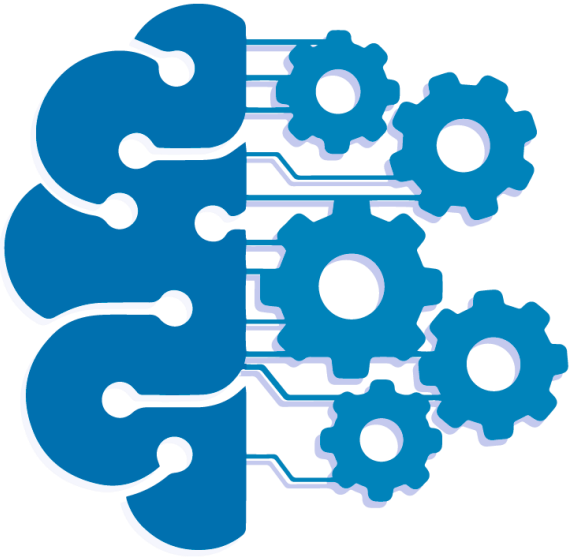
УНИВЕРСИТЕТ ИТМО

А.В. Кугаевских, Д.И. Муромцев, О.В. Кирсанова

КЛАССИЧЕСКИЕ МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ



Санкт-Петербург
2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

А.В. Кугаевских, Д.И. Муромцев, О.В. Кирсанова

Классические методы машинного обучения

УЧЕБНОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлениям подготовки 09.04.01 «Информатика и вычислительная техника», 09.04.04 «Программная инженерия» в качестве учебного пособия для реализации основных профессиональных образовательных программ высшего образования магистратуры

Санкт-Петербург
2022

А.В. Кугаевских, Д.И. Муромцев, О.В. Кирсанова. Классические методы машинного обучения. – СПб: Университет ИТМО, 2022. – 53 с.

Рецензент(ы):
Пальчунов Д.Е., доктор физико-математических наук, ведущий научный сотрудник, Институт математики СО РАН.

В настоящем учебном пособии рассмотрены методы машинного обучения, уже ставшие классикой: кластеризация, ассоциативные правила, обучение с учителем, ансамблевые методы.

Учебное пособие предназначено для освоения соответствующих разделов дисциплин «Архитектура систем ИИ», «Инструментальные средства для систем ИИ», «Методология инженерии программных систем ИИ», «Фундаментальные основы организации систем искусственного интеллекта».

УНИВЕРСИТЕТ ИТМО

Университет ИТМО – национальный исследовательский университет, ведущий вуз России в области информационных, фотонных и биохимических технологий. Альма-матер победителей международных соревнований по программированию – ICPC (единственный в мире семикратный чемпион), Google Code Jam, Facebook Hacker Cup, Яндекс.Алгоритм, Russian Code Cup, Topcoder Open и др. Приоритетные направления: IT, фотоника, робототехника, квантовые коммуникации, трансляционная медицина, Life Sciences, Art&Science, Science Communication. Входит в ТОП-100 по направлению «Автоматизация и управление» Шанхайского предметного рейтинга (ARWU) и занимает 74 место в мире в британском предметном рейтинге QS по компьютерным наукам (Computer Science and Information Systems). С 2013 по 2020 гг. – лидер Проекта 5–100.

Оглавление

1. Введение

Настоящее учебное пособие может быть использовано для подготовки к экзаменам и выполнению лабораторных работ по дисциплинам: дисциплин «Архитектура систем ИИ», «Инструментальные средства для систем ИИ», «Методология инженерии программных систем ИИ», «Фундаментальные основы организации систем искусственного интеллекта», а также для разработки ВКР по тематикам, требующим применения машинного обучения. Рассматриваемый теоретический материал будет полезен для самостоятельной работы студентов при их взаимодействии с библиотеками машинного обучениями.

В 1997 году Том Митчелл дал определение машинному обучению в своей классической книге «Машинное обучение» [1]:

Говорят, что компьютерная программа **обучается** на опыте E относительно некоторого класса задач T и меры качества P, если качество на задачах из T, измеренное с помощью P, возрастает с ростом опыта E.

И в этом определении читается главное отличие машинного обучения от разработки программ. В результате работы алгоритма машинного обучения, который, конечно же, программируется, на некоторых данных D получается модель.

Модель - математический объект, описывающий соотношение между описанием примеров (samples) X, чаще всего в виде признаков, и целевыми переменными (targets) Y. Обобщающая способность модели не может быть идеальной, поэтому всегда имеет место быть ошибка e.

$$Y = f(X) + e$$

За долгие годы с помощью машинного обучения решали разные задачи, которые могут быть сведены к нескольким базовым классам T, из которых основными являются: классификация и регрессия.

Задача **классификации** ставится, когда нам необходимо присвоить примеру метку класса или несколько меток классов (в этом случае, задача называется multi-labeling classification) на основании отличительных особенностей, именуемых признаками. При этом классы представляют собой конечное множество, $Y = \{1, \dots, K\}$. В задаче **регрессии** нам интересно по последовательному предъявлению примеров восстановить некоторую зависимость и на ее основе предсказать будущие значения, $Y = \mathbb{R}^M$.

В зависимости от задачи, имеющихся данных и лени разработчика выбирается вид машинного обучения. Чаще всего применяют обучение с учителем, больше всего хотят работающий алгоритм обучения без учителя. Когда разметка всех имеющихся данных слишком дорога, применяют обучение с частичным привлечением учителя.

Виды машинного обучения:

1. Обучение с учителем (supervised learning) – есть размеченные данные (для каждого примера есть «решение или метка класса», но не только, размеченные данные характеризуются еще и значимой информацией о признаках или зонах интереса, например, прямоугольными рамками нужных объектов на изображении);

2. С частичным привлечением учителя (semi-supervised learning) – для части прецедентов задается пара «ситуация, решение», а для части - только «ситуация»;

3. Трансдуктивное обучение (transduction learning) - обучение, когда прогноз предполагается делать только для примеров из тестовой выборки;

4. Обучение без учителя (unsupervised learning) – есть неразмеченные данные («ситуация»), требуется сгруппировать объекты;

5. Обучение с подкреплением (reinforcement learning) – есть размечаемые данные («ситуация, предполагаемое решение»). Алгоритм обучения работает через вознаграждение за правильное решение или наказание за неправильное;

6. Активное обучение (active learning) - обучаемый алгоритм может самостоятельно назначать следующую исследуемую ситуацию, на которой станет известен верный ответ;

7. Многозадачное обучение (multi-task learning) - одновременное обучение группе взаимосвязанных задач, для каждой из которых задаются свои пары «ситуация, решение»;

8. Многовариантное обучение (multi-instant learning) - обучение, когда примеры могут быть объединены в группы, в каждой из которых для всех примеров имеется «ситуация», но только для одного из них (причем, неизвестно какого) имеется пара «ситуация, решение»;

9. Динамическое обучение или онлайн обучение – предполагает адаптацию к постоянно меняющимся данным.

Было придумано множество алгоритмов, относящихся к сфере машинного обучения. Не все из них выдержали испытание временем. Например, метод ближайших соседей хоть и прост в реализации, и быстр в исполнении, но другие методы превосходят его по качеству работы, поэтому мне непонятно, зачем его до сих пор изучают на машинном обучении. Поэтому в данном пособии речь пойдет только об актуальных методах широкого применения.

Алгоритмы машинного обучения можно разделить по принципу, лежащему в основе построения модели. Геометрические алгоритмы преобразуют пространство данных либо строя разделяющую гиперповерхность, либо разнося данные в пространстве. Вероятностные алгоритмы придуманы под влиянием теории вероятностей. Логические алгоритмы используют логическую взаимосвязь между описанием данных. Ансамблевые алгоритмы призваны комбинировать несколько моделей для общего улучшения качества.

1. Геометрические
    a. K-ближайших соседей (kNN)
    b. Логистическая регрессия
    c. Машины опорных векторов (SVM)

d. Кластеризация K-средних (K-means)

2. Вероятностные
    a. Байесовский классификатор
    b. Поиск максимального правдоподобия (EM)
    c. Скрытые марковские модели (HMM)
3. Логические
    a. Деревья решений
    b. Ассоциативные правила
4. Ансамбли
    a. Усиление (boosting)
    b. Bagging
    c. Стэковое обобщение

Выбор того или иного алгоритма обусловливается задачей, которую предстоит решить, и имеющимися данными, их природой, полнотой и качеством описания. Разработчики самой популярной на настоящее время библиотеки для анализа данных scikit-learn на своем сайте [2] опубликовали диаграмму для выбора алгоритма, рисунок 1.

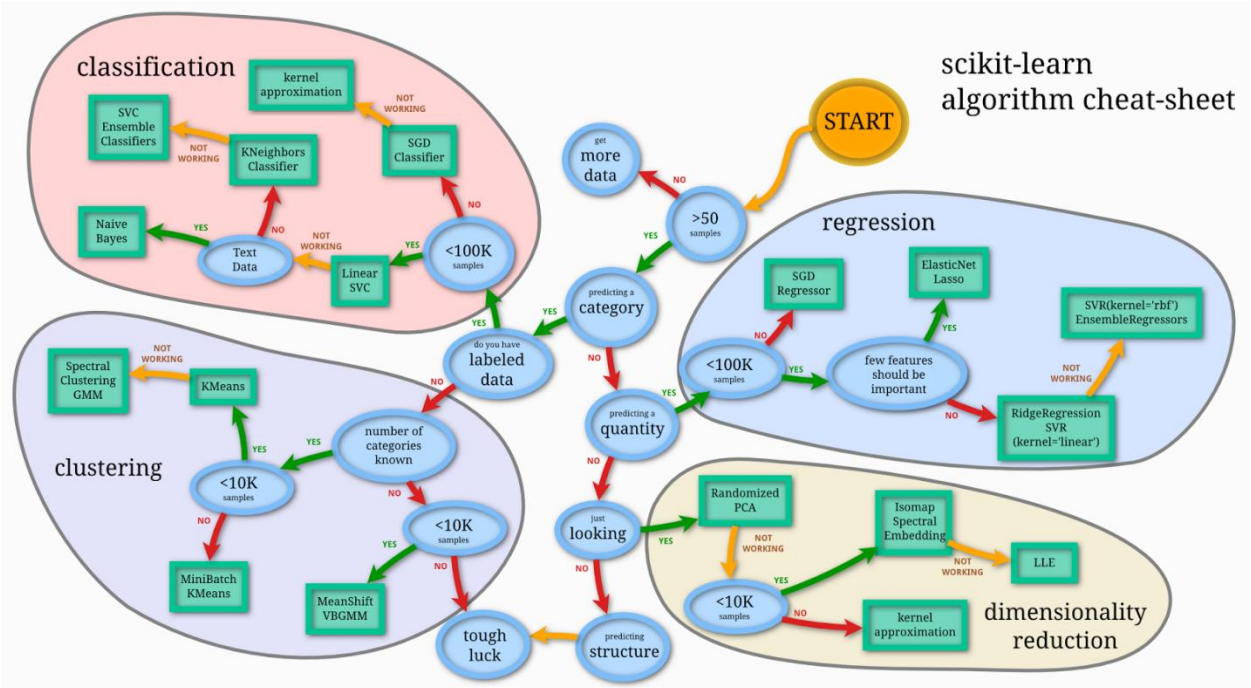


Рисунок 1 – Диаграмма выбора алгоритма машинного обучения [2]

Еще одним важным ингредиентом машинного обучения помимо алгоритмов являются данные, некоторые даже утверждают, что это самый важный ингредиент, «новая нефть» и прочий «тяжелый бред» маркетологов. Генеральная совокупность при решении задачи недоступна, наш набор данных (dataset), над которым мы проводим эксперименты по построению аппроксимирующей модели, может лишь отражать ее. При этом важно, чтобы набор данных, который по сути является выборочной совокупностью, был репрезентативен. Репрезентативность набора данных предполагает, что в нем

хорошо представлены разные случаи моделируемого процесса. В противном случае это может стать причиной смещения оценок, про которые мы поговорим подробнее в главе 5.

В построении модели участвуют данные из набора, разделенные по трем выборкам.

Обучающая выборка (training set) используется исключительно для обучения модели. Математически она представляет собой произведение вектора экземпляров (X) и вектора целевых переменных (Y)

$$S^l = X \times Y = \{(x_1, y_1), \dots, (x_l, y_l)\}$$

Тестовая выборка (testing set) – полностью независимая от обучающей выборка, предназначенная для оценки качества работы модели. Без ее применения нельзя гарантировать несмещенность оценок модели (также называемую переобучением).

Валидационная выборка (development set, validation set) применяется для подбора параметров, выбора признаков и других решений по алгоритму. Чаще всего валидационная выборка составляет 10% от всего набора данных.

Разделение имеющего набора данных на обучающую и тестовую выборку должно соответствовать некоторым правилам:

1. Соотношение обучающей и тестовой выборок должно быть сильно в пользу обучающей, чаще всего используют разделение 70% и 30%, 80% и 20%, 60% и 40%.
2. Распределение признаков в обучающей и тестовой выборках должны иметь одинаковые характеристики. Обратное называется ковариантным сдвигом. Например, как в обучающую, так и в тестовую выборки должны попасть данные, собранные в разное время, чтобы избежать сосредоточения на ложных корреляциях условий сбора данных, причиной которых может быть нарушение методологии сбора данных, смена устройств и т.д.
3. При разделении нужно учитывать несбалансированность классов (imbalanced data), сохраняя одинаковое соотношение экземпляров между классами в обеих выборках. С несбалансированностью надо поступать очень осторожно, обучение модели «в лоб» на таком наборе данных может привести к искажению обобщающей способности из-за того, что модель обучится на классе с преобладанием экземпляров (мажорном классе) и совершенно проигнорирует классы с небольшим количеством экземпляров в выборке (минорные классы). Для этого можно применить самый и простой и в то же время самый эффективный метод – изменить порог разделения классов.
4. Чем выше размер обучающей выборки, тем выше точность. Теоретически… На самом деле наступает момент, когда с ростом размера выборки качество обучения либо выходит на плато, либо даже начинает падать. Это можно отследить по кривой обучения, показывающей зависимость между размером выборки и качеством модели. Малая обучающая выборка может плохо сказаться на обобщающей способности. Старайтесь, чтобы на каждый класс

было не меньше 1000 примеров. Тем не менее, и на 100 примерах можно обучить качественную модель, во всяком случае до 2012 года так и делали.

Если размер обучающей выборки Вас не устраивает, а возможности добрать реальных данных нет или это слишком дорого, можно применить **аугментацию**. Например, для изображений это может быть применение геометрических преобразований (аффинных, проективных и т.п.), цветовые преобразования (яркость, контрастность, оттенок), добавление бликов, тени, тумана. Но с синтетическими данными всегда есть опасность того, что они не будут репрезентативны реальному миру, где будет функционировать модель.

Больше всего дискуссий в контексте машинного обучения вызывает необходимость знания математики для успешного создания моделей. Во-первых, как говорил Ломоносов, математика тем и хороша, что ум в порядок приводит, а во-вторых, без знания математики многие моменты в машинном обучении остаются на уровне слесарного станка и случайного выбора его параметров. Нередко приходиться модифицировать алгоритм машинного обучения под специфику задачи и имеющиеся данные, а сделать это качественно без понимания работы метода невозможно. Поэтому перед изучением машинного обучения не лишним будет вспомнить математику.

Потребуются знания линейной алгебры, в особенности понятие вектора и векторного пространства, сингулярное разложение матрицы, собственные вектора и собственные значения, квадратичные формы, тензоры. Рекомендуемая литература: Тыртышников Е.Е. Матричный анализ и линейная алгебра. – М.: Физматлит, 2007 [3]; Ильин В.А., Позняк Э.Г. Линейная алгебра. – М.: Физматлит, 2005 [4].

Еще одна обязательная область - математический анализ, в особенности предел функции, непрерывные функции, дифференцирование, функции многих переменных, градиент, интегрирование. Рекомендуемая литература: Ильин В.А., Позняк Э.Г. Основы математического анализа: в 2 ч. - М.: Наука. Физматлит, 1998 [5].

Многие оптимизационные техники, в частности градиентный спуск, рассматриваются в дисциплине «методы оптимизации». Рекомендуемая литература: Поляк Б.Т. Введение в оптимизацию. – М.: Наука, 1983 [6].

Для понимания деревьев решений и ансамблей потребуется понимание математической логики. Рекомендуемая литература: Игошин В.И. Математическая логика и теория алгоритмов, 2-е изд. – М.: Издательский дом «Академия», 2008 [7].

Теория вероятностей и математическая статистика. Помимо стандартных понятий вероятности, в том числе совместной вероятности, математического ожидания, распределений случайных величин, предельных теорем также понадобятся условное математическое ожидание, многомерное нормальное распределение, марковские цепи, расхождение Кульбака-Лейблера и расхождение Йенсена-Шеннона, генеральная и выборочная совокупности, а также другие понятия математической статистики. Рекомендуемая литература: Вентцель Е.С. Теория вероятностей [8]; Кремер Н.Ш. Теория вероятностей и

математическая статистика, 2-е изд. – М.: Юнити-Дана, 2004 [9]; Боровков А.А. Математическая статистика, 4-е изд. – СПб.: Издательство «Лань», 2010 [10].

Потребуются хорошие знания теории информации, в особенности понятия энтропии и ее видов, в частности условная энтропия, перекрестная энтропия, понятие перплексии. Также будет необходимо понимание прироста информации и взаимной информации. Рекомендуемая литература: Павлов Ю.Н. Теория информации для бакалавров. -М.: Издательство МГТУ им. Н.Э. Баумана, 2016 [11]; Панин В.В. Основы теории информации, 4-е изд. – М.: Бином. Лаборатория знаний, 2012 [12].

Дискретная математика, в особенности комбинаторика и теория графов. Рекомендуемая литература: Андерсон Дж. Дискретная математика и комбинаторика. – М.: Вильямс, 2004 [13]; Хаггарти Р. Дискретная математика для программистов, 2-е изд. – М.: Техносфера, 2012 [14].

Данное пособие не претендует на полноту описания методов машинного обучения. Регрессионный анализ и метод поиска максимального правдоподобия хорошо описывается в курсе теории вероятности. Для более глубокого понимания остальных методов можно порекомендовать книгу: Флах П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных. – М.: ДМК Пресс, 2015 [15].

Обучение с подкреплением мы в данном пособии рассматривать не будем, самой лучшей книгой является Саттон Р.С., Барто Э.Дж. Обучение с подкреплением: Введение. 2-е изд. – М.: ДМК Пресс, 2020 [16], которая и рекомендуется всем как обязательное чтение.

Самой лучшей на данный момент книгой по теории вычислительного обучения можно считать Шалев-Шварц Ш., Бен-Давид Ш. Идеи машинного обучения: от теории к алгоритмам. – М.: ДМК Пресс, 2019 [17].

2. Ассоциативные правила

2.1. Алгоритм APRIORI

Ассоциативные правила были придуманы в 1966 году и применены в 1993 для обнаружения закономерностей в транзакциях POS-терминалов в супермаркетах. И по сей день основным направлением их применения является анализ потребительской корзины для вывода рекомендаций. Но вывод закономерностей на основании статистики может быть применен и в других областях. Поэтому ассоциативные правила развились в выявление часто встречающихся шаблонов (frequent pattern mining) в наборе данных.

Пусть у нас имеются часто встречающиеся наборы объектов $T = \{i_j | i_i \in I\}$ в большом множестве наборов $D = \{T_1, \dots, T_m\}$, F – некоторый произвольный набор объектов и D_F - множество транзакций T, в которые входит набор F.

Таблица 1. Пример множества наборов

Номер транзакции T	Набор объектов
1	Чипсы, водка, пиво
2	Кокосы, кола, ром, сигары
3	Кола, чипсы
4	Пиво, чипсы
5	Ром, сигары

Тогда ассоциативное правило определяется как импликация вида $X \Rightarrow Y$, где операнды $X, Y \subseteq I$. Условно можно сказать, что ассоциативное правило напоминает условный оператор или продукционное правило: если выполняется первый операнд X (или левая часть), то результатом будет второй операнд (или правая часть) Y. Пользуясь нашим примером из таблицы 1, правило может выглядеть как $\{\text{кокосы}, \text{ром}\} \Rightarrow \{\text{кола}\}$.

Для того, чтобы автоматизировать процесс составления ассоциативных правил, был разработан алгоритм Apriori [18]. Основная идея алгоритма заключается в построении всевозможных наборов и генерации по ним правил. Для этого рассчитываются два параметра: поддержка набора и достоверность правила.

Поддержка (support) набора F – отношение количества транзакций, в которые входит набор F, к общему количеству транзакций

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

Достоверность (confidence) – вероятность того, что из наличия в транзакции набора X следует наличие в ней набора Y. Показывает, насколько правило оказывается верным.

Отношение числа транзакций, содержащих наборы X и Y, к числу транзакций, содержащих набор X:

$$conf(X \Rightarrow Y) = \frac{|D_F|}{|D_X|} = \frac{supp(X \cup Y)}{supp(X)}$$

В терминах теории вероятностей достоверность можно понимать как оценку условной вероятности $P(E_Y|E_X)$ нахождения правой части правила в транзакции при условии нахождения левой части там же.

Формализовать алгоритм можно следующим образом:

Шаг 1. Присвоить $k = 1$ и выполнить отбор всех 1-элементных наборов, у которых поддержка больше minsupp

Шаг 2. $k = k + 1$

Шаг 3. Если не удается создать k -элементные наборы, то выход, иначе на шаг 4.

Шаг 4. Создать множество k -элементных наборов кандидатов в частые наборы. Для этого необходимо объединить в k -элементные кандидаты $(k-1)$-элементные частые наборы.

Шаг 5. Для каждой транзакции T из множества D выбрать кандидатов из Ck, присутствующих в транзакции T. Для каждого набора из построенного

множества Ck удалить набор, если хотя бы одно из его $(k-1)$) подмножеств не является часто встречающимся

Шаг 6. Для каждого кандидата из множества Ck увеличить $supp$ на 1.

Шаг 7. Выбрать только кандидатов из множества Ck, у которых $supp > minsupp$. Вернуться к шагу 2.

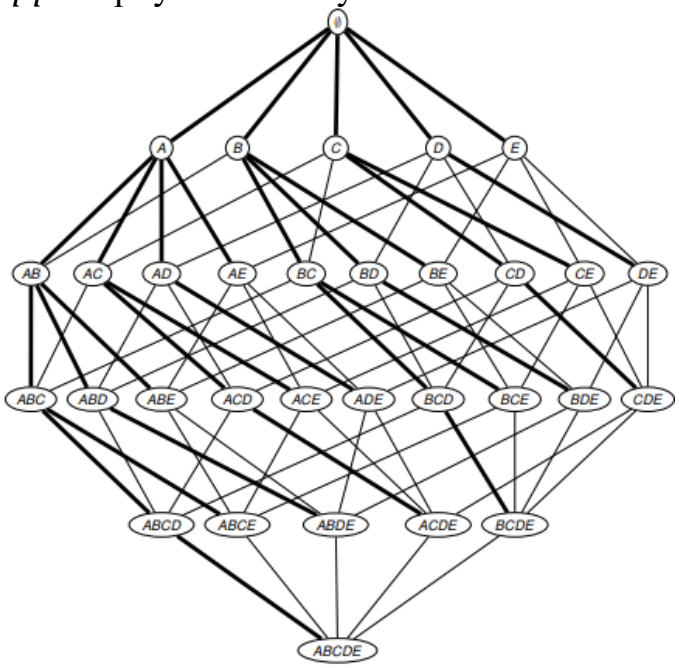


Рисунок 2 – Генерация наборов алгоритмом Apriori

Правила, созданные алгоритмом Apriori, не лишены противоречивости, наблюдаемой на больших наборах. Поэтому было введено еще несколько концепций: улучшение и уверенность, применимые к правилам.

Улучшение (lift, в некоторых источниках improvement) – показывает, полезнее ли правило случайного угадывания.

Отношение числа транзакций, содержащих наборы X и Y, к произведению количества транзакций, содержащих набор X, и количества транзакций, содержащих набор Y.

$$impr(X \Rightarrow Y) = \frac{|D_F|}{|D_X||D_Y|} = \frac{supp(X \cup Y)}{supp(X) * supp(Y)}$$

Если $impr > 1$, то предсказание с помощью правила вероятнее случайного угадывания, в противном случае используют отрицание правила $X \Rightarrow not(Y)$.

Уверенность (conviction) – показывает, насколько правило будет неверным, если ассоциация межу X и Y была чистой случайностью.

Отношение ожидаемой частоты, что X встречается без Y, и наблюдаемой частоты неверных предсказаний.

$$conv(X \Rightarrow Y) = \frac{1 - supp(Y)}{1 - conf(X \Rightarrow Y)}$$

Алгоритм Apriori весьма прост в реализации, но очень жаден до памяти.

2.2. Алгоритм FP-Growth

В качестве альтернативы Apriori был разработан алгоритм FP-Growth [19], основной идеей которого является построение префиксного дерева из транзакций. Для каждой транзакции считаем поддержку, затем сортируем наборы по убыванию поддержки. Отказываемся от генерации кандидатов и в итоге строим префиксное дерево.

При первом сканировании формируем упорядоченный набор элементов транзакций, сортируя их в транзакции по общей частоте встречаемости и отбирая те из них, которые превышают порог минимальной поддержки

Если для очередного предмета в FP-дереве уже содержится узел, то для предмета не создается новый узел, а индекс существующего увеличивается на 1. В противном случае для этого предмета создается новый узел, и ему присваивается индекс 1.

Для иллюстрации работы алгоритма воспользуемся примером из [19]. Имея набор объектов, вводим порог минимальной поддержки равной 3 и сортируем набор, таблица 2.

Таблица 2. Пример множества наборов для FP-Growth [19]

Номер транзакции *T*	**Набор объектов**	**Отсортированный набор**
1	f; a; c; d; g; i; m; p	f; c; a; m; p
2	a; b; c; f; l; m; o	f; c; a; b;m
3	b; f; h; j; o	f; b
4	b; c; k; s; p	c; b; p
5	a; f; c; e; l; p; m; n	f; c; a; m; p

На рисунке 3 изображено получившееся дерево. В отсортированном наборе транзакции 1 первым идет объект f, он нам ранее не встречался, поэтому создаем узел, делая его из потомков корня. От него создаем узел c и так далее для всех объектов транзакции 1. Переходим к транзакции 2, узлы f, c, a уже есть, поэтому мы проходим по ним, увеличивая их частоту на 1, далее появляется объект b, создаем для него узел. Хотя объект m у нас уже был, но не в этой транзакции, поэтому мы создаем для него потомка от узла b, а не увеличиваем частоту узла, являющегося потомком a. Точно также поступаем с объектом b из транзакции 3. Для объекта c из транзакции 4 у нас нет узла, идущего от корня, поэтому для этой транзакции создаем свою ветку. Транзакция 5 повторяет транзакцию 4, поэтому просто увеличиваем частоту в каждом соответствующем узле.

Для каждого предмета в FP-дереве, представленного своим узлом, можно указать путь, т.е. последовательность узлов, которую надо пройти от корневого узла до узла, связанного с данным предметом. Если предмет представлен в нескольких ветвях FP-дерева, то таких путей будет насколько.

Такой набор путей называется условным базисом предмета (conditional base). Каждый путь в базисе состоит из двух частей — префикса и суффикса. Префикс — это собственно последовательность узлов, которые образуют путь. Суффикс — это сам узел, к которому «прокладывается» путь. Таким образом, в условном базисе все пути будут иметь различные префиксы и одинаковый суффикс.

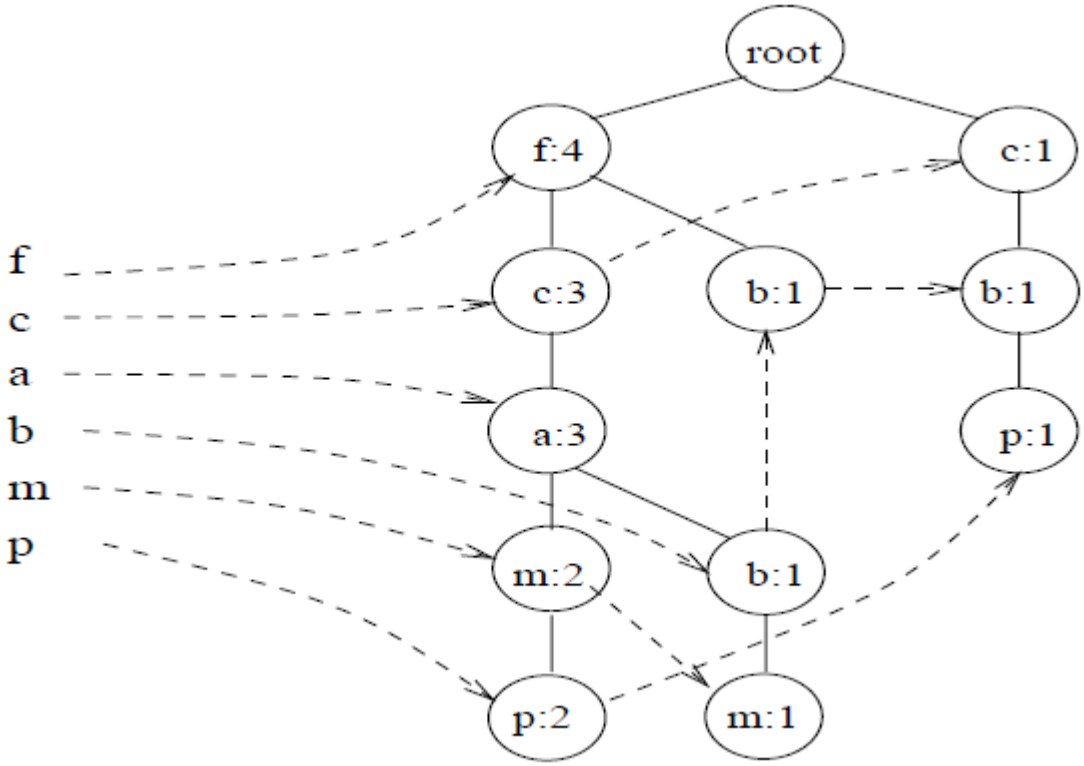


Рисунок 3 – Префиксное дерево для таблицы 2.

Выбираем предмет (например, b) и находим в дереве все пути, которые ведут к узлам этого предмета. Иными словами, для a это будет набор {fb, fcab, cb}. Затем для каждого пути подсчитываем, сколько раз данный предмет встречается в нем, и записываем это в виде (fb, 1), (fcab, 1) и (cb, 1). Удалим сам предмет (суффикс набора) из ведущих к нему путей. После останутся только префиксы: {f, fca, c}. Подсчитаем, сколько раз каждый предмет появляется в префиксах путей, полученных на предыдущем шаге, и упорядочим в порядке убывания этих значений, получив новый набор транзакций.

На его основе построим новое FP-дерево, которое назовем **условным FP-деревом** (conditional FP-tree), поскольку оно связано только с одним объектом (в нашем случае, a).

В условном FP-дереве найдем все предметы (узлы), для которых поддержка (количество появлений в дереве) больше или равна minsupp. Если узел встречается два или более раза, то его индексы, т.е. частоты появлений предмета в условном базисе, суммируются.

Начиная с корня дерева, записываем пути, которые ведут к каждому узлу, для которого поддержка/индекс больше или равны minsupp, возвращаем назад предмет (суффикс), удаленный на шаге 2, и подсчитываем индекс/поддержку, полученную в результате.

3. Кластеризация

Кластеризация – задача разбиения заданной выборки объектов (ситуаций) на подмножества, называемые **кластерами**, так, чтобы каждый кластер состоял из **схожих** объектов, а объекты разных кластеров существенно отличались.

Пусть X — множество объектов, Y — множество меток кластеров. Задана функция расстояния между объектами $d(x, c)$. Имеется конечная обучающая выборка объектов $X^m = \{x_1, \dots, x_m\} \subset X$.

Требуется разбить выборку на непересекающиеся подмножества, называемые кластерами, так, чтобы каждый кластер состоял из объектов, близких по метрике d, а объекты разных кластеров существенно отличались. При этом каждому объекту $x_i \in X^m$ приписывается номер кластера y_i.

Алгоритм кластеризации — это функция $a: X \to Y$, которая любому объекту $x \in X$ ставит в соответствие номер кластера $y \in Y$.

Множество в некоторых случаях известно заранее, однако чаще ставится задача определить оптимальное число кластеров с точки зрения того или иного критерия качества кластеризации.

Главное в кластеризации – выбрать параметры и подобрать метрику, которая будет рассчитываться по значениям параметров и определять близость объектов. При этом данные для кластеризации должны удовлетворять некоторым требованиям:

– показатели не должны сильно коррелировать между собой, мультиколлинеарность не идет на пользу правильному расчету метрики;

– показатели должны быть безразмерными;

– распределение показателей должно быть близко к нормальному;

– показатели должны отвечать требованию «устойчивости», под которой понимается отсутствие влияния на их значения случайных факторов;

– выборка должна быть однородна, не содержать «выбросов».

3.1.k-средних (k-means)

Это самый классический и самый популярный алгоритм кластеризации. Заданное фиксированное число k кластеров наблюдения сопоставляются кластерам так, что средние в кластере (для всех переменных) максимально возможно отличаются друг от друга. Алгоритм стремится минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров: $V = \sum_{i=1}^{k} \sum_{x_j \in S_i} \left(x_j - \mu_i\right)^2$, где k - число кластеров, S_i - полученные кластеры $i = 1,2, \dots, k$ и μ_i - центры масс векторов $x_j \in S_i$. В данном случае используется евклидова метрика, но метрик много и под задачу стоит подбирать свою метрику, евклидова метрика хоть и стоит, что называется, по умолчанию, но не всегда оптимально решает задачу.

Метрики расстояний

Евклидово: $d(x, x') = \sqrt{\sum_{i=1}^{n} (x_i - x_i')^2}$

Квадрат евклидова расстояния. Применяется для придания большего веса более отдаленным друг от друга объектам: $d(x, x') = \sum_{i=1}^{n} (x_i - x_i')^2$.

Манхэттенское расстояние (расстояние L_1). В большинстве случаев эта мера расстояния приводит к таким же результатам, как и для обычного расстояния Евклида. Однако для этой меры влияние отдельных больших разностей (выбросов) уменьшается: $d(x, x') = \sum_{i=1}^{n} |x_i - x_i'|$.

Расстояние Чебышева. Это расстояние может оказаться полезным, когда нужно определить два объекта как «различные», если они различаются по какой-либо одной координате: $d(x, x') = max(|x_i - x'_i|)$.

Степенное расстояние. Применяется в случае, когда необходимо увеличить или уменьшить вес, относящийся к размерности, для которой соответствующие объекты сильно отличаются: $d(x, x') = \sqrt[r]{\sum_{i=1}^{n}(x_i - x'_i)^p}$.

Расстояние Махаланобиса. Учитывает корреляции между переменными: $d(x, x') = \sqrt{(x - x')COV^{-1}(x - x')^T}$.

Обобщенно алгоритм k-means выглядит следующим образом:

1. Первоначальное распределение объектов по кластерам.
    a. Выбирается число k, и на первом шаге эти точки считаются "центрами" кластеров. Каждому кластеру соответствует один центр;
    b. Выбор начальных центроидов может осуществляться следующим образом:
        i. выбор k -наблюдений для максимизации начального расстояния;
        ii. случайный выбор k -наблюдений;
        iii. выбор первых k -наблюдений.
    c. В результате каждый объект назначен определенному кластеру
2. Итеративный процесс:
    a. Вычисляются центры кластеров. Объекты опять перераспределяются;
    b. Процесс вычисления центров и перераспределения объектов продолжается до тех пор, пока не выполнено одно из условий:
        i. кластерные центры стабилизировались, т.е. все наблюдения принадлежат кластеру, которому принадлежали до текущей итерации;
        ii. число итераций равно максимальному числу итераций.

Неприятность заключаются в том, что не гарантируется достижение глобального минимума суммарного квадратичного отклонения V, а только одного из локальных минимумов. Результат зависит от выбора исходных центров кластеров, их оптимальный выбор неизвестен. Большая проблема – это число кластеров, его надо знать заранее.

На сходимость влияет выбор начальных центроидов. Есть хорошая схема, которую почему-то не так часто вспоминают, это **k-means++** [20]. Эта модификация меняет первый шаг алгоритма k-means следующим образом:

1. Выбрать первый центроид случайным образом (среди всех точек);
2. Для каждой точки найти значение квадрата расстояния до ближайшего центроида (из тех, которые уже выбраны) dx^2;
3. Выбрать из этих точек следующий центроид так, чтобы вероятность выбора точки была пропорциональна вычисленному для неё квадрату расстояния:

a. На шаге 2 нужно параллельно с расчётом dx^2 подсчитывать сумму $\mathrm{sum}(dx^2)$;
b. После накопления суммы найти значение $Rnd = random(0,1) * \mathrm{sum}$;
c. Rnd случайным образом укажет на число из интервала $[0; \mathrm{sum})$, и нам остаётся только определить, какой точке это соответствует. Для этого нужно снова начать подсчитывать сумму $\mathrm{s}(dx^2)$ до тех пор, пока сумма не превысит Rnd. Как только это случится, суммирование останавливается, и мы можем взять текущую точку в качестве центроида.

4. Повторять шаги 2 и 3 до тех пор, пока не будут найдены все необходимые центроиды.

3.2.DBSCAN

Если мы не знаем заранее точного количества кластеров, то можно применить алгоритм DBSCAN [21]. Этот алгоритм основан на плотности, т.е. группирует точки, которые тесно расположены. Соответственно, кластеры могут быть произвольной формы. Точки, которые расположены одиноко в областях с малой плотностью, помечаются как выбросы.

DBSCAN быстрее на больших объемах и более устойчив к выбросам. Как для любого алгоритма кластеризации, качество зависит от выбранной метрики расстояния.

Алгоритм различает три вида точек: корневые, граничные и шумовые.

Основные (корневые) – формируют кластер со всеми точками, достижимыми из этой. Точка p является основной, если по меньшей мере $minPts$ точек находятся в ε -окрестности ($U_\varepsilon(x) = \{u \in U : \rho(x,u) \le \varepsilon\}$). Эти точки прямо достижимы из p:

$$|U_\varepsilon(x)| \ge minPts$$

Достижимые по плотности (граничные). Точка q прямо достижима из p, если q находится на расстоянии, не большем ε, от точки p, и p – основная. Точка A q -достижима из p, если имеется путь $p_1, \dots, p_n$ с $p_1 = p$ и $p_n = q$, где каждая точка p_{i+1} достижима прямо из p_i.

Алгоритм DBSCAN:

1. $U \coloneqq X$ – непомеченные, $a \coloneqq 0$ – счетчик кластеров;
2. Пока $U \neq \emptyset$:
   a. Взять случайную точку $x \in U$;
   b. Если $|U_\varepsilon(x)| < minPts$, то:
      i. Пометить x как, возможно, шумовую
   c. Иначе:
      i. Создать новый кластер $K \coloneqq U_\varepsilon(x)$, $a = a + 1$;
      ii. Для всех $x' \in K$, не помеченных или шумовых;
      iii. Если $|U_\varepsilon(x)| \ge minPts$, то $K \coloneqq K \cup U_\varepsilon(x)$ иначе пометить x' как граничную точку кластера K;

iv. $a_i := a$ для всех $x_i \in K$;
v. $U := U \backslash K$.

Параметр ε может быть выбран с помощью графа ближайших соседей. $minPts$ может быть получен из размерности D набора данных $minPts \geq D + 1$.

На рисунке 4 показано сравнение разных алгоритмов кластеризации на разных конфигурациях данных.

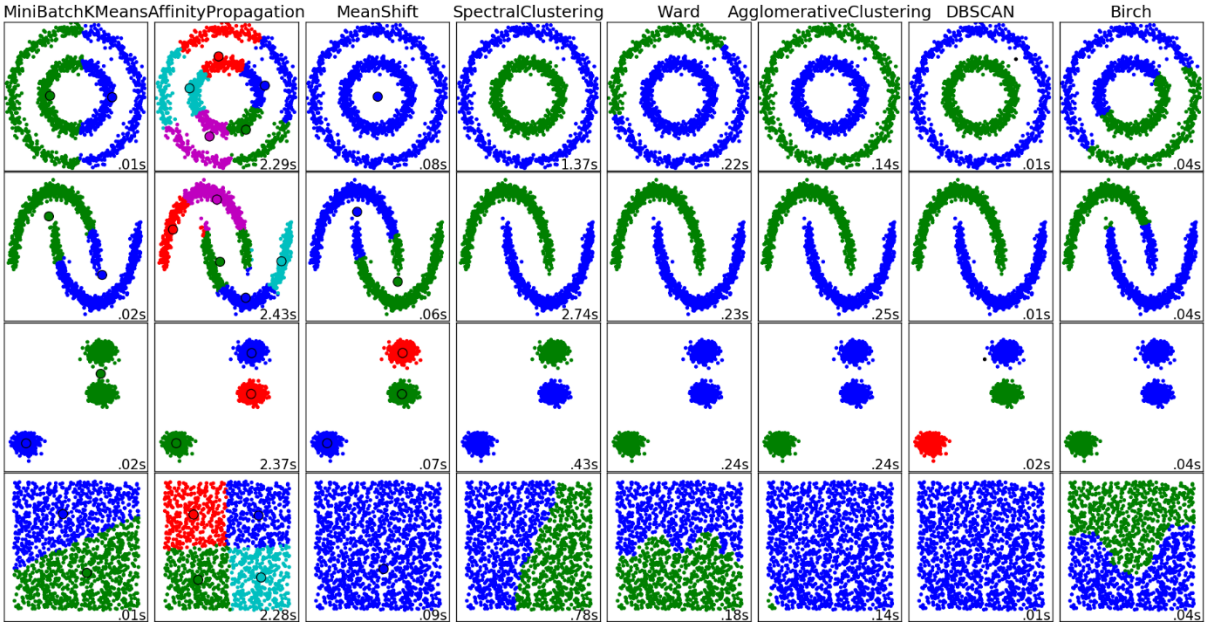


Рисунок 4 – Сравнение алгоритмов кластеризации [22]

3.3.Нечеткая кластеризация C-средних (Fuzzy C-means)

До этого мы рассматривали только случаи, когда данные могут принадлежать только одному кластеру, но бывают ситуации, когда данные могут принадлежать нескольким кластерам одновременно. Для этого был разработан алгоритм Fuzzy C-means на основе нечеткой логики [23, 24].

Сначала выбираем число классов M, меру нечеткости $1 < m < \infty$, функцию расстояний $d(c, x)$ (обычно $d(c, x) = \|X - C\|^2$) и критерий окончания поиска $0 < \varepsilon < 1$

1. Задаем матрицу $U^0 = \{u_{ij}(x_i, c_j) : x_i \in X, c_j \in C\}$ весов принадлежности точки $x_i \in X$ к кластеру j с центром c_j для всех точек и кластеров:
   a. можно взять принадлежности к кластеру из k-means;
   b. рандомно, ограничения $0 < u_{ij} < 1$, $0 < \sum_{i=1}^{N} u_{ij} < 1$ для $\forall 0 \leq j \leq C$ и $\sum_{j=1}^{C} u_{ij} = 1$ для $\forall 0 \leq i \leq N$);
2. Вычисляем центроиды:

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

3. Перевычисляем веса:

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

4. Проверяем функцию потерь $\|U^k - U^{k-1}\| < \varepsilon$ (чаще всего достаточно $max_{ij} \|u_{ij}^k - u_{ij}^{k-1}\| < \varepsilon$):
    a. если да, то заканчиваем, если нет, то переходим к шагу 2.

3.4.Метрики качества

Большие трудности вызывает оценка качества кластеризации. Предложено большое количество метрик, которые можно разделить на внутренние (например, силуэт выборки и индекс Дэвиса-Болдуина) и внешние (используют информацию об истинном разбиении на кластеры, например, ARI, AMI и самая популярная V-мера)

Силуэт сначала определяется отдельно для каждого объекта в выборке.

Пусть a – среднее расстояние от данного объекта до объектов из того же кластера, b – среднее расстояние от данного объекта до объектов из ближайшего кластера. Тогда силуэт объекта

$$s = \frac{b - a}{\max(a, b)}$$

Далее рассчитывается силуэт выборки – средняя величина силуэтов объектов данной выборки.

Силуэт показывает, насколько среднее расстояние до объектов своего кластера отличается от среднего расстояния до объектов других кластеров. Данная величина лежит в диапазоне [-1, 1].

– значения, близкие к -1 – плохая (разрозненная) кластеризация;

– значения, близкие к нулю – кластеры пересекаются и накладываются друг на друга;

– значения, близкие к 1 – «плотные» четко выделенные кластеры.

Индекс Дэвиса-Болдуина вычисляет компактность как расстояние от объектов кластера до их центроидов, а отделимость - как расстояние между центроидами. для роста качества кластеризации он должен минимизироваться. На практике считается, что силуэт – более качественная метрика.

$$DB(C) = \frac{1}{K} \sum_{c_k \in C} \max_{c_l \in C \setminus c_k} \left(\frac{S(c_k) + S(c_l)}{\|\bar{c}_k - \bar{c}_l\|} \right)$$

$$S(c_k) = \frac{1}{|c_k|} \sum_{x_i \in c_k} \|x_i - \bar{c}_k\|$$

ARI

Пусть n – число объектов в выборке, a – число пар объектов, имеющих одинаковые метки и находящихся в одном кластере, b – число пар объектов,

имеющих различные метки и находящихся в разных кластерах. Тогда Rand index (RI) выражает схожесть двух разных кластеризаций

$$RI = \frac{2(a+b)}{n(n-1)}$$

Adjusted Rand index

$$ARI = \frac{RI - E[RI]}{\max RI - E[RI]}$$

AMI

Взаимная информация

$$MI(U,V) = \sum_{i=1}^{|U|}\sum_{j=1}^{|V|}\frac{\left|U_i \cap V_j\right|}{N}\log_2\frac{N\left|U_i \cap V_j\right|}{|U_i|\left|V_j\right|}$$

Adjusted Mutual Information

$$AMI = \frac{MI - E[MI]}{avg\big(H(U), H(V)\big) - E[MI]}$$

V-мера

Сначала рассчитываем однородность и полноту.

Однородность (K – результат кластеризации, C – истинное разбиение)

$$h = 1 - \frac{H(C|K)}{H(C)}$$

$$H(C|K) = -\sum_{k=1}^{K}\sum_{c=1}^{C}\frac{a_{ck}}{N}\log_2\left(\frac{a_{ck}}{\sum_{c=1}^{C} a_{ck}}\right)$$

$$H(C) = -\sum_{c=1}^{C}\frac{\sum_{k=1}^{K} a_{ck}}{C}\log_2\left(\frac{\sum_{k=1}^{K} a_{ck}}{C}\right)$$

Полнота

$$c = 1 - \frac{H(K|C)}{H(K)}$$

После этого, можно вывести V-меру

$$v = 2\frac{hc}{h+c}$$

4. Линейные методы

4.1.Линейная регрессия

Линейная регрессия хорошо описана в курсе статистики, здесь ограничимся кратким описанием, которое поможет нам перейти к логистической регрессии, которая для машинного обучения уже более интересна.

Итак, у нас есть некоторые данные, по которым мы предполагаем линейную зависимость, рисунок 4. Соответственно, нам надо эту зависимость восстановить

в виде параметров функции. К слову, линейная регрессия очень чувствительна к мультиколлинеарности.

Пусть имеется обучающая выборка $X^l = (x_i, y_i)_{i=1}^l$, $x_i \in \mathbb{R}^n$, $y_i \in \mathbb{R}$ и тестовая выборка $X^k = (\widetilde{x_i}, \widetilde{y_i})_{i=1}^k$.

Модель линейной регрессии ($w \in \mathbb{R}^n$) представляет собой параметрическую функцию:

$$a(x, w) = \sum_{j=1}^{n} w_j f_j(x)$$

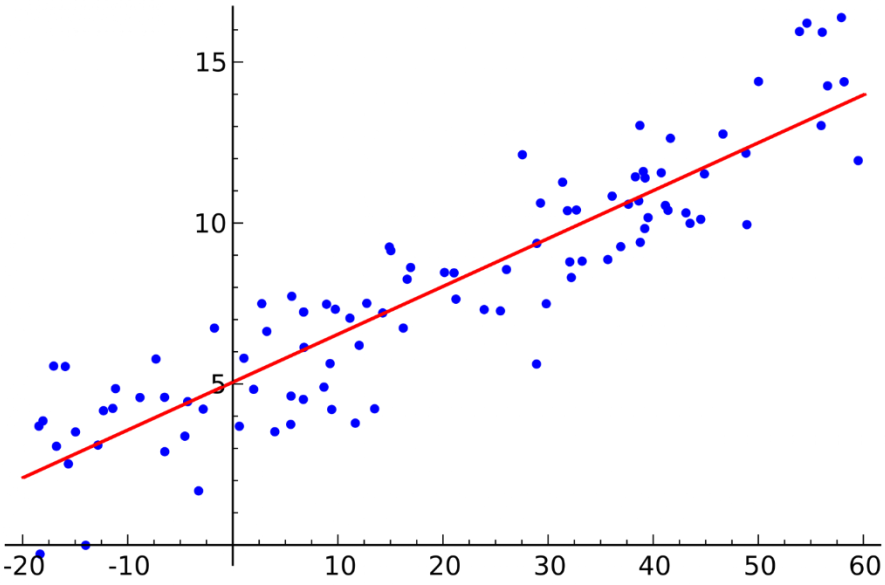


Рисунок 4 – Пример линейной регрессии

Чтобы наша модель аппроксимировала данные из обучающей выборки наилучшим образом, мы должны подобрать вектор весов W. Вручную это делать достаточно муторно. Задача подбора значений вектора весов при наличии функционала, определяющего стоимость (он так и называется, функция стоимости или функция потерь), является задачей безусловной оптимизации.

Линейная регрессия – это простая модель, поэтому мы можем применить метод наименьших квадратов. В качестве функции потерь используется среднеквадратическая ошибка (MSE) $L(a, y) = (a - y)^2$, которую мы минимизируем, что по сути своей является уменьшением ошибки на обучающей выборке:

$$Q(w) = \sum_{i=1}^{l} (a(x_i, w) - y_i)^2 \to \min_w$$

Тогда проверка качества модели на тестовой выборке будет представлять собой сравнение прогноза модели и соответствующего реального значения из тестовой выборки:

$$\bar{Q}(w) = \frac{1}{k}\sum_{i=1}^{k}(a(\tilde{x}_i, w) - \tilde{y}_i)^2$$

Перейдем к матричным обозначениям:

$$A = \begin{pmatrix} a_1(x_1) & \cdots & a_n(x_1) \\ \vdots & \ddots & \vdots \\ a_1(x_l) & \cdots & a_n(x_l) \end{pmatrix} Y = \begin{pmatrix} y_1 \\ \vdots \\ y_l \end{pmatrix} W = \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}$$

Соответственно, обучение запишется в виде:

$$Q(w) = \|AW - Y\|^2 \to \min_w$$

Необходимым условием минимума будет равенство градиента по вектору W нулю в точке минимума

$$\frac{\partial Q}{\partial w}(w) = 2A^T(AW - Y) = 0$$

Получаем систему

$$A^T AW = A^T Y$$

Выражаем искомый вектор весов

$$W^* = (A^T A)^{-1} A^T Y = A^+ Y$$

$$Q(W^*) = \|P_A Y - Y\|^2$$

где $P_A = AA^+ = A(A^T A)^{-1}A^T$ - проекционная матрица.

В таком виде можно заметить сходство матричного уравнения поиска вектора весов с сингулярным разложением матрицы (SVD).

Произвольная $l \times n$ матрица A представима в виде **сингулярного разложения**, т.е. произведения трех матриц:

$$A = VDU^T.$$

Основные свойства:

$-$ $l \times n$ матрица $V = (v_1, \dots, v_n)$ ортогональна, $V^T V = I_n$, v_j - собственные векторы матрицы AA^T;

$-$ $n \times n$ матрица $U = (u_1, \dots, u_n)$ ортогональна, $U^T U = I_n$, u_j - собственные векторы матрицы $A^T A$;

$-$ $n \times n$ матрица $D = diag\left(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_n}\right)$ диагональна, λ_j - собственные значения матриц $A^T A$ и AA^T.

Применяя сингулярное разложение, получаем решение:

$$A^+ = (UDV^T VDU^T)^{-1} UDV^T = UD^{-1}V^T = \sum_{j=1}^{n} \frac{1}{\sqrt{\lambda_j}} u_j v_j^T$$

$$W^* = A^+ Y = UD^{-1}V^T Y = \sum_{j=1}^{n} \frac{1}{\sqrt{\lambda_j}} u_j \left(v_j^T y\right)$$

$$AW^* = P_A Y = (VDU^T)UD^{-1}V^T Y = VV^T Y = \sum_{j=1}^{n} v_j \left(v_j^T y\right)$$

$$\|W^*\|^2 = \|UD^{-1}V^T Y\|^2 = \|D^{-1}V^T Y\|^2 = \sum_{j=1}^{n} \frac{1}{\lambda_j} \left(v_j^T y\right)^2$$

4.2.Линейная классификация

Классификация объектов, как мы условились в самом начале, имеет дело с признаками этих объектов, по которым происходит процесс разделения на классы. Так как объекты могут быть представлены точками в многомерном пространстве, размерность которого определяется количеством признаков, то можно применить математическую разделимость. Некоторые не пересекающиеся множества могут быть некоторым образом разделены в пространстве, но принцип разделимости требует доказательства обоснованности применения в каждом конкретном случае.

Различают линейную и нелинейную разделимость. Линейная разделимость предполагает, что между объектами разных классов можно провести разделяющую гиперплоскость. На рисунке 5 визуализированы объекты, принадлежащие двум классам, как видно, мы можем спокойно провести линию, которая их разделит, это и есть пример линейной разделимости.

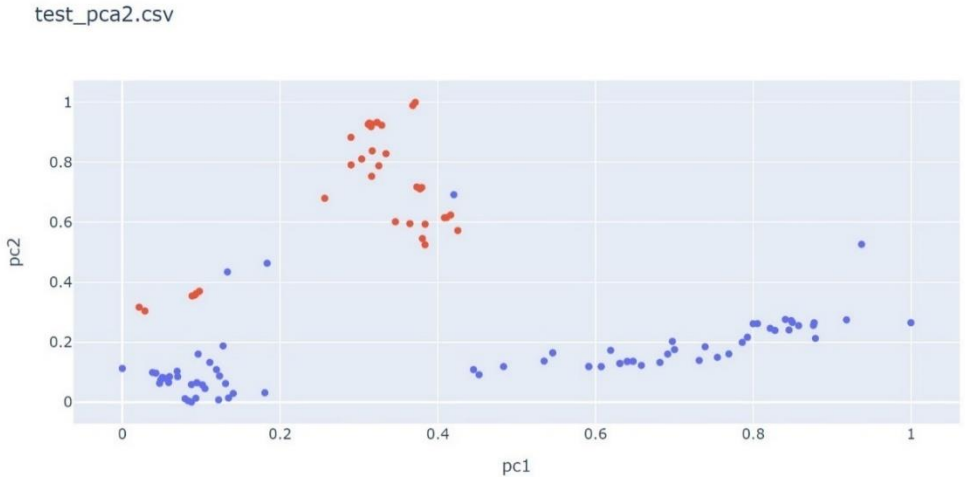


Рисунок 5 – Пример линейной разделимости двух классов

На рисунке 6 показаны объекты двух классов, которые с помощью линии мы разделить не можем. Это, как нетрудно догадаться, нелинейная разделимость.

Несмотря на всю сложность, нелинейно разделимые данные все же можно классифицировать. Об этом говорит теорема Ковера, которая обосновывает, почему же работают такие методы, как нейронные сети.

Теорема Ковера о разделимости гласит, что нелинейное преобразование сложной задачи классификации образов в пространство более высокой размерности повышает вероятность линейной разделимости образов.

Но начнем обзор методов классификации мы все же с линейной разделимости, а именно с логистической регрессии. Логично, что она предполагает бинарную классификацию ($y_i \in \{-1, +1\}$) и пороговую функцию потерь $L(a, y) = [ay < 0]$.

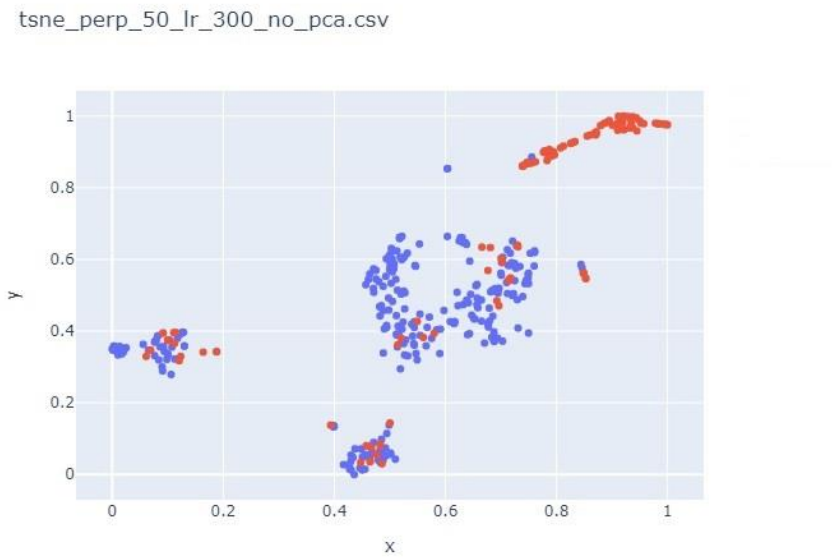


Рисунок 6 – Пример нелинейной разделимости двух классов

Модель линейной классификации ($w \in \mathbb{R}^n$), знак показывает принадлежность к тому или иному классу:

$$a(x, w) = sign \sum_{j=1}^{n} w_j f_j(x)$$

Такую модель можно обучить с помощью минимизации эмпирического риска, но она будет склонна к переобучению, и ее тяжело интерпретировать в плане отступа объекта от разделяющей плоскости. Можно посмотреть на эту задачу с позиции предсказания вероятности отнесения к классу +1, $P\{y = 1|x\} = f(z)$. И в этом поможет логистическая регрессия.

Пусть $P(X)$ – вероятность события X. Тогда отношение вероятностей того произойдет событие или нет, будет $OR(X) = \frac{P(X)}{1-P(X)}$. Но $OR(X) \in [0, \infty)$, нам же нужен диапазон $(-\infty, \infty)$, для этого возьмем логарифм $\ln OR(X)$.

Разделяющая гиперплоскость задается функцией $z = w_0 + w_1 x_1 + w_2 x_2$

Тогда отношение шансов $OR(X) = e^z$. Имея прогноз шанса отнесения к классу +1, можно вычислить вероятность из обратной функции отношения шансов:

$$P = \frac{OR}{1 + OR} = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}} = \sigma(w^T x)$$

Таким образом, мы получили сигмоиду (функция логистического отклика, обратное логит-преобразование). Так как сигмоида является основой для построения формального нейрона, то можно считать, что каждый отдельный нейрон в слое строит логистическую регрессию.

Веса логистической регрессии подбираются через максимизацию правдоподобия для распределения Бернулли.

Пусть $X \times Y$ – выборка с плотностью $p(x, y|w) = P(y|x, w)p(x)$. Тогда оценка максимального правдоподобия для w будет (другими словами, мы максимизируем отступ на объекте x_i):

$$\prod_{i=1}^{l} p(x_i, y_i|w) = \prod_{i=1}^{l} P(y_i, x_i|w)p(x_i) \to \max_w$$

Оптимизировать произведение неудобно, поэтому переходят к логарифму правдоподобия (log-likelihood, log-loss):

$$L(w) = \sum_{i=1}^{l} \log P(y_i, x_i|w) \to \max_w$$

Обучение логистической регрессии будет максимизацией функции правдоподобия:

$$Q(w) = \sum_{i=1}^{l} \log\big(1 + exp(-\langle w, x_i\rangle y_i)\big)$$

Подбор весов можно осуществить и с помощью **градиентного спуска**. Тогда задача обучения эквивалентна *поиску глобального безусловного минимума функции потерь*. Кратко вспомним, что по этому поводу говорит теория оптимизации.

Пусть у нас имеется непрерывно дифференцируемая функция $f(x)$. Численное решение задачи $f(x^*) = \min_{x \in \mathbb{R}} f(x)$ связано с построением последовательности $\{x^k\}$ точек, обладающих свойством $f(x^{k+1}) < f(x^k)$. Преобразование точки x^k в x^{k+1} представляет собой итерацию.

Общее правило построения последовательности

$$x^{k+1} = x^k + t_k d^k$$

x^0 - начальная точка поиска (задается исходя из условий задачи), d^k - приемлемое направление перехода (направление спуска), t_k - величина шага.

Направление спуска d^k должно удовлетворять условию $(\nabla f(x^k), d^k) < 0$, обеспечивающему убывание функции, примером является направление вектора антиградиента.

Градиентом $\nabla f(x)$ непрерывно дифференцируемой функции $f(x)$ в точке x называется вектор-столбец, элементами которого являются частные производные первого порядка, вычисленные в данной точке:

$$\nabla f(x) = \begin{pmatrix} \dfrac{\partial f(x)}{\partial x_1} \\ \dfrac{\partial f(x)}{\partial x_2} \\ \vdots \\ \dfrac{\partial f(x)}{\partial x_n} \end{pmatrix}$$

Градиент функции направлен по нормали к поверхности уровня, т.е. перпендикулярно к касательной плоскости, проведенной в точке x, в сторону наибольшего возрастания функции в данной точке.

Поверхностью уровня функции $f(x)$ называется множество точек, в которых функция принимает постоянное значение.

В случае машинного обучения вместо x подставляем вектор весов w. Для нахождения точки глобального минимума используют **метод наискорейшего градиентного спуска**.

Пусть дана функция $f(x)$, ограниченная снизу на множестве $\mathbb{R}^n$ и имеющая непрерывные частные производные во всех его точках.

Требуется найти локальный минимум функции (в нашем случае, функции потерь) на множестве допустимых решений $X = \mathbb{R}^n$, т.е. найти точку $x^* \in \mathbb{R}^n$, что $f(x^*) = \min\limits_{x \in \mathbb{R}^n} f(x)$.

Стратегия решения задачи состоит в построении последовательности точек $\{x^k\}$ точек, обладающих свойством $f(x^{k+1}) < f(x^k)$. Точки последовательности вычисляются по правилу

$$x^{k+1} = x^k - t_k \nabla f(x^k)$$

где x^0 - начальная точка поиска (в нашем случае, начальные значения весов), величина шага t_k определяется для каждого значения k из условия:

$$\varphi(t_k) = f\left(x^k - t_k \nabla f(x^k)\right) \to \min_{t_k}$$

Решения этой задачи может осуществляться с использованием необходимого условия минимума $\frac{d\varphi}{dt_k} = 0$ с последующей проверкой достаточного условия минимума $\frac{d^2\varphi}{dt_k^2} > 0$.

4.3.Оценка качества классификации

Для того, чтобы избежать неприятности переобучения, проводится оценка качества модели на тестовой выборке. Но даже в этом случае может возникнуть смещение оценок, поэтому прибегают к технике **кросс-валидации**, которая также известна под именами скользящего контроля или перекрестной проверки. Осуществить такой контроль можно с помощью двух методов: метод отложенных данных (holdout method) и контроль по k-блокам (k-fold cross-validation).

Метод отложенных данных предполагает разделение обучающей и тестовой выборок в соотношении 70-30 (чаще всего) или 60-40 или 80-20. Оценка ошибки будет близка к ошибке модели на новых данных, но при этом сильно зашумлена. Для борьбы с шумом многократно случайно разделяют обучающую и тестовую выборку, параметр ошибки при этом усредняют. Но в процессе итераций каждая точка данных будет попадать в тестовое подмножество различное число раз, что может привести к смещению. Не всегда такая борьба с шумом имеет смысл, лучше уж прибегнуть к контролю по k-блокам.

Контроль по k-блокам показан на рисунке 7. Данные случайным образом делятся на k непересекающихся подмножеств (5, 10 или 20). Затем на k−1 частях данных производится обучение модели, а оставшаяся часть данных используется для тестирования. Процедура повторяется k раз. После циклического перебора всех k подмножеств полученная оценка усредняется.

С процедурой тестирования качества разобрались. Теперь рассмотрим метрики качества. Они должны быть независимы от параметров модели и позволять оценивать объективно качество предсказания меток классов.

Для вычисления разных метрик качества составляют матрицу неточностей (confusion matrix), таблица 3, которая позволяет структурировать количество верно предсказанных меток и ошибочных меток классов. В ячейках матрицы указывается количество объектов, отнесенных к тому или иному классу.

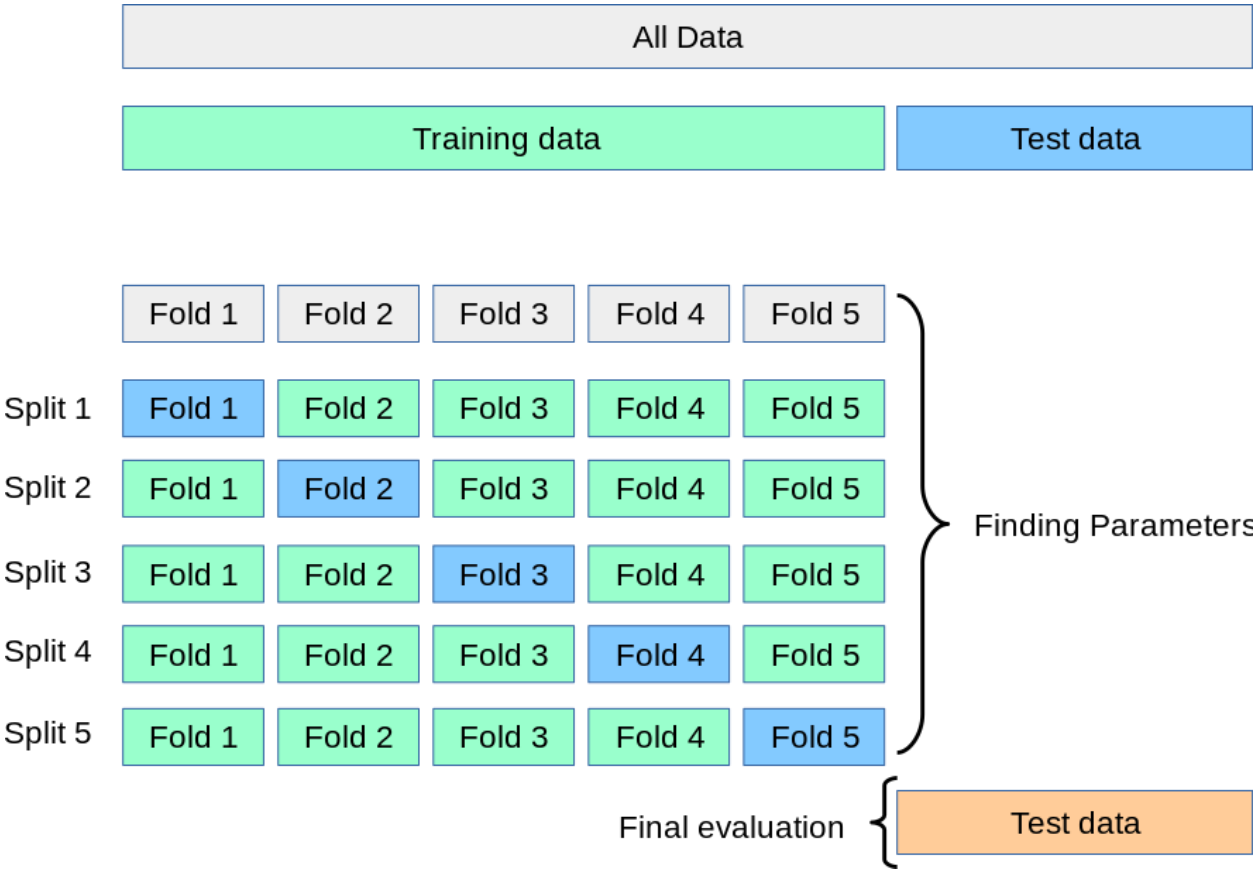


Рисунок 7 – Схема метода контроля по k-блокам [25]

Таблица 3. Матрица неточностей

		Предсказанные моделью классы	
		A(x)=1	A(x)=-1
Реальные классы	Y=1	True positive	False negative
	Y=-1	False positive	True negative

Ошибки классификации бывают двух видов: False Positive и False Negative. В статистике первый вид ошибок называют ошибкой I-го рода (нулевая гипотеза

неверно отвергнута), а второй — ошибкой II-го рода (нулевая гипотеза неверно принята).

Общая сумма протестированных объектов складывается из значений матрицы неточностей.

$$N = TP + FP + FN + TN$$

По матрице неточностей можно вычислить несколько весьма популярных метрик: достоверность (самая популярная метрика), полноту и точность (всегда идут парой) и другие, более специфичные для разных областей наук метрики.

Достоверность (в других источниках, правильность) показывает долю правильных ответов.

$$accuracy = \frac{TP + TN}{N}$$

Оценка модели только по достоверности удовлетворит нас, если классы имеют одинаковое значение, и не позволяет нам учесть неравную важность FP и FN, что критично, например, в медицине. Для этого используется полнота и точность.

Полнота (способность обнаруживать данный класс). В медицине этот показатель называется чувствительность (sensitivity). Высокая чувствительность показывает правильность выявления больных пациентов, у которых действительно есть это заболевание.

$$recall = \frac{TP}{TP + FN}$$

Точность (способность отличать этот класс от других):

$$precision = \frac{TP}{TP + FP}$$

Специфичность (specificity), чаще всего применяется в медицинской статистике. Высокая специфичность позволяет отсеять людей, у которых действительно нет этого заболевания.

$$TNR = \frac{TN}{TN + FP}$$

Коэффициент корреляции Мэтьюса (в статистике известен как фи-коэффициент) может применяться как мера качества для бинарной классификации при высоком дисбалансе классов. Принимает значение в диапазоне [-1,1], где 1 – идеальное предсказание, 0 – случайное предсказание, -1 – полное расхождение между предсказанием и наблюдением.

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Помимо приведенных показателей, по матрице неточностей можно рассчитать очень много других, специфичных для своих предметных областей, например, FDR для геномики.

Полнота и точность противоречат друг другу. Всегда можно повысить полноту до 1 при очень низкой точности. Баланс между этими двумя показателями отображает **F-мера**, которая является средним гармоническим полноты и точности:

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}},\ \alpha \in [0,1]$$

$$F = \frac{(\beta^2+1)PR}{\beta^2 P + R},\ \beta^2 = \frac{(1-\alpha)}{\alpha}, \beta^2 \in [0,\infty)$$

Обычная зависимость полноты и точности показана на рисунке 8, но такое бывает не всегда, в некоторых задачах необходимо отдавать предпочтение полноте ($\beta > 1$) или точности ($\beta < 1$). В противном случае можно использовать **сбалансированную F-меру** ($\beta = 1$):

$$F_1 = \frac{2PR}{P+R}$$

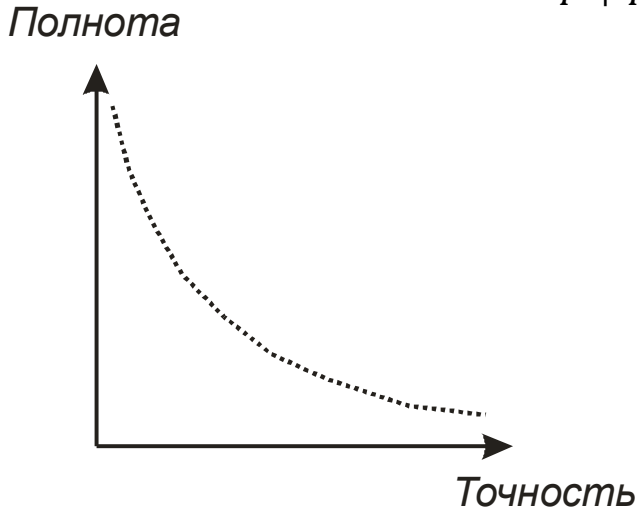


Рисунок 8 – Зависимость полноты и точности

Другой весьма популярный способ оценить качество классификационных моделей – это **ROC-кривые** (ROC – рабочая характеристика приемника, метод был разработан для военных радаров в 1941 году, отсюда и название).

В случае бинарной классификации получаются вероятности принадлежности объектов к классам, но, чтобы отнесение объекта к классу имело место быть, необходимо выбрать порог, который будет разделять классы, рисунок 9. Интуитивно хочется взять порог, равный 0,5, но это не всегда возможно, особенно на несбалансированной по классам выборке. В этом случае порог необходимо двигать в сторону того или иного класса. Также на значение порога может повлиять требования бизнес-процесса, в который мы встраиваем модель.

Поэтому, если мы хотим оценить качество работы модели без привязки к конкретному значению порога, можно применить ROC-кривую, рисунок 10. Каждая точка на кривой соответствует некоторому значению порога. Кривая строится в координатах FPR (ось абсцисс) и TPR (ось ординат), обе эти величины растут при уменьшении порога.

Клиент	Вероятность невозврата
Mike	0.78
Jack	0.45
Larry	0.13
Kate	0.06
William	0.03
Jessica	0.02

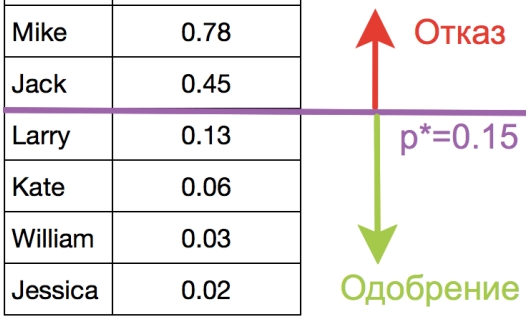


Рисунок 9 – Выбор порога для разделения двух классов

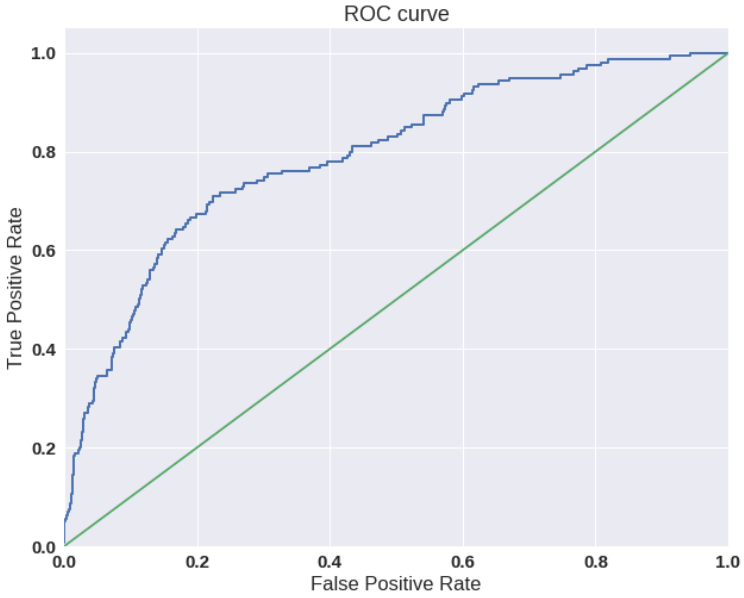


Рисунок 10 – Пример ROC-кривой

TPR (true positive rate) показывает чувствительность алгоритма классификации и равна recall. FPR (false positive rate) отображает долю объектов negative класса, предсказанных неверно:

$$FPR = \frac{FP}{FP + TN}$$

ROC-кривая строится для бинарной классификации, в случае многоклассовой – для каждого класса своя ROC-кривая.

Сравнить несколько моделей по ROC-кривым визуально бывает трудно, поэтому применяют показатель «площади под кривой» (**AUC**), дающий численную оценку. AUC эквивалентна вероятности, что классификатор присвоит больший вес случайно выбранной положительной сущности, чем случайно выбранной отрицательной

$$A = \int_{-\infty}^{\infty} y(T)x'(T)dT = \int_{-\infty}^{\infty} TPR(T)FPR'(T)dT = \int_{-\infty}^{\infty} TPR(T)P_0(T)dT = \langle TPR \rangle$$

В идеальном случае $A = 1$. Чем она больше, тем алгоритм лучше работает. Если $A = 0.5$ (что на ROC-кривой отображается совпадением графика и

диагонали), то наша модель, по сути, осуществляет случайное угадывание. Если $A < 0.5$, то классификатор действует с точностью до наоборот.

Еще одним графиком, позволяющим оценить качество модели, является **DET-кривая** (detection error tradeoff, компромисс ошибки обнаружения), отображающая частоту ложных отклонений по сравнению с частотой ложных приемов. Изначально DET-кривая была предложена для задач распознавания речи [26], но может быть применена в любых задачах детекции.

Принцип построения похож на построение ROC-кривой, отличаются только оси: по оси абсцисс откладывается FPR, по оси ординат – FNR (False negative rate).

$$FNR = \frac{FN}{FN + TP} = 1 - TPR = 1 - recall$$

Для задач регрессии есть свои метрики качества модели. Наиболее популярны средняя абсолютная ошибка (MAE), среднеквадратическая ошибка (MSE), RMSE, коэффициент детерминации (R^2).

Средняя абсолютная ошибка (MAE)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|a(x_i) - y_i|$$

Среднеквадратическая ошибка (MSE) применяется, когда надо подчеркнуть большие ошибки, но поэтому она более чувствительна к выбросам, чем MAE. Чем ошибка меньше, тем лучше, но важно учитывать масштаб данных. Чтобы MSE имела размерность исходных данных, из нее извлекают корень и получают RMSE, что затрудняет сравнение на разных наборах. Избежать этого недостатка позволяет коэффициент детерминации.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(a(x_i) - y_i)^2$$

$$RMSE = \sqrt{MSE}$$

Коэффициент детерминации (R^2) – доля дисперсии, объясненная моделью, в общей дисперсии целевой переменной. Чем ближе к 1, тем модель лучше объясняет данные. Модели с коэффициентов детерминации больше 0,8 можно считать хорошими. $\bar{y}$ – среднее арифметическое y_i.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(a(x_i) - y_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

SMAPE (Symmetric Mean Absolute Percentage Error) рассматривает не абсолютные, а относительные ошибки на объектах:

$$SMAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{2|a(x_i) - y_i|}{y_i + a(x_i)}$$

5. Машины опорных векторов

5.1. Метод опорных векторов (SVM)

Продолжаем рассматривать линейную классификацию. И в этой главе узнаем про самый эффективный метод линейной классификации – машины опорных векторов, которые являются развитием теории распознавания образов Вапника и Червоненкиса. В настоящее время машины опорных векторов применяются не только для классификации, но и для регрессии, но об этом чуть позже.

Наша задача, по-прежнему – построить оптимальную разделяющую гиперплоскость, которая разделит два класса наилучшим образом. Для этого алгоритм ищет точки данных, которые ближе всего к гиперплоскости, они называются *опорными векторами*. Цель алгоритма – максимизировать расстояние *зазора* – расстояние между опорными векторами и гиперплоскостью. Чем больше зазор, тем меньше будет средняя ошибка классификатора.

Гиперплоскость – n-1 мерная подплоскость в n-мерном евклидовом пространстве, которая разделяет пространство на две отдельные части.

Все основные элементы метода опорных векторов показаны на рисунке 11.

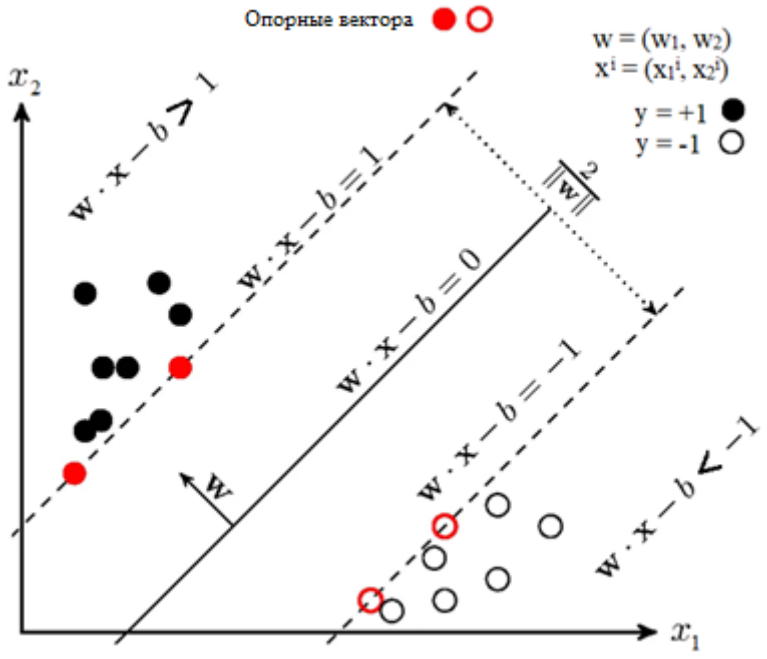


Рисунок 11 – Данные, опорные вектора и разделяющая гиперплоскость в построении SVM [27]

Для того, чтобы построить машину опорных векторов, надо найти уравнение разделяющей гиперплоскости для двух классов $F(x) = sign(w^T x - b)$, где w – вектор нормали к разделяющей гиперплоскости. А для этого надо найти w, b, которые максимизируют зазор.

Дальше понадобится **теорема Каруша-Куна-Таккера**, поэтому вспомним ее.

Пусть поставлена задача нелинейного программирования с ограничениями

$$\begin{cases} f(x) \to \min\limits_{x \in X} \\ g_i(x) \le 0, i = 1 \dots m \\ h_j(x) = 0, j = 1 \dots k \end{cases}$$

Если x – точка локального минимума при наложенных ограничениях, то существуют такие множители μ_i и λ_j, что для функции Лагранжа $L(x; \mu, \lambda)$ выполняются условия:

$$\begin{cases} \dfrac{\partial L}{\partial x} = 0, \;\; L(x; \mu, \lambda) = f(x) + \sum\limits_{i=1}^{m} \mu_i g_i(x) + \sum\limits_{j=1}^{k} \lambda_j h_j(x) \\ g_i(x) \le 0, h_j(x) = 0 \;\; (\text{исходные ограничения}) \\ \mu_i \ge 0 \;\; (\text{двойственные ограничения}) \\ \mu_i g_i(x) = 0 \;\; (\text{условие дополняющей нежесткости}) \end{cases}$$

При этом искомая точка является седловой точкой функции Лагранжа: минимумом по x и максимумом по двойственным переменным μ.

Классический метод опорных векторов был разработан для линейной разделимости, который впоследствии был обобщен до линейной неразделимости. Рассмотрим оба варианта.

Случай линейной разделимости

Задача – минимизировать $\|w\|^2$ при условии $y_i(w \cdot x_i - b) \ge 1$

По теореме Каруша-Куна-Таккера она эквивалентна двойственной задаче поиска седловой точки функции Лагранжа (λ – вектор двойственных переменных)

$$\begin{cases} L(w, b, \lambda) = \dfrac{1}{2}\|w\|^2 - \sum\limits_{i=1}^{n} \lambda_i (y_i(w \cdot x_i - b) - 1) \to \min\limits_{w,b} \max\limits_{\lambda} \\ \lambda_i \ge 0 \end{cases}$$

Сведем эту задачу к эквивалентной задаче квадратичного программирования

$$L(w, b, \lambda) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j (x_i \cdot x_j) \to \max_{\lambda}$$

Решить эту задачу можно градиентными методами или покоординатным спуском. Если мы ее решили, то w и b можно найти по формулам (следует из необходимых условий существования седловой точки):

$$w = \sum_{i=1}^{n} \lambda_i y_i x_i$$
$$b = w \cdot x_i - y_i$$

Случай линейной неразделимости (soft-margin SVM)

Для того, чтобы алгоритм мог работать в случае, если классы линейно неразделимы, позволим ему допускать ошибки на обучающей выборке с

величиной ошибки e_i. На практике делают так из-за того, что гарантировать линейную разделимость невозможно.

Задача – минимизировать $\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} e_i$ при условии $y_i(w * x_i - b) \geq 1 - e_i$, где C - параметр настройки метода, который позволяет регулировать отношение между максимизацией ширины разделяющей полосы и минимизацией суммарной ошибки

$$\begin{cases} L(w,b,e,\lambda,\eta) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \lambda_i (y_i(w \cdot x_i - b) - 1) - \\ \quad - \sum_{i=1}^{n} e_i(\lambda_i + \eta_i - C) \to \min_{w,b,e} \max_{\lambda,\eta} \\ \lambda_i, e_i, \eta_i \geq 0 \end{cases}$$

Сведем эту задачу к эквивалентной задаче квадратичного программирования

$$\begin{cases} L(w,b,\lambda) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j (x_i \cdot x_j) \to \max_{\lambda} \\ 0 \leq \lambda_i \leq C \\ \sum_{j=1}^{n} \lambda_i y_i = 0 \end{cases}$$

Эта задача решается также с помощью градиентных методов или покоординатного спуска. Если мы решили данную задачу, то w и b можно найти по формулам:

$$w = \sum_{i=1}^{n} \lambda_i y_i x_i$$

$$b = w \cdot x_i - y_i$$

Решение задачи оптимизации с мягким зазором разделяет обучающую выборку на три множества:

$\lambda_i = 0$ – те, что лежат вне или на границе зазора;

$0 < \lambda_i < C$ – опорные векторы на границе зазора;

$\lambda_i = C$ – внутри или на границе зазора.

Ядерные SVM

В 1992 году была предложена вариация машины опорных векторов для нелинейной классификации. Все элементы обучающей выборки вкладываются в пространство более высокой размерности с помощью специального отображения $\varphi: \mathbb{R}^n \to X$. При этом отображение φ выбирается так, чтобы в новом пространстве X выборка была линейно разделима (согласно теореме Ковера о разделимости). Тогда уравнение гиперплоскости примет вид $F(x) = sign(w^T \varphi(x) - b)$.

Обозначим через выражение $k(x, x') = \langle \varphi(x), \varphi(x') \rangle$ - ядро классификатора

$$L(w,b,\lambda) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j K(x_i, x_j) \to \max_{\lambda}$$

$$0 \le \lambda_i \le C$$

$$\sum_{j=1}^{n} \lambda_i y_i = 0$$

Существует много разных ядер классификатора, наиболее популярные представлены ниже и на рисунке 12.

– Полиномиальное однородное степени d $k(x, x') = (x \cdot x')^d$;

– Полиномиальное неоднородное $k(x, x') = (x \cdot x' + 1)^d$;

– Радиальная базисная функция $k(x, x') = e^{-\gamma\|x-x'\|^2}, \gamma > 0$;

– Гиперболический тангенс $k(x, x') = tanh(\alpha x \cdot x' + \beta), \alpha > 0, \beta < 0$.

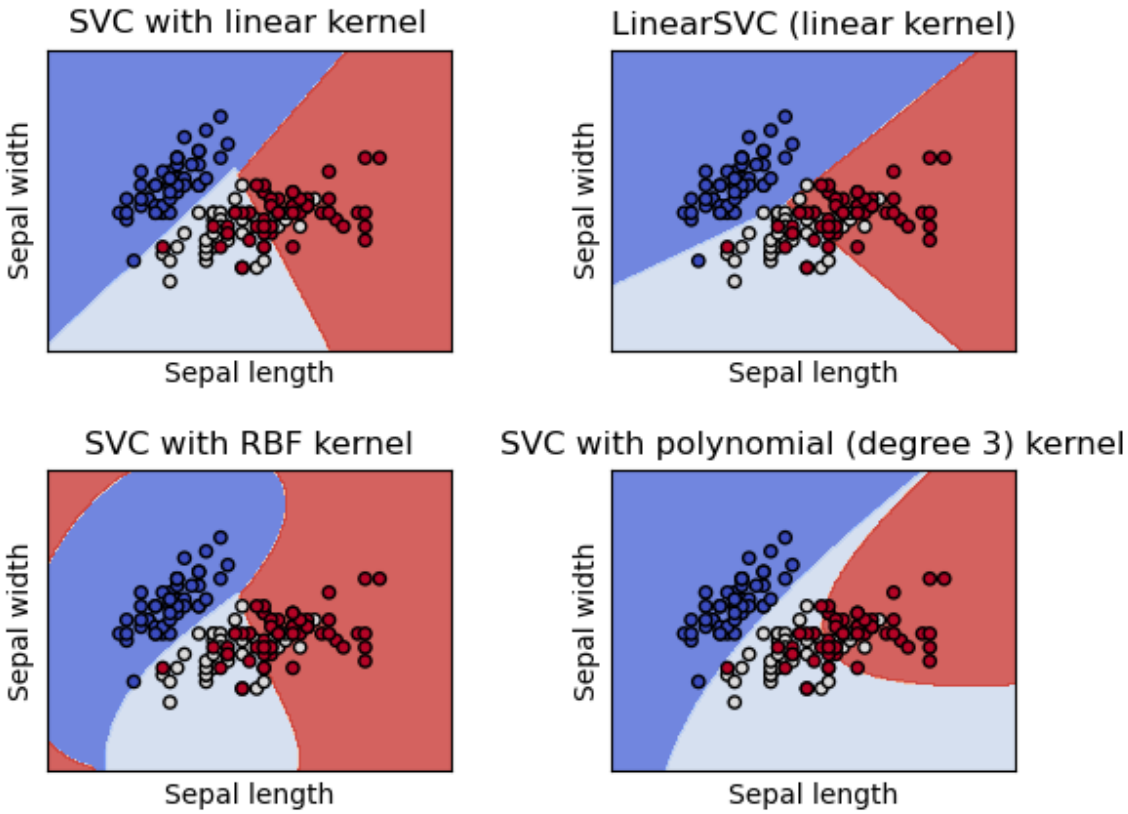


Рисунок 12 – Отличие разных ядер машин опорных векторов [28]

5.2. Метод релевантных векторов (RVM)

Метод релевантных векторов [29] имеет тот же функционал, что и машины опорных векторов, но обеспечивает вероятностную классификацию.

Как и в SVM, при некоторых $\lambda_i \ge 0$ веса определяются так:

$$w = \sum_{i=1}^{n} \lambda_i y_i x_i$$

Опорным векторам x_i соответствуют $\lambda_i \neq 0$. Какие из коэффициентов лучше обнулить?

Пусть регуляризатор зависит не от w, а от λ_i. Пусть λ_i независимые, гауссовские, с дисперсиями α_i:

$$p(\lambda) = \frac{1}{(2\pi)^{n/2}\sqrt{\alpha_1 \dots \alpha_n}} \exp\left(-\sum_{i=1}^{n} \frac{\lambda_i^2}{2\alpha_i}\right)$$

Тогда функционал обучения перепишется в виде

$$\sum_{i=1}^{n} \big(1 - y_i(w(\lambda) \cdot x_i - b)\big) + \frac{1}{2}\sum_{i=1}^{n}\left(\ln \alpha_i + \frac{\lambda_i^2}{\alpha_i}\right) \to \min_{\lambda,\alpha}$$

Для решения задачи классификации алгоритм будет выглядеть следующим образом [30].

Имеется обучающая выборка $X^l = (x_i, y_i)_{i=1}^{l}$ и матрица обобщенных признаков $\Phi^{n,m} = \left(\phi_j(x_i)\right)_{j=1}^{m}$.

Инициализируем гиперпараметры $\alpha_i = 1$, $\beta = 1$, пределы изменения α и w (AB и WB), максимальное количество итераций (как правило сходится за 50 итераций).

На каждой итерации:

1. Заполняем гиперпараметрами α_i диагональную матрицу $A = diag(\alpha_1, \dots, \alpha_m)$;
2. Считаем матрицу $\Sigma = (\beta\Phi^T\Phi + A)^{-1}$;
3. Считаем гессиан $w_{MP} = \Sigma\beta\Phi^T y$;
4. Для $j = 1, \dots, m$:
    a. Если $w_{MP,j} < WB$ или $\alpha_j > AB$, то:
        i. $w_{MP,j} = 0$;
        ii. $\alpha_j = \infty$;
        iii. $\gamma_j = 0$;
    b. Иначе:
        i. $\gamma_j = \alpha_j^{(old)}\Sigma_{jj}$;
        ii. $\alpha_j = \frac{\gamma_j}{w_{MP,j}^2}$;
5. $\beta_i = \frac{n - \sum_{i=1}^{n}\gamma_i}{\|t - \Phi w\|^2} w_{MP,i}^2$.

RVM обеспечивает гораздо более разреженные решения, хотя может застрять в локальном экстремуме.

6. Деревья решений

6.1. Алгоритм C4.5

На рисунке 13 изображена задача «ирисы Фишера», предполагающая классификацию на 3 класса, линейными классификаторами ее решить невозможно, нелинейные будут избыточны из-за малого количества признаков.

Но с этой задачей справятся деревья решений (decision trees), которые последовательно применяют решающие правила (предикаты).

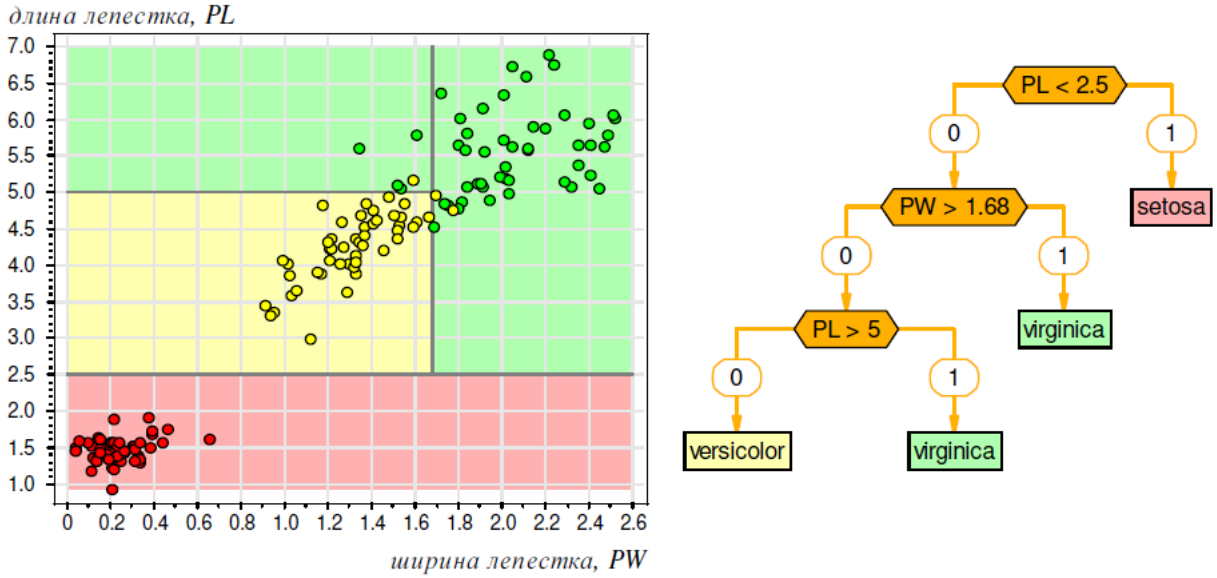


Рисунок 13 – Задача «ирисы Фишера»

Дерево решений – способ представления правил в иерархической, последовательной структуре, где каждому объекту соответствует единственный узел, дающий решение. В результате формируется покрывающий набор конъюнкций.

Каждый узел дерева содержит признак, ребра – значения признака, листы – метки классов, пример такого дерева показан на рисунке 13 справа. Классы должны быть дискретными. Каждый пример должен однозначно относиться к одному из классов. Справа показаны решающие поверхности, порожденные деревом решений.

C4.5 [31] – алгоритм построения дерева решений, количество потомков у узла не ограничено. Решает только задачи классификации. В нем есть важное требование: количество классов должно быть значительно меньше количества записей в исследуемом наборе данных.

Пусть T – множество примеров, где каждый элемент описывается m атрибутами, C_j - метка класса.

Процесс построения дерева будет происходить итеративно сверху вниз.

На первом шаге мы имеем пустое дерево (имеется только корень) и исходное множество T (ассоциированное с корнем). Требуется разбить исходное множество на подмножества. Делается через выбор одного из атрибутов в качестве проверки.

Тогда в результате разбиения получаются n (по числу значений атрибута) подмножеств и, соответственно, создаются n потомков корня, каждому из которых поставлено в соответствие свое подмножество, полученное при разбиении множества T.

В процессе построения любого дерева решений необходимо выбрать критерий разбиения (в случае C4.5 это прирост информации), правило остановки

(при небольшой выборке дерево строят до ее исчерпания, при большой – применяют отсечение).

Отсечение ветвей (pruning) - эвристический метод. Идем от листов к корню, помечая по некоторому критерию, например качество классификации, узлы на удаление. Вместо узлов ставится лист с меткой класса с наибольшим количеством исходов в этом поддереве.

Критерий разбиения

Пусть мы имеем проверку X (в качестве проверки может быть выбран любой атрибут), которая принимает n значений $A_1, A_2, \dots, A_n$. Тогда разбиение T по проверке X даст нам подмножества $T_1, T_2, \dots, T_n$, при X, равном соответственно $A_1, A_2, \dots, A_n$. $freq(C_j, T)$ – количество примеров из множества T, относящихся к классу C_j.

Оценка среднего количества информации, необходимого для определения класса примера из множества T (энтропия):

$$Info(T) = -\sum_{j=1}^{k} \frac{freq(C_j, T)}{|T|} * \log_2 \left(\frac{freq(C_j, T)}{|T|} \right)$$

Оценка среднего количества информации, необходимого для определения класса примера из множества T после разбиения множества T по X (условная энтропия):

$$Info_X(T) = \sum_{i=1}^{n} \left(\frac{|T_i|}{|T|} * Info(T_i) \right)$$

Оценка потенциальной информации, получаемой при разбиении множества T на n подмножеств. Эта оценка необходима для учета атрибутов с уникальными значениями.

$$split_{info(X)} = -\sum_{i=1}^{n} \left(\frac{|T_i|}{|T|} * \log_2 \left(\frac{|T_i|}{|T|} \right) \right)$$

Нормированный прирост информации

$$Gain\_ratio(X) = \frac{Info(T) - Info_X(T)}{split_{info(X)}}$$

Критерий $Gain\_ratio$ считается для всех атрибутов. Выбирается атрибут с максимальным $Gain\_ratio$. Этот атрибут будет являться проверкой в текущем узле дерева, а затем по этому атрибуту производится дальнейшее построение дерева.

Такие же рассуждения можно применить к полученным подмножествам $T_1, T_2, \dots, T_n$ и продолжить рекурсивно процесс построения дерева, до тех пор, пока в узле не окажутся примеры из одного класса.

Для примера возьмем таблицу 4 из [31].

Таблица 4. Пример данных для построения дерева решений

№	Признаки				Класс
	Outlook	Temperature	Humidity	Windy	
1	Sunny	Hot	High	False	N
2	Sunny	Hot	High	True	N
3	Overcast	Hot	High	False	P
4	Rain	Mild	High	False	P
5	Rain	Cool	Normal	False	P
6	Rain	Cool	Normal	True	N
7	Overcast	Cool	Normal	True	P
8	Sunny	Mild	High	False	N
9	Sunny	Cool	Normal	False	P
10	Rain	Mild	Normal	False	P
11	Sunny	Mild	Normal	True	P
12	Overcast	Mild	High	True	P
13	Overcast	Hot	Normal	False	P
14	Rain	Mild	High	True	N

Посчитаем критерий разбиения для всех признаков, чтобы определить какой признак будет помещен в корень дерева. В начале у нас полный набор данных, поэтому энтропия,

$$Info(T) = -\left(\frac{5}{14} * \log_2\left(\frac{5}{14}\right) + \frac{9}{14} * \log_2\left(\frac{9}{14}\right)\right)$$

Для признака Outlook имеем 3 значения, каждое из которых дает свое подмножество T_i:

$$Info_X(T) = \left(\frac{5}{14} * -\left(\frac{3}{5} * \log_2\left(\frac{3}{5}\right) + \frac{2}{5} * \log_2\left(\frac{2}{5}\right)\right)\right) + \left(\frac{4}{14} * -\left(\frac{4}{4} * \log_2\left(\frac{4}{4}\right) + \frac{0}{4} * \log_2\left(\frac{0}{4}\right)\right)\right) + \left(\frac{5}{14} * -\left(\frac{3}{5} * \log_2\left(\frac{3}{5}\right) + \frac{2}{5} * \log_2\left(\frac{2}{5}\right)\right)\right)$$

Точно также считается для оставшихся признаков. Затем выбираем признак с максимальным нормированным приростов информации и помещаем его в корень. Следующим шагом заполняем узлы по значениям признака в корне, но в этот раз множество T будет состоять не из 14 строк, а из такого количества строк, где признак в корне принял то или иное значение.

6.2. Алгоритм CART

CART (Classification and Regression Tree) [32] – это алгоритм построения бинарного дерева решений. Решает задачи классификации и регрессии. Каждый узел имеет двух потомков, соответственно, в правый потомок уходим в случае

выполнения правила в узле, в левый – невыполнения. В качестве **критерия разбиения** применяется неопределенность Джини:

$$Gini(T) = 1 - \sum_{i=1}^{n} p_i^2$$

где p_i – относительная частота класса i в T

Если набор T разбивается на две части $T1$ и $T2$ с числом примеров в каждом N_1 и N_2 соответственно, тогда показатель качества разбиения будет равен:

$$Gini_{split}(T) = \frac{N_1}{N} * Gini(T1) + \frac{N_2}{N} * Gini(T2)$$

Наилучшим разбиением считается то, для которого $Gini_{split}$ минимально.

Алгоритм CART предполагает уникальный алгоритм **отсечения** - minimal cost-complexity tree pruning.

Пусть $|T|$ – число листов дерева, $R(T) = 1 - accuracy$ – ошибка классификации. Тогда полная стоимость (оценка затраты-сложность) дерева:

$$C_\alpha(T) = R(T) + \alpha|T|$$

$\alpha|T|$ – штраф за сложность дерева (параметр α выбирается большим нуля)

Менее ветвистое дерево, дающее большую ошибку классификации, может стоить меньше, чем дающее меньшую ошибку, но более ветвистое.

На основании $C_\alpha(T)$ строим последовательность уменьшающихся поддеревьев, где можно получить следующее дерево в последовательности, применив отсечение к текущему.

Для каждого узла t в дереве T_1 (открывает последовательность) выбираются слабые связи, для которых величина $g(t)$ является наименьшей:

$$g(t) = \frac{R(t) - R(T_{1,t})}{|T_{1,t}| - 1}$$

Дерево T_1 отсекается в этих узлах, чтобы получить дерево T_2, и так далее. Финальное дерево выбирается по минимальной ошибке классификации или на основании перекрестной проверки.

7. Вероятностные модели

7.1.Байесовский классификатор

Пусть X – множество описаний объектов, Y – множество номеров классов. На множестве пар «объект, класс» $X \times Y$ определена вероятностная мера P, которая задает вероятности принадлежности объектов к классам. Имеется конечная обучающая выборка независимых наблюдений $X^m = \{(x_1, y_1), \dots, (x_m, y_m)\}$, полученных согласно вероятностной мере P.

В этом случае задача классификации делится на две:

– Построение оптимального классификатора при известных плотностях классов;

– Восстановление плотностей классов по обучающей выборке.

Пусть для каждого класса $y \in Y$ известна априорная вероятность P_y того, что появится объект класса y, и плотности распределения $p_y(x)$ каждого из классов (называемые функциями правдоподобия классов).

Требуется построить алгоритм классификации $a(x)$, доставляющий минимальное значение функционалу среднего риска, определяемому как математическое ожидание ошибки:

$$R(a) = \sum_{y \in Y} \sum_{s \in Y} \lambda_y P_y P_{(x,y)}\{a(x) = s|y\}$$

где λ_y - цена ошибки или штраф за отнесение объекта класса y к какому-либо другому классу

Решением этой задачи является алгоритм

$$a(x) = \arg\max_{y \in Y} \lambda_y P_y p_y(x)$$

$P\{y|x\} = P_y p_y(x)$ - апостериорная вероятность того, что объект x принадлежит классу y.

Если классы равнозначимы, $\lambda_y P_y = const(y)$, то объект x просто относится к классу с наибольшим значением плотности распределения в точке x.

Восстановление плотностей классов

По заданной подвыборке объектов класса y требуется построить эмпирические оценки априорных вероятностей P_y и функций правдоподобия $p_y(x)$. В качестве оценки априорных вероятностей берут долю объектов данного класса в обучающей выборке.

Восстановление плотностей можно сделать двумя способами:

– Параметрическое восстановление плотности при дополнительном предположении, что плотности нормальные (гауссовские), приводит к нормальному дискриминантному анализу и линейному дискриминанту Фишера;

– Непараметрическое восстановление плотности приводит к методу парзеновского окна.

Разделение смеси распределений может быть сделано с помощью EM-алгоритма. Дополнительное предположение, что плотности компонент смеси являются радиальными функциями, приводит к методу радиальных базисных функций. Обычно в качестве компонент смеси берут гауссовские плотности.

7.2.Наивные байесовский классификатор

Байесовский классификатор трудно вычислять из-за сложности оценки плотностей, поэтому чаще всего используют наивный байесовский классификатор (naïve bayes), который основан на использовании метода максимального правдоподобия.

Предполагается, что объекты описываются независимыми признаками $f_j: X \to D_j$ с плотностями распределения $p_j(\xi|y)$. Тогда функции правдоподобия классов представимы в виде произведения одномерных плотностей по признакам, $x^j \equiv f_j(x)$:

$$p(x|y) = p_1(x^1|y) \dots p_n(x^n|y)$$

Прологарифмировав, получим классификатор ($\hat{P}$ и $\widehat{p_j}$ - эмпирические оценки):

$$a(x) = \arg\max_{y \in Y} \left(\ln \lambda_y \hat{P}(y) + \sum_{j=1}^{n} \ln \widehat{p_j}\left(x^j | y\right) \right)$$

Предположение о независимости существенно упрощает задачу, так как оценить n одномерных плотностей гораздо легче, чем одну n-мерную плотность.

Предполагается, что одномерные плотности экспоненциальны:

$$p\left(x^j | y; \theta_{yj}, \varphi_{yj}\right) = \exp\left(\frac{x^j \theta_{yj} - c\left(\theta_{yj}\right)}{\varphi_{yj}} + h\left(x^j, \varphi_{yj}\right) \right)$$

Тогда задача максимизации log-правдоподобия

$$L(\theta, \varphi) = \sum_{j=1}^{n} \sum_{y \in Y} \left(\sum_{x_i \in X_y} \ln p\left(x_i^j | y; \theta_{yj}, \varphi_{yj}\right) \right) \to \max_{\theta, \varphi}$$

распадается на независимые подзадачи для каждого (y, j)

$$\sum_{x_i \in X_y} \left(\frac{x^j \theta_{yj} - c\left(\theta_{yj}\right)}{\varphi_{yj}} + h\left(x^j, \varphi_{yj}\right) \right) \to \max_{\theta_{yj}, \varphi_{yj}}$$

Решение (поиск параметров) θ_{yj} через среднее значение признака j в классе y:

$$\frac{\partial L}{\partial \theta_{yj}} = 0 \Rightarrow c'^{\left(\theta_{yj}\right)} = \sum_{x_i \in X_y} \frac{x_i^j}{|X_y|} \equiv \bar{x}_{yj} \Rightarrow \theta_{yj} = [c']^{-1}\left(\bar{x}_{yj}\right)$$

Решение φ_{yj} не всегда выражается из уравнения $\frac{\partial L}{\partial \varphi_{yj}} = 0$, но для распределений Пуассона, Бернулли, биномиального $\varphi_{yj} = 1$

Для гауссовского распределения (и если не зависит от y)

$$\varphi_{yj} = \frac{1}{l} \sum_{i=1}^{l} \left(x_i^j - \bar{x}_{yj}\right)^2$$

В итоге линейный классификатор ($h\left(x^j, \varphi_{yj}\right)$ сокращается, если не зависит от y

$$a(x) = \arg\max_{y \in Y} \left(\sum_{j=1}^{n} x^j \frac{\theta_{yj}}{\varphi_{yj}} + \ln\left(\lambda_y P(y)\right) - \sum_{j=1}^{n} \frac{c\left(\theta_{yj}\right)}{\varphi_{yj}} + h\left(x^j, \varphi_{yj}\right) \right)$$

$$\frac{\theta_{yj}}{\varphi_{yj}} = w_{yj}, \ \ln\left(\lambda_y P(y)\right) - \sum_{j=1}^{n} \frac{c\left(\theta_{yj}\right)}{\varphi_{yj}} = b_y$$

Помимо метода наивного байеса, в машинном обучении применяют скрытые марковские модели. Это статистическая модель, имитирующая работу процесса, похожего на марковский процесс с неизвестными параметрами. В свое

время их разработали для распознавания речи, и по сей день они применяются в распознавании аудио, видео, текста и т.п., хотя уже и уступают свои позиции реккурентным нейронным сетям. Хороший обзор скрытых марковских моделей написали Рабинер [33] и Мерков [34].

8. Ансамблевые методы

8.1. Бэггинг (Bagging, Bootstrap aggregating)

Bootstrap aggregating [35] - метод создания различающихся моделей на основе различных случайных выборок из исходного набора данных, так называемых бутстрэп-выборок (усиливающих выборок).

Бутстрэп – это случайный выбор данных из датасета и представление их в модель, затем данные возвращаются в датасет, и процесс повторяется. Вероятность того, что конкретный пример не входит в усиливающую выборку размера n равна $(1 - 1/n)^n$. При $n \to \infty$ она стремится к $1/e$. На усиливающих выборках обучаются базовые алгоритмы классификации. Результат определяется голосованием. По итогу строится кусочно-линейная разделяющая плоскость.

Вспоминая дилемму «смещение-разброс», применение бэггинга уменьшает только разброс.

8.1.1. Random Forest

Самый яркий представитель бэггинга – это алгоритм Random Forest (случайны лес). Random Forest [36] является композицией (ансамблем) множества решающих деревьев (применяется CART-дерево). Как было видно на рисунке 27, решающие поверхности дерева решений очень резкие, что может приводить к переобучению, сгладить это позволяет комбинация деревьев решений. Прогноз получается в результате агрегирования ответов множества деревьев: для задач классификации используется голосование, для регрессии - выбор среднего значения.

Обучение деревьев происходит независимо друг от друга (на разных подмножествах). Оптимальное число деревьев подбирается таким образом, чтобы минимизировать ошибку классификатора на тестовой выборке.

Пусть N – размер обучающей выборки, M – количество признаков, $m \approx \sqrt{M}$ – неполное количество признаков для обучения для классификации и $m \approx M/3$ для регрессии. Благодаря такому выбору признаков и данных удается снизить корреляции между деревьями решений.

Сначала генерируем случайную подвыборку с повторениями размером N из обучающей выборки. По ней строим дерево решений, признаки для которого выбираем из m случайно отобранных, не применяем отсечение.

8.2. Усиление (Boosting)

8.2.1. AdaBoost

Усиление (boosting) в процессе обучения строит композицию из базовых алгоритмов обучения для улучшения их эффективности. Метод бустинга в чём-то схож с методом бэггинга: берётся множество одинаковых моделей и объединяется, чтобы получить сильного ученика. Но разница заключается в том, что модели приспосабливаются к данным последовательно, то есть каждая модель будет исправлять ошибки предыдущей.

Алгоритм AdaBoost [37] вызывает слабый классификатор в цикле. После каждого вызова обновляется распределение весов, которые отвечают важности каждого из объектов обучающего множества для классификации. На каждой итерации веса каждого неверно классифицированного объекта возрастают, таким образом новый классификатор «фокусирует своё внимание» на этих объектах. Бустинг устойчив к переобучению.

Хотя базовые модели для бустинга – это модели с низким разбросом и высоким смещением, применение техники усиления позволяет уменьшить и смещение, и разброс.

На рисунке 14 показано сравнение разделения двух классов разными алгоритмами машинного обучения, в том числе бэггинга и бустинга.

AdaBost решает задачу бинарной классификации $Y = \{-1, +1\}$. Имеется обучающая выборка $x_i \epsilon X^{(m)}$ и базовые алгоритмы классификации $h_1, \dots, h_T$.

Функционал качества композиции – число ошибок на обучающей выборке:

$$Q_T = \sum_{i=1}^{m} \left[y_i \sum_{t=1}^{T} \alpha_t h_t(x_i) < 0 \right] \to \min$$

Инициализация весов объектов $D_1(i) = \frac{1}{m}$

Для каждого $t = 1, \dots, T$:

1. Находим классификатор $h_t: X \to Y$, который минимизирует взвешенную ошибку классификации $e_j = \sum_{i=1}^{m} D_t(i)\left[y_i \neq h_j(x_i)\right]$;
2. Если $e_j \geq 0.5$, то остановка;
3. Минимум функционала Q_T достигается при коэффициенте доверия $\alpha_t = \frac{1}{2} \ln \frac{1-e_t}{e_t}$, где e_t - взвешенная ошибка классификатора h_t;
4. Обновляем $D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$, где Z_t - нормализующий параметр, выбранный так, чтобы $\sum_{i=1}^{m} D_{t+1}(i) = 1$.

Строим результирующий классификатор:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$$

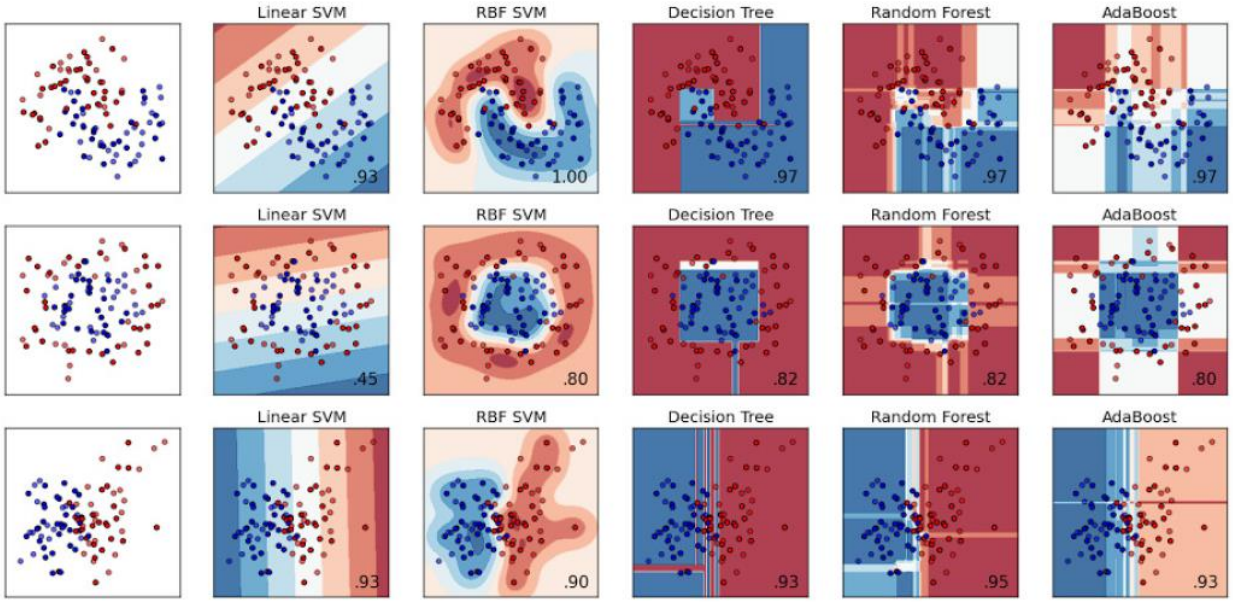


Рисунок 14 – Сравнение разделения двух классов с помощью разных алгоритмов [38]

8.2.2. Градиентный бустинг

Градиентный бустинг [39] использует типичный алгоритм градиентного спуска для решения задачи. Когда приходит время добавить новый слабый алгоритм в ансамбль, находится оптимальный вектор сдвига, улучшающий предыдущий ансамбль алгоритмов. Этот вектор сдвига является антиградиентом от функции ошибок работы предыдущего ансамбля моделей. Благодаря вектору сдвигов мы знаем, какие значения должны принимать объекты обучающей выборки. А поскольку нам надо найти очередной алгоритм в композиции, то находим тот, при использовании которого минимизируется отклонение ответов от истинных.

Обучаем α_t, h_t при фиксированных предыдущих

Функционал качества с заданной гладкой функцией потерь $L(h, y)$:

$$Q = \sum_{i=1}^{m} L\left(\sum_{t=1}^{T} \alpha_t h_t(x_i) + \alpha h(x_i), y_i\right) \to \min_{\alpha,h}$$

Пусть $f_{T-1} = (\sum_{t=1}^{T} \alpha_t h_t(x_i))_{i=1}^{m}$ - вектор текущего приближения, тогда вектор следующего приближения $f_T = (\sum_{t=1}^{T} \alpha_t h_t(x_i) + \alpha h(x_i))_{i=1}^{m}$ можно найти с помощью градиентного метода минимизации $Q(f), f \in \mathbb{R}^m$:

$$f_{T,i} = f_{T-1,i} - \beta g_i$$

где $g_i = L'_f\left(f_{T-1,i}, y_i\right)$ - компоненты вектора градиента, β – градиентный шаг.

Это похоже на добавление базового алгоритма, а значит, нам надо найти такой базовый алгоритм h_t, чтобы вектор $\left(h_t(x_i)\right)_{i=1}^{m}$ приближал вектор антиградиента $(-g_i)_{i=1}^{m}$:

$$h_t = \arg\min_h \sum_{i=1}^{m} (h(x_i) + g_i)^2$$

По сравнению с AdaBoost в алгоритме меняется содержимое итерации:

1. $h_t = \arg\min_h \sum_{i=1}^m \big(h(x_i) + L'(f_i, y_i)\big)^2$ - базовый алгоритм, приближающий антиградиент;
2. $\alpha_t = \arg\min_{\beta>0} \sum_{i=1}^m L(f_i + \beta h_t(x_i), y_i)$ - задача одномерной минимизации;
3. $f_i = f_i + \alpha_t h_t(x_i)$ - обновление вектора значений на объектах выборки.

Каждый следующий базовый алгоритм обучается, чтобы исправить ошибки предыдущих.

Существует вариация градиентного бустинга - стохастический градиентный бустинг, в котором при оптимизации используется не вся выборка, а случайная подвыборка, по аналогии с бэггингом. При этом улучшается сходимость и обобщающая способность.

8.2.3. XGBoost

Очень популярной библиотекой, реализующей градиентный бустинг, является xgboost [40]. Посмотрим на то, как она работает. В качестве слабых алгоритмов в ней используются CART-деревья (w_k - значение в листе k, K – множество листьев, $B_k(x)$ - бинарный индикатор, что x попадает в лист k):

$$h(x, w) = \sum_{k \in K} w_k B_k(x)$$

В функционале качества применяются L_0 и L_2 регуляризаторы:

$$Q(w) = \sum_{i=1}^m L(a(x_i) + h(x_i, w), y_i) + \gamma|K| + \frac{\lambda}{2}\sum_{k \in K} w_k^2 \to \min_w$$

где $a(x_i) = \sum_{t=1}^T \alpha_t h_t(x_i)$ - ранее построенная часть ансамбля

Для оптимизации функционала качества используют приближенное аналитическое решение

Разложим функционал в ряд Тейлора до второго члена $L(a + b, y) \approx L(a, y) + bL'(a, y) + \frac{b^2}{2}L''(a, y)$:

$$\Phi(w) = \sum_{i=1}^m \left(g_i h_i + \frac{1}{2} b_i h_i^2\right) + \gamma|K| + \frac{\lambda}{2}\sum_{k \in K} w_k^2 \to \min_w$$

где $g_i = L'(a(x_i), y_i)$, $b_i = L''(a(x_i), y_i)$

При условии $\frac{\partial \Phi(w)}{\partial w_k}$ оптимальное значение листа k

$$w_k = -\frac{\sum_i g_i B_k(x_i)}{\lambda + \sum_i b_i B_k(x_i)}$$

Подставляя вес w_k обратно в $\Phi(w)$ выводим критерий ветвления:

$$\Phi(B_1, \dots, B_k) = -\frac{1}{2}\sum_{k \in K} \frac{(\sum_i g_i B_k(x_i))^2}{\lambda + \sum_i b_i B_k(x_i)} + \gamma|K| \to \min$$

8.2.4. CatBoost

Еще одной популярной библиотекой является CatBoost, разработанный компанией Яндекс. Алгоритм, лежащий в ее основе, основан на градиентном бустинге. В стандартном градиентном бустинге есть две проблемы:

– переобучение в градиентах: g_i вычисляются в тех же точках x_i, по которым ансамбль $a_{t-1}(x)$ обучался аппроксимировать y_i. Решением является упорядоченный бустинг;

– надо обрабатывать категориальные признаки. Для этого можно использовать статистику по целевому признаку, которая вычисляется по перестановкам X^{rj}:

$$\tilde{f}(x) = \frac{\sum_{x_i \in X^{rj}} [f(x_i) = f(x)] y_i + \gamma p}{\sum_{x_i \in X^{rj}} [f(x_i) = f(x)] y_i + \gamma}$$

Упорядоченный бустинг

Пусть X^{rj} - подвыборка первых 2^j объектов из $\sigma_r\left(X^{(m)}\right)$, $g_{ti} = L'\left(a_{t-1}^{rj}(x_i), y_i\right)$ - градиент в точке (x_i, y_i) для модели-полуфабриката, которая по ней не обучалась, где $j = \log_2(i-1)$

1. Сгенерировать случайные перестановки $\sigma_0, \sigma_1, \dots, \sigma_s$
2. Для всех $t = 1, \dots, T$:
    a. Выбрать случайно перестановку σ_r;
    b. Вычислить несмещенный вектор градиента g_{ti};
    c. Вычислить $h_t = \arg\min_h \sum_{i=1}^{m} (h(x_i) + g_{ti})^2$ по X^{rj};
    d. Для всех деревьев h_t^{rj}:
        i. Скопировать общую структуру из h_t;
        ii. Вычислить значения в листьях по X^{rj};
    e. Вычислить значения в листьях для h_t по X^{0j};
    f. Вычислить α_t и обновить f_i.

8.3. Стэкинг

Техника стэкинга позволяет решать задачу машинного обучения совершенно разнородными моделями одновременно, что бывает полезно на сложных данных, которые не могут быть аппроксимированы одной моделью, даже нейронными сетями. Собственно, каскад нейронных сетей и строится по принципу стэкинга. А саму нейронную сеть можно считать стэкингом логистических регрессий.

Блэндинг – самая простая форма стэкинга. Обучающую выборку делят на две части: на первой обучаются базовые алгоритмы, их ответы получают на второй части и тестовой выборке. Эти ответы могут служить новыми признаками (называются «мета-признаки») для обучения мета-алгоритма, который проверяют на тестовой выборке, рисунок 15. Такое разделение обучающей выборки – редкостное расточительство, поэтому применяют стэкинг.

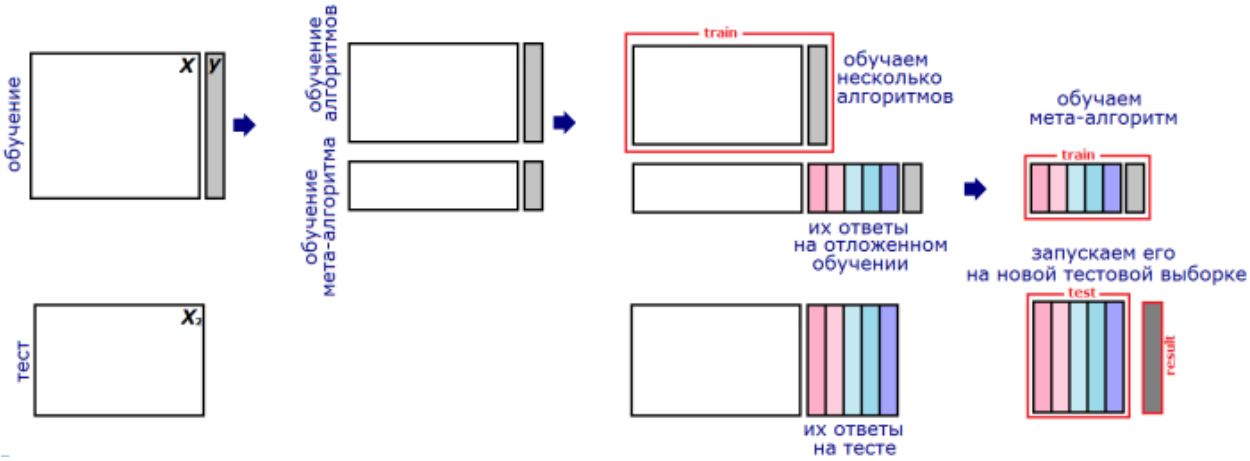


Рисунок 15 – Схема блэндинга [41]

Стэкинг, рисунок 16, предполагает разбиение выборки на k блоков (fold). Для объекта из выборки, который находится в k блоке, делается предсказание с помощью слабых моделей, которые были обучены на $k-1$ блоках. Этот процесс повторяется для каждого блока. Таким образом, создается набор прогнозов слабых моделей для каждого объекта выборки, на них в итоге обучается метамодель. В итоге мы вместо одного метапризнака $h_t(x)$ имеем k похожих $h_{tj}(x)$. Для получения результата усредняем метапризнаки $h_t(x) = \frac{1}{k}\sum_{j=1}^{k} h_{tj}(x)$.

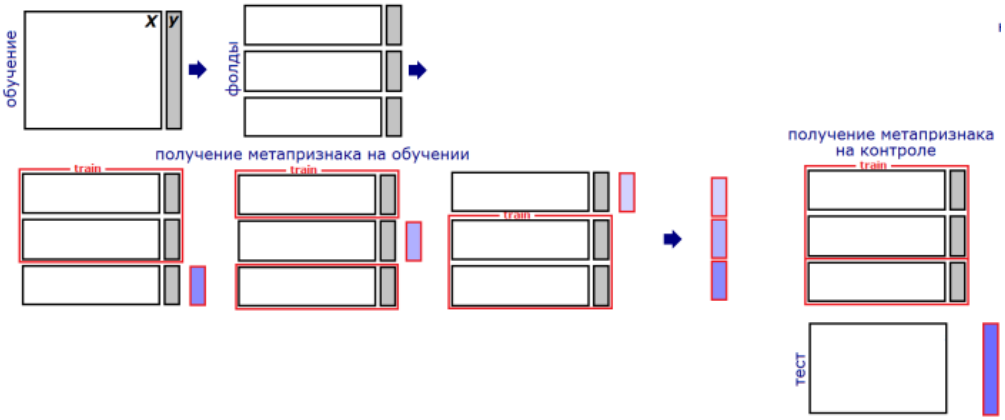


Рисунок 16 – Схема стэкинга [41]

9. Контрольные вопросы

– Чем отличается задача классификации от задачи кластеризации?
– В каких случаях применяются тестовая и валидационная выборки?
– Что такое аугментация и каковы нюансы ее применения?
– Как формируются правила в алгоритмах Apriori и FP-Growth?
– Как влияют параметры алгоритма DBSCAN на качество кластеризации?
– Как влияет начальное распределение данных на форму кластеров?
– Какие способы кросс-валидации применяют для оценки качества классификации и почему?
– В чем отличие линейной и нелинейной разделимости?
– В чем нюансы применения ROC-кривых?

– Как используется теорема Каруша-Куна-Таккера в поиске опорных векторов?

– Как на расчет прироста информации влияют пропуски в данных?

– Какие из ансамблевых методов позволяют комбинировать различные алгоритмы машинного обучения?

Список использованных источников

1. Mitchell T.M. Machine learning. McGraw-Hill, New York, 1997
2. Choosing the right estimator [Электронный ресурс]. – Режим доступа: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
3. Тыртышников Е.Е. Матричный анализ и линейная алгебра. – М.: Физматлит, 2007
4. Ильин В.А., Позняк Э.Г. Линейная алгебра. – М.: Физматлит, 2005.
5. Ильин В.А., Позняк Э.Г. Основы математического анализа: в 2 ч. - М.: Наука. Физматлит, 1998
6. Поляк Б.Т. Введение в оптимизацию. – М.: Наука, 1983
7. Игошин В.И. Математическая логика и теория алгоритмов, 2-е изд. – М.: Издательский дом «Академия», 2008
8. Вентцель Е.С. Теория вероятностей
9. Кремер Н.Ш. Теория вероятностей и математическая статистика, 2-е изд. – М.: Юнити-Дана, 2004
10. Боровков А.А. Математическая статистика, 4-е изд. – СПб.: Издательство «Лань», 2010
11. Павлов Ю.Н. Теория информации для бакалавров. -М.: Издательство МГТУ им. Н.Э. Баумана, 2016
12. Панин В.В. Основы теории информации, 4-е изд. – М.: Бином. Лаборатория знаний, 2012.
13. Андерсон Дж. Дискретная математика и комбинаторика. – М.: Вильямс, 2004
14. Хаггарти Р. Дискретная математика для программистов, 2-е изд. – М.: Техносфера, 2012.
15. Флах П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных. – М.: ДМК Пресс, 2015
16. Саттон Р.С., Барто Э.Дж. Обучение с подкреплением: Введение. 2-е изд. – М.: ДМК Пресс, 2020
17. Шалев-Шварц Ш., Бен-Давид Ш. Идеи машинного обучения: от теории к алгоритмам. – М.: ДМК Пресс, 2019.
18. Agrawal R. et al. Fast discovery of association rules // Advances in Knowledge Discovery and Data Mining. AAAI Press, 1996. P. 307–328.
19. Han J. et al. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach // Data Mining and Knowledge Discovery. 2004. Vol. 8, № 1. P. 53–87.

20. Arthur D., Vassilvitskii S. k-means++: the advantages of careful seeding // Proceedings of the 18th annual ACM-SIAM symposium on Discrete algorithms. 2007. P. 1027–1035.
21. Ester M. et al. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise // Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. 1996. P. 226–231.
22. Comparing different clustering algorithms on toy datasets [Электронный ресурс]. – Режим доступа: https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html
23. Dunn J.C. A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters // Journal of Cybernetics. 1973. Vol. 3, № 3. P. 32–57.
24. Bezdek J.C. Pattern recognition with fuzzy objective function algorithms. New York: Plenum Press, 1981. 256 p.
25. Cross-validation: evaluating estimator performance [Электронный ресурс]. – Режим доступа: https://scikit-learn.org/stable/modules/cross_validation.html
26. Martin A. et al. The DET curve in assessment of detection task performance. 1997. P. 1895–1898.
27. SVM [Электронный ресурс]. – Режим доступа: https://staesthetic.files.wordpress.com/2014/02/svm.png?w=1060
28. Support Vector Machines [Электронный ресурс]. – Режим доступа: https://scikit-learn.org/stable/modules/svm.html
29. Tipping M. The Relevance Vector Machine // Advances in Neural Information Processing Systems / ed. Solla S., Leen T., Müller K. MIT Press, 2000. Vol. 12.
30. Метод релевантных векторов [Электронный ресурс]. – Режим доступа: http://www.machinelearning.ru/wiki/index.php?title=RVM
31. Quinlan J.R. C4.5: programs for machine learning. San Mateo, Calif: Morgan Kaufmann Publishers, 1993. 302 p.
32. Breiman L. et al. Classification and Regression Trees. Wadsworth, Belmont, California, 1984
33. Rabiner L.R. A tutorial on hidden Markov models and selected applications in speech recognition // Proc. IEEE. 1989. Vol. 77, № 2. P. 257–286.
34. Мерков А.Б. Распознавание образов: Построение и обучение вероятностных моделей. – М.: ЛЕНАНД, 2020
35. Breiman L. Bagging Predictors: Technical Report 421. Berkeley, USA: University of California, 1994.
36. Breiman L. Random Forests // Machine Learning. 2001. Vol. 45, № 1. P. 5–32.
37. Freund Y., Schapire R.E. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting // Journal of Computer and System Sciences. 1997. Vol. 55, № 1. P. 119–139.

38. Classifier comparison [Электронный ресурс]. – Режим доступа: https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html
39. Friedman J.H. Stochastic gradient boosting // Computational Statistics & Data Analysis. 2002. Vol. 38, № 4. P. 367–378.
40. Chen T., Guestrin C. XGBoost: A Scalable Tree Boosting System // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco California USA: ACM, 2016. P. 785–794.
41. Дьяконов А.Г. Стекинг (Stacking) и блендинг (Blending) [Электронный ресурс]. – Режим доступа: https://dyakonov.org/2017/03/10/стекинг-stacking-и-блендинг-blending/

А.В. Кугаевских, Д.И. Муромцев, О.В. Кирсанова

Классические методы машинного обучения

Учебное пособие

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49, литер А