum_{\mathbf{x}_j \in C_i} K(\mathbf{x}_j, \mathbf{x}_j) - \sum_{i=1}^k \frac{1}{n_i} \sum_{\mathbf{x}_a \in C_i} \sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b) \\ &= \sum_{j=1}^n K(\mathbf{x}_j, \mathbf{x}_j) - \sum_{i=1}^k \frac{1}{n_i} \sum_{\mathbf{x}_a \in C_i} \sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b) \end{aligned} \quad (43)$$

Таким образом, ядерная K-средних SSE целевая функция может быть выражена исключительно в терминах функции ядра. Как и K-средних, для минимизации целевой функции SSE мы применяем жадный итеративный



подход. Основная идея состоит в том, чтобы присвоить каждой точке ближайшее среднее значение в пространстве признаков, что приведет к новой кластеризации, которая, в свою очередь, может быть использована для получения новых оценок для средних значений кластера. Однако основная трудность заключается в том, что мы не можем явно вычислить среднее значение каждого кластера в пространстве признаков. К счастью, явное получение средних кластера не требуется; все операции могут быть выполнены в терминах функции ядра $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$.

Рассмотрим расстояние от точки $\phi(\mathbf{x}_j)$ до среднего значения μ_i^ϕ в пространстве признаков, которое можно вычислить как

$$\begin{aligned} \|\phi(\mathbf{x}_j) - \mu_i^\phi\|^2 &= \|\phi(\mathbf{x}_j)\|^2 - 2\phi(\mathbf{x}_j)^T \mu_i^\phi + \|\mu_i^\phi\|^2 \\ &= \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_j) - \frac{2}{n_i} \sum_{\mathbf{x}_a \in C_i} \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_a) + \frac{1}{n_i^2} \sum_{\mathbf{x}_a \in C_i} \sum_{\mathbf{x}_b \in C_i} \phi(\mathbf{x}_a)^T \phi(\mathbf{x}_b) \\ &= K(\mathbf{x}_j, \mathbf{x}_j) - \frac{2}{n_i} \sum_{\mathbf{x}_a \in C_i} K(\mathbf{x}_a, \mathbf{x}_j) + \frac{1}{n_i^2} \sum_{\mathbf{x}_a \in C_i} \sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b) \end{aligned} \quad (44)$$

Таким образом, расстояние от точки до среднего значения кластера в пространстве признаков можно вычислить, используя только ядерные операции. На этапе назначения кластера ядерного К-средних мы присваиваем точку ближайшему среднему кластеру следующим образом:

$$\begin{aligned} C^*(\mathbf{x}_j) &= \arg \min_i \left\{ \|\phi(\mathbf{x}_j) - \mu_i^\phi\|^2 \right\} \\ &= \arg \min_i \left\{ K(\mathbf{x}_j, \mathbf{x}_j) - \frac{2}{n_i} \sum_{\mathbf{x}_a \in C_i} K(\mathbf{x}_a, \mathbf{x}_j) + \frac{1}{n_i^2} \sum_{\mathbf{x}_a \in C_i} \sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b) \right\} \\ &= \arg \min_i \left\{ \frac{1}{n_i^2} \sum_{\mathbf{x}_a \in C_i} \sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b) - \frac{2}{n_i} \sum_{\mathbf{x}_a \in C_i} K(\mathbf{x}_a, \mathbf{x}_j) \right\} \end{aligned} \quad (45)$$

где мы опускаем член $K(\mathbf{x}_j, \mathbf{x}_j)$, потому что он остается одинаковым для всех k кластеров и не влияет на решение о назначении кластера. Также обратите внимание, что первый член - это просто среднее попарное ядерное значение для кластера C_i , которое не зависит от точки $\mathbf{x}_j$. Фактически, это



квадрат нормы кластерного среднего в пространстве признаков. Второй член в два раза больше среднего значения ядра для точек в C_i относительно x_j .

Алгоритм 3 показывает псевдокод для метода ядерного K-средних. Он начинается с первоначального случайного разбиения точек на k кластеров. Затем он итеративно обновляет назначения кластера, переназначая каждую точку ближайшему среднему значению в пространстве признаков с помощью уравнения (45). Чтобы облегчить вычисление расстояния, он сначала вычисляет среднее значение ядра, то есть квадрат нормы среднего значения кластера, для каждого кластера (для цикла в строке 5). Затем он вычисляет среднее значение ядра для каждой точки x_j с точками в кластере C_i (для цикла в строке 7). На этапе назначения основного кластера эти значения используются для вычисления расстояния x_j от каждого из кластеров C_i и x_j присваиваются ближайшему среднему значению. Эта информация о переназначении используется для повторного разделения точек на новый набор кластеров. То есть все точки x_j , которые ближе к среднему значению C_i , составляют новый кластер для следующей итерации. Этот итерационный процесс повторяется до сходимости.



Алгоритм 3: Ядерный К-средних

KERNEL-KMEANS($\mathbf{K}, k, \epsilon$):

```

1  $t \leftarrow 0$ 
2  $\mathcal{C}^t \leftarrow \{C_1^t, \dots, C_k^t\}$  // Randomly partition points into  $k$  clusters
3 repeat
4    $t \leftarrow t + 1$ 
5   foreach  $C_i \in \mathcal{C}^{t-1}$  do // Compute squared norm of cluster means
6      $\text{sqnorm}_i \leftarrow \frac{1}{n_i^2} \sum_{\mathbf{x}_a \in C_i} \sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b)$ 
7   foreach  $\mathbf{x}_j \in \mathbf{D}$  do // Average kernel value for  $\mathbf{x}_j$  and  $C_i$ 
8     foreach  $C_i \in \mathcal{C}^{t-1}$  do
9        $\text{avg}_{ji} \leftarrow \frac{1}{n_i} \sum_{\mathbf{x}_a \in C_i} K(\mathbf{x}_a, \mathbf{x}_j)$ 
10  // Find closest cluster for each point
11  foreach  $\mathbf{x}_j \in \mathbf{D}$  do
12    foreach  $C_i \in \mathcal{C}^{t-1}$  do
13       $d(\mathbf{x}_j, C_i) \leftarrow \text{sqnorm}_i - 2 \cdot \text{avg}_{ji}$ 
14       $j^* \leftarrow \text{argmin}_i \{d(\mathbf{x}_j, C_i)\}$ 
15       $C_{j^*}^t \leftarrow C_{j^*}^{t-1} \cup \{\mathbf{x}_j\}$  // Cluster reassignment
16   $\mathcal{C}^t \leftarrow \{C_1^t, \dots, C_k^t\}$ 
17 until  $1 - \frac{1}{n} \sum_{i=1}^k |C_i^t \cap C_i^{t-1}| \leq \epsilon$ 

```

Для проверки сходимости мы проверяем, есть ли какие-либо изменения в кластерных назначениях точек. Количество точек, не меняющих кластеры, задается как сумма $\sum_{i=1}^k |C_i^t \cap C_i^{t-1}|$, где t задает текущую итерацию. Доля точек, переназначенных другому кластеру в текущей итерации, задается как

$$\frac{n - \sum_{i=1}^k |C_i^t \cap C_i^{t-1}|}{n} = 1 - \frac{1}{n} \sum_{i=1}^k |C_i^t \cap C_i^{t-1}|$$

Ядерный К-средних останавливается, когда доля точек с новыми назначениями кластера падает ниже некоторого порога $\epsilon \geq 0$. Например, можно выполнять итерацию, пока никакие точки не изменяют кластеры.

Вычислительная сложность

Вычисление среднего значения ядра для каждого кластера C_i занимает время $O(n^2)$ по всем кластерам. Вычисление среднего значения ядра для каждой точки по отношению к каждому из k кластеров также занимает $O(n^2)$ времени. Наконец, вычисление ближайшего среднего для каждой точки и переназначения кластера занимает $O(kn)$ времени. Общая вычислительная





сложность ядерного K-средних составляет $O(tn^2)$, где t - это количество итераций до сходимости. Сложность I/O составляет $O(t)$ проходов по ядерной матрице K .

4.5 Гауссова смесь распределений без максимизации и с максимизацией ожиданий

Метод k-средних является примером «жесткой» кластеризации, где каждая точка может принадлежать только одному кластеру. Теперь мы обобщаем подход для рассмотрения «мягкого» распределения в кластеры, так что у каждой точки есть вероятность принадлежать каждому кластеру.

Пусть D состоит из n точек x_j в d -мерном пространстве $\mathbb{R}^d$. Пусть X_a - случайная величина, соответствующая a -ому атрибуту. Мы также используем X_a для обозначения a -того вектора столбца, соответствующий n выборкам данных из X_a . Пусть $X = (X_1, X_2, \dots, X_d)$ - векторная случайная величина, определенная через d -атрибуты с помощью x_j , являющимися образцами данных из X .

Модель Гауссовой смеси

Мы предполагаем, что каждый кластер характеризуется многомерным нормальным распределением, то есть:

$$f_i(x) = f(x_j | \mu_i, \Sigma_i) = \frac{1}{(2\pi)^{-\frac{d}{2}} |\Sigma_i^{-1}|^{\frac{1}{2}}} \exp \left\{ -\frac{(x_j - \mu_i)^T \Sigma_i^{-1} (x_j - \mu_i)}{2} \right\} \quad (46)$$

где $\mu_i \in \mathbb{R}^d$ - среднее значение кластера и $\Sigma_i \in \mathbb{R}^{d \times d}$ - ковариационная матрица неизвестные параметры. $f_i(x)$ - плотность вероятности x , принадлежности к кластеру C_i . Мы предполагаем, что функция плотности вероятности X задается как модель гауссовой смеси для всех k нормалей кластеров, определяемых как

$$f(x) = \sum_{j=1}^k f_i(x) P(C_i) = \sum_{j=1}^k f(x | \mu, \Sigma) P(C_i) \quad (47)$$

где априорные вероятности $P(C_i)$ называются параметрами смеси, которые должны удовлетворять условию

$$\sum_{i=1}^k P(C_i) = 1$$



Таким образом, модель гауссовой смеси характеризуется средним значением μ_i , ковариационной матрицей Σ_i , и вероятностью смеси $P(C_i)$ для каждого из k нормальных распределений. Мы компактно запишем множество всех параметров модели в виде:

$$\theta = \{\mu_1, \Sigma_1, P(C_1), \dots, \mu_k, \Sigma_k, P(C_k)\}$$

Оценка максимального правдоподобия

С учетом набора данных $\mathbf{D}$, определим правдоподобие θ как условную вероятность данных $\mathbf{D}$ с учетом параметров модели θ , обозначенную как $P(\mathbf{D}|\theta)$. Потому что каждая из n точек x_j считается случайной выборкой из X (т. е. независимой, с распределением как X), правдоподобие θ задается как

$$P(\mathbf{D}|\theta) = \prod_{j=1}^n f(x_j)$$

Целью оценки максимального правдоподобия (MLE) является выбор параметров θ , которые максимизируют вероятность, то есть:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \{P(\mathbf{D}|\theta)\}$$

Максимизация логарифма функции правдоподобия - это типичное решение, поскольку оно превращает произведение по точкам в суммирование, а максимальное значение правдоподобия и логарифмической вероятности совпадают. Т.е. MLE максимизирует

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \{\ln P(\mathbf{D}|\theta)\}$$

где функция логарифмического правдоподобия задается как:

$$\ln P(\mathbf{D}|\theta) = \sum_{j=1}^n \ln f(x_j) = \sum_{j=1}^n \ln \left(\sum_{i=1}^k f(x|\mu, \Sigma) P(C_i) \right) \quad (48)$$

Непосредственная максимизация логарифмической вероятности над θ - весьма трудная задача. Вместо этого мы можем использовать подход максимизации ожидания (ЕМ) для нахождения оценок максимального правдоподобия для параметров θ . ЕМ - это двухэтапный итерационный подход, который начинается с начального предположения для параметров θ . Учитывая текущие оценки для θ , на шаге ожидания ЕМ вычисляет апостериорные вероятности кластера $P(C_i|x_j)$ теорему Байеса:



$$P(C_i | \mathbf{x}_j) = \frac{P(C_i \text{ and } \mathbf{x}_j)}{P(\mathbf{x}_j)} = \frac{P(\mathbf{x}_j | C_i)P(C_i)}{\sum_{a=1}^k P(\mathbf{x}_j | C_a)P(C_a)}$$

Поскольку каждый кластер моделируется как многомерное нормальное распределение [ур-е (46)], вероятность $\mathbf{x}_j$ данного кластера можно получить, рассматривая небольшой интервал $\varepsilon > 0$ с центром в точке $\mathbf{x}_j$, следующим образом:

$$P(\mathbf{x}_j | C_i) \simeq 2\varepsilon \cdot f(\mathbf{x}_j | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = 2\varepsilon \cdot f_i(\mathbf{x}_j)$$

Апостериорная вероятность C_i с $\mathbf{x}_j$:

$$P(C_i | \mathbf{x}_j) = \frac{f_i(\mathbf{x}_j)P(C_i)}{\sum_{a=1}^k f_a(\mathbf{x}_j)P(C_a)} \quad (49)$$

И $P(C_i | \mathbf{x}_j)$ можно рассматривать как вес или вклад точки $\mathbf{x}_j$ в кластеризацию C_i . Далее, на этапе максимизации, используя веса $P(C_i | \mathbf{x}_j)$ ЕМ заново пересчитывает $\boldsymbol{\theta}$, то есть он заново оценивает параметры $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$, и $P(C_i)$ для каждого кластера C_i . Пересчитанное среднее значение задается как средневзвешенное значение всех точек, пересчитанная ковариационная матрица задается как взвешенная ковариация по всем парам измерений, а пересчитанная априорная вероятность для каждого кластера задается как доля весов, которые вносят вклад в этот кластер. Мы начнем с применения алгоритма кластеризации ЕМ для одномерных и общих d -мерных случаев.

ЕМ в одномерном пространстве

Рассмотрим набор данных $\mathbf{D}$, состоящий из одного атрибута X , где каждая точка $x_j \in \mathbb{R}$ ($j = 1, \dots, n$) - случайная выборка из X . Для модели смеси [ур-е 47], мы используем одномерные нормали для каждого кластера:

$$f_i(\mathbf{x}) = f(x_j | \mu_i, \sigma_i^2) = \frac{1}{\sqrt{2\pi}\sigma_j} \exp \left\{ -\frac{(\mathbf{x}_j - \boldsymbol{\mu}_i)^2}{2\sigma_i^2} \right\}$$

с параметрами кластера μ_i , σ_i^2 и $P(C_i)$. Подход ЕМ состоит из трех этапов: инициализации, шага ожидания и шага максимизации.

Инициализация

Для каждого кластера C_i для ($i = 1, 2, \dots, k$), мы можем случайным образом инициализировать параметры кластер μ_i , σ_i^2 и $P(C_i)$. Среднее значение μ_i выбирается равномерно случайным образом из диапазона



возможных значений для X . Обычно предполагается, что начальная дисперсия задается в виде $\sigma_i^2 = 1$. Наконец, предыдущие вероятности кластера инициализируются в $P(C_i) = \frac{1}{k}$, так что каждый кластер имеет равную вероятность.

Шаг ожидания

Предположим, что для каждого из k кластеров мы имеем оценку параметров, а именно: среднее значение μ_i , дисперсия σ_i^2 и априорная вероятность $P(C_i)$. Учитывая эти значения апостериорные вероятности кластеров вычисляются с помощью ур-я (49):

$$P(C_i | \mathbf{x}_j) = \frac{f(x_j | \mu_i, \sigma_i^2) P(C_i)}{\sum_{a=1}^k f(x_j | \mu_a, \sigma_a^2) P(C_a)}$$

Для удобства мы используем обозначение $w_{ij} = P(C_i | x_j)$, рассматривая апостериорные вероятности как вес или вклад точки x_j в кластер C_i . Далее, пусть:

$$\mathbf{w}_i = (w_{i1}, w_{i2}, \dots, w_{in})^T$$

обозначим весовой вектор для кластера C_i через все n точек.

Шаг максимизации

При условии, что все апостериорные значения вероятности или веса $w_{ij} = P(C_i | x_j)$ известны, шаг максимизации, как следует из названия, вычисляет оценки максимального правдоподобия параметров кластера путем пересчета μ_i , σ_i^2 и $P(C_i)$. Пересчитанное значение для среднего значения кластера, μ_i , вычисляется как средневзвешенное значение из всех пунктов:

$$\mu_i = \frac{\sum_{j=1}^n w_{ij} x_j}{\sum_{j=1}^n w_{ij}}$$

В терминах вектора веса $\mathbf{w}_i$ и вектора атрибутов $X = (x_1, x_2, \dots, x_n)^T$ - мы можем переписать как:

$$\mu_i = \frac{\mathbf{w}_i^T X}{\mathbf{w}_i^T \mathbf{1}}$$

Пересчитанное значение кластерной дисперсии вычисляется как взвешенная дисперсия по всем точкам:





$$\sigma_i^2 = \frac{\sum_{j=1}^n w_{ij} (x_j - \mu_i)^2}{\sum_{j=1}^n w_{ij}}$$

Пусть $Z_i = X - \mu_i \mathbf{1} = (x_1 - \mu_i, x_2 - \mu_i, \dots, x_n - \mu_i)^T = (z_{1i}, z_{2i}, \dots, z_{in})^T$

будет отцентрированным вектором атрибутов для кластера C_i , и пусть Z_i^S -квадратный вектор, заданный как $Z_i^S = (z_{i1}^2, z_{i2}^2, \dots, z_{in}^2)^T$. Дисперсия может быть компактно выражена в терминах произведения между вектором веса и квадратом центрированного вектора:

$$\sigma_i^2 = \frac{\mathbf{w}_i^T Z_i^S}{\mathbf{w}_i^T \mathbf{1}}$$

Наконец, априорная вероятность кластера C_i пересчитывается как доля веса, принадлежащего C_i , от общего веса:

$$P(C_i) = \frac{\sum_{j=1}^n w_{ij}}{\sum_{a=1}^k \sum_{j=1}^n w_{aj}} = \frac{\sum_{j=1}^k w_{ij}}{\sum_{j=1}^k 1} = \frac{\sum_{j=1}^k w_{ij}}{n} \quad (50)$$

где мы воспользовались тем фактом, что

$$\sum_{i=1}^n w_{ij} = \sum_{i=1}^n P(C_i | x_j) = 1$$

В векторной нотации априорная вероятность может быть записана как:

$$P(C_i) = \frac{\mathbf{w}_i^T \mathbf{1}}{n}$$

Итерация

Начиная с исходного набора значений параметров кластера μ_i , σ_i^2 и $P(C_i)$ для всех $i = 1, \dots, k$, алгоритм ЕМ применяет шаг ожидания для вычисления весов $w_{ij} = P(C_i | x_j)$. Эти значения затем используются на этапе максимизации для вычисления обновленных параметров кластера μ_i , σ_i^2 и $P(C_i)$. Как шаг ожидания, так и максимизация применяются итеративно до тех пор, пока, например, не произойдет конвергенция, например, пока средние не изменятся очень мало от одной итерации к другой.

ЕМ в d -мерном пространстве

Теперь рассмотрим метод ЭМ в d измерениях, где каждый кластер характеризуется многомерным нормальным распределением [ур-е. (46)], с параметрами μ_i , Σ_i и $P(C_i)$. Таким образом, для каждого кластера C_i , нам нужно оценить d -мерный вектор среднего:





$$\mu_i = (\mu_1, \mu_2, \dots, \mu_n)^T$$

и ковариационной матрицей $d \times d$:

$$\Sigma = \begin{pmatrix} (\sigma_1^i)^2 & \sigma_{12}^i & \dots & \sigma_{1d}^i \\ \sigma_{21}^i & (\sigma_2^i)^2 & \dots & \sigma_{2d}^i \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{d1}^i & \sigma_{d2}^i & \dots & (\sigma_d^i)^2 \end{pmatrix}$$

Поскольку ковариационная матрица симметрична, мы должны оценить: $\binom{d}{2} = \frac{d(d-1)}{2}$ попарных ковариаций и d дисперсии, в общей сложности $\frac{d(d+1)}{2}$ параметры для Σ_i . Такое количество может быть большим для практических целей, потому что у нас может быть недостаточно данных, чтобы надежно оценить все из них. Например, если $d = 100$, то мы должны оценить $100 * \frac{101}{2} = 5050$ параметров! Одним из упрощений является предположение, что все измерения являются независимыми, что приводит к диагональной ковариационной матрице:

$$\Sigma_i = \begin{pmatrix} (\sigma_1^i)^2 & 0 & \dots & 0 \\ 0 & (\sigma_2^i)^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & (\sigma_d^i)^2 \end{pmatrix}$$

При предполагаемой независимости у нас есть только d параметров для оценки диагональной ковариационной матрицы.

Инициализация

Для каждого кластера C_i , где $i = 1, \dots, k$, мы случайным образом инициализируем среднее значение μ_i путем выбора наугад значения μ_{ia} для каждого измерения X_a равномерно распределённого на диапазоне X_a . Ковариационная матрица инициализируется как матрица тождества $d \times d$ $\Sigma_i = I$. Наконец, апостериорные вероятности кластера инициализируются $P(C_i) = 1$, так что каждый кластер имеет равную вероятность.

Шаг ожидания

На шаге ожидания мы вычисляем апостериорную вероятность кластера C_i для точки x_j , используя ур-е. (49), при $i = 1, \dots, k$ и $j = 1, \dots, n$. Как и прежде, мы используем сокращенные обозначение $w_{ij} = P(C_i|x_j)$, чтобы



обозначить тот факт, что $P(C_i|x_j)$ можно рассматривать как вес вклада точки x_j в кластер C_i , и мы используем обозначение $w_i = (w_1, w_2, \dots, w_n)^T$ чтобы определить весовой вектор для кластера C , через все n точек.

Шаг максимизации

При заданных весах w_{ij} , на этапе максимизации мы повторно вычисляем, μ_i , Σ_i и $P(C_i)$. Среднее значение μ_i для кластера C_i можно оценить как:

$$\mu_i = \frac{\sum_{j=1}^n w_{ij} x_j}{\sum_{j=1}^n w_{ij}} \quad (51)$$

которое может быть компактно выражено в матричном виде как:

$$\mu_i = \frac{\mathbf{D}^T \mathbf{w}_i}{\mathbf{w}_i^T \mathbf{1}}$$

Пусть $Z_i = \mathbf{D} - \mathbf{1}\mu_i^T$ - центрированная матрица данных для кластера C_i . Пусть $\mathbf{z}_{ji} = x_j - \mu_i \in \mathbb{R}^d$ - j -я центрированную точку в Z_i . Мы можем выразить Σ_i :

$$\Sigma_i = \frac{\sum_{j=1}^n w_{ij} \mathbf{z}_{ji} \mathbf{z}_{ji}^T}{\mathbf{w}_i^T \mathbf{1}} \quad (52)$$

Учитывая попарное представление атрибутов, ковариация между измерениями X_a и X_b оценивается как:

$$\sigma_{ab}^i = \frac{\sum_{j=1}^n w_{ij} (x_{ja} - \mu_{ia})(x_{jb} - \mu_{ib})}{\sum_{j=1}^n w_{ij}}$$

где x_{ja} и μ_{ia} - значения a -го измерения для x_j и μ_i , соответственно.

Наконец, априорная вероятность $P(C_i)$ для каждого кластера - это то же самое, что и одномерный случай [ур-е. (50)], приведенное в виде:

$$P(C_i) = \frac{\sum_{j=1}^k w_{ij}}{n} = \frac{\mathbf{w}_i^T \mathbf{1}}{n} \quad (53)$$

Алгоритм кластеризации ЭМ

Псевдокод для многомерного алгоритма кластеризации ЭМ приведен в алгоритме 4. После инициализации μ_i , Σ_i и $P(C_i)$ для всех $i = 1, \dots, k$, шаги ожидания и максимизации повторяются до тех пор, пока не произойдет конвергенция. Для проверки конвергенции мы проверяем, что $\sum_i \|\mu_i^t - \mu_i^{t-1}\|^2 \leq \varepsilon$, где $\varepsilon > 0$ - порог сходимости, и t обозначает итерацию. Другими





словами, итерационный процесс продолжается до тех пор, пока изменение в среднем кластера не станет очень маленьким.

Алгоритм 4: Алгоритм максимизации ожидания (ЕМ)

EXPECTATION-MAXIMIZATION ($\mathbf{D}, k, \epsilon$):

```

1  $t \leftarrow 0$ 
  // Initialization
2 Randomly initialize  $\mu_1^t, \dots, \mu_k^t$ 
3  $\Sigma_i^t \leftarrow \mathbf{I}, \forall i = 1, \dots, k$ 
4  $P^t(C_i) \leftarrow \frac{1}{k}, \forall i = 1, \dots, k$ 
5 repeat
6    $t \leftarrow t + 1$ 
   // Expectation Step
7   for  $i = 1, \dots, k$  and  $j = 1, \dots, n$  do
8      $w_{ij} \leftarrow \frac{f(\mathbf{x}_j | \mu_i, \Sigma_i) \cdot P(C_i)}{\sum_{a=1}^k f(\mathbf{x}_j | \mu_a, \Sigma_a) \cdot P(C_a)}$  // posterior probability  $P^t(C_i | \mathbf{x}_j)$ 
   // Maximization Step
9   for  $i = 1, \dots, k$  do
10     $\mu_i^t \leftarrow \frac{\sum_{j=1}^n w_{ij} \cdot \mathbf{x}_j}{\sum_{j=1}^n w_{ij}}$  // re-estimate mean
11     $\Sigma_i^t \leftarrow \frac{\sum_{j=1}^n w_{ij} (\mathbf{x}_j - \mu_i) (\mathbf{x}_j - \mu_i)^T}{\sum_{j=1}^n w_{ij}}$  // re-estimate covariance matrix
12     $P^t(C_i) \leftarrow \frac{\sum_{j=1}^n w_{ij}}{n}$  // re-estimate priors
13 until  $\sum_{i=1}^k \|\mu_i^t - \mu_i^{t-1}\|^2 \leq \epsilon$ 
```

Вычислительная сложность

Для шага ожидания, чтобы вычислить апостериорные вероятности кластера, нам нужно инвертировать Σ_i и вычислить его определитель $|\Sigma_i|$, который занимает $O(d^3)$. Для k кластеров время равно $O(kd^3)$. На шаге ожидания, оценка плотности $f(\mathbf{x}_j | \mu_i, \Sigma_i)$ требуется $O(d^2)$, для n точек и k кластеров это потребует $O(knd^2)$. На шаге максимизации, время обновления для Σ_i для k кластеров по требует $O(knd^2)$. Таким образом, вычислительная сложность метода ЕМ, составляет $O(t(kd^3 + knd^2))$ где t -число итераций. Если мы используем диагональную ковариационную матрицу, то обратная матрица и определитель могут быть вычислены за $O(d)$. Вычисление плотности в точке занимает $O(d)$ времени, так что время для шага ожидания равно $O(knd)$. Шаг максимизации также занимает $O(knd)$ для повторной оценки Σ_i . Таким образом, общее время для диагональной ковариационной матрицы равно $O(tknd)$. Сложность ввода-вывода для алгоритма ЕМ - это



$O(t)$, полное сканирование базы данных, потому что мы читаем весь набор точек в каждой итерации.

5 ФАКТОРНЫЙ И КОМПОНЕНТНЫЙ АНАЛИЗ

Компонентный анализ (components analysis) - многомерный статистический метод снижения размерности, применяемый для изучения взаимосвязей между значениями количественных переменных. Задача компонентного анализа состоит в преобразовании исходной системы взаимосвязанных переменных в новую систему некоррелированных обобщенных показателей или ортогональных показателей. Новые некоррелированные показатели называются компонентами.

Алгоритм компонентного анализа состоит в последовательном извлечении компонент. Для единственности решения в методе главных компонент компоненты должны быть упорядочены по убыванию доли объясняемой суммарной дисперсии исходных переменных. Первая компонента характеризует наибольшую долю вариации исходных переменных, вторая компонента объясняет наибольшую долю дисперсии, не объясняемой первой компонентой и т.д. В результате компонентного анализа число полученных некоррелированных компонент совпадает с числом исходных переменных. Т.е. классический компонентный анализ сохраняет размерность пространства переменных.

Каждой извлеченной компоненте соответствует характеристика, называемая собственным значением. Собственное значение показывает часть вариации исходных переменных, объясняемую компонентой. В компонентном анализе, если используется корреляционная матрица, каждая переменная стандартизована и ее дисперсия равна 1. Следовательно, если число исходных переменных k , то суммарная дисперсия равна k . Компонентный анализ сохраняет всю суммарную дисперсию, поэтому сумма всех собственных значений равна числу исходных переменных.

На основе полученных собственных значений рассчитывается матрица нагрузок и дается интерпретация компонент. На основе полученной матрицы нагрузок в компонентном анализе может быть произведено вращение



факторов для получения простой структуры матрицы нагрузок. Классический компонентный анализ, в отличие от факторного анализа, не предполагает вращение матрицы нагрузок. Но вращение настолько полезная возможность методов снижения размерности, что алгоритмы вращения были разработаны для компонентного анализа.

На основе окончательной матрицы нагрузок рассчитывают индивидуальные значения главных компонент для каждого объекта наблюдения. Индивидуальные значения главных компонент на объектах представляют собой линейную комбинацию исходных переменных для каждого фактора. На практике для интерпретации и дальнейшего анализа используют компоненты, удовлетворяющие следующим условиям:

- их собственные значения должны быть больше 1 - это означает, что компонента более информативна, чем стандартизированная переменная;
- компонента должна иметь хотя бы одну нагрузку больше критического значения - это означает, что компонента тесно связана, по крайней мере, с одной исходной переменной.

Отобранные для дальнейшего анализа компоненты называют главными компонентами. Отсюда название метода - метод главных компонент. Индивидуальные значения главных компонент могут быть использованы для дальнейшего статистического анализа, например:

- построения уравнения регрессии на главные компоненты;
- кластеризации наблюдений по главным компонентам.

Очень часто под термином «факторный анализ» понимают методы факторного и компонентного анализов. На самом деле, компонентный анализ представляет собой самостоятельный метод снижения размерности.

Факторный и компонентный анализы - оба метода снижения размерности. Оба метода решают в принципе одну и ту же задачу и поэтому результаты похожи. В матрицах нагрузок первый общий фактор и первая главная компонента, как правило, совпадают. Второй фактор может уже существенно отличаться от второй главной компоненты и т.д. Иногда даже





число общих факторов может существенно отличаться от числа весомых главных компонент.

Предпосылки методов факторного и компонентного анализа различаются. Цель компонентного анализа - объяснить всю корреляцию между переменными и всю суммарную дисперсию исходных переменных. Число первоначально извлеченных компонент совпадает с числом исходных переменных. Хотя в дальнейшем анализе используются только главные компоненты.

В факторный анализ с самого начала предполагает, что число извлеченных факторов будет существенно меньше числа первоначальных переменных. В факторном анализе извлеченные новые переменные-факторы в принципе не могут объяснить полностью суммарную дисперсию исходных переменных и их корреляции.

Компонентный анализ используют чаще как метод избавления от мультиколлинеарности объясняющих переменных в регрессионном анализе. Главные компоненты всегда не коррелированы между собой. Общие факторы могут быть коррелированными или слабо коррелированными между собой.

В факторном анализе делается больший акцент на интерпретации факторов. В компонентном анализе делается больший акцент на сокращение размерности пространства за счет некоррелированных переменных.

Факторный анализ (factor analysis) - многомерный статистический метод, применяемый для изучения взаимосвязей между значениями количественных переменных. Основная идея факторного анализа заключается в том, что имеющиеся зависимости между большим числом исходных наблюдаемых переменных определяются существованием гораздо меньшего числа скрытых или латентных переменных, называемых факторами.

Главными целями факторного анализа являются: сокращение числа переменных и определения структуры взаимосвязей между переменными. Поэтому факторный анализ используется или как метод сокращения данных или как метод классификации. Факторный анализ позволяет исследователю



описать объект измерения с одной стороны всесторонне, учитывая множество исходных тесно взаимосвязанных между собой переменных, а с другой стороны компактно с помощью небольшого числа переменных.

Методы факторного анализа доступны во всех профессиональных статистических пакетах обработки данных: SPSS, SAS, R, Statistica и др.

5.1 Метод главных компонент

Метод главных компонент (РСА) - это метод, который ищет r -мерный базис, наилучшим образом описывающий дисперсию данных. Направление с наибольшей проецируемой дисперсией называется первым главным компонентом. Ортогональное направление, которое захватывает вторую по величине проецируемую дисперсию, называется вторым главным компонентом и т.д. Как мы увидим, направление, которое максимизирует дисперсию, также минимизирует среднеквадратичную ошибку.

Наилучшая линейная аппроксимация

Мы начнем с $r = 1$, т.е. с одномерного подпространства или прямой $\mathbf{u}$, которая наилучшим образом аппроксимирует $\mathbf{D}$, с точки зрения дисперсии проецируемых точек. Это приведет к общей методике РСА для наилучшего $1 \leq r \leq d$ размерного базиса для $\mathbf{D}$.

Без потери общности, мы предполагаем, что $\mathbf{u}$ имеет величину $\|\mathbf{u}\|^2 = \mathbf{u}^T \mathbf{u} = 1$; в противном случае можно продолжать увеличивать проецируемую дисперсию, просто увеличивая величину $\mathbf{u}$. Мы также предполагаем, что данные центрированы так, что среднее значение $\boldsymbol{\mu} = \mathbf{0}$.

Проекция x_i на вектор $\mathbf{u}$ берется как

$$x'_i = \left(\frac{\mathbf{u}^T \mathbf{x}_i}{\mathbf{u}^T \mathbf{u}} \right) \mathbf{u} = (\mathbf{u}^T \mathbf{x}_i) \mathbf{u} = a_i \mathbf{u}$$

где скаляр

$$a_i = \mathbf{u}^T \mathbf{x}_i$$

дает координату x'_i вдоль прямой $\mathbf{u}$. Заметим, что поскольку средняя точка $\boldsymbol{\mu} = \mathbf{0}$, ее координата вдоль $\mathbf{u}$ равна $\mu_u = 0$.

Необходимо выбрать направление $\mathbf{u}$ так, чтобы дисперсия проецируемых точек была максимальной. Проецируемая дисперсия вдоль $\mathbf{u}$ берется как:





$$\begin{aligned}
\sigma_u^2 &= \frac{1}{n} \sum_{i=1}^n (a_i - \mu_u)^2 \\
&= \frac{1}{n} \sum_{i=1}^n (\mathbf{u}^T \mathbf{x}_i)^2 \\
&= \frac{1}{n} \sum_{i=1}^n \mathbf{u}^T (\mathbf{x}_i \mathbf{x}_i^T) \mathbf{u} \\
&= \mathbf{u}^T \left(\frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^T \right) \mathbf{u} \\
&= \mathbf{u}^T \Sigma \mathbf{u}
\end{aligned} \tag{54}$$

где Σ - это ковариационная матрица для центрированных данных $\mathbf{D}$.

Чтобы максимизировать проецируемую дисперсию, мы должны решить задачу ограниченной оптимизации, а именно, чтобы максимизировать σ_u^2 с учетом ограничения $\mathbf{u}^T \mathbf{u} = 1$. Это может быть решено путем введения множителя Лагранжа α для ограничения, чтобы получить задачу безусловной максимизации.

$$\max_{\mathbf{u}} J(\mathbf{u}) = \mathbf{u}^T \Sigma \mathbf{u} - \alpha (\mathbf{u}^T \mathbf{u} - 1) \tag{55}$$

Приравнивая производную $J(\mathbf{u})$ по $\mathbf{u}$ к нулевому вектору, получаем

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{u}} J(\mathbf{u}) &= \mathbf{0} \\
\frac{\partial}{\partial \mathbf{u}} (\mathbf{u}^T \Sigma \mathbf{u} - \alpha (\mathbf{u}^T \mathbf{u} - 1)) &= \mathbf{0} \\
2\Sigma \mathbf{u} - 2\alpha \mathbf{u} &= \mathbf{0} \\
\Sigma \mathbf{u} &= \alpha \mathbf{u}
\end{aligned} \tag{56}$$

Это означает, что α является собственным значением ковариационной матрицы Σ , с соответствующим собственным вектором $\mathbf{u}$. Далее, взяв скалярное произведение с $\mathbf{u}$ по обе стороны от уравнения (56) получаем

$$\mathbf{u}^T \Sigma \mathbf{u} = \mathbf{u}^T \alpha \mathbf{u}$$

Из уравнения (54) имеем

$$\begin{aligned}
\sigma_u^2 &= \alpha \mathbf{u}^T \mathbf{u} \\
\text{или } \sigma_u^2 &= \alpha
\end{aligned} \tag{57}$$



Таким образом, чтобы максимизировать проецируемую дисперсию σ_u^2 , необходимо выбрать наибольшее собственное значение матрицы Σ . Другими словами, доминирующий собственный вектор $\mathbf{u}_1$ задает направление наибольшей дисперсии, также называемое первым главным компонентом, т.е. $\mathbf{u} = \mathbf{u}_1$. Кроме того, наибольшее собственное значение λ_1 определяет проецируемую дисперсию, т.е. $\sigma_u^2 = \alpha = \lambda_1$.

Минимальная квадратичная ошибка

Теперь покажем, что направление, которое максимизирует проецируемую дисперсию, также минимизирует среднеквадратичную ошибку. Как и раньше, предположим, что набор данных $\mathbf{D}$, был центрирован путем вычитания среднего значения из каждой точки. Для точки $\mathbf{x}_i \in \mathbf{D}$ пусть $\mathbf{x}'_i$ обозначает ее проекцию вдоль направления $\mathbf{u}$, и пусть $\boldsymbol{\epsilon}_i = \mathbf{x}_i - \mathbf{x}'_i$ обозначает вектор ошибки. Среднеквадратичная ошибка (MSE) условия оптимизации определяется как

$$MSE(\mathbf{u}) = \frac{1}{n} \sum_{i=1}^n \|\boldsymbol{\epsilon}_i\|^2 \quad (58)$$

$$\begin{aligned} &= \frac{1}{n} \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{x}'_i\|^2 \\ &= \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \mathbf{x}'_i)^T (\mathbf{x}_i - \mathbf{x}'_i) \\ &= \frac{1}{n} \sum_{i=1}^n (\|\mathbf{x}_i\|^2 - 2\mathbf{x}_i^T \mathbf{x}'_i + (\mathbf{x}'_i)^T \mathbf{x}'_i) \end{aligned} \quad (59)$$

Заметив, что $\mathbf{x}'_i = (\mathbf{u}^T \mathbf{x}_i) \mathbf{u}$, имеем

$$\begin{aligned} &= \frac{1}{n} \sum_{i=1}^n (\|\mathbf{x}_i\|^2 - 2\mathbf{x}_i^T (\mathbf{u}^T \mathbf{x}_i) \mathbf{u} + ((\mathbf{u}^T \mathbf{x}_i) \mathbf{u})^T (\mathbf{u}^T \mathbf{x}_i) \mathbf{u}) \\ &= \frac{1}{n} \sum_{i=1}^n (\|\mathbf{x}_i\|^2 - 2(\mathbf{u}^T \mathbf{x}_i)(\mathbf{x}_i^T \mathbf{u}) + (\mathbf{u}^T \mathbf{x}_i)(\mathbf{x}_i^T \mathbf{u}) \mathbf{u}^T \mathbf{u}) \\ &= \frac{1}{n} \sum_{i=1}^n (\|\mathbf{x}_i\|^2 - (\mathbf{u}^T \mathbf{x}_i)(\mathbf{x}_i^T \mathbf{u})) \end{aligned}$$





$$\begin{aligned}
&= \frac{1}{n} \sum_{i=1}^n \|x_i\|^2 - \frac{1}{n} \sum_{i=1}^n \mathbf{u}^T (x_i x_i^T) \mathbf{u} \\
&= \frac{1}{n} \sum_{i=1}^n \|x_i\|^2 - \mathbf{u}^T \left(\frac{1}{n} \sum_{i=1}^n x_i x_i^T \right) \mathbf{u} \\
&= \sum_{i=1}^n \frac{\|x_i\|^2}{n} - \mathbf{u}^T \Sigma \mathbf{u}
\end{aligned} \tag{60}$$

Заметим, что общая дисперсия центрированных данных (т.е. $\mu = \mathbf{0}$) получается как

$$\text{var}(\mathbf{D}) = \frac{1}{n} \sum_{i=1}^n \|x_i - \mathbf{0}\|^2 = \frac{1}{n} \sum_{i=1}^n \|x_i\|^2$$

Далее, имеем

$$\text{var}(\mathbf{D}) = \text{tr}(\Sigma) = \sum_{i=1}^d \sigma_i^2$$

Таким образом, можно переписать уравнение (60) как

$$MSE(\mathbf{u}) = \text{var}(\mathbf{D}) - \mathbf{u}^T \Sigma \mathbf{u} = \sum_{i=1}^d \sigma_i^2 - \mathbf{u}^T \Sigma \mathbf{u}$$

Поскольку первый член $\text{var}(\mathbf{D})$ является константой для данного набора данных $\mathbf{D}$, вектор $\mathbf{u}$, который минимизирует $MSE(\mathbf{u})$, максимизирует второй член, проецируемую дисперсию $\mathbf{u}^T \Sigma \mathbf{u}$. Поскольку мы знаем, что $\mathbf{u}_1$, доминирующий собственный вектор матрицы Σ , максимизирует проецируемую дисперсию, имеем

$$MSE(\mathbf{u}_1) = \text{var}(\mathbf{D}) - \mathbf{u}_1^T \Sigma \mathbf{u}_1 = \text{var}(\mathbf{D}) - \mathbf{u}_1^T \lambda_1 \mathbf{u}_1 = \text{var}(\mathbf{D}) - \lambda_1 \tag{61}$$

Таким образом, главный компонент $\mathbf{u}_1$, являющийся направлением, которое максимизирует проецируемую дисперсию, также является направлением, которое минимизирует среднеквадратичную ошибку MSE .

Наилучшая двумерная аппроксимация

Теперь нас интересует наилучшее двумерное приближение к $\mathbf{D}$. Как и раньше, предположим, что $\mathbf{D}$ уже центрирован, т.ч. $\mu = \mathbf{0}$. Мы уже вычислили направление с наибольшей дисперсией, а именно $\mathbf{u}_1$, которое



является собственным вектором с соответствующим наибольшим собственным значением λ_1 матрицы Σ . Теперь мы хотим найти другое направление $\mathbf{v}$, которое также максимизирует проецируемую дисперсию, но ортогонально $\mathbf{u}_1$. Согласно формуле (3.9) проецируемая на $\mathbf{v}$ дисперсия определяется как

$$\sigma_v^2 = \mathbf{v}^T \Sigma \mathbf{v}$$

Далее мы потребуем, чтобы $\mathbf{v}$ был единичным вектором, ортогональным $\mathbf{u}_1$, т.е.

$$\mathbf{v}^T \mathbf{u}_1 = 0$$

$$\mathbf{v}^T \mathbf{v} = 1$$

Тогда условие оптимизации определяется как

$$\max_{\mathbf{v}} J(\mathbf{v}) = \mathbf{v}^T \Sigma \mathbf{v} - \alpha(\mathbf{v}^T \mathbf{v} - 1) - \beta(\mathbf{v}^T \mathbf{u}_1 - 0) \quad (62)$$

Взяв производную $J(\mathbf{v})$ по $\mathbf{v}$ и приравняв ее к нулевому вектору, получаем

$$2\Sigma\mathbf{v} - 2\alpha\mathbf{v} - \beta\mathbf{u}_1 = \mathbf{0} \quad (63)$$

Если умножить левую часть на $\mathbf{u}_1^T$, то получим

$$2\mathbf{u}_1^T \Sigma \mathbf{v} - 2\alpha \mathbf{u}_1^T \mathbf{v} - \beta \mathbf{u}_1^T \mathbf{u}_1 = 0$$

$2\mathbf{v}^T \Sigma \mathbf{u}_1 - \beta = 0$, откуда следует, что

$$\beta = 2\mathbf{v}^T \lambda_1 \mathbf{u}_1 = 2\lambda_1 \mathbf{v}^T \mathbf{u}_1 = 0$$

При выводе выше мы использовали тот факт, что $\mathbf{u}_1^T \Sigma \mathbf{v} = \mathbf{v}^T \Sigma \mathbf{u}_1$, и что $\mathbf{v}$ ортогонален $\mathbf{u}_1$. Подставляя $\beta = 0$ в уравнение (3.18), получаем

$$2\Sigma\mathbf{v} - 2\alpha\mathbf{v} = \mathbf{0}$$

$$\Sigma\mathbf{v} = \alpha\mathbf{v}$$

Это означает, что $\mathbf{v}$ - это еще один собственный вектор Σ . Так же, как в уравнении (57), имеем $\sigma_v^2 = \alpha$. Чтобы максимизировать дисперсию по $\mathbf{v}$, мы должны выбрать $\alpha = \lambda_2$, второе по величине собственное значение Σ , причем второй главный компонент задается соответствующим собственным вектором, т.е. $\mathbf{v} = \mathbf{u}_2$.

Общая проецируемая дисперсия

Пусть $\mathbf{U}_2$ - матрица, столбцы которой соответствуют двум главным компонентам



$$U_2 = \begin{pmatrix} | & | \\ u_1 & u_2 \\ | & | \end{pmatrix}$$

Для точки $x_i \in D$ координаты в двумерном подпространстве, натянутом на u_1 и u_2 , могут быть вычислены по формуле (51) следующим образом

$$a_i = U_2^T x_i$$

Предположим, что каждая точка $x_i \in \mathbb{R}^d$ в D была спроецирована для получения ее координат $a_i \in \mathbb{R}^d$, что дало новый набор данных A . Кроме того, поскольку предполагается, что D центрирован ($\mu = 0$), координаты спроецированного среднего также равны нулю, т.к. $U_2 \mu = U_2^T 0 = 0$.

Общая дисперсия для A определяется как

$$\begin{aligned} \text{var}(A) &= \frac{1}{n} \sum_{i=1}^n \|a_i - 0\|^2 \\ &= \frac{1}{n} \sum_{i=1}^n (U_2^T x_i)^T (U_2^T x_i) \\ &= \frac{1}{n} \sum_{i=1}^n x_i^T (U_2 U_2^T) x_i \\ &= \frac{1}{n} \sum_{i=1}^n x_i^T P_2 x_i \end{aligned} \quad (64)$$

Где P_2 - матрица ортогональной проекции [уравнение (53)], которая определяется как

$$P_2 = U_2 U_2^T = u_1 u_1^T + u_2 u_2^T$$

Подставляя это в (3.19), проецируемая общая дисперсия определяется как

$$\begin{aligned} \text{var}(A) &= \frac{1}{n} \sum_{i=1}^n x_i^T P_2 x_i \\ &= \frac{1}{n} \sum_{i=1}^n x_i^T (u_1 u_1^T + u_2 u_2^T) x_i \\ &= \frac{1}{n} \sum_{i=1}^n (u_1^T x_i)(x_i^T u_1) + \frac{1}{n} \sum_{i=1}^n (u_2^T x_i)(x_i^T u_2) \end{aligned} \quad (65)$$





$$= \mathbf{u}_1^T \Sigma \mathbf{u}_1 + \mathbf{u}_2^T \Sigma \mathbf{u}_2 \quad (66)$$

Поскольку $\mathbf{u}_1$ и $\mathbf{u}_2$ - собственные вектора Σ , имеем $\Sigma \mathbf{u}_1 = \lambda_1 \mathbf{u}_1$ и $\Sigma \mathbf{u}_2 = \lambda_2 \mathbf{u}_2$, т.е.

$$\text{var}(\mathbf{A}) = \mathbf{u}_1^T \Sigma \mathbf{u}_1 + \mathbf{u}_2^T \Sigma \mathbf{u}_2 = \mathbf{u}_1^T \lambda_1 \mathbf{u}_1 + \mathbf{u}_2^T \lambda_2 \mathbf{u}_2 = \lambda_1 + \lambda_2 \quad (67)$$

Таким образом, сумма собственных значений представляет собой общую дисперсию проецируемых точек, а первые два главных компонента максимизируют эту дисперсию.

Среднеквадратичная ошибка

Теперь мы продемонстрируем, что первые два главных компонента также минимизируют среднеквадратичную ошибку. Среднеквадратичная ошибка задается как

$$\begin{aligned} \text{MSE} &= \frac{1}{n} \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{x}'_i\|^2 \\ &= \frac{1}{n} \sum_{i=1}^n (\|\mathbf{x}_i\|^2 - 2\mathbf{x}_i^T \mathbf{x}'_i + (\mathbf{x}'_i)^T \mathbf{x}'_i), \text{ используя (3.14)} \\ &= \text{var}(\mathbf{D}) + \frac{1}{n} \sum_{i=1}^n (-2\mathbf{x}_i^T \mathbf{P}_2 \mathbf{x}_i + (\mathbf{P}_2 \mathbf{x}_i)^T \mathbf{P}_2 \mathbf{x}_i), \text{ используя (52), что } \mathbf{x}'_i = \mathbf{P}_2 \mathbf{x}_i \\ &= \text{var}(\mathbf{D}) - \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i^T \mathbf{P}_2 \mathbf{x}_i) \\ &= \text{var}(\mathbf{D}) - \text{var}(\mathbf{A}), \text{ используя (3.20)} \end{aligned} \quad (68)$$

Таким образом, MSE минимизируется именно тогда, когда общая проецируемая дисперсия $\text{var}(\mathbf{A})$ максимальна. Из (3.22) имеем

$$\text{MSE} = \text{var}(\mathbf{D}) - \lambda_1 - \lambda_2$$

Наилучшая r-мерная аппроксимация

Теперь нас интересует наилучшая r-мерная аппроксимация к $\mathbf{D}$, где $2 < r \leq d$. Предположим, что мы уже вычислили первые $j-1$ главных компонент или собственных векторов, $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_{j-1}$, соответствующих $j-1$ наибольшим собственным значениям матрицы Σ , для $1 \leq j \leq r$. Чтобы вычислить j-й новый базисный вектор $\mathbf{v}$, мы должны убедиться, что он нормализован до единичной длины, т.е. $\mathbf{v}^T \mathbf{v} = 1$, и ортогонален всем



предыдущим компонентам $\mathbf{u}_i$, т.е. $\mathbf{u}_i^T \mathbf{v} = 0$, для $1 \leq i < j$. Как и до этого, проецируемая дисперсия по $\mathbf{v}$ задается как

$$\sigma_v^2 = \mathbf{v}^T \Sigma \mathbf{v}$$

В сочетании с ограничениями на $\mathbf{v}$ это приводит к следующей задаче максимизации с множителями Лагранжа:

$$\max_{\mathbf{v}} J(\mathbf{v}) = \mathbf{v}^T \Sigma \mathbf{v} - \alpha(\mathbf{v}^T \mathbf{v} - 1) - \sum_{i=1}^{j-1} \beta_i(\mathbf{u}_i^T \mathbf{v} - 0)$$

Взяв производную $J(\mathbf{v})$ по $\mathbf{v}$ и приравняв ее к нулевому вектору, получаем

$$2\Sigma\mathbf{v} - 2\alpha\mathbf{v} - \sum_{i=1}^{j-1} \beta_i \mathbf{u}_i = \mathbf{0} \quad (69)$$

Если умножить левую часть на $\mathbf{u}_k^T$ для $1 \leq k < j$, то получим

$$\begin{aligned} 2\mathbf{u}_k^T \Sigma \mathbf{v} - 2\alpha \mathbf{u}_k^T \mathbf{v} - \beta_k \mathbf{u}_k^T \mathbf{u}_k - \sum_{\substack{i=1 \\ i \neq k}}^{j-1} \beta_i \mathbf{u}_k^T \mathbf{u}_i &= 0 \\ 2\mathbf{v}^T \Sigma \mathbf{u}_k - \beta_k &= 0 \\ \beta_k = 2\mathbf{v}^T \lambda_k \mathbf{u}_k = 2\lambda_k \mathbf{v}^T \mathbf{u}_k &= 0 \end{aligned}$$

где мы использовали тот факт, что $\Sigma \mathbf{u}_k = \lambda_k \mathbf{u}_k$, т.к. $\mathbf{u}_k$ является собственным вектором, которому соответствует k -е наибольшее собственное значение λ_k матрицы Σ . Таким образом, мы находим, что $\beta_i = 0$ для всех $i < j$ в уравнении (69), откуда следует

$$\Sigma \mathbf{v} = \alpha \mathbf{v}$$

Чтобы максимизировать дисперсию по $\mathbf{v}$, мы устанавливаем $\alpha = \lambda_j$, j -е наибольшее собственное значение матрицы Σ , где $\mathbf{v} = \mathbf{u}_j$ задает j -ю главную компоненту.

Таким образом, чтобы найти наилучшее r -мерное приближение к $\mathbf{D}$, мы вычисляем собственные значения матрицы Σ . Поскольку Σ является положительно полуопределенной, все ее собственные числа должны быть неотрицательными, и мы можем отсортировать их в порядке убывания следующим образом:

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_r \geq \lambda_{r+1} \dots \geq \lambda_d \geq 0$$





Затем мы выбираем r наибольших собственных значений и соответствующие им собственные векторы, чтобы сформировать лучшее r -мерное приближение.

Общая проецируемая дисперсия

Пусть U_r - r -мерная матрица базисных векторов

$$U_r = \begin{pmatrix} | & | & \cdot & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_r \\ | & | & \cdot & | \end{pmatrix}$$

с матрицей проекции, заданной как

$$P_r = U_r U_r^T = \sum_{i=1}^r \mathbf{u}_i \mathbf{u}_i^T$$

Пусть A обозначает набор данных, сформированный координатами проецируемых точек в r -мерном подпространстве, т.е. $\mathbf{a}_i = U_r^T \mathbf{x}_i$, и пусть $\mathbf{x}'_i = P_r \mathbf{x}_i$ обозначает спроецированную точку в исходном d -мерном пространстве. Из (64), (66) и (67) проецируемая дисперсия определяется как

$$\text{var}(A) = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i^T P_r \mathbf{x}_i = \sum_{i=1}^r \mathbf{u}_i^T \Sigma \mathbf{u}_i = \sum_{i=1}^r \lambda_i$$

Таким образом, общая проецируемая дисперсия - это просто сумма r наибольших собственных значений матрицы Σ .

Среднеквадратичная ошибка

На основании вывода для уравнения (68), среднеквадратичная ошибка в r -измерениях может быть записана как

$$\begin{aligned} \text{MSE} &= \frac{1}{n} \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{x}'_i\|^2 \\ &= \text{var}(\mathbf{D}) - \text{var}(A) \\ &= \text{var}(\mathbf{D}) - \sum_{i=1}^r \mathbf{u}_i^T \Sigma \mathbf{u}_i \\ &= \text{var}(\mathbf{D}) - \sum_{i=1}^r \lambda_i \end{aligned}$$

Первые r главных компонент максимизируют проецируемую дисперсию $\text{var}(A)$ и, таким образом, они также минимизируют MSE.





Общая дисперсия

Обратите внимание, что общая дисперсия $\mathbf{D}$ инвариантна к изменению базисных векторов. Следовательно, имеем следующее тождество:

$$\text{var}(\mathbf{D}) = \sum_{i=1}^d \sigma_i^2 = \sum_{i=1}^d \lambda_i$$

Выбор размерности

Часто мы можем не знать, сколько измерений r использовать для хорошей аппроксимации. Одним из критериев выбора r является вычисление доли общей дисперсии, охваченной первыми r главными компонентами

$$f(r) = \frac{\lambda_1 + \lambda_2 + \dots + \lambda_r}{\lambda_1 + \lambda_2 + \dots + \lambda_d} = \frac{\sum_{i=1}^r \lambda_i}{\sum_{i=1}^d \lambda_i} = \frac{\sum_{i=1}^r \lambda_i}{\text{var}(\mathbf{D})} \quad (70)$$

Учитывая определенный желаемый порог дисперсии, скажем α , начиная с первого главного компонента, мы продолжаем добавлять дополнительные компоненты и останавливаемся на наименьшем значении r , для которого $f(r) \geq \alpha$. Другими словами, мы выбираем наименьшее количество измерений так, что подпространство, охватываемое этими r измерениями, захватывает по крайней мере долю α общей дисперсии. На практике α обычно устанавливается равным 0.9 или выше, т.ч. уменьшенный набор данных фиксирует не менее 90% общей дисперсии.

Алгоритм 5 демонстрирует псевдокод для алгоритма метода главных компонент PCA. Центрируются входные данные $\mathbf{D} \in \mathbb{R}^{n \times d}$ вычитанием среднего значения из каждой точки. Затем вычисляются собственные векторы и собственные значения ковариационной матрицы Σ . Учитывая желаемый порог дисперсии α , выбирается наименьший набор измерений r , который охватывает по меньшей мере долю α общей дисперсии. В заключении, вычисляются координаты каждой точки в новом r -мерном подпространстве главных компонентов, чтобы получить новую матрицу данных $\mathbf{A} \in \mathbb{R}^{n \times r}$.





АЛГОРИТМ 5. Метод главных компонент PCA

PCA ($\mathbf{D}, \alpha$):

- 1 $\boldsymbol{\mu} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$ // compute mean
- 2 $\mathbf{Z} = \mathbf{D} - \mathbf{1} \cdot \boldsymbol{\mu}^T$ // center the data
- 3 $\boldsymbol{\Sigma} = \frac{1}{n} (\mathbf{Z}^T \mathbf{Z})$ // compute covariance matrix
- 4 $(\lambda_1, \lambda_2, \dots, \lambda_d) = \text{eigenvalues}(\boldsymbol{\Sigma})$ // compute eigenvalues
- 5 $\mathbf{U} = (\mathbf{u}_1 \ \mathbf{u}_2 \ \dots \ \mathbf{u}_d) = \text{eigenvectors}(\boldsymbol{\Sigma})$ // compute eigenvectors
- 6 $f(r) = \frac{\sum_{i=1}^r \lambda_i}{\sum_{i=1}^d \lambda_i}$, for all $r = 1, 2, \dots, d$ // fraction of total variance
- 7 Choose smallest r so that $f(r) \geq \alpha$ // choose dimensionality
- 8 $\mathbf{U}_r = (\mathbf{u}_1 \ \mathbf{u}_2 \ \dots \ \mathbf{u}_r)$ // reduced basis
- 9 $\mathbf{A} = \{\mathbf{a}_i \mid \mathbf{a}_i = \mathbf{U}_r^T \mathbf{x}_i, \text{ for } i = 1, \dots, n\}$ // reduced dimensionality data

Геометрия метода главных компонент

Геометрически, когда $r = d$, PCA соответствует ортогональному изменению базиса, так что общая дисперсия фиксируется суммой дисперсий по каждому из главных направлений $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_d$, и далее, все ковариации равны нулю. Это можно увидеть, посмотрев на коллективное действие полного набора главных компонент, которые могут быть организованы в ортогональную матрицу $d \times d$.

$$\mathbf{U} = \begin{pmatrix} | & | & \cdot & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_d \\ | & | & \cdot & | \end{pmatrix}$$

$$\mathbf{U}^{-1} = \mathbf{U}^T$$

Каждый главный компонент $\mathbf{u}_i$ соответствует собственному вектору ковариационной матрицы $\boldsymbol{\Sigma}$, т.е.

$$\boldsymbol{\Sigma} \mathbf{u}_i = \lambda_i \mathbf{u}_i \text{ for all } 1 \leq i \leq d$$

Который можно компактно записать в матричных обозначениях следующим образом:

$$\boldsymbol{\Sigma} \begin{pmatrix} | & | & \cdot & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_d \\ | & | & \cdot & | \end{pmatrix} = \begin{pmatrix} | & | & \cdot & | \\ \lambda_1 \mathbf{u}_1 & \lambda_2 \mathbf{u}_2 & \dots & \lambda_d \mathbf{u}_d \\ | & | & \cdot & | \end{pmatrix}$$

$$\boldsymbol{\Sigma} \mathbf{U} = \mathbf{U} \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_d \end{pmatrix}$$





$$\Sigma \mathbf{U} = \mathbf{U} \Lambda \quad (71)$$

Если мы умножим уравнение (3.26) слева на $\mathbf{U}^{-1} = \mathbf{U}^T$, то получим

$$\mathbf{U}^T \Sigma \mathbf{U} = \mathbf{U}^T \mathbf{U} \Lambda = \Lambda = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_d \end{pmatrix}$$

Это означает, что если мы изменим базис на $\mathbf{U}$, то ковариационная матрица Σ изменится на аналогичную матрицу Λ , которая фактически является ковариационной матрицей в новом базисе. Тот факт, что Λ является диагональной, подтверждает, что после изменения базиса все ковариации исчезают, и остается только дисперсия по каждой из главных компонент, причем дисперсия по каждому новому направлению $\mathbf{u}_i$ задается соответствующим собственным значением λ_i .

Стоит отметить, что в новом базисе уравнение

$$\mathbf{x}^T \Sigma^{-1} \mathbf{x} = 1 \quad (72)$$

Определяет d-мерный эллипсоид (или гиперэллипс). Собственные векторы $\mathbf{u}_i$ матрицы Σ , т.е. главные компоненты, являются направлениями главных осей эллипсоида. Квадратные корни собственных значений, т.е. $\sqrt{\lambda_i}$, являются длинами полуосей эллипса.

Умножая (3.26) справа на $\mathbf{U}^{-1} = \mathbf{U}^T$, имеем

$$\Sigma = \mathbf{U} \Lambda \mathbf{U}^T \quad (73)$$

Предполагая, что Σ обратимая или невырожденная, имеем

$$\Sigma^{-1} = (\mathbf{U} \Lambda \mathbf{U}^T)^{-1} = (\mathbf{U}^{-1})^T \Lambda^{-1} \mathbf{U}^{-1} = \mathbf{U} \Lambda^{-1} \mathbf{U}^T$$

Где

$$\Lambda^{-1} = \begin{pmatrix} \frac{1}{\lambda_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\lambda_d} \end{pmatrix}$$





Подставив Σ^{-1} в уравнение (3.27) и используя тот факт, что $x = Ua$ из (47), где $a = (a_1, a_2, \dots, a_d)^T$ представляет координаты x в новом базисе, получаем

$$\begin{aligned}x^T \Sigma^{-1} x &= 1 \\(a^T U^T) U \Lambda^{-1} U^T (Ua) &= 1 \\a^T \Lambda^{-1} a &= 1 \\\sum_{i=1}^d \frac{a_i^2}{\lambda_i} &= 1\end{aligned}$$

что и есть уравнение эллипса с центром в 0 и длинной полуосей $\sqrt{\lambda_i}$. Таким образом $x^T \Sigma^{-1} x = 1$ или эквивалентно $a^T \Lambda^{-1} a = 1$ в новом базисе главных компонент определяет d -мерный эллипсоид, где длины полуосей равны стандартным отклонениям (квадратный корень из дисперсии $\sqrt{\lambda_i}$) вдоль каждой оси. Точно так же уравнение $x^T \Sigma^{-1} x = s$ или эквивалентное $a^T \Lambda^{-1} a = s$, для различных значений скалярной величины s , представляет концентрические эллипсоиды.

5.2 Метод независимых компонент

Анализ независимых компонент (АНК, англ. *Independent Component Analysis, ICA*), называемый также **Метод независимых компонент** (МНК) — это вычислительный метод в обработке сигналов для разделения многомерного сигнала на аддитивные подкомпоненты. Этот метод применяется при предположении, что подкомпоненты являются негауссовыми сигналами и что они статистически независимы друг от друга. АНК является специальным случаем слепого разделения сигнала.

Анализ независимых компонент пытается разложить множественный сигнал на независимые негауссовы сигналы. Например звук обычно является сигналом, который состоит из сложения в каждый момент одиночных t -сигналов, идущих из нескольких источников. Вопрос заключается в том, возможно ли разделить эти источники, выделяя их из общего сигнала. Если допущение статистической независимости верно, слепое разделение независимых компонент смешанного сигнала даст очень



хорошие результаты. Метод также применяется для анализа сигналов, которые могут быть и не смешанными.

Простым приложением АНК является «задача о шумной вечеринке», когда собеседники слышат друг друга, выделяя голос собеседника из общего сигнала, состоящего из шума одновременно говорящих людей в помещении и шумной улицы за окном. Обычно задача упрощается предположением, что задержка по времени или эхо отсутствуют. Заметим, что отфильтрованный и задержанный сигнал является копией зависимой компоненты, и тогда допущение статистической независимости не нарушено.

Важно также учитывать, что если представлено N источников, нужно по меньшей мере N наблюдений (например микрофонов, если наблюдаемый сигнал — аудио), чтобы обнаружить исходные сигналы. В этом случае матрица квадратна ($J=D$), где D входная размерность данных, а J — размерность модели). Иначе получаем и исследуем недоопределённый ($J>D$) или переопределённый ($J<D$) случай.

Метод АНК — разделение смешанных сигналов, базируется на двух допущениях и трёх эффектах источников смешанного сигнала, что даёт очень хорошие результаты. Двумя допущениями являются:

Источники сигналов независимы друг от друга.

Значения каждого источника сигнала имеют негауссово распределение.

Тремя эффектами источника смешанного сигнала являются:

Независимость: как в и допущении 1, источники сигналов независимы, однако их смесь не является независимой от источников, потому что смесь сигналов имеет одни и те же источники.

Нормальность: согласно центральной предельной теореме, распределение суммы независимых случайных переменных с конечной дисперсией стремится к гауссовому распределению. Попросту говоря, сумма двух независимых случайных переменных обычно имеет распределение более близкое к гауссовому, чем любое из двух исходных случайных



переменных. Здесь мы рассматриваем каждый сигнал как случайную переменную.

Сложность: временная сложность любой смеси сигналов больше, чем сложность одного сигнала, более простого по его составляющим.

Эти принципы составляют базовые основы АНК. Если сигналы, которые нам удалось извлечь из смеси, независимы, подобно исходным сигналам, и имеют негауссовы гистограммы или имеют малую сложность, подобную сигналу источников, они должны быть сигналами источников.

АНК находит независимые компоненты (которые называются факторами, скрытыми переменными или источниками) путём максимизации статистической независимости оцениваемых компонент. Можно выбрать один из многих путей для определения заменителя независимости, и этот выбор определит форму алгоритма АНК. Два наиболее широких определения независимости АНК:

1. Минимизация взаимной информации
2. Максимизация негауссовости

Семейство алгоритмов АНК с минимизацией взаимной информации (англ. *Minimization-of-Mutual information*, MMI) использует такие меры, как расхождение Кульбака – Лейблера и максимальная энтропия. Семейство алгоритмов АНК с максимизацией негауссовости использует коэффициент эксцесса и негэнтропию.

Типичным алгоритмам АНК свойственно использование следующих методов:

- центрирование (вычитание среднего и создание нулевого среднего для сигнала);
- избавление от корреляции (обычно с помощью спектрального разложения матрицы);
- снижение размерности в качестве препроцессорных шагов для снижения сложности проблемы и создания действительного итеративного алгоритма.

Декорреляция и снижение размерности могут быть получены методом главных компонент или сингулярным разложением. Декорреляция





обеспечивает методу такие условия, когда все размерности трактуются одинаково *и установлены априори* до прогона алгоритма. Хорошо известные алгоритмы для АНК: infomax, FastICA, JADE, ядерный независимый компонентный анализ и многие другие. В общем случае АНК не сможет определить действительное число источников сигналов, единственно правильный порядок или масштаб (включая знак) сигналов.

АНК важен для слепого разделения сигнала и имеет много практических приложений. Метод тесно связан с поиском (или даже является частным случаем поиска) факториального кодирования данных, то есть нового векторного представления каждого вектора данных таким образом, чтобы он был однозначно закодирован результирующим кодовым вектором (кодирование без потерь), при этом компоненты кода статистически независимы.

Линейный анализ независимых компонент может быть разделён на случай без шумов и случай с шумами, где АНК без шумов является частым случаем АНК с шумом. Нелинейный АНК следует считать отдельным случаем.

Общее определение

Данные представлены наблюдаемым случайным вектором $\mathbf{x} = (x_1, \dots, x_m)^T$, а скрытые компоненты случайным вектором $\mathbf{s} = (s_1, \dots, s_n)^T$. Задачей построения алгоритма является преобразование наблюдаемых данных $\mathbf{x}$ с помощью статического преобразования $\mathbf{W}$ в наблюдаемый вектор максимально независимых компонент $\mathbf{s} = \mathbf{W}\mathbf{x}$, измеренных некоторой функцией независимости $F(s_1, \dots, s_n)$.

Генерирующая модель

Линейная АНК без шума

Компоненты x_i наблюдаемого случайного вектора $\mathbf{x} = (x_1, \dots, x_m)^T$ генерируются как сумма независимых компонент $s_k, k = 1, \dots, n$:





$$x_i = a_{i,1}s_1 + \dots + a_{i,k}s_k + \dots + a_{i,n}s_n$$

взвешенных весами $a_{i,k}$.

Та же генерирующая модель может быть записана в векторном виде

как $\mathbf{x} = \sum_{k=1}^n s_k \mathbf{a}_k$, где наблюдаемый случайный вектор $\mathbf{x}$ представлен базисными векторами $\mathbf{a}_k = (a_{1,k}, \dots, a_{m,k})^T$. Базисные вектора $\mathbf{a}_k$ образуют столбцы матрицы смешивания $\mathbf{A} = (\mathbf{a}_1, \dots, \mathbf{a}_n)$ и генерирующая формула может быть записана как $\mathbf{x} = \mathbf{A}\mathbf{s}$, где $\mathbf{s} = (s_1, \dots, s_n)^T$.

Если дана модель и реализации $x_1, \dots, x_n$ случайного вектора $\mathbf{x}$, задачей является оценка как матрицы смешивания $\mathbf{A}$, так и источников $\mathbf{s}$. Это делается путём адаптивного вычисления векторов $\mathbf{w}$ и установления функции цены, которая либо максимизирует негауссовость вычисленного $s_k = \mathbf{w}^T \mathbf{x}$ или минимизирует взаимную информацию. В некоторых случаях априорное знание распределения вероятности источников может быть использовано в функции цены.

Исходные источники $\mathbf{s}$ могут быть извлечены путём умножения наблюдаемых сигналов $\mathbf{x}$ на обратную к матрице смешивания $\mathbf{W} = \mathbf{A}^{-1}$, которая известна также как не смешивающая матрица. Здесь предполагается, что матрица смешивания квадратная ($n=m$). Если число базисных векторов больше размерности наблюдаемых векторов $n > m$, задача является переопределённой, но остаётся разрешимой с помощью псевдообратной матрицы.

Линейный АНК с шумом

С добавочным предположением о нулевом среднем и не коррелирующим гауссовым шумом $\mathbf{n} \sim N(0, \text{diag}(\Sigma))$, модель АНК принимает форму $\mathbf{x} = \mathbf{A}\mathbf{s} + \mathbf{n}$.





Нелинейный АНК

Смесь источников не обязательно должна быть линейной. Используя нелинейную функцию смешивания $f(\cdot|\theta)$ с параметрами θ нелинейной моделью АНК будет $x = f(s|\theta) + n$.

Различимость

Независимые компоненты различимы с точностью до перестановки и масштабирования источников. Эта различимость требует, чтобы:

- Максимум один из источников s_k был гауссовым,
- Число наблюдаемых смесей m должно быть не меньше числа

компонент $n: m \geq n$. Это эквивалентно высказыванию, что матрица смеси A должна иметь полный ранг, чтобы существовала обратная ей смесь.

Бинарный анализ независимых компонент

Специальным вариантом АНК является Бинарный АНК, в котором как источники сигнала, так и мониторы имеют двоичную форму, а наблюдения от мониторов являются дизъюнктивной смесью бинарных независимых источников. Задача, как было показано, имеет приложения во многих областях, включая медицинскую диагностику, многокластерное назначение, сетевую томографию и управление ресурсами интернета.

Пусть $x_1, x_2, \dots, x_m$ является набором бинарных переменных из m мониторов и $y_1, y_2, \dots, y_n$ является набором бинарных переменных из n источников. Связи источник-монитор представлены (неизвестной) смешанной матрицей G , где $g_{ij} = 1$ указывает, что сигнал от i -го источника может наблюдаться j -м монитором. Система работает следующим образом: в любое время, если источник i активен ($y_i = 1$) и он связан с монитором j ($g_{ij} = 1$), то монитор j будет наблюдать некоторую активность ($x_j = 1$). Формально мы имеем:



$$x_i = \bigvee_{j=1}^n (g_{ij} \wedge y_j), i = 1, 2, \dots, m,$$

где $\wedge$ является булевым И (англ. AND), а $\vee$ является булевым ИЛИ (англ. OR). Заметим, что шум не моделируется явно, а трактуется как независимые источники.

Описанная выше проблема может быть эвристически решена (при предположении, что переменные непрерывны) путём применения метода FastICA на бинарных наблюдаемых данных для получения смешанной матрицы G (получены вещественные значения), затем применяем технику округления на G для получения бинарных значений. Этот подход, как было показано, даёт крайне неточный результат.

Другим методом является использование динамического программирования — матрица рекурсивно разбивает наблюдения X на подматрицы и алгоритм вывода прогоняется на этих подматрицах. Ключевое наблюдение, которое ведёт к этому алгоритму: подматрица X^0 матрицы X , где $x_{ij} = 0 \forall j$ соответствует несмещённой матрице наблюдений скрытых компонент, которые не имеют связи с i -м монитором. Результаты экспериментов показывают, что этот подход точен при умеренном уровне шумов.

Аппарат обобщённой бинарной АНК вводит более широкое описание проблемы, которое не требует какого-либо знания о порождающей модели. Другими словами, этот метод пытается разложить источник на независимые компоненты (на столько, насколько возможно создать алгоритм без потери какой-либо информации) без предварительных допущений применения способа, при помощи которого он был получен. Хотя эта задача достаточно сложна, она может быть точно решена с помощью метода ветвей и границ или точно ограничена сверху умножением матрицы на вектор.



6 УСИЛЕНИЕ ОБУЧЕНИЯ И КОНТРОЛЯ

6.1 Марковский процесс принятия решений

В математике Марковский процесс принятия решения (MDP) - это дискретный во времени стохастический процесс управления. Он обеспечивает математическую основу для моделирования принятия решений в ситуациях, когда результаты частично случайны и частично находятся под контролем лица, принимающего решения. MDP полезны для изучения задач оптимизации, решаемых с помощью динамического программирования и обучения с подкреплением. MDP были известны, по крайней мере, еще в 1950-х годах; Основной объем исследований марковских процессов принятия решений стал результатом книги Рональда Ховарда 1960 года «Динамическое программирование и марковские процессы». Они используются во многих дисциплинах, включая робототехнику, автоматическое управление, экономика и производство. Название MDP происходит от русского математика Андрея Маркова, поскольку они являются продолжением цепей Маркова.

На каждом временном шаге процесс находится в некотором состоянии s , и лицо, принимающее решение, может выбрать любое действие a , которое доступно в состоянии s . В ответ процесс на следующем временном шаге случайным образом переходит в новое состояние s' и дает лицу, принимающему решение, соответствующее вознаграждение $R_a(s, s')$.

Вероятность того, что процесс перейдет в свое новое состояние s' зависит от выбранного действия. В частности, он задается функцией перехода состояния $P_a(s, s')$. Таким образом, следующее состояние s' зависит от текущего состояния s и действия лица, принимающего решение a . Но учитывая s и a , он условно не зависит от всех предыдущих состояний и действий; другими словами, переходы состояний MDP удовлетворяют свойству Маркова.

Марковские процессы принятия решений являются расширением цепей Маркова; разница заключается в добавлении действий (предоставление возможности выбора) и вознаграждений (мотивация). И наоборот, если для



каждого состояния существует только одно действие (например, «ждать») и все вознаграждения одинаковы (например, «ноль»), процесс принятия решения Маркова сводится к цепи Маркова.

Чтобы определить марковский процесс принятия решений, нужно задать 4-кортеж $(S, A, P(\cdot, \cdot), R(\cdot, \cdot))$, где

- S - конечное множество состояний,
- A - конечное множество действий (часто представляется в виде множеств A_s , доступных из состояния s),
- $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ вероятность, что действие a в состоянии s во время t приведет в состояние s' ко времени $t+1$,
- $R_a(s, s')$ - вознаграждение, получаемое после перехода в состояние s' из состояния s с вероятностью перехода $P_a(s, s')$.

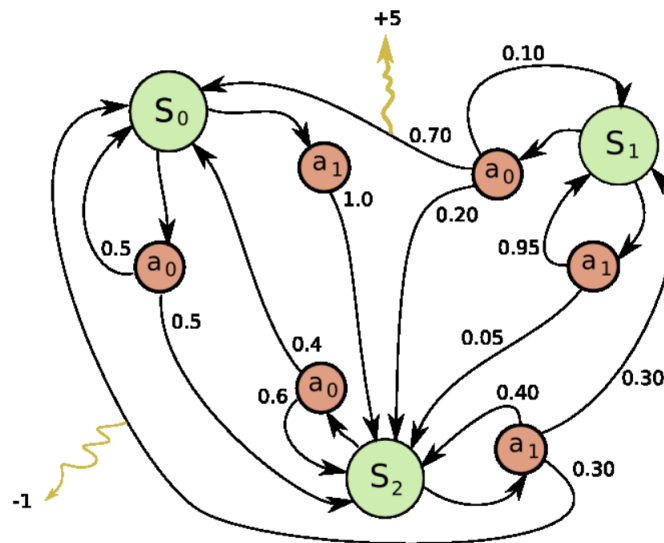


Рис. 21 Пример MDP с 3 состояниями и 2 действиями

6.2 Итерация значений и политик

Итерация значения

Итерация значения (Bellman 1957), которая также называется обратной индукцией. Функция π не используется; вместо этого значение $\pi(s)$



вычисляется в пределах $V(s)$ всякий раз, когда это необходимо. Подставляем вычисление $\pi(s)$ в вычисление $V(s)$, что дает объединенный шаг:

$$V_{i+1}(s) := \max_a \left\{ \sum_{s'} P_a(s'|s) (R_a(s, s') + \gamma V_i(s')) \right\},$$

где i - номер итерации. Итерация значения начинается с $i = 0$ и V_0 как предположение функция значения. Затем он выполняет итерацию, многократно вычисляя V_{i+1} для всех состояний s , пока V не сойдется с левой частью, равной правой (которая является «уравнением Беллмана» для этой задачи). Статья Ллойда Шепли 1953 года о стохастических играх включала в качестве особого случая метод итерации значений для MDP, но это было признано только позже.

Итерация политики

В итерации политики (Howard 1960) первый шаг выполняется один раз, а затем второй шаг повторяется до тех пор, пока она не сойдется. Затем снова выполняется первый шаг и так далее.

Вместо повторения второго шага до сходимости его можно сформулировать и решить как набор линейных уравнений. Эти уравнения просто получаются путем преобразования $s = s'$ в уравнение второго шага. Таким образом, повторение шага два до сходимости можно интерпретировать как решение линейных уравнений с помощью релаксации (итерационный метод). Этот вариант имеет то преимущество, что существует определенное условие остановки: когда массив π не меняется при применении шага 1 ко всем состояниям, алгоритм завершен.

Итерация политики обычно медленнее, чем итерация значения для большого количества возможных состояний.

Итерация измененной политики

В итерации измененной политики (van Nunen 1976 ; Puterman & Shin 1978) первый шаг выполняется один раз, а затем второй шаг повторяется несколько раз. Затем снова выполняется первый шаг и так далее.

Приоритетное сканирование

В этом варианте шаги предпочтительно применяются к состояниям, которые в некотором роде важны - независимо от того, основаны ли они на





алгоритме (были большие изменения в V или π вокруг этих состояний в последнее время) или в зависимости от использования (эти состояния близки к начальному состоянию или иным образом представляют интерес для человека или запрограммировать с использованием алгоритма).

6.3 Q - обучение

На основе получаемого от среды вознаграждения агент формирует функцию полезности Q , что впоследствии дает ему возможность уже не случайно выбирать стратегию поведения, а учитывать опыт предыдущего взаимодействия со средой. Одно из преимуществ Q-обучения – то, что оно в состоянии сравнить ожидаемую полезность доступных действий, не формируя модели окружающей среды. Применяется для ситуаций, которые можно представить в виде МППР.

Таким образом, алгоритм это функция качества от состояния и действия:

$$Q : S \times A \rightarrow \mathbb{R},$$

Перед обучением Q инициализируется случайными значениями. После этого в каждый момент времени t агент выбирает действие a_t , получает награду r_t , переходит в новое состояние s_{t+1} , которое может зависеть от предыдущего состояния s_t и выбранного действия, и обновляет функцию Q . Обновление функции использует взвешенное среднее между старым и новым значениями:

$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}},$$

где r_t это награда, полученная при переходе из состояния s_t в состояние s_{t+1} , и α это скорость обучения ($0 < \alpha \leq 1$).

Алгоритм заканчивается, когда агент переходит в терминальное состояние s_{t+1} .



Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	1	0	0	0	0	0	0
	2	0	0	0	0	0	0
	3	0	0	0	0	0	0
	4	0	0	0	0	0	0
	5	0	0	0	0	0	0
	6	0	0	0	0	0	0
	7	0	0	0	0	0	0
	8	0	0	0	0	0	0
	9	0	0	0	0	0	0
100	0	0	0	0	0	0	

Training

↓

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	1	0	0	0	0	0	0
	2	0	0	0	0	0	0
	3	0	0	0	0	0	0
	4	0	0	0	0	0	0
	5	0	0	0	0	0	0
	6	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	7	0	0	0	0	0	0
	8	0	0	0	0	0	0
	9	0	0	0	0	0	0
100	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603	



```
fun Q-learning( $S, A, R, T, \alpha, \gamma$ ):  
  for  $s \in S$ :  
    for  $a \in A$ :  
       $Q(s, a) = \text{rand}()$   
  while  $Q$  is not converged:  
     $s = \forall s \in S$   
    while  $s$  is not terminal:  
       $\pi(s) = \text{argmax}_a Q(s, a)$   
       $a = \pi(s)$   
       $r = R(s, a)$   
       $s' = T(s, a)$   
       $Q(s', a) = (1 - \alpha)Q(s', a) + \alpha(r + \gamma \max_{a'} Q(s', a'))$   
       $s = s'$   
  return  $Q$ 
```

6.4 Обучение с подкреплением

В обучении с подкреплением существует агент (agent) взаимодействует с окружающей средой (environment), предпринимая действия (actions). Окружающая среда дает награду (reward) за эти действия, а агент продолжает их предпринимать.

Алгоритмы с частичным обучением пытаются найти стратегию, приписывающую состояниям (states) окружающей среды действия, одно из которых может выбрать агент в этих состояниях.

Среда обычно формулируется как марковский процесс принятия решений (МППР) с конечным множеством состояний, и в этом смысле алгоритмы обучения с подкреплением тесно связаны с динамическим программированием. Вероятности выигрышей и перехода состояний в МППР обычно являются величинами случайными, но стационарными в рамках задачи.

При обучении с подкреплением, в отличие от обучения с учителем, не предоставляются верные пары "входные данные-ответ", а принятие субоптимальных решений (дающих локальный экстремум) не ограничивается явно. Обучение с подкреплением пытается найти компромисс между исследованием неизученных областей и применением имеющихся знаний





(exploration vs exploitation). Баланс изучения-применения при обучении с подкреплением исследуется в задаче о многоруком бандите.

Формально простейшая модель обучения с подкреплением состоит из:

- множества состояний окружения (states) S ;
- множества действий (actions) A ;
- множества вещественнозначных скалярных "выигрышей" (rewards).

В произвольный момент времени t агент характеризуется состоянием $s_t \in S$ и множеством возможных действий $A(s_t)$. Выбирая действие $a \in A(s_t)$, он переходит в состояние s_{t+1} и получает выигрыш r_t . Основываясь на таком взаимодействии с окружающей средой, агент, обучающийся с подкреплением, должен выработать стратегию $\pi: S \rightarrow A$, которая максимизирует величину $R = r_0 + r_1 + \dots + r_n$ в случае МППР, имеющего терминальное состояние, или величину:

$$R = \sum_t \gamma^t r_t,$$

для МППР без терминальных состояний (где $0 \leq \gamma \leq 1$ — дисконтирующий множитель для "предстоящего выигрыша").

Таким образом, обучение с подкреплением особенно хорошо подходит для решения задач, связанных с выбором между долгосрочной и краткосрочной выгодой.

Постановка задачи обучения с подкреплением

S — множество состояний среды

Игра агента со средой:

- инициализация стратегии $\pi_1(a|s)$ и состояния среды s_1 ;
- для всех $t=1 \dots T$:
- агент выбирает действие $a_t \sim \pi_t(a|s_t)$;
- среда генерирует награду $r_{t+1} \sim p(r|a_t, s_t)$ и новое состояние $s_{t+1} \sim p(s|a_t, s_t)$;



- агент корректирует стратегию $\pi_{t+1}(a|s)$.

Это марковский процесс принятия решений (МППР), если $P(s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \dots, s_1, a_1) = P(s_{t+1} = s', r_{t+1} = r | s_t, a_t)$

МППР называется финитным, если $|A| < \infty$, $|S| < \infty$.

Алгоритмы

Теперь, когда была определена функция выигрыша, нужно определить алгоритм, который будет использоваться для нахождения стратегии, обеспечивающей наилучший результат.

Наивный подход к решению этой задачи подразумевает следующие шаги:

- опробовать все возможные стратегии;
- выбрать стратегию с наибольшим ожидаемым выигрышем.

Первая проблема такого подхода заключается в том, что количество доступных стратегий может быть очень велико или бесконечно. Вторая проблема возникает, если выигрыши стохастические — чтобы точно оценить выигрыш от каждой стратегии потребуется многократно применить каждую из них. Этих проблем можно избежать, если допустить некоторую структуризацию и, возможно, позволить результатам, полученным от пробы одной стратегии, влиять на оценку для другой. Двумя основными подходами для реализации этих идей являются оценка функций полезности и прямая оптимизация стратегий.

Подход с использованием функции полезности использует множество оценок ожидаемого выигрыша только для одной стратегии π (либо текущей, либо оптимальной). При этом пытаются оценить либо ожидаемый выигрыш, начиная с состояния s , при дальнейшем следовании стратегии π ,

$$V(s) = E[R|s, \pi],$$

либо ожидаемый выигрыш, при принятии решения a в состоянии s и дальнейшем соблюдении π ,

$$Q(s, a) = E[R|s, \pi, a],$$



Если для выбора оптимальной стратегии используется функция полезности Q , то оптимальные действия всегда можно выбрать как действия, максимизирующие полезность.

Если же мы пользуемся функцией V , необходимо либо иметь модель окружения в виде вероятностей $P(s'|s, a)$, что позволяет построить функцию полезности вида

$$Q(s, a) = \sum_{s'} V(s') P(s'|s, a),$$

либо применить т.н. метод исполнитель-критик, в котором модель делится на две части: критик, оценивающий полезность состояния V , и исполнитель, выбирающий подходящее действие в каждом состоянии.

Имея фиксированную стратегию π , оценить $E[R|\cdot]$ при $\gamma = 1$ можно просто усреднив непосредственные выигрыши. Наиболее очевидный способ оценки при $\gamma \in (0, 1)$ – усреднить суммарный выигрыш после каждого состояния. Однако для этого требуется, чтобы МППР достиг терминального состояния (завершился).

Поэтому построение искомой оценки при $\gamma \in (0, 1)$ неочевидно. Однако, можно заметить, что R образуют рекурсивное уравнение Беллмана:

$$E[R|s_t] = r_t + \gamma E[R|s_{t+1}],$$

Подставляя имеющиеся оценки V и применяя метод градиентного спуска с квадратичной функцией ошибок, мы приходим к алгоритму обучения с временными воздействиями (temporal difference (TD) learning). В простейшем случае и состояния, и действия дискретны и можно придерживаться табличных оценок для каждого состояния.

6.5 Частично наблюдаемый марковский процесс принятия решения

Частично наблюдаемый марковский процесс принятия решений (POMDP) является обобщением марковского процесса принятия решений (MDP). POMDP моделирует процесс принятия решений агентом, в котором предполагается, что динамика системы определяется MDP, но агент не





может напрямую наблюдать за базовым состоянием. Вместо этого он должен поддерживать распределение вероятностей по набору возможных состояний на основе набора наблюдений и вероятностей наблюдений, а также лежащего в основе MDP.

Структура POMDP достаточно общая, чтобы моделировать множество реальных последовательных процессов принятия решений. Приложения включают проблемы навигации роботов, техническое обслуживание машин и планирование в условиях неопределенности в целом. Общая структура марковских процессов принятия решений с несовершенной информацией была описана Карлом Йоханом Остромом в 1965 году в случае дискретного пространства состояний, и она была дополнительно изучена в операциях . исследование сообщества, в котором была придумана аббревиатура POMDP. Позднее он был адаптирован для задач в искусственном интеллекте и автоматизированном планировании Лесли П. Келблинг и Майкл Л. Литтман .

Точное решение POMDP дает оптимальное действие для каждого возможного убеждения по состояниям мира. Оптимальное действие максимизирует (или минимизирует) ожидаемое вознаграждение (или стоимость) агента на возможно бесконечном горизонте. Последовательность оптимальных действий известна как оптимальная политика агента для взаимодействия со своим окружением.

Формальное определение

POMDP с дискретным временем моделирует отношения между агентом и его окружение. Формально POMDP - это кортеж из семи элементов $(S, A, T, R, \Omega, O, \gamma)$, где

S - набор состояний,

A - набор действий,

T - набор вероятностей условного перехода между состояниями,

$R : S \times A \rightarrow \mathbb{R}$ - функция вознаграждения.

Ω - набор наблюдений,

O - набор условных вероятностей наблюдения, а

$\gamma \in [0, 1]$ - коэффициент дисконтирования.



В каждый период времени среда находится в некотором состоянии $s \in S$. Агент выполняет действие $a \in A$, которое вызывает переход среды в состояние s' с вероятностью $T(s'|s, a)$. В то же время агент получает наблюдение $o \in \Omega$, которое зависит от нового состояния окружающей среды, s' , а для только что предпринятого действия a с вероятностью $O(o|s', a)$. Наконец, агент получает вознаграждение r , равное $R(s, a)$. Затем процесс повторяется. Цель состоит в том, чтобы агент выбирал действия на каждом временном шаге, которые максимизируют его ожидаемое будущее

дисконтированное вознаграждение: $E \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$ - награда, полученная в определенный момент t . Коэффициент дисконтирования γ определяет, насколько немедленное вознаграждение предпочтительнее более отдаленного вознаграждения. Когда $\gamma = 0$, агент заботится только о том, какое действие принесет наибольшее ожидаемое немедленное вознаграждение; когда $\gamma = 1$, агент заботится о максимизации ожидаемой суммы будущих вознаграждений.

ЛИТЕРАТУРА

1. Ведение в машинное обучение. Рязань: 2020. - 46 с. Интернет. <http://kro.digitalr.ru/media/preza/Основы%20машинного%20обучения.pdf>.
2. Воронина В.В., Михеев А.В., Ярушкина Н.Г., Святков К.В. Теория и практика машинного обучения. Ульяновск: УлЕТУ, 2017. - 289 с.
3. Миронов А.М. Машинное обучение. М.: МГУ, 2019. - 83 с.
4. Вьюгин В.В. Математические основы машинного обучения и прогнозирования. Москва, издательство МЦНМО, 2018. 384 с.
5. Николенко С., Кадури А., Архангельская Е. Глубокое обучение. Погружение в мир нейронных сетей, Питер, 2018. 480 с.
6. Skansi S. Introduction to Deep Learning: from Logical Calculus to Artificial Intelligence. Springer, 2018. 191 с.
7. Мюллер А., Гвидо С. Введение в машинное обучение с помощью Python, Москва, 2017. 393 с.





8. Харрисон, Мэтт. Машинное обучение: карманный справочник. Краткое руководство по методам структурированного машинного обучения на Python. Пер. с англ. - СПб.: ООО "Диалектика", 2020. - 320 с. : ил. - Парал.тит. англ.
9. Лимановская, О.В. Основы машинного обучения : учебное пособие / О.В. Лимановская, Т.И. Алферьева ; Мин-во науки и высшего образования РФ. - Екатеринбург: Изд-во Урал. ун-та, 2020. - 88 с.
10. Мюллер Д.П., Массарон Л. Искусственный интеллект. СПб.: ООО Диалектика, 2019. - 384 с.
11. *Математические методы обучения по прецедентам (теория обучения машин)* К. В. Воронцов http://www.ccas.ru/voron_voron@ccas.ru