Федеральное государственное образовательное бюджетное учреждение высшего образования

«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»
(Финансовый университет)

Департамент анализа данных, принятия решении й и финансовых технологий

М.В. Коротеев

Технологии анализа данных и машинное обучение

УЧЕБНОЕ ПОСОБИЕ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1 семестр

Для студентов, обучающихся по направлениям 09.03.03 «Прикладная информатика»
(программа подготовки бакалавра)

Одобрено Советом Департамента анализа данных, принятия решении й и финансовых технологий (протокол №5 от 20 ноября 2018 г.)

Москва – 2018

УДК 004(075.8)
ББК 32.972
К68

Авторы:

Коротеев М.В., канд. экон. наук, доцент Департамента анализа данных, принятия решении й и финансовых технологий Финансового университета при Правительстве РФ (3 п.л.)

Рецензент:
Макрушин С.В., канд. экон. наук, доцент Департамента анализа данных, принятия решении й и финансовых технологий Финансового университета при Правительстве РФ.

К68 Коротеев М.В.

Технологии анализа данных и машинное обучение. Учебное пособие для самостоятельной работы. 1 семестр: Учебное пособие. – М.: Финансовый университет, департамент анализа данных, принятия решении й и финансовых технологий, 2018. – 48 с.

Цель пособия – предоставить студенту основной теоретический материал, необходимый для успешного освоение дисциплины «Технологии анализа данных и машинное обучение». Пособие содержит разъяснения основных теоретических положений задач обучения с учителем и без учителя, а также диагностики систем машинного обучения.

Учебное пособие предназначено для бакалавров направления подготовки 09.03.03 «Прикладная информатика»

УДК 004(075.8)
ББК 32.972

Учебное издание

Коротеев *Михаил Викторович,*

Технологии анализа данных и машинное обучение.
Учебное пособие для самостоятельной работы,
1 семестр

Компьютерный набор, верстка: М.В. Коротеев

Формат 60х90/16. Гарнитура *Times New Roman.*
Усл. п.л. 3,0. Изд. № - 2018. Заказ № ____
Электронное издание

© ФГОБУ ВО «Финансовый университет при Правительстве Российской Федерации», 2018.
© Департамент анализа данных, принятия решении й и финансовых технологий, 2018.
© Коротеев Михаил Викторович, 2018.

О Г Л А В Л Е Н И Е

Что такое машинное обучение?

В настоящее время общепринятыми являются два определения машинного обучения. Артур Самуэль определяет его как «поле обучения, которое дает компьютерам возможность учиться, не будучи явно запрограммированным». Это более старое, неформальное определение.

Том Митчелл предлагает более современное определение: “Говорят, что компьютерная программа учится на опыте E по отношению к некоторому классу задач T и показателю производительности P, если его производительность при задачах в T, измеряемая через P, улучшается с опытом E.”

Пример: игра в шашки.

E = опыт игры во многие игры шашек;

T = задача игры в шашки;

P = вероятность того, что программа выиграет следующую игру.

В общем, любая проблема машинного обучения может быть отнесена к одной из двух широких классов задач:

- обучение с учителем;
- обучение без учителя.

Обучение с учителем

В обучении с учителем нам предоставляется набор данных и уже известно, как должен выглядеть наш правильный вывод, имея представление о том, что между входом и выходом существует взаимосвязь.

Задачи обучения с учителем подразделяются на проблемы регрессии и классификации. В задаче регрессии мы пытаемся предсказать результаты по непрерывной шкале, что означает, что мы пытаемся сопоставить входные переменные с некоторой непрерывной функцией. В проблеме классификации мы вместо этого пытаемся предсказать результаты в дискретной шкале. Другими словами, мы пытаемся отобразить входные переменные в дискретные категории.

Пример 1:

Учитывая данные о размерах домов на рынке недвижимости, попробуйте предсказать их цену. Цена как функция размера - это непрерывная выходная переменная, поэтому это проблема регрессии.

Мы могли бы превратить этот пример в проблему классификации, вместо этого сделав наш вывод о том, продает ли дом больше или меньше

запрашиваемой цены. Здесь мы классифицируем дома по цене на две дискретные категории.

Вообще, почти все задачи регрессии можно переформулировать как задачи классификации. Поэтому часто в программных пакетах методы регрессии и классификации не разделяются четко как два принципиально разных класса задач. Даже если алгоритмы для решения довольно существенно различаются, между этими двумя классами задач, несомненно существует взаимосвязь.

Пример 2:

(a) Регрессия. Рассматривая изображение человека, мы должны предсказать его возраст на основе данной картины.

(b) Классификация. Рассматривая изображение человека, мы должны предсказать, является ли он/она школьником, студентом или взрослым. Еще один пример классификации - банки должны решить, давать ли кому-либо кредит на основании его кредитной истории.

Обучение без учителя

Обучение без учителя, с другой стороны, позволяет нам решать проблем, без полного понимания, как выглядят наши результаты. Мы можем получить структуру из данных, где мы не обязательно знаем влияние переменных. Мы можем получить эту структуру, объединив данные на основе отношений между переменными в данных.

При обучении без учителя нет обратной связи, основанной на известных значениях выходной переменной, то есть нет учителя, который бы исправил нашу модель.

Примеры:

Кластеризация: возьмите сборник из 1000 эссе, написанных об одной теме, и найдите способ автоматически группировать эти эссе на небольшие группы, которые каким-то образом похожи разными переменными, такими как частота слов, длина предложения, количество страниц и так далее.

Другой пример: «Алгоритм коктейльной вечеринки», который может найти структуру в беспорядочных данных (например, идентификация отдельных голосов и музыки из сетки звуков на вечеринке).

Зачем изучать машинное обучение?

В настоящее время, объем данных, которые генерирует человек в своей деятельности экспоненциально увеличивается. Специалисты оценивают, что только за первое десятилетие XXI века человечество создало больше

информации, чем за всю предыдущую свою историю. И это не только и не столько информация в виде книг, статей, картин, видео. Основной объем информации составляет всякого рода числовая информация, отражающая всяческие аспекты жизни человека: экономическая и социальная, данные о продажах товаров, услуг, биржевые сводки, логи поведения пользователей Интернет, персональных компьютеров, мобильных устройств, телефонные разговоры. Примеров собираемой информации можно придумать бесконечное количество. И с каждым днем все больше и больше аспектов каждодневного человеческого поведения не просто происходит в физической реальности, а протоколируется, сохраняется и обрабатывается теми или иными операторами обработки данных.

Конечно, не стоит думать, что это коллективный сознательный процесс по внедрению тотальной слежки за каждым жителем Земли. Информация о представляла собой ценность с давних времен: правителям - для лучшего понимания экономики государства, ученым - для построения обобщающих теорий о реальности, частным компаниям - для анализа пользовательских предпочтений и извлечения прибыли.

Кроме того, параллельно растет и объем научной информации. Астрономы собирают данные о небе с большим разрешением и в большем диапазоне частот. Биологи научились строить компьютерные модели синтеза протеинов. Медики могут наблюдать за работой человеческого тела при помощи новейших технологий визуализации. Физики строят сложнейшие математические модели и проводят грандиозные эксперименты. Все это порождает колоссальный объем информации, которую нужно хранить и анализировать.

Технологической предпосылкой для развития науки о данных послужило развитие вычислительной техники и систем хранение больших объемов информации. Это позволило сохранять практически всю информацию, которая может понадобиться в будущем. Если раньше информация хранилась и передавалась в основном в письменном виде (а еще раньше - вообще в устной форме), то сегодня мы располагаем технологиями хранения, поиска и доступа к большим объемам информации.

Конечно, мало толку просто хранить информацию. Если она не используется для анализа и извлечения знаний - она бесполезна. И в достоверной информации почти всегда содержатся скрытые закономерности, тенденции, зависимости. Извлечением таких скрытых знаний в массивах информации и занимается анализа данных (data mining).

Преобразование информации в знания так или иначе сопровождается созданием каких-либо моделей реального мира. Например, анализируя какой-либо массив экономической информации мы можем проследить зависимость одного показателя от другого и, используя методы математической статистики, проверить достоверность такой зависимости и даже вывести

формулу расчета одного показателя на основании другого (или нескольких других). Однако, в настоящее время информации накопилось настолько много, что вручную проанализировать такой объем при помощи людей-ученых не представляется возможным. В таком случае, в игру вступают методы машинного обучения, которые способны на основании большого объема данных самостоятельно (более или менее) построить модель задачи, алгоритм решения которой человек даже не может подробно описать.

Возьмем как пример одну из весьма популярных сейчас задач - создание беспилотных автомобилей. Эту задачу невозможно решить без применения машинного обучения. Ведь, даже если вы - прекрасный водитель, вы не сможете написать подробную программу для компьютера, которая учитывала все факторы и была бы способна вести автомобиль хотя бы так же хорошо, как человек. Слишком много обстоятельств пришлось бы принимать в расчет - конфигурация дороги, разметка, наличие, положение, скорость и ускорение других автомобилей, наличие пешеходов. Кроме того, не стоит забывать, что чтобы вообще выделить такие высокоуровневые понятия как “автомобиль” и “пешеход”, их нужно распознавать на изображении с камер и сенсоров, при помощи которых беспилотный автомобиль получает информацию об окружающем мире. А это отдельная и совсем нетривиальная задача. Программировать беспилотный автомобиль без машинного обучения было бы кошмарно кропотливым, невероятно сложных и обидно неблагодарным занятием, ведь при малейшем изменении обстановки, вся логика работы программы может в корне измениться.

С другой стороны, не стоит воспринимать машинное обучение как решающую все проблемы магию, способную справиться с любой задачей. Методы машинного обучения основаны на довольно простых математических алгоритмах, имеют множество проблем и ограничений и способны решать только определенные задачи. Основные ограничения машинного обучения связаны со скоростью обучения. Грамотно спроектированная модель может решать невероятно сложные задачи, например, распознавания образов, но кто сказал, что это происходит мгновенно. Сложные глубокие нейронные сети требуют работы суперкомпьютеров для их обучения и этот процесс может занимать часы, дни и даже месяцы. Хотя, более простые модели можно создавать и обучать практически мгновенно на довольно скромных вычислительных машинах.

Связь с другими дисциплинами

Машинное обучение, как научная дисциплина, основывается на инструментарии математической многокритериальной оптимизации. Поэтому для понимания принципов работы основных алгоритмов требуется довольно приличное знание математики.

Для понимания самого принципа обучения вам понадобятся знания из области математического анализа, а именно - дифференцирование функций нескольких переменных (multivariate calculus). Оценка качества модели, достоверности и надежности предсказания потребуют знания основ теории вероятности и математической статистики. Если вы хотите понимать доказательства работоспособности алгоритмов машинного обучения, готовьтесь применять глубокие знания в математическом анализе и линейной алгебре.

Даже если вы хотите всего лишь уметь применять простые модели, вам необходимо владеть основами программирования. Вообще, машинное обучение как раздел математики не привязан к конкретному языку программирования, но традиционно, существуют языки, более широко применяющиеся для машинного обучения и имеющие развитую систему библиотек, фреймворков и других инструментов. В частности, одним из самых распространенных языков для данных целей является Python. В его экосистеме присутствуют все необходимые библиотеки для использования практически всех существующих алгоритмов и методов машинного обучения и анализа данных.

Для практического применения обучающихся моделей также неплохо бы владеть теорией вычислений и методами анализа асимптотик алгоритмов.

Линейная регрессия с одной переменной

Представление модели

Напомним, что в задачах регрессии мы принимаем входные переменные и пытаемся подогнать выход на непрерывную ожидаемую функцию результата. Линейная регрессия с одной переменной также известна как «парная линейная регрессия».

Одномерная линейная регрессия используется, когда вы хотите предсказать одно выходное значение *y*, зависящее от одного входного значения *x*. Мы проводим обучение с учителем, это означает, что мы уже имеем представление о том, какие значения выходной переменной соответствуют некоторым значениям входной переменной.

Функция гипотезы

Наша прогностическая функция (функция гипотезы, модель) имеет общий вид:

$$\hat{y} = h_b(x) = b_0 + b_1 x$$

Обратите внимание, что это похоже на уравнение прямой. в данном случае, мы пытаемся подобрать функцию *h(x)* таким образом, чтобы отобразить данные нам значения *x* в данные значения *y*.

Допустим, мы имеем следующий обучающий набор данных:

входная переменная x	выходная переменная y
0	4
1	7
2	7
3	8

Мы можем составить случайную гипотезу с параметрами $b_0 = 2, b_1 = 2$. Тогда для входного значения *x=1, y=4*, что на 3 меньше данного. Задача регрессии состоит в нахождении таких параметров функции гипотезы, чтобы она отображала входные значения в выходные как можно более точно, или, другими словами, описывала линию, наиболее точно ложащуюся в данные точки на плоскости (x, y).

Функция ошибки

Мы можем измерить точность нашей функции гипотезы, используя функцию ошибки. Для этого требуется средняя (фактически чуть усложненная версия среднего арифметического) всех результатов вычисления гипотезы с входами x по сравнению с фактическим выходом y.

$$J(b_0, b_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_b(x_i) - y_i)^2$$

По сути своей, это половина среднего квадрата разницы между прогнозируемым и фактическим значением выходной переменной.

Эту функцию называют «функцией квадрата ошибки» или «среднеквадратичной ошибкой» (mean squared error, MSE). Среднее значение уменьшено вдвое для удобства вычисления градиентного спуска, так как производная квадратичной функции будет отменять множитель 1/2.

Теперь мы можем конкретно измерить точность нашей предсказывающей функции по сравнению с правильными результатами, которые мы имеем, чтобы мы могли предсказать новые результаты, которых у нас нет.

Если мы попытаемся представить это наглядно, наш набор данных обучения будет разбросан по плоскости x-y. Мы пытаемся подобрать прямую линию, которая проходит через этот разбросанный набор данных. Наша цель -

получить наилучшую возможную линию. Лучшая линия будет такой, чтобы средние квадраты вертикальных расстояний рассеянных точек от линии были наименьшими. В лучшем случае линия должна проходить через все точки нашего набора данных обучения. В таком случае значение J будет равно 0.

Метод градиентного спуска

Таким образом, у нас есть наша функция гипотезы, и у нас есть способ оценить, насколько хорошо конкретная гипотеза вписывается в данные. Теперь нам нужно подобрать параметры в функции гипотезы. Вот где приходит на помощь метод градиентного спуска.

Представьте себе, что мы нарисуем нашу функцию гипотезы на основе ее параметров b_0 и b_1 (фактически мы представляем график функции стоимости как функцию оценок параметров).

Отложим b_0 на оси x и b_1 на оси y, с функцией стоимости на вертикальной оси z. Точки на нашем графике будут результатом функции стоимости, используя нашу гипотезу с этими конкретными параметрами.

Мы будем знать, что нам удалось подобрать оптимальные параметры, когда наша функция стоимости находится в самом низу на нашем графике, то есть когда ее значение является минимальным.

То, как мы это делаем - это используя производную (касательную линию к функции) нашей функции стоимости. Наклон касательной является производной в этой точке, и это даст нам направление движения в сторону самого крутого уменьшения значения функции. Мы делаем шаги вниз по функции стоимости в направлении с самым крутым спусками, а размер каждого шага определяется параметром α, который называется скоростью обучения.

Алгоритм градиентного спуска:

повторяйте до сходимости:

$$b_j := b_j - \alpha \frac{\partial}{\partial b_j} J(b_0, b_1)$$

где j=0,1 - представляет собой индекс номера признака.

Линейная регрессия с несколькими переменными

Линейная регрессия с несколькими переменными также известна как «множественная линейная регрессия». Введем обозначения для уравнений, где мы можем иметь любое количество входных переменных:

$x^{(i)}$ - вектор-столбец всех значений признаков i-го обучающего примера;

$x_j^{(i)}$ - значение j-го признака i-го обучающего примера;

m - количество примеров в обучающей выборке;

n - количество признаков;

X - матрица признаков;

b - вектор параметров регрессии.

Заметим, что в будущем для удобства примем, что $x_0^{(i)} = 1$ для всех i. Другими словами, мы для удобства введем некий суррогатный признак, для всех наблюдений равный единице. это сильно упростит математические выкладки, особенно в матричной форме.

Теперь определим множественную форму функции гипотезы следующим образом, используя несколько признаков:

$$h_b(x) = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

Используя определение матричного умножения, наша многопараметрическая функция гипотезы может быть кратко представлена в виде: *h(x) = B X*.

Функция ошибки

Для множественной регрессии функция ошибки от вектора параметров b выглядит следующим образом:

$$J(b) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_b(x^{(i)}) - y^{(i)}\right)^2$$

Или в матричной форме:

$$J(b) = \frac{1}{2m}(Xb - \vec{y})^T(Xb - \vec{y})$$

Метод градиентного спуска для множественной регрессии определяется следующими уравнениями:

повторять до сходимости:

$$b_0 := b_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_b(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)}$$

$$b_1 := b_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_b(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)}$$

$$b_2 := b_2 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_b(x^{(i)}) - y^{(i)}) \cdot x_2^{(i)}$$

. . .

Или в матричной форме:

$$b := b - \frac{\alpha}{m} X^T (Xb - \vec{y})$$

Нормализация признаков

Мы можем ускорить сходимость метода градиентного спуска, получив каждое из наших входных значений примерно в том же диапазоне. Это связано с тем, что b будет быстро сходиться на малых диапазонах и медленно на больших диапазонах, и поэтому будет колебаться неэффективно до оптимального, если переменные очень неравномерны.

Способ предотвратить это - изменить диапазоны наших входных переменных, чтобы они были примерно одинаковыми. В идеале

$$-1 \leq x \leq 1 \text{ или же } -0{,}5 \leq x \leq 0{,}5.$$

Это не точные требования; мы только пытаемся ускорить процесс. Цель состоит в том, чтобы получить все входные переменные в примерно один из этих диапазонов, дать или взять несколько.

Два метода для этого - масштабирование признаков и нормализация по среднему. Масштабирование признаков заключается в делении входных значений на размах выборки (то есть максимальное значение минус минимальное значение) входной переменной, в результате чего новый диапазон составляет всего 1. Нормализация по среднему включает в себя вычитание среднего значения входной переменной из значений для этой входной переменной, в результате чего новое среднее значение для этой переменной равно нулю. Чтобы реализовать оба этих метода, отрегулируйте свои входные значения, как показано в этой формуле:

$$x_i := \frac{x_i - \mu_i}{s_i}$$

Где μ_i - среднее значение признака i, а s_i - стандартное отклонение этого признака.

Советы по методу градиентного спуска

1. Отладка градиентного спуска.

Сделайте график с количеством итераций по оси x. Теперь построим функцию стоимости J(b) по числу итераций градиентного спуска. Если J(b) когда-либо возрастает, то, вероятно, вам необходимо уменьшить α.

2. Автоматический тест сходимости.

Объявите сходимость, если J(b) уменьшится меньше чем на E на одной итерации, где E - некоторое небольшое значение, такое как 10^{-3}. Однако на практике трудно выбрать это пороговое значение.

Было доказано, что если скорость обучения α достаточно мала, то J(b) будет уменьшаться на каждой итерации.

3. Суррогатные признаки

Мы можем улучшить наши функции и форму нашей функции гипотез несколькими способами. Мы можем объединить несколько признаков в один. Например, мы можем объединить x1 и x2 в новый признак x3, взяв x1·x2.

Полиномиальная регрессия

Наша функция гипотезы не обязательно должна быть линейной (прямой), если это не соответствует данным.

Мы можем изменить поведение или кривую нашей функции гипотезы, сделав ее квадратичной, кубической или квадратной корневой функцией (или любой другой формой).

Например, если наша функция гипотезы $\hat{y} = h_b(x) = b_0 + b_1 x$, то мы можем добавить еще один признак, основанный на x1, получив квадратичную функцию $\hat{y} = h_b(x) = b_0 + b_1 x + b_2 x^2$ или кубическую функцию $\hat{y} = h_b(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3$. В кубической функции мы по сути ввели два новых признака: $x_2 = x_2^2, x_3 = x_1^3$. Точно таким же образом, мы можем создать, например, такую функцию: $\hat{y} = h_b(x) = b_0 + b_1 x + b_2 \sqrt{x}$.

Одна важная вещь, о которой следует помнить, заключается в том, что если вы выбираете свои функции таким образом, масштабирование признаков становится очень важным. Например, если x имеет диапазон 1 - 1000, тогда диапазон x^2 становится 1 - 1000000, а диапазон x^3 становится 1 - 1000000000.

Нормальное уравнение

«Нормальное уравнение» - это метод нахождения оптимальных параметров регрессии без итераций:

$$b = (X^T X)^{-1} X^T y$$

Нет необходимости выполнять масштабирование признаков, если мы решаем регрессию с помощью нормального уравнения.

Метод решения через нормальное уравнение имеет ряд преимуществ по сравнению с методом градиентного спуска:

1. Нет необходимости в нормализации признаков;
2. Не нужно выбирать скорость обучения;
3. Не требует вычисления частных производных функции ошибки;

Однако, у него есть и недостатки:

1. Имеет асимптотику $O(n^3)$ по сравнению с $O(n^2)$ у градиентного спуска. Поэтому довольно медленно работает при больших n.
2. Требует вычисления обратной матрицы. В некоторых случаях матрица $X^T X$ может быть вырожденной, что затруднит использование нормального уравнения.

Логистическая регрессия

Теперь мы переходим от регрессионных проблем к задачам классификации. Пусть название «логистическая регрессия» не вводит в заблуждение; метод назван таким образом по историческим причинам и на самом деле является подходом к проблемам классификации, а не регрессионным проблемам.

Бинарная классификация

Вместо того, чтобы наш выходной вектор y был непрерывным диапазоном значений, он будет равен только 0 или 1.

$y \in \{0,1\}$

Где 0 обычно принимается как «отрицательный класс» и 1 как «положительный класс», но вы можете назначить ему какое-либо представление. На данный момент мы занимаемся только двумя классами, называемыми проблемой двоичной или бинарной классификации.

Один из методов - использовать линейную регрессию и отображать все предсказания, превышающие 0,5, как 1 и все меньше 0,5 в качестве 0. Этот метод не работает, потому что классификация на самом деле не является линейной функцией.

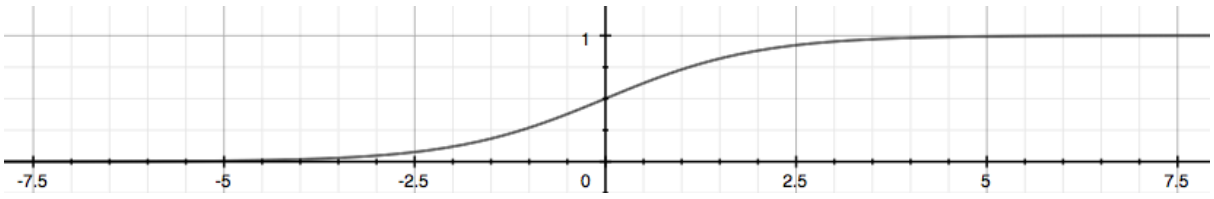
Представление гипотезы

Наша гипотеза должна удовлетворять:

$0 \leq h_b(x) \leq 1$

В этой новой форме задачи используется «сигмоидальная функция», также называемая «логистическая функция»:

$$h_b(x) = g(b^T x)$$
$$z = b^T x$$
$$g(z) = \frac{1}{1 + e^{-z}}$$



Показанная здесь функция g(z) отображает любое вещественное число на интервал (0, 1), что делает его полезным для преобразования произвольнозначной функции в функцию, более подходящую для классификации.

Начнем с нашей старой гипотезы (линейной регрессии), за исключением того, что мы хотим ограничить диапазон до 0 и 1. Это достигается путем подключения $b^T x$ в логистическую функцию.

h_b даст нам вероятность того, что наш результат равен 1. Например, $h_b(x) = 0,7$ дает нам вероятность 70%, что наш выход равен 1.

$$h_b(x) = P(y = 1|x; b) = 1 - P(y = 0|x; b)$$
$$P(y = 0|x; b) + P(y = 1|x; b) = 1$$

Наша вероятность того, что наше предсказание равна 0, является просто дополнением нашей вероятности того, что она равна 1 (например, если вероятность того, что она равна 1, равна 70%, то вероятность того, что она равна 0, равна 30%).

Граница принятия решений

Чтобы получить нашу дискретную классификацию 0 или 1, мы можем перевести вывод функции гипотезы следующим образом:

$$h_b(x) \geq 0.5 \rightarrow y = 1$$
$$h_b(x) < 0.5 \rightarrow y = 0$$

Логистическая функция g ведет себя таким образом, что когда ее вход больше или равен нулю, его выход больше или равен 0,5. Следует запомнить:

$$z = 0, e^0 = 1 \Rightarrow g(z) = 1/2$$
$$z \rightarrow \infty, e^{-\infty} \rightarrow 0 \Rightarrow g(z) = 1$$
$$z \rightarrow -\infty, e^{\infty} \rightarrow \infty \Rightarrow g(z) = 0$$

Отсюда:

$$z = 0, e^0 = 1 \Rightarrow g(z) = 1/2$$
$$z \to \infty, e^{-\infty} \to 0 \Rightarrow g(z) = 1$$
$$z \to -\infty, e^{\infty} \to \infty \Rightarrow g(z) = 0$$

Граница принятия решения - это линия, которая разделяет область, где y = 0 и где y = 1. Она создается нашей функцией гипотезы.

Пример:

$$b = \begin{bmatrix} 5 & -1 & 0 \end{bmatrix}^T$$
$$y = 1 \; if \; 5 + (-1)x_1 + 0x_2 \geq 0$$
$$5 - x_1 \geq 0$$
$$-x_1 \geq -5$$
$$x_1 \leq 5$$

В этом случае наша граница решения является прямой вертикальной линией, расположенной на графике, где $x_1 = 5$, а все слева от нее обозначает y = 1, а все вправо обозначает y = 0.

Функция ошибки

Мы не можем использовать одну и ту же функцию ошибки, которую мы используем для линейной регрессии, потому что логистическая функция вызовет волнообразный вывод, вызывая множество локальных оптимумов. Другими словами, это не будет выпуклая функция.

Вместо этого наша функция стоимости для логистической регрессии выглядит так:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$, где

$$\text{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \text{ if y} = 1$$

$$\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \text{ if y} = 0$$

Чем больше наша гипотеза отключена от y, тем больше выходная функция стоимости. Если наша гипотеза равна y, то наша стоимость равна 0:

$$\text{Cost}(h_b(x), y) = 0 \text{ if } h_b(x) = y$$
$$\text{Cost}(h_b(x), y) \to \infty \text{ if } y = 0 \text{ and } h_b(x) \to 1$$
$$\text{Cost}(h_b(x), y) \to \infty \text{ if } y = 1 \text{ and } h_b(x) \to 0$$

Если наш правильный ответ y равен 0, тогда функция стоимости будет равна 0, если наша функция гипотезы также выдает 0. Если наша гипотеза приближается к 1, то функция стоимости будет приближаться к бесконечности.

Если наш правильный ответ y равен 1, функция стоимости будет равна 0, если наша функция гипотезы выйдет 1. Если наша гипотеза приближается к 0, то функция стоимости приблизится к бесконечности.

Заметим, что запись функции стоимости таким образом гарантирует, что J(b) выпукла для логистической регрессии.

Градиентный спуск

Мы можем сжать два условных случая функции стоимости в один случай:

$$\text{Cost}(h_b(x), y) = -y\ \log(h_b(x)) - (1-y)\log(1-h_b(x))$$

Обратите внимание, что когда y равно 1, то второй член будет равен нулю и не повлияет на результат. Если y равно 0, то первый член будет равен нулю и не повлияет на результат.

Мы можем полностью выписать всю нашу функцию затрат следующим образом:

$$J(b) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_b(x^{(i)})) + (1-y^{(i)})\log(1-h_b(x^{(i)}))]$$

Частная производная от J (θ)

Сначала вычислим производную от сигмоидной функции (она будет полезна при нахождении частной производной от J(θ)):

$$\begin{aligned}\sigma(x)' &= \left(\frac{1}{1+e^{-x}}\right)' = \frac{-(1+e^{-x})'}{(1+e^{-x})^2} = \frac{-1'-(e^{-x})'}{(1+e^{-x})^2} = \frac{0-(-x)'(e^{-x})}{(1+e^{-x})^2} = \frac{-(-1)(e^{-x})}{(1+e^{-x})^2} = \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \left(\frac{1}{1+e^{-x}}\right)\left(\frac{e^{-x}}{1+e^{-x}}\right) = \sigma(x)\left(\frac{+1-1+e^{-x}}{1+e^{-x}}\right) = \sigma(x)\left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) = \sigma(x)(1-\sigma(x))\end{aligned}$$

Теперь мы готовы найти полученную частную производную:

$$\begin{aligned}\frac{\partial}{\partial b_j}J(b) &= \frac{\partial}{\partial b_j}\frac{-1}{m}\sum_{i=1}^{m}\left[y^{(i)}log(h_b(x^{(i)})) + (1-y^{(i)})log(1-h_b(x^{(i)}))\right] \\ &= -\frac{1}{m}\sum_{i=1}^{m}\left[\frac{y^{(i)}\sigma(b^Tx^{(i)})(1-\sigma(b^Tx^{(i)}))\frac{\partial}{\partial b_j}b^Tx^{(i)}}{h_b(x^{(i)})} + \frac{-(1-y^{(i)})\sigma(b^Tx^{(i)})(1-\sigma(b^Tx^{(i)}))\frac{\partial}{\partial b_j}b^Tx^{(i)}}{1-h_b(x^{(i)})}\right] \\ &= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}(1-h_b(x^{(i)}))x_j^{(i)} - (1-y^{(i)})h_b(x^{(i)})x_j^{(i)}\right] \\ &= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} - y^{(i)}h_b(x^{(i)}) - h_b(x^{(i)}) + y^{(i)}h_b(x^{(i)})\right]x_j^{(i)} \\ &= \frac{1}{m}\sum_{i=1}^{m}\left[h_b(x^{(i)}) - y^{(i)}\right]x_j^{(i)}\end{aligned}$$

Помните, что общая форма градиентного спуска:

$$b_j := b_j - \alpha\frac{\partial}{\partial b_j}J(b_0, b_1)$$

Мы можем разработать производную часть, используя вышеприведенное дифференцирование, чтобы получить:

$$b_j := b_j - \frac{\alpha}{m}\sum_{i=1}^{m}(h_b(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Обратите внимание, что этот алгоритм идентичен тому, который мы использовали в линейной регрессии. Мы все равно должны одновременно обновлять все значения в b.

Сопряженный градиентный спуск, BFGS и L-BFGS - это более сложные и быстрые способы оптимизации b, которые можно использовать вместо градиентного спуска. Лучше не писать программные реализации этих более сложных алгоритмов самостоятельно (если только вы не являетесь экспертом в численных вычислениях), но вместо этого используйте библиотеки, поскольку они уже протестированы и сильно оптимизированы. Библиотека scikit learn реализует многие эти алгоритмы.

Многоклассовая классификация: один против всех

Теперь мы рассмотрим классификацию данных более чем в двух категориях. Вместо y = {0,1} мы расширим наше определение так, чтобы y = {0,1 ... n}.

В этом случае мы делим нашу задачу на n + 1 (+1, потому что индекс начинается с 0) двоичных задач классификации; в каждом из них мы прогнозируем вероятность того, что 'y' является членом одного из наших классов.

$$y \in \{0, 1...n\}$$
$$h_b^{(0)}(x) = P(y = 0|x; b)$$
$$h_b^{(1)}(x) = P(y = 1|x; b)$$
$$\cdots$$
$$h_b^{(n)}(x) = P(y = n|x; b)$$
$$\text{prediction} = \max_i(h_b^{(i)}(x))$$

Мы в основном выбираем один класс, а затем объединяем всех остальных в один второй класс. Мы делаем это неоднократно, применяя двоичную логистическую регрессию к каждому случаю, а затем используем гипотезу, которая вернула наивысшее значение в качестве нашего прогноза.

Метод опорных векторов

Метод опорных векторов (support vector machines, SVM) является еще одним типом алгоритма машинного обучения с учителем для задач классификации.

Иногда он работает быстрее и точнее логистической регрессии. Чтобы использовать метод опорных векторов, мы модифицируем первый член функции стоимости $-\log(h_b(x)) = -\log\left(\frac{1}{1+e^{-b^T x}}\right)$, так что при $b^T x$ (здесь и далее будем обозначать эту величину как z) больше 1, оно выводит 0. Кроме того, для значений z, меньших 1, мы будем использовать прямую убывающую линию вместо сигмовидной кривой (в англоязычной литературе это называется функцией hinge loss). Аналогично, мы модифицируем второй член функции стоимости $-\log(1-h_{b(x)}) = -\log\left(1-\frac{1}{1+e^{-b^T x}}\right)$, так что, когда z меньше -1, алгоритм выдает 0. Мы также модифицируем его так, чтобы при значениях z больше -1, вместо сигмоидной кривой мы используем прямую растущую линию.

Мы будем обозначать соответствующие компоненты функции ошибки как $\text{cost}_1(z)$ и $\text{cost}_0(z)$ (соответственно, обратите внимание, что $\text{cost}_1(z)$ является стоимостью для классификации при y = 1 и $\text{cost}_0(z)$ - это стоимость классификации при y = 0), и мы можем определить их следующим образом (где k - произвольная константа, определяющая величину наклона линии):

$z = b^T x$

$\text{cost}_0(z) = \max(0, k(1+z))$

$\text{cost}_1(z) = \max(0, k(1-z))$

Вспомните полную функцию стоимости из регуляризованной логистической регрессии. Мы можем преобразовать ее в функцию стоимости для векторных машин поддержки, заменив cost_0 и cost_1.

$$J(b) = \frac{1}{m}\sum_{i=1}^{m} y^{(i)}\ \text{cost}_1(b^T x^{(i)}) + (1-y^{(i)})\ \text{cost}_0(b^T x^{(i)}) + \frac{\lambda}{2m}\sum_{j=1}^{n} b_j^2$$

Кроме того, общепринятое соглашение диктует, что мы регуляризуем с использованием фактора C вместо λ, тогда функция ошибки будет выглядеть следующим образом:

$$J(b) = C\sum_{i=1}^{m} y^{(i)}\ \text{cost}_1(b^T x^{(i)}) + (1-y^{(i)})\ \text{cost}_0(b^T x^{(i)}) + \frac{1}{2}\sum_{j=1}^{n} b_j^2$$

Это эквивалентно умножению уравнения на $C = \frac{1}{\lambda}$ и, таким образом, приводит к тем же значениям при оптимизации. Теперь, когда мы хотим еще больше регуляризовать (то есть сократить возможность переобучения), мы уменьшаем C, и когда мы хотим регуляризовать меньше (то есть уменьшать возможность недообучения), мы увеличиваем C.

Наконец, обратите внимание, что значение функции гипотезы в методе опорных векторов не интерпретируется как вероятность того, что y будет

равен 1 или 0 (как для гипотезы логистической регрессии). Вместо этого он выводит либо 1, либо 0 дискретно.

Полезный способ думать о методе опорных векторов - как о методе, максимизирующем расстояние между классами и границей принятия решения. Когда мы установим нашу константу C в очень большое значение (например, 100 000), наша оптимизирующая функция будет ограничивать b так, чтобы уравнение суммы ошибки каждого примера равна 0. Мы накладываем следующие ограничения на b:

$$b^T x \geq 1$$

Если C очень велико, мы должны выбрать такие параметры b, что:

$$\sum_{i=1}^{m} y^{(i)} \text{cost}_1(b^T x) + (1 - y^{(i)}) \text{cost}_0(b^T x) = 0$$

Напомним, что граница принятия решения в логистической регрессии - это линия, отделяющая положительные и отрицательные примеры. В SVM граница решения имеет особое свойство, заключающееся в том, что она как можно дальше от положительного и отрицательного примеров.

SVM будет отделять отрицательные и положительные примеры с большим отрывом. Этот большой запас достигается только тогда, когда C очень большой.

Данные называются линейно разделимыми, когда прямая линия, плоскости или гиперплоскость (в зависимости от размерности данных) может отделить положительные и отрицательные примеры. Увеличение и уменьшение C аналогично уменьшению и увеличению λ и может упростить вид границы принятия решения.

Ядра

Ядра (kernels) позволяют нам создавать сложные нелинейные классификаторы с использованием метода опорных векторов.

При данном x, можно ввести новые признаки в зависимости от близости x к определенным заранее выбранным точкам (ориентирам), скажем $l^{(1)}, l^{(2)}, l^{(3)}$. Для этого мы находим «подобие» x и некоторой точки $l^{(i)}$:

$$f_i = similarity(x, l^{(i)}) = \exp(-\frac{||x - l^{(i)}||^2}{2\sigma^2})$$

Эта функция «подобия» называется гауссовским ядром. Это конкретный пример ядра. Существует несколько свойств функции подобия:

- Если $x \approx l^{(i)}$, тогда $f_i = \exp(-\frac{\approx 0^2}{2\sigma^2}) \approx 1$

- Если x далек от $l^{(i)}$, тогда $f_i = \exp(-\frac{(big)^2}{2\sigma^2}) \approx 0$

Другими словами, если x и ориентир близки, то сходство будет близким к 1, и если x и ориентир находятся далеко друг от друга, сходство будет близко к 0.

Каждый ориентир дает нам набор признаков для нашей модели:

$l^{(1)} \rightarrow f_1$

$l^{(2)} \rightarrow f_2$

$l^{(3)} \rightarrow f_3$

. . .

$h_b(x) = b_1 f_1 + b_2 f_2 + b_3 f_3 + \ldots$

σ^2 - параметр гауссовского ядра, и его можно модифицировать, чтобы увеличить или уменьшить область действия нашей функции f_i. В сочетании с изменением значений внутри b, мы можем выбрать эти ориентиры, чтобы получить общую форму границы принятия решения.

Один из способов получить ориентиры - разместить их в тех же точках гиперпространства, что и все учебные примеры. Это дает нам набор ориентиров, с одним ориентиром на каждый пример обучения. Это дает нам «вектор функций» $f_{(i)}$ для всех наших изначальных признаков, например $x_{(i)}$. Мы можем также установить $f_0 = 1$, чтобы соответствовать b_0. Таким образом, данный пример обучения $x_{(i)}$.

Теперь, чтобы получить параметры b, мы можем использовать алгоритм минимизации SVM, но с заменой $x^{(i)}$ на $f^{(i)}$:

$$\min_b C \sum_{i=1}^{m} y^{(i)} \text{cost}_1(b^T f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(b^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} b_j^2$$

Использование ядер для генерации $f_{(i)}$ не является исключительным для SVM и может также применяться к логистической регрессии. Однако из-за вычислительной оптимизации в алгоритмах SVM, ядра работают с SVM, намного быстрее, чем с другими алгоритмами, поэтому ядра почти всегда встречаются вместе только с SVM.

Выбор параметров SVM

Выбор C (напомним, что $C = \frac{1}{\lambda}$) производится с учетом следующих условия:

- Если C велико, то мы получаем более высокую дисперсию / низкое смещение

- Если C мало, то мы получаем более низкую дисперсию / более высокое смещение

Другим параметром, который мы должны выбрать, является σ^2 для функции гауссовского ядра:

- При большом σ^2 характеристики f_i изменяются более плавно, что приводит к более высокому смещению и более низкой дисперсии.
- При малом σ^2 функции f_i изменяются менее плавно, вызывая более низкое смещение и более высокую дисперсию.

В практическом применении выбор, который вам нужно сделать, это:

- Выбор параметра C
- Выбор ядра (функция подобия)
    - Нет ядра («линейное» ядро) - дает стандартный линейный классификатор. Выберем, когда n велико, а m мало
    - Гауссовское ядро (описанное выше) - нужно выбрать σ^2. Выберите, когда n мало и m велико
    - Полиномиальное: $\mathbf{k}(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + \text{const})^d$
    - Радиальная базисная функция: $\mathbf{k}(\mathbf{x}, \mathbf{x}') = e^{-\gamma ||\mathbf{x} - \mathbf{x}'||^2}$, $\gamma > 0$.
    - Гауссова радиальная базисная функция: $\mathbf{k}(\mathbf{x}, \mathbf{x}') = e^{-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}}$.
    - Сигмоид: $\mathbf{k}(\mathbf{x}, \mathbf{x}') = \tanh(\kappa \langle \mathbf{x}, \mathbf{x}' \rangle + c), \ \kappa > 0, c < 0$.

Выполняйте масштабирование признаков перед использованием гауссовского ядра.

Не все функции подобия являются допустимыми ядрами. Они должны удовлетворять теореме Мерсера, которая гарантирует правильную работу оптимизаций пакета SVM и не расходится.

Регуляризация

Проблема переобучения

Регуляризация предназначена для решения проблемы переобучения.

Недообучение - это проблема выбора гипотезы, когда форма нашей функции h плохо отражает тренд данных. Обычно это вызвано слишком простой функцией или использует слишком мало функций. например. если взять $h_{theta(x) = theta_0 + theta_1 x_1 + theta_2 x_2}$, то мы делаем первоначальное предположение о том, что линейная модель хорошо подгоняет учебные данные и сможет обобщить, но это может быть не так.

С другой стороны, чрезмерная или высокая дисперсия или переобучение вызвано функцией гипотезы, которая подходит к имеющимся данным, но не позволяет хорошо обобщать предсказание новых данных. Обычно это вызвано сложной функцией, которая создает много ненужных кривых и углов, не связанных с данными.

Эта терминология применяется как к линейной, так и к логистической регрессии. Существует два основных варианта решения проблемы переобучения:

1) Уменьшить количество признаков:

a) Вручную выберите, какие признаки сохранить.

b) Использовать алгоритм выбора модели.

2) Регуляризация

Сохраните все признаки, но уменьшите параметры b_j.

Регуляризация работает хорошо, когда у нас много полезных признаков.

Функция ошибки

Если мы переобучаем нашу функцию гипотезы, мы можем уменьшить вес, который привносят некоторые члены в нашей функции, увеличивая их вклад в функцию ошибки.

Скажем, мы хотели сделать следующую функцию более квадратичной:

$$b_0 + b_1x + b_2x^2 + b_3x^3 + b_4x^4$$

Мы хотим устранить влияние b_3 и b_4. Если мы не избавимся от этих признаков и не изменим форму нашей гипотезы, мы можем вместо этого изменить нашу функцию стоимости:

$$min_b \frac{1}{2m}\sum_{i=1}^{m}(h_b(x^{(i)}) - y^{(i)})^2 + 1000 \cdot b_3^2 + 1000 \cdot b_4^2$$

В конце мы добавили два дополнительных термина, чтобы раздуть стоимость b_3 и b_4. Теперь, чтобы функция ошибки приблизилась к нулю, нам нужно будет уменьшить значения b_3 и b_4 до нуля. Это в свою очередь значительно уменьшит значения b_3x^3 и b_4x^4 в нашей функции.

Мы могли бы также регуляризовать все наши параметры в одном суммировании:

$$min_b \frac{1}{2m}\left[\sum_{i=1}^{m}(h_b(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} b_j^2\right]$$

где λ или лямбда - параметр регуляризации. Он определяет, насколько сильно штрафуются высокие значения параметров регрессии.

Используя вышеприведенную функцию стоимости с дополнительным суммированием, мы можем сгладить выход нашей функции гипотез, чтобы уменьшить переобучение. Если лямбда выбрана слишком большой, она может слишком сильно сгладить функцию и вызвать недооценку.

Регулярная линейная регрессия

Мы можем применить регуляризацию как к линейной регрессии, так и к логистической регрессии. Сначала рассмотрим случай линейной регрессии.

Градиентный спуск

Мы изменим нашу функцию градиентного спуска, чтобы отделить b_0 от остальных параметров, потому что мы не хотим штрафовать b_0:

$$b_0 := b_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_b(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$b_j := b_j - \alpha \left[\left(\frac{1}{m} \sum_{i=1}^{m} (h_b(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) + \frac{\lambda}{m} b_j \right] \qquad j \in \{1, 2...n\}$$

Множитель $\frac{\lambda}{m} b_j$ выполняет регуляризацию.

С некоторыми манипуляциями наше правило обновления также может быть представлено как:

$$b_j := b_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_b(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Первый член в приведенном выше уравнении, $1 - \alpha\frac{\lambda}{m}$ всегда будет меньше 1. Интуитивно вы можете видеть его как уменьшение значения b_j на некоторую величину при каждом обновлении.

Обратите внимание, что второй член теперь точно такой же, как и раньше.

Нормальное уравнение

Теперь давайте подходим к регуляризации с использованием альтернативного метода неитеративного нормального уравнения.

Чтобы добавить в регуляризацию, уравнение совпадает с нашим оригиналом, за исключением того, что мы добавляем еще один член в круглые скобки:

$$b = \left(X^T X + \lambda \cdot L\right)^{-1} X^T y$$

$$L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$$

L - матрица с 0 в левом верхнем углу и 1 вниз по диагонали, 0 - везде вне диагонали. Она должна иметь размерность (n + 1) × (n + 1). Интуитивно это - единичная матрица (хотя мы не включаем x_0), умноженная на одно действительное число λ.

Напомним, что если m ≤ n, то матрица $X^T X$ необратима. Однако, когда мы добавляем член λ·L, то $X^T X + L$ становится обратимой матрицей.

Регулярная логистическая регрессия

Мы можем регуляризовать логическую регрессию так же, как мы регуляризируем линейную регрессию. Начнем с функции ошибки. Можно регуляризовать функцию ошибки, добавив к концу дополнительное слагаемое:

$$J(b) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\ \log(h_b(x^{(i)})) + (1-y^{(i)})\ \log(1-h_b(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n} b_j^2$$

Как и при линейной регрессии, мы захотим отдельно обновить b_0 и остальные параметры, потому что мы не хотим регуляризовать b_0.

$$b_0 := b_0 - \alpha\ \frac{1}{m}\ \sum_{i=1}^{m}(h_b(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$b_j := b_j - \alpha\ \left[\left(\frac{1}{m}\ \sum_{i=1}^{m}(h_b(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}b_j\right] \qquad j \in \{1, 2...n\}$$

Это идентично алгоритму градиентного спуска, представленному для линейной регрессии.

Единичный вектор постоянного признака

Как оказалось, очень важно добавить постоянный признак в свой обучающий набор, прежде чем начинать обучение модели. Обычно этот признак - всего лишь набор из единиц для всех ваших учебных примеров. Конкретно, если X - матрица признаков, то X_0 является вектором с единицами.

Предположим, вы проектируете модель, которая предсказывает цену дома на основе некоторых признаков. В этом случае чем помогает этот единичный

вектор? Предположим, что простая модель включает признаки, которые прямо пропорциональны ожидаемой цене, то есть если признак X_i увеличивается, ожидаемая цена y также будет увеличиваться. Так, например, мы могли бы иметь два признака: размер дома в $м^2$ и количество комнат.

Когда вы обучаете свою модель, вы начнете с добавления одного вектора X_0. Затем вы можете после обучения узнать, что вес для вашей первоначальной функции - это некоторое значение b_0. Но что это значит на практике? Давайте предположим, что кто-то знает, что у вас есть рабочая модель для цен на жилье. Оказывается, что для этого примера, если они спросят, сколько денег они могут ожидать, продавая дом, вы можете сказать, что как минимум b_0 долларов (или рандов), прежде чем вы даже используете свою обученную модель. Как и вышеприведенная аналогия, постоянная b_0 несколько устойчива, когда все ваши входы являются нулями. Конкретно, это цена дома без комнат, который не занимает никакую площадь.

Однако это объяснение имеет некоторые недостатки, потому что, если у вас есть некоторые функции, которые снижают цену, например возраст дома, то b_0 не может быть абсолютным минимумом цены. Это связано с тем, что возраст может снизить цену. Более простой и грубый способ выразить это состоит в том, что компонент b_0 вашей модели представляет собой неотъемлемое смещение модели. Другие функции затем выражают импульс, чтобы отойти от этой позиции смещения.

Признак «смещения» (bias feature) - это просто способ переместить вектор наилучшего соответствия, чтобы лучше соответствовать данным. Например, рассмотрим проблему обучения с помощью одного признака X_1. Формула без функции X_0 - это просто $b_1 * X_1 = y$. Это показано как линия, которая всегда проходит через начало координат с наклоном y / theta. слагаемое x_0 позволяет линии проходить через другую точку на оси y. Это почти всегда будет лучше соответствовать. Не все же наилучшие подходящие линии проходят через начало координат (0,0)?

Диагностика систем машинного обучения

Ошибки в ваших прогнозах можно устранить с помощью следующих способов:

1. Получение дополнительных примеров в обучающую выборку;
2. Попытка использования меньших наборов признаков;
3. Попытка использования дополнительных признаков;
4. Попытки использования полиномиальных признаков;
5. Увеличение или уменьшение λ.

Не следует просто выбирать один из этих путей наугад. Мы рассмотрим диагностические методы для выбора одного из вышеуказанных решений путем тщательного исследования процесса обучения модели.

Оценка качества модели

Гипотеза может иметь низкую ошибку для примеров из обучающей выборки, но все же быть неточной (из-за переобучения). Используя набор обучающих примеров, мы можем разбить данные на два набора: набор для обучения и набор для тестов.

Новая процедура с использованием этих двух наборов:

1. Обучите b и минимизируйте $J_{train}(b)$ с помощью обучающего набора;
2. Вычислите ошибку тестового набора $J_{test}(b)$

Ошибка тестового набора

для регрессии:

$$J_{test}(b) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_b(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

для классификации:

$$J(b) = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_b(x_{test}^{(i)}), y_{test}^{(i)})$$

. где

$$err(h_b(x), y) = \begin{matrix} 1 & \text{if } h_b(x) \geq 0.5 \; and \; y = 0 \; or \; h_b(x) < 0.5 \; and \; y = 1 \\ 0 & else \end{matrix}$$

Ошибка классификации дает нам долю тестового набора, которая была ошибочно классифицирована.

Выбор модели (model selection)

Просто потому, что алгоритм обучения хорошо подходит для тренировочного набора, это не значит, что это хорошая гипотеза. Ошибка вашей гипотезы, измеренная в наборе данных, с которым вы подготовили параметры, будет ниже, чем на любом другом наборе данных.

Чтобы выбрать модель вашей гипотезы, вы можете проверить каждую степень полинома и посмотреть на результат ошибки. Метод без использования третьего, валидационного набора (обратите внимание: это плохой метод - не используйте его) выглядел бы следующим образом:

1. Оптимизируйте параметры b, используя обучающий набор для каждой степени полинома.

2. Найдите степень полинома d с наименьшей ошибкой, используя тестовый набор.
3. Оцените ошибку обобщения, также используя тестовый набор с $J_{test}(b^{(d)})$, (d = theta от полинома с более низкой ошибкой);

В этом случае мы обучили одну переменную d или степень полинома, используя тестовый набор. Это приведет к тому, что наше значение ошибки будет больше для любого другого набора данных.

Подобные переменные, не являющиеся параметрами гипотезы, но так или иначе влияющие на точность нашей модели называются гиперпараметрами модели. Обучение гиперпараметров - более сложный и долгий путь и он требует проведения специальной процедуры.

Использование валидационного набора

Чтобы решить эту проблему, мы можем ввести третий набор, валидационный набор, чтобы использовать его как промежуточный набор для обучения гиперпараметров. Тогда наш тестовый набор даст нам точную, не оптимистичную ошибку.

Один из возможных примеров разбивки нашего набора данных на три набора:
1. Обучающий набор (train set): первые 60%;
2. Валидационный набор (validation set): следующие 20%;
3. Тестовый набор (test set): последние 20%.

Этот лишь одна из самых простых схем разбиения набора данных. С помощью специальной техники - кросс-валидации или перекрестной проверки - процесс разбиения набора автоматизируется и становится более робастным. Пока не будем рассматривать эту продвинутую технику выбора модели.

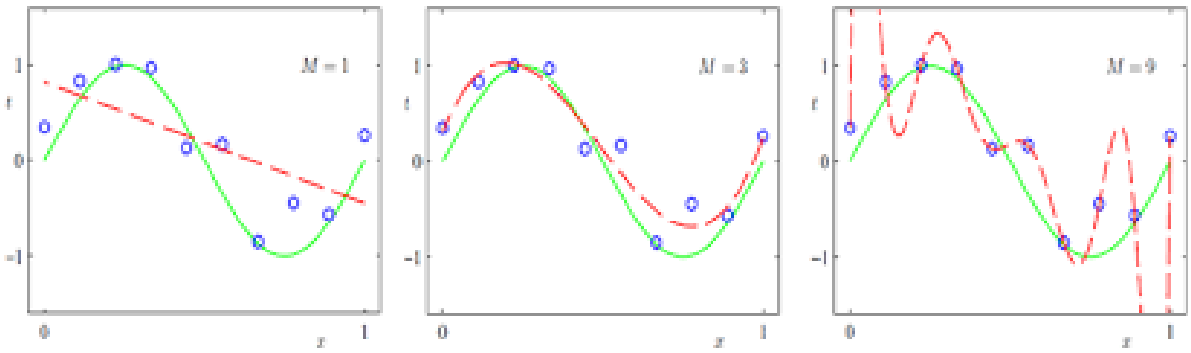
Теперь мы можем вычислить три отдельных значения ошибки для трех разных наборов.

Метод выбора гипотезы с помощью валидационного набора (обратите внимание: этот метод предполагает, что мы не используем регуляризацию и на валидационном наборе)
1. Оптимизируйте параметры b, используя набор тренировок для каждой степени полинома.
2. Найдите степень полинома d с наименьшей ошибкой, используя валидационный набор.
3. Оцените ошибку обобщения, используя тестовый набор с $J_{test}(b^{(d)})$(где b - параметры от полинома с более низкой ошибкой);

Таким образом, степень полинома d не была обучена с использованием тестового набора.

Имейте в виду, что использование валидационного набора для выбора d означает, что мы также не можем использовать его для процесса проверки правильности установки значения лямбда.

Диагностика недо- и переобучения

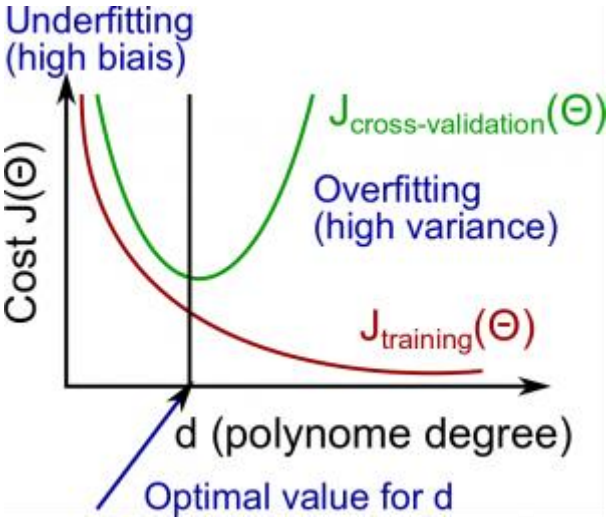


В этом разделе мы рассмотрим взаимосвязь между степенью полинома d и недо- или переобучением нашей гипотезы. Нам нужно различать, является ли проблемой, которая способствует плохим предсказаниям, смещение (bias) или дисперсия (variance) функции гипотезы. Высокое смещение чаще недообучается, а высокая дисперсия чаще переобучается. Нам нужно найти золотую середину между этими двумя крайностями.

Ошибка обучения будет уменьшаться по мере увеличения степени d многочлена. В то же время ошибка перекрестной проверки будет уменьшаться по мере увеличения d до точки, а затем она увеличивается с увеличением d, образуя выпуклую кривую.

- Высокое смещение (недообучение): как $J_{train}(b)$, так и $J_{CV}(b)$ будут высокими. Кроме того, $J_{CV}(b) \approx J_{train}(b)$.
- Высокая дисперсия (переобучение): $J_{train}(b)$ будет низкой, а $J_{CV}(b)$ будет намного больше, чем $J_{train}(b)$.

При выборе из многих значений d можно построить зависимость ошибок на обучающем и валидационных наборах на графике зависимости от d. Такой график наглядно показывает области недо- и переобучения и выглядит подобно следующему:



Регуляризация и переобучение

Вместо того, чтобы смотреть на степень d, способствующую смещению / дисперсии, теперь мы рассмотрим параметр регуляризации λ.

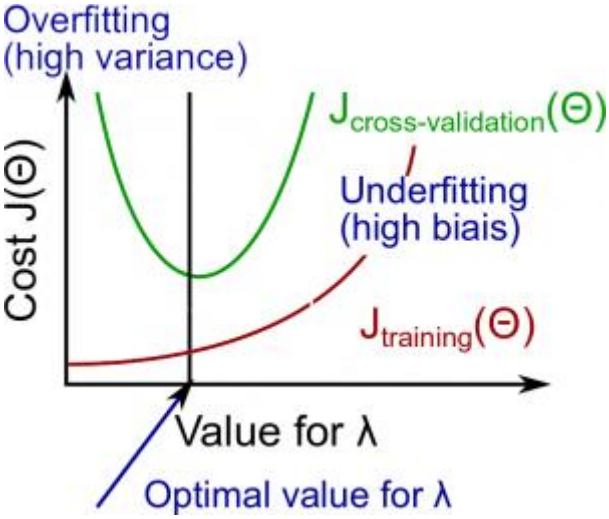
- Большой λ: высокое смещение (недообучение);
- Малый λ: высокая дисперсия (переобучение).

Большая лямбда сильно штрафует все параметры b, что значительно упрощает линию нашей результирующей функции, поэтому вызывает недообучение.

Отношение λ к набору тренировок и набору дисперсий выглядит следующим образом:

- Низкий λ: $J_{train}(b)$ низкий, а $J_{CV}(b)$ высокий (высокая дисперсия / переобучение);
- Оптимальное значение λ: $J_{train}(b)$ и $J_{CV}(b)$ низки и $J_{train}(b) \approx J_{CV}(b)$;
- Большой λ: оба $J_{train}(b)$ и $J_{CV}(b)$ будут высокими (недообучение / высокое смещение).

Опять же, можно построить зависимость ошибок от параметра регуляризации для наглядного поиска оптимального значения λ:

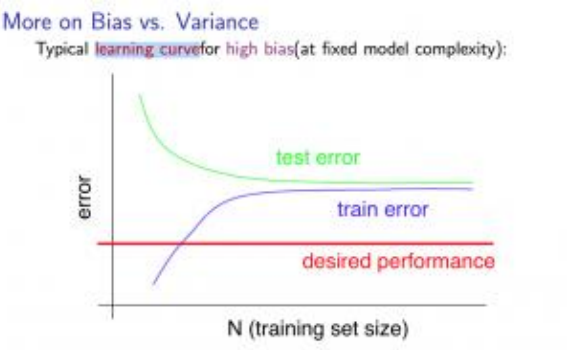


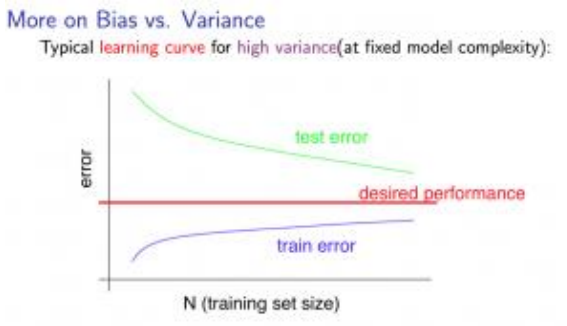
Чтобы выбрать вид модели и параметр регуляризации λ одновременно, нам нужно:

1. Создайте список возможных значения λ (например, λ∈ {0, 0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64, 1.28, 2.56 ,5.12, 10.24});
2. Создайте набор моделей с разной степенью или любыми другими вариантами;
3. Начните итерировать по λ и для каждого λ итерировать по всем моделям, чтобы обучать параметры b;
4. Вычислите ошибку валидационного набора, используя обученные b (вычисленные с λ) на $J_{CV}(b)$ без регуляризации или λ = 0;
5. Выберите лучшую комбинацию, которая дает самую низкую ошибку на валидационном наборе;
6. Используя наилучшую комбинацию b и λ, примените ее на $J_{test}(b)$, чтобы увидеть, дает ли она хорошее обобщение задачи.

Кривые обучения

Тренировка квадратичной функции на трех примерах легко даст нулевую ошибку, потому что мы всегда можем найти квадратичную кривую, которая точно касается 3 точек. По мере увеличения обучающего набора ошибка для квадратичной функции будет возрастать. Значение ошибки будет стабилизироваться после определенного значения m или размера набора тренировок.





Если алгоритм обучения страдает от высокого смещения, получение большего количества данных по обучению не может (само по себе) существенно улучшить эффективность. Если алгоритм обучения страдает от большой дисперсии, скорее всего, получение большего количества данных по обучению повысит общую эффективность обучения.

Типичным правилом при выполнении диагностики является:

- Дополнительные примеры в обучающей выборке устраняют высокую дисперсию, но не помогают при высоком смещении;
- Меньшие функции устраняют высокую дисперсию, но помогают пир высоком смещении;
- Дополнительные признаки устраняют высокое смещение, но не высокую дисперсию;
- Добавление полиномиальных признаков устраняют высокое смещение, но не высокую дисперсию;
- При использовании градиентного спуска уменьшение лямбда может исправить высокое смещение, и увеличение лямбда может исправить высокую дисперсию;
- При использовании нейронных сетей небольшие нейронные сети более подвержены недобучению, а большие - переобучению. Перекрестная проверка размера сети - это хороший способ выбора архитектуры нейронной сети.

Рекомендуемый подход к решению проблем машинного обучения:

- Начните с простого алгоритма, быстро реализуйте его и проверьте его на ранней стадии.
- Стройте кривые обучения, чтобы решить, поможет ли добавление данных, признаков или что-то другое
- Анализ ошибок: вручную проверьте ошибки на примерах в валидационном наборе и попытайтесь определить общие зависимости.

Важно, чтобы результаты проверки качества модели были выражены как одно интегральное числовое значение. В противном случае сложно оценить производительность вашего алгоритма.

Метрики ошибок для смещенных классов

Иногда трудно сказать, является ли уменьшение ошибки на самом деле улучшением алгоритма.

Например: при прогнозировании диагнозов рака на обучающей выборке, где у 0,5% примеров есть рак, мы обнаруживаем, что наш алгоритм после обучения имеет ошибку в 1%. Однако, если бы мы просто классифицировали каждый отдельный пример как 0, то наша ошибка уменьшилась бы до 0,5%, хотя мы не улучшили алгоритм.

Обычно такая ситуация происходит при использовании наборов данных со смещенными классами; то есть, когда один из классов очень редок во всем наборе данных. Или, по-другому говоря, когда у нас есть больше примеров из одного класса, чем из другого класса.

Для этого мы можем использовать метрики, связанные с понятиями полноты и точности(precision/recall).

Рассмотрим возможные варианты классификации обучающих примеров в задаче бинарной классификации:

- Прогнозируемый: 1, Фактический: 1 - Истинный положительный (true positive, TP);
- Прогнозируемый: 0, Фактический: 0 - Истинный отрицательный (true negative);
- Прогнозируемый: 0, Фактический, 1 - Ложноотрицательный (false negative);
- Прогнозируемый: 1, Фактический: 0 - Ложноположительный (false positive).

Классическая метрика точности (accuracy) показывает следующее: из всех пациентов, какую долю мы предсказали правильно, то есть $A = \frac{TP + TN}{TP + FP + TN + FN}$.

Тем не менее, у этой метрики есть одна особенность которую необходимо учитывать. Она присваивает всем примерам одинаковый вес, что может быть некорректно в случае если распределение документов в обучающей выборке сильно смещено в сторону какого-то одного или нескольких классов. В этом случае у классификатора есть больше информации по этим классам и соответственно в рамках этих классов он будет принимать более адекватные решения. На практике это приводит к тому, что вы имеете точность, скажем, 80%, но при этом в рамках какого-то конкретного класса классификатор работает из рук вон плохо не определяя правильно даже треть документов.

Точность (precision): из всех пациентов, у которых мы предсказали y=1, какая доля действительно имеет рак?

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

Полнота (recall): из всех пациентов, у которых действительно рак, у какой доли мы его распознали?

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

Эти две метрики дают нам лучшее представление о том, как работает наш классификатор. Мы хотим, чтобы точность и отзыв были высокими.

В примере в начале раздела, если мы классифицируем всех пациентов как 0, тогда наш отзыв будет $\frac{0}{0 + f} = 0$, поэтому, несмотря на более низкий процент ошибок, мы можем быстро увидеть, что этот алгоритм хуже.

Обратите внимание: если алгоритм предсказывает только один класс, как в одном из примеров, точность не определена, так как невозможно делить на 0.

Нам может потребоваться уверенное предсказание двух классов с использованием логистической регрессии. Один из способов - увеличить наш порог предсказания:

- Прогнозировать 1, если: $h_b(x) \geq 0.7$
- Прогнозировать 0, если: $h_b(x) < 0.7$

Таким образом, мы прогнозируем рак, только если у пациента есть вероятность как минимум 70%. Выполняя это, мы будем иметь более высокую точность, но более низкую полноту.

В противоположном примере мы можем снизить наш порог:

- Прогнозировать 1, если: $h_b(x) \geq 0.3$
- Прогнозировать 0, если: $h_b(x) < 0.3$

Таким образом, мы получаем очень безопасное предсказание. Это приведет к более высокой полноте, но более низкой точности. Чем выше пороговое значение, тем больше точность и тем ниже полнота. Чем ниже порог, тем больше полнота и тем ниже точность. Чтобы превратить эти две метрики в одно число, мы можем взять значение F.

Один из способов - взять среднее: $\frac{P+R}{2}$. Это не работает. Если мы прогнозируем все y = 0, то это приведет к среднему показателю, несмотря на то, что полнота такого алгоритма равна 0. Если мы прогнозируем все примеры как y = 1, то очень высокая полнота вызовет среднее значение, несмотря на точность 0.

Лучше всего вычислить оценку F1 (F-Score) - среднее геометрическое из этих двух показателей:

$$\text{F-Score} = 2\frac{PR}{P+R}$$

Чтобы показатель F был высоким, как точность, так и отзыв должны быть высокими.

Данные для машинного обучения

В некоторых случаях «неполный алгоритм» при наличии достаточных данных может превосходить превосходный алгоритм с меньшим количеством данных.

Мы должны выбрать наши признаки таким образом, чтобы иметь достаточно информации для построения адекватной модели. Полезным эмпирическим тестом для проверки достаточности набора признаков является следующее правило: учитывая вход x, сможет ли эксперт-человек уверенно предсказать y?

Если у нас есть алгоритм с низким смещением, то чем больше мы используем обучающий набор, тем меньше мы будем переообучаться и тем точнее алгоритм будет работать на тестовой выборке. Таким образом можно сформулировать золотое правило машинного обучения: нужно взять как можно больше данных и как можно более вариативный алгоритм.

Обучение с помощью больших наборов данных

В некоторых областях существует возможность использовать для машинного обучения весьма большие наборы данных. Наборы данных часто могут приближаться к таким размерам, как m = 100 000 000. В этом случае наш шаг градиентного спуска должен будет суммировать все сто миллионов примеров. Мы хотим попытаться избежать этого - подходы для этого описаны ниже.

Стохастический градиентный спуск

Стохастический градиентный спуск (stochastic gradient decsent, SGD) является альтернативой классическому (или пакетному, batch gradient descent, BGD) градиентному спуску, более эффективен и масштабируется для больших наборов данных.

Стохастический градиентный спуск формализуетсяя другим, но схожим образом:

$$cost(b, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_b(x^{(i)}) - y^{(i)})^2$$

Единственная разница в вышеперечисленной функции затрат заключается в устранении константы m в $$ \dfrac{1}{2} $$

$$J_{train}(b) = \frac{1}{m}\sum_{i=1}^{m} cost(b, (x^{(i)}, y^{(i)}))$$

Алгоритм выглядит следующим образом

Случайно «перетасовать» набор данных

Для i = 1 ... m

$$b_j := b_j - \alpha(h_b(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Этот алгоритм будет пытаться одновременно подгонять один пример обучения. Таким образом, мы можем добиться прогресса в градиентном спуске без необходимости сначала сканировать все примеры обучения. Стохастический градиентный спуск вряд ли сходится на глобальном минимуме и вместо этого будет беспорядочно перемещаться вокруг него, но обычно дает результат, достаточно близкий. Стохастический градиентный

спуск обычно занимает 1-10 проходов через набор данных, чтобы приблизиться к глобальному минимуму.

Мини-пакетный градиентный спуск

Мини-пакетный градиентный спуск иногда может быть даже быстрее, чем стохастический градиентный спуск. Вместо использования всех m-примеров, как при пакетном градиентном спуске, и вместо использования только одного примера, как при стохастическом градиентном спуске, мы будем использовать некоторое промежуточное число примеров b.

Типичные значения для b варьируются от 2 до 100 или около того.

Например, при b = 10 и m = 1000:

Повторение:

Для i = 1,11,21,31, ..., 991

$$b_j := b_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_b(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

Мы просто суммируем более десяти примеров за раз. Преимущество вычисления более чем одного примера за раз заключается в том, что мы можем использовать векторизованные реализации над примерами b.

Стохастическая градиентная сходимость

Как выбрать скорость обучения α для стохастического градиентного спуска? Кроме того, как мы отлаживаем стохастический градиентный спуск, чтобы убедиться, что он приближается к глобальному оптимуму?

Одна из стратегий заключается в построении средней стоимости гипотезы, применяемой к кажущимся 1000 примерам обучения. Мы можем вычислить и сохранить эти затраты во время итераций спуска градиента.

С меньшей скоростью обучения возможно, что вы можете получить немного лучшее решение со стохастическим градиентным спуском. Это связано с тем, что стохастический градиентный спуск будет колебаться и скакать по глобальному минимуму, и он будет делать небольшие случайные скачки с меньшей скоростью обучения.

Если вы увеличите количество примеров, в среднем превышающих производительность вашего алгоритма, линия графика станет более плавной. При очень небольшом числе примеров для среднего, линия будет слишком шумной, и будет трудно найти тренд.

Одна из стратегий, позволяющих фактически сходиться на глобальном минимуме, заключается в медленном уменьшении α по времени. Однако это не часто делается, потому что люди не хотят возиться с еще большим количеством гиперпараметров.

Онлайн обучение

При непрерывном потоке пользователей на веб-сайт мы можем запустить бесконечный цикл, который получает (x, y), где мы собираем некоторые действия пользователя для функций в x, чтобы предсказать некоторое поведение y.

Вы можете обновлять b для каждой отдельной пары (x, y) по мере их сбора. Таким образом, вы можете адаптироваться к новым действиям пользователей, так как вы постоянно обновляете параметры модели.

Кластеризация

Обучение без учителя является противоположностью обучению с учителем, потому что оно использует немаркированный набор тренировок, а не размеченный. Другими словами, у нас нет вектора y ожидаемых результатов, у нас есть только набор данных, в которых мы можем найти структуру.

Кластеризация (или кластерный анализ) — это задача разбиения множества объектов на группы, называемые кластерами. Внутри каждой группы должны оказаться «похожие» объекты, а объекты разных группы должны быть как можно более отличны. Главное отличие кластеризации от классификации состоит в том, что перечень групп четко не задан и определяется в процессе работы алгоритма.

Кластеризация хороша для:

- Сегментация рынка
- Анализ социальной сети
- Организация компьютерных кластеров
- Астрономический анализ данных

Представление модели

Пусть X — множество объектов, Y — множество номеров (имён, меток) кластеров. Задана функция расстояния между объектами $\rho(x, x')$. Имеется конечная обучающая выборка объектов $X^m = \{x_1, \ldots, x_m\} \subset X$. Требуется разбить выборку на непересекающиеся подмножества, называемые кластерами, так, чтобы каждый кластер состоял из объектов, близких по

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

Наши основные переменные:

- K (количество кластеров)
- Тренировочный набор $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$
- Где $x^{(i)} \in \mathbb{R}^n$

Алгоритм K средних

Алгоритм k-средних (K-Means) является самым популярным и широко используемым алгоритмом для автоматической группировки данных в когерентные подмножества.

Основная идея алгоритма выражена в следующих шагах:

1. Случайно инициализируйте две точки в наборе данных, называемые центроидами кластера.
2. Назначение кластера: назначьте все примеры в одну из двух групп на основе того, какой кластер-центроид ближе всего к примеру.
3. Переместите центроид: вычислите средние значения для всех точек внутри каждой из двух групп кластеров кластеров, а затем переместите точки центроида кластера на эти средние.
4. Повторно запустите (2) и (3), пока мы не найдем наши кластеры.

Обратите внимание, что мы не будем использовать соглашение $x_0 = 1$.

Первый цикл алгоритма - это назначение центроидов кластеров. Мы создаем вектор c, где $c^{(i)}$ представляет центроид, назначенный примеру $x^{(i)}$. Мы можем написать операцию шага назначения кластеров математически следующим образом:

$$c^{(i)} = argmin_k \; ||x^{(i)} - \mu_k||^2$$

Каждый элемент $c^{(i)}$ содержит индекс центроида, который имеет минимальное расстояние до $x^{(i)}$.

По соглашению мы возводим правую сторону в квадрат, что делает функцию, которую мы пытаемся свести к минимуму, более быстро возрастающей. Это в основном просто удобное соглашение. Но соглашение, которое помогает уменьшить количество вычислений, потому что евклидово расстояние требует квадратного корня, но отменяется.

Второй цикл алгоритма - это перемещение центроидов, где мы перемещаем каждый центроид в среднем по своей группе. Более формально уравнение для этого цикла выглядит следующим образом:

$$\mu_k = \frac{1}{n}[x^{(k_1)} + x^{(k_2)} + \cdots + x^{(k_n)}] \in \mathbb{R}^n$$

Где каждый из $x^{(k_1)}, x^{(k_2)}, \ldots, x^{(k_n)}$ - примеры обучения, назначенные группе k.

Если у вас есть центроид кластера, которому соответствуют 0 точек, вы можете случайно повторно инициализировать этот центроид для новой точки. Вы также можете просто исключить эту группу кластеров.

После ряда итераций алгоритм сходится, и новые итерации не влияют существенно на положение центроидов кластеров.

Некоторые наборы данных не имеют реального внутреннего разделения или естественной структуры. Алгоритмы типа К-средних могут равномерно сегментировать такие данные в подмножества, поэтому даже в этом случае могут быть полезны.

Функция ошибки

Вспомним некоторые параметры, которые мы использовали в нашем алгоритме:

$c^{(i)}$ - индекс кластера (1,2, ..., K), к которому в настоящее время назначается пример x (i)

μ_k - кластеризованный центроид k ($\mu_k \in \mathbb{R}^n$)

$\mu_{c^{(i)}}$ - кластерный кластер кластера, которому присвоен пример $x^{(i)}$;

Используя эти переменные, мы можем определить нашу функцию стоимости:

$$J(c^{(i)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K) = \frac{1}{m}\sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$$

Наша цель оптимизации состоит в том, чтобы свести к минимуму все наши параметры, используя указанную выше функцию стоимости:

$min_{c,\mu}(c, \mu)$

То есть мы находим все значения в наборах *c*, представляющие все наши кластеры, и μ, представляющие все наши центроиды, которые минимизируют среднее значение расстояний каждого примера обучающей выборки до его соответствующего кластерного центра.

На этапе назначения кластеров наша цель заключается в следующем:

Минимизируйте J(...) с $c^{(1)}, \ldots, c^{(m)}$ (удерживая $\mu_1, \ldots, \mu_K$ фиксированным)

В шаге централизованного шага наша цель:

Минимизируйте J (...) с помощью $\mu_1, \ldots, \mu_K$

В ходе работы алгоритма k-средних общая ошибка модели не может увеличиваться. При нормальном ходе обучения она всегда должна уменьшаться.

Существует рекомендуемый метод изначального задания центроидов кластеров - случайная инициализация. Причем:

1. Убедитесь, что количество ваших кластеров меньше числа ваших учебных примеров.
2. Случайно выберите примеры обучения K.
3. Установите $\mu_1, \ldots, \mu_K$, равный этим примерам K.

Алгоритм K-средних может застревать в локальных оптимумах. Чтобы уменьшить вероятность этого, вы можете запустить алгоритм для многих различных случайных инициализаций. В случаях, когда K <10, настоятельно рекомендуется запускать цикл случайных инициализаций.

Выбор количества кластеров

Выбор K может быть довольно произвольным и неоднозначным. Метод локтя: отслеживайте стоимость J и количество кластеров K. Функция стоимости должна уменьшаться по мере увеличения количества кластеров, а затем выходить на плато. Выберите K в точке, где функция ошибки начинает выравниваться (точка локтя, elbow point). Однако довольно часто кривая очень постепенная, поэтому нет четкого локтя.

J всегда будет уменьшаться по мере увеличения K. Единственное исключение - если k-средства застревают в плохом локальном оптимуме.

Другой способ выбора K - наблюдать, насколько хорошо k-означает выполнение в нисходящей цели. Другими словами, вы выбираете K, который окажется наиболее полезным для какой-то цели, которую вы пытаетесь достичь от использования этих кластеров.

Обзор методов кластеризации

Существует большое количество алгоритмов кластеризации. Среди них выделяют линейные и нелинейные алгоритмы. К линейным алгоритмам относятся:

1. линейный алгоритм k-средних;
2. нечеткий алгоритм кластеризации c-mean;

3. иерархический алгоритм кластеризации;
4. алгоритм кластеризации Гаусса (EM);
5. алгоритм кластеризации пороговых значений качества;

Некоторые алгоритмы нелинейной кластеризации:

- алгоритм кластеризации на основе MST;
- алгоритмы кластеризации ядра k-mean;
- алгоритм кластеризации на основе плотности;

Понижение размерности

Постановка задачи

Вы когда-нибудь работали над набором данных с более чем тысячей функций? Как насчет более 50 000 функций? Это очень сложная задача, особенно если вы не знаете, с чего начать. Наличие большого числа переменных является одновременно благом и проклятием. Замечательно, что у нас есть масса данных для анализа, но проводить его одновременно сложно из-за размера.

Невозможно проанализировать каждую переменную на микроскопическом уровне. Может потребоваться несколько дней или месяцев для проведения значимого анализа. Не говоря уже о количестве вычислительной мощности, которое это займет. Нам нужен лучший способ справиться с большими размерными данными, чтобы мы могли быстро извлечь из него шаблоны и идеи.

Мы можем существенно уменьшить количество имеющихся признаков, если у нас много избыточных данных. Для этого мы найдем две сильно коррелированные признаки, на их основе создадим новый признак, который точно описывает оба.

Выполнение уменьшения размерности уменьшит общие данные, которые мы должны хранить в памяти компьютера, и ускорит алгоритм обучения.

При уменьшении размерности мы уменьшаем количество признаков, а не наше количество примеров. Наша переменная m останется прежнего размера; n, число признаков, каждый из которых переносится из $x^{(1)}$ в $x^{(m)}$, будет уменьшено.

Нелегко визуализировать данные более трех измерений. Мы можем уменьшить размер наших данных до 3 или менее, чтобы можно было построить визуализацию, облегчающую понимание зависимостей в данных.

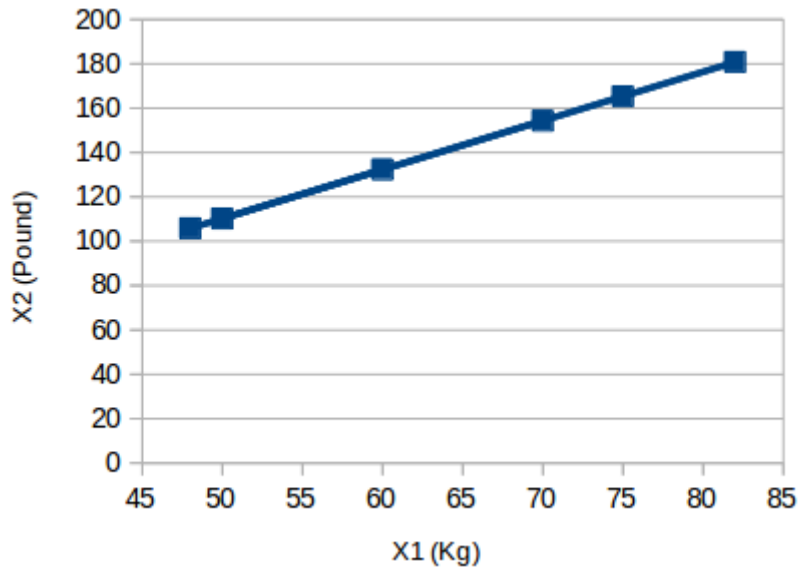
Нам нужно найти новые функции z_1, z_2 (и, возможно, z_3), которые могут эффективно суммировать все остальные функции. Пример: сотни признаков,

связанных с экономической системой страны, могут быть объединены в один, которую можно охарактеризовать «Экономическая деятельность».

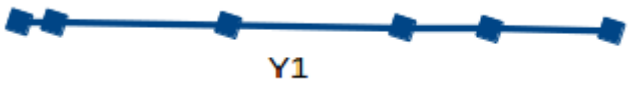
По мере того, как объем генерации и сбор аданных все больше возрастает, визуализация и выведение умозаключений становятся все более сложными. Одним из наиболее распространенных способов визуализации является использование диаграмм. Предположим, что у нас есть 2 переменные: возраст и высота.

Теперь рассмотрим случай, в котором мы имеем, скажем, 100 переменных (p = 100). В этом случае мы можем иметь 100 (100-1) / 2 = 5000 разных графиков. Не имеет смысла визуализировать каждый из них отдельно, не так ли? В таких случаях, когда мы имеем большое количество переменных, лучше выбрать подмножество этих переменных (p << 100), которое фиксирует столько информации, сколько исходный набор переменных.

Поясним это на простом примере. Рассмотрим приведенное ниже изображение:



Здесь мы имеем веса подобных объектов в Kg (X1) и Pound (X2). Если мы будем использовать обе эти переменные, они будут передавать аналогичную информацию. Таким образом, имеет смысл использовать только одну переменную. Мы можем преобразовать данные из 2D (X1 и X2) в 1D (Y1), как показано ниже:



Аналогично, мы можем уменьшить p размерности данных в подмножество k размерностей (k << n). Это называется уменьшением размерности.

Метод главных компонент

Наиболее популярным алгоритмом сокращения размерности является метод главных компонент (principal component analysis, PCA). В этом методе переменные преобразуются в новый набор переменных, которые являются линейной комбинацией исходных переменных. Эти новые переменные известны как основные или главные компоненты. Они получаются таким образом, что первый компонент учитывает большую часть возможного изменения исходных данных, после чего каждый последующий компонент имеет максимально возможную дисперсию.

Второй главный компонент должен быть ортогонален первому главному компоненту. Другими словами, он делает все возможное, чтобы зафиксировать дисперсию данных, которые не были захвачены первым основным компонентом. Для двумерного набора данных могут быть только два главных компонента.

Учитывая две функции: x_1 и x_2, мы хотим найти одну такую переменную, которая эффективно описывает обе функции одновременно. Затем мы сопоставляем наши старые функции с этой новой строкой, чтобы получить новую отдельную функцию. То же самое можно сделать с тремя функциями, где мы сопоставляем их с плоскостью.

Цель PCA - уменьшить среднее значение всех расстояний каждой функции до линии проецирования. Это называется ошибка проецирования. Уменьшите от 2d до 1d: найдите направление (вектор $u^{(1)} \in \mathbb{R}^n$, на который нужно проецировать данные, чтобы минимизировать ошибку проецирования.

Более общий случай таков:

Понижение от n-мерности до k-размерности: найдите k векторов $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$ на которые нужно проецировать данные, чтобы свести к минимуму ошибка проектирования.

Если мы перейдем от 3d к 2d, мы будем проектировать наши данные на два направления (плоскость), поэтому k будет равно 2.

PCA не является линейной регрессией. В линейной регрессии мы минимизируем квадрат ошибки от каждой точки до нашей линии предиктора. Это вертикальные расстояния. В PCA мы минимизируем кратчайшее расстояние или кратчайшие ортогональные расстояния до наших точек данных. В более общем плане, в линейной регрессии, мы берем все наши примеры в x и применяем параметры в Θ для предсказания y. В PCA мы берем ряд функций $x_1, x_2, \ldots, x_n$ и находим среди них наиболее близкий общий набор данных. Мы не пытаемся предсказать какой-либо результат, и мы не применяем к ним тета-веса.

Алгоритм анализа основных компонентов

Основные компоненты чувствительны к шкале измерения, теперь, чтобы исправить эту проблему, мы должны всегда стандартизировать переменные перед применением PCA. Прежде чем мы сможем применить PCA, необходимо выполнить предварительную обработку данных. Данный набор тренировок: $x^{(1)}$, $x^{(2)}$, ..., $x^{(m)}$, необходимо привести к одной шкале соответствующими методами (масштабирование признаков/ нормализация по среднему):

$$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{s_j}$$

1. Вычислить ковариационную матрицу

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)})(x^{(i)})^T$$

2. Вычислить собственные векторы ковариационной матрицы Σ

3. Взять первые k столбцов матрицы U и вычислить z.

Мы будем назначать первые k столбцов U переменной, называемой «Ur». Это будет n × k матрица. Мы вычисляем z так:

$$z^{(i)} = U_r^T \cdot x^{(i)}$$

Выбор количества основных компонентов

Как выбрать k, также называемое числом главных компонентов? Напомним, что k - размерность, которую мы уменьшаем до.

Один из способов выбора k состоит в следующем:

1. Учитывая среднюю квадратную ошибку проекции: $dfrac1m \sum_{i=1}^{m} ||x^{(i)} - x_{approx}^{(i)}||^2$,
2. Также дано полное изменение данных: $\frac{1}{m} \sum_{i=1}^{m} ||x^{(i)}||^2$,
3. Выберите k как наименьшее значение, такое, что: $\frac{\frac{1}{m} \sum_{i=1}^{m} ||x^{(i)} - x_{approx}^{(i)}||^2}{\frac{1}{m} \sum_{i=1}^{m} ||x^{(i)}||^2} \leq 0.01$.

Другими словами, квадратная ошибка проекции, деленная на общее изменение, должна быть меньше одного процента, так что 99% дисперсии сохраняется.

1. Алгоритм выбора k
2. Попробуйте PCA с k = 1,2, ...
3. Вычислить U_{r}, z, x

Проверьте приведенную выше формулу, что 99% дисперсии сохраняется. Если нет, перейдите к шагу 1 и увеличьте k. Это дает нам матрицу S. Мы можем проверить 99% сохраняемой дисперсии с использованием матрицы S следующим образом:

$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

Рекомендации по применению PCA

Наиболее распространенным применением PCA является ускорение обучения с учителем.

Учитывая набор тренировок с большим количеством функций (например, $x^{(1)}, \ldots, x^{(m)} \in \mathbb{R}^{10000}$), мы можем использовать PCA для уменьшения числа признаков в каждом примере обучающего набора (например, $z^{(1)}, \ldots, z^{(m)} \in \mathbb{R}^{1000}$).

Обратите внимание, что мы должны определить сокращение PCA от $x^{(i)} z^{(i)}$ только на обучающем наборе, а не на валидационных или тестовых наборах. Вы можете применить отображение $z^{(i)}$ к вашим наборам перекрестной проверки и тестирования после того, как они определены в наборе обучения.

Вот некоторые из преимуществ применения уменьшения размерности к набору данных:

- Пространство, необходимое для хранения данных, уменьшается по мере уменьшения количества измерений
- Меньшие размеры приводят к меньшему времени вычисления / обучения
- Некоторые алгоритмы не работают хорошо, когда в данных присутствует большая размерность. Поэтому необходимо уменьшить размерность, чтобы алгоритм был полезным
- Он решает проблему мультиколлинеарности, удаляя избыточные признаки. Например, у вас есть две переменные - «время, затрачиваемое на беговую дорожку в считанные минуты» и «сжигаемые калории». Эти переменные сильно коррелируют, поскольку чем больше времени вы проводите на беговой дорожке, тем больше калорий вы будете сжигать. Следовательно, нет смысла хранить оба значения, поскольку только один из них делает то, что вам нужно
- Это помогает визуализировать данные. Как обсуждалось ранее, очень сложно визуализировать данные в более высоких измерениях, поэтому

сокращение нашего пространства до 2D или 3D может позволить нам более четко строить и наблюдать рисунки

Не следует использовать PCA в качестве попытки предотвратить переобучение. Мы могли бы подумать, что сокращение количества признаков с помощью PCA было бы эффективным способом решения проблемы переобучения. Это может работать, но не рекомендуется, поскольку он не учитывает значения наших результатов y. Использование только регуляризации будет по крайней мере столь же эффективным.

Не предполагайте заранее, что вам необходимо использовать PCA. Сначала попробуйте выполнить полный алгоритм машинного обучения без PCA. Затем используйте PCA, если это необходимо.

Обзор других методов понижения размерности

Метод главных компонент далеко не единственный метод, использующийся для понижения размерности в данных. В статье (Sharma 2018) приводится обзор следующих широко распространенных алгоритмов:

1. Удаление отсутствующих значений (Missing Value Ratio): если в наборе данных слишком много отсутствующих значений, мы используем этот подход для уменьшения числа переменных. Мы можем отбросить переменные с большим количеством недостающих значений в них;
2. Фильтр с низкой дисперсией (Low Variance filter): мы применяем этот подход для определения и отбрасывания постоянных переменных из набора данных. Целевая переменная не подвержена чрезмерному воздействию переменных с низкой дисперсией, и, следовательно, эти переменные можно безопасно удалить;
3. Фильтр с высокой корреляцией (High Correlation filter): пара переменных с высокой корреляцией увеличивает мультиколлинеарность в наборе данных. Таким образом, мы можем использовать эту технику, чтобы найти высококоррелированные признаки и соответственно отбросить их;
4. Случайный лес: это один из наиболее часто используемых методов, который говорит о важности каждого признака, присутствующего в наборе данных. Мы можем найти важность каждого признака и сохранить самые лучшие признаки, что приведет к уменьшению размерности;
5. Методы последовательного исключения признаков и последовательного включения признаков требуют большого вычислительного времени и поэтому обычно используются для небольших наборов данных;
6. Факторный анализ: этот метод лучше всего подходит для ситуаций, когда мы имеем сильно коррелированный набор переменных. Он делит переменные на основе их корреляции в разные группы и представляет каждую группу с коэффициентом;

7. Анализ главных компонент: это один из наиболее широко используемых методов обработки линейных данных. Он делит данные на набор компонентов, которые стараются объяснить как можно большую дисперсию;
8. Анализ независимых компонент (ICA): мы можем использовать ICA для преобразования данных в независимые компоненты, которые описывают данные с использованием меньшего количества компонентов;
9. ISOMAP: Мы используем этот метод, когда данные сильно нелинейны;
10. t-SNE: Этот метод также хорошо работает, когда данные сильно нелинейны. Он отлично работает для визуализации;
11. UMAP: Этот метод хорошо работает для данных с высокой размерностью. Его время работы короче по сравнению с t-SNE.

Учебное издание

Коротеев Михаил Викторович
кандидат экономических наук, доцент Департамента анализа данных, принятия решении й и финансовых технологий Финансового университета при Правительстве Российской Федерации

ТЕХНОЛОГИИ АНАЛИЗА ДАННЫХ И МАШИННОЕ ОБУЧЕНИЕ
УЧЕБНОЕ ПОСОБИЕ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ К ЗАЧЕТУ, 1 семестр

Для студентов, обучающихся по направлению 38.03.01 «Экономика» и 38.03.02 «Менеджмент»
(программа подготовки бакалавра)

Компьютерный набор, верстка:
М.В. Коротеев

Вычитка и корректура выполнены авторами

Формат 60x90/16. Гарнитура Times New Roman.
3п.л. 2018 г. Электронное издание

© Финансовый университет при Правительстве Российской Федерации», 2018.
© Департамент анализа данных, принятия решении й и финансовых технологий, 2018.
© М.В. Коротеев 2018.