Линейные модели

Конспект Яндекс учебника https://education.yandex.ru/handbook/ml

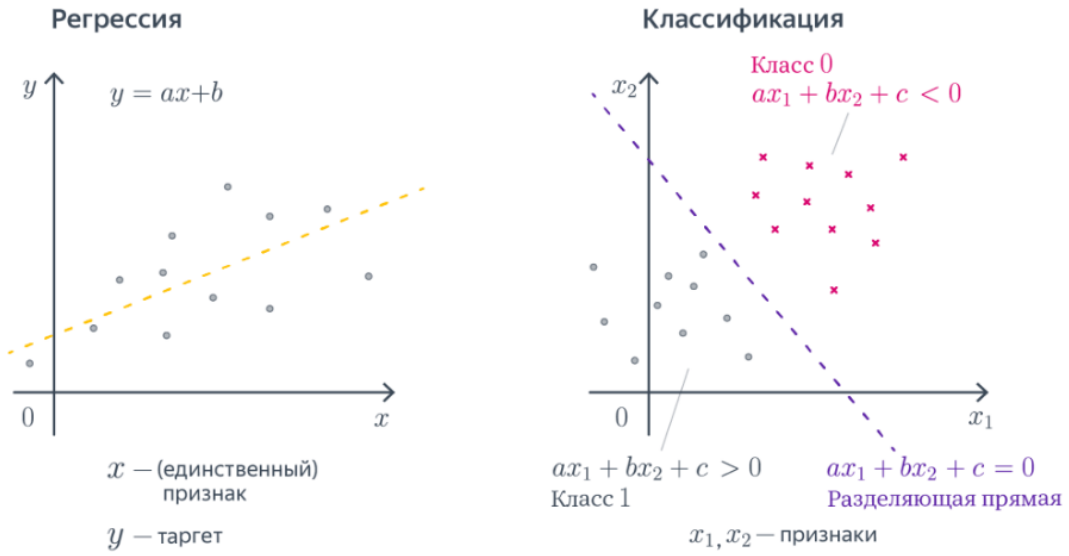
Введение

В общем виде, пусть у вас есть матрица X, каждой строчке которой вам нужно задать категориальный признак(классификация) или числовой(регрессия). (Ex: определить, мошенническая ли транзакция; вычислить окупаемость разработки шахты). Понятно, что в реальности существует практически точный ответ на нашу задачу, а нашей целью является выдать ответ, максимально похожий на "правду". Предсказывать ответ мы будем с помощью семейства уравнений

$$y = w_1 x_1 + \ldots + w_D x_D + w_0,$$

линейная модель

где y - наш предполагаемый объект, target; вектор X - числовые признаки, на основе которых мы делаем прогноз, features; D - размерность X; w*0* - свободный член, bias.



классификация и регрессия

То есть, нашей задачей является подбор чисел для вектора W, который при подстановке в наше уравнения выдаст лучший ответ.

Может быть такое, что целевая переменная зависит не только от коэффициентов W, но и от значений некоторых функций коэффициентов W (ex: sin, log, e.t.c). В таком случае мы можем провести замену переменной и свести задачу к линейной модели, увеличив её размерность.

Стоит помнить, что нам никто не гарантирует, что все наши признаки будут числовые. В случае категориального признака нужно провести hot encoding (разбиение категориального признака на k признаков, каждый i-ый из которых будет либо 0 (исходный признак не принадлежит i-ому классу), либо 1 (искодный признак принадлежит i-ому классу). Но в таком случае сумма столбцов, на которые мы разбили наш признак, всегда будет равна 1 (см. карт.),

	weight	pet_type_cat	pet_type_dog	pet_type_hamster	color_brown	color_gray	color_red	color_white
0	0.600000	0	0	0	0	0	1	0
1	3.000000	1	0	0	0	0	0	1
2	0.800000	0	0	1	1	0	0	0
3	5.000000	1	0	0	0	1	0	0
4	7.000000	0	1	0	0	0	0	0

one hot encoding

а значит, если мы будем обладать информацией о k - 1 столбиков, на которые мы разбили категориальный признак, то мы точно можем сказать значение оставшегося. То есть после one hot encoding'а мы можем, а на самом деле должны, убрать один из новоиспечённых признаков, чтобы уменьшить размерность (так как лишние фичи ведут к переобучению - не предсказанию, а подгонке ответа).

Плюсы:

Сразу можно сказать, что линейные модели хорошо интерпретированы, так как W является вектором весов, поэтому легко сказать почему наша модель нашла такой ответ.

Минусы:

Подходит для узкого класса задач (очевидно, что очень небольшое количество вещей в жизни можно выразить линейно).

Желание подогнать линейную модель под решаемую задачу приводит к излишнему поиску, от чего может зависить таргет, то есть к введению новых признаков (feature engineering), что вредит интерпретируемости.

В случае приближённой линейной зависимости(когда какую-то строчку практически можно выразить с помощью линейной комбинации других) X, веса W получаются нереалистичными.

Нужно осторожно выкидывать фичи, веса у которых малы, так как нельзя точно сказать, правильное ли это решение.

Веса могут значительно меняться в зависимости от размера обучаемой выборки (чем она больше, тем точнее веса).

Функции потерь (loss functions)

Как понять, хорошо ли мы предсказали тагрет? Давайте введём какую-то функцию потерь, которая будет показывать, насколько сильно наш ответ отличается от эталонного. В таком случае нашей задачей будет являться минимизация этой функции.

Для удобства записи провернём математический фокус:

$$(x_{i1} \quad \dots \quad x_{iD}) \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_D \end{pmatrix} + w_0 = (1 \quad x_{i1} \quad \dots \quad x_{iD}) \cdot \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_D \end{pmatrix}$$

Поскольку это сильно упрощает запись, в дальнейшем мы будем считать, что это уже сделано и зависимость имеет вид просто $f_w(x_i) = \langle w, x_i \rangle$.

избавление от w0

Так как эталонным ответом является вектор каких-то значений и наше предсказание является числовым вектором, то за loss function можно принять расстояние между ними(как его считать вариантов много).

1) Квадрат L2-нормы вектора разницы(евклидово расстояние) предсказаний модели и y:

$$\mathcal{L}(y - \hat{y}) = \sum_{i=1}^{n} (y - \hat{y}_i)^2$$

L2 loss function

2) Среднеквадратичное отклонение, Mean Squared Error, MSE

$$L(f, X, y) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \langle x_i, w \rangle)^2$$

MSE

Лучше L2, потому что не зависит от размера выборки

ЛИНЕЙНАЯ РЕГРЕССИЯ

Как находить вектор W через метод наименьших квадратов (МНК)?
Точный метод

Нам нужно найти минимум

$$\|\mathbf{Y} - X\mathbf{W}\|^2$$

МНК

Возьмём частные производные по W и приравняем к нулю

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = -2X^T(\mathbf{Y} - X\mathbf{W}) = 0$$

применяем матричное дифференцирование

$$X^T\mathbf{Y} = X^TX\mathbf{W}$$

получаем это

$$\mathbf{W} = (X^TX)^{-1}X^T\mathbf{Y}$$

а затем это

Посмотря на эту формулу, можно заметить, что мы вычисляем обратную матрицу от квадратной, значит эта матрица должна быть невырожденной (определитель не ноль). Обычно матрица получается невырожденной, так как определитель ноль, если одну строку можно выразить через другую, а обычно такого нет, а если вдруг есть, то мы избавляемся от этого. Зато ситуация, когда определитель близок к нулю встречается часто, что приводит к вычилительным проблемам.

$$O(D^2N + D^3)$$

D - фичи, N - строки

Можно ускорить, если применять продвинутые методы перемножения матриц. Метод МНК имеет недостатки в виде поиска обратной матрицы - вычислительная сложность, а также небольшое колебание значений приводит к большому разбросу W.

Нахождение W через МНК, приближённый метод

Функция, которую мы минимизируем, является гладкой и выпуклой, значит у неё существует градиент - вектор функции в точке, который направлен в сторону наибольшего роста функции, антиградиент - в сторону наибольшей убыли. Таким образом, если мы подсчитали на основе некоторого предварительного вектора W нашу loss function, то подсчитав антиградиент этой функции, мы можем сдвинуть вектор W в сторону антиградиента, тем самым минимизируя функцию. Ну и нам нужно задать шаг сдвига - темп обучения. Такой алгоритм называется градиентным спуском. Теоритически выбор стартовой точки влияет на скорость алгоритма, но на практике проверено, что этим можно пренебречь. Чем больше число обусловленности матрицы X (высота элипсоподобных выпуклостей на графике), тем хуже алгоритму. Шаг алгоритма является гиперпараметром и сильно влияет на его работу (не перескочим ли экстремум), скорее всего его придётся подбирать отдельно, также как и максимальное число итераций или порог (tolerance - значение допустимой ошибки).

$$O(NDS)$$

S - итерации

В общем, градиентный спуск выглядит посимпатичнее (для меня).

Стохастический градиентный спуск (SGD)

На предыдущем этапе мы пересчитывали градиент на каждой итерации, попробуем это оптимизировать. Введём термин batch (или mini-batch) - какой-то кусок выборки, подвыборка ((x*i*, y*i*)). Тогда мы можем считать градиент не по всей выборке, а по какой-то подвыборке.
Как делить выборку на батчи? Все алгоритмы долгие, поэтому обычно шафлят исходную выборку, а затем, определив размер батча, делят выборку линейно. Тогда обычно вместо вместо количество итераций (шагов алгоритма) задаётся другой гиперпараметр - число эпох (сколько раз мы

пробежим через всю выборку). При этом градиент пересчитываем после каждого батча.

$$O(NDE)$$

E - эпохи

В принципе, сложность примерно такая же, но в стохастическом мы в N / B раз больше обновили градиент, то есть он становится более шумным, но считать быстрее. Точность в принципе сильно не отличается, зависит от графиков, существуют дальнейшие модификации.

Градиентные спуски могут работать с любыми дифференцируемыми лосс функциями, с MAE, например.

Регуляризация

Бывает так, что веса получаются неинтерпретируемые (отрицательные, которые в реальном мире не могут быть отрицательными), это скорее всего означает что мы столкнулись с проблемами, которые мы описали выше. Помочь с этим нам может регуляризация - ограничение возможного диапазона значений весов. Обычно регуляризуют с помощью L1 или L2 - норм.

Вместо исходной задачи теперь предлагается решить такую:

$$\min_w L(f, X, y) = \min_w (|Xw - y|_2^2 + \lambda |w|_k^k)$$

λ – это очередной параметр, а $|w|_k^k$ – это один из двух вариантов:

$$|w|_2^2 = w_1^2 + \ldots + w_D^2$$

или

$$|w|_1^1 = |w_1| + \ldots + |w_D|$$

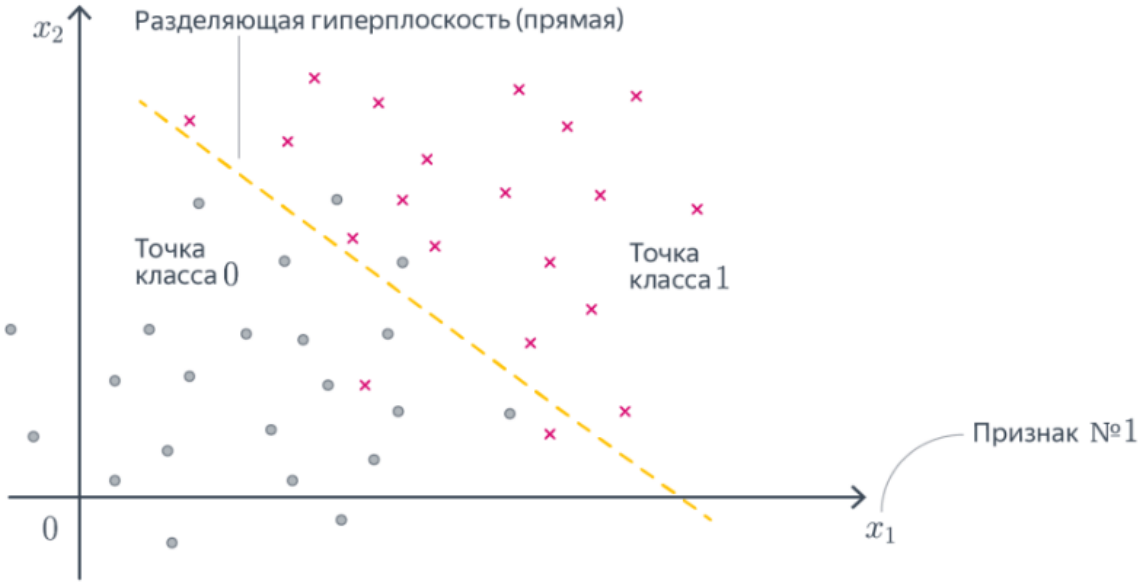
Добавка $\lambda |w|_k^k$ называется **регуляризационным членом** или **регуляризатором**, а число λ – **коэффициентом регуляризации**.

лямбда - отдельно подбираемый гиперпараметр

L1 регуляризация приводит к занулению маловажных фич, удаление которых помогает бороться с коллинеарностью.

ЛИНЕЙНАЯ КЛАССИФИКАЦИЯ

Поговорим сначала о простом частом случае - бинарной классификации.



бинарная классификация

Очевидно, что в большинстве случаев выборка не является линейно разделимой

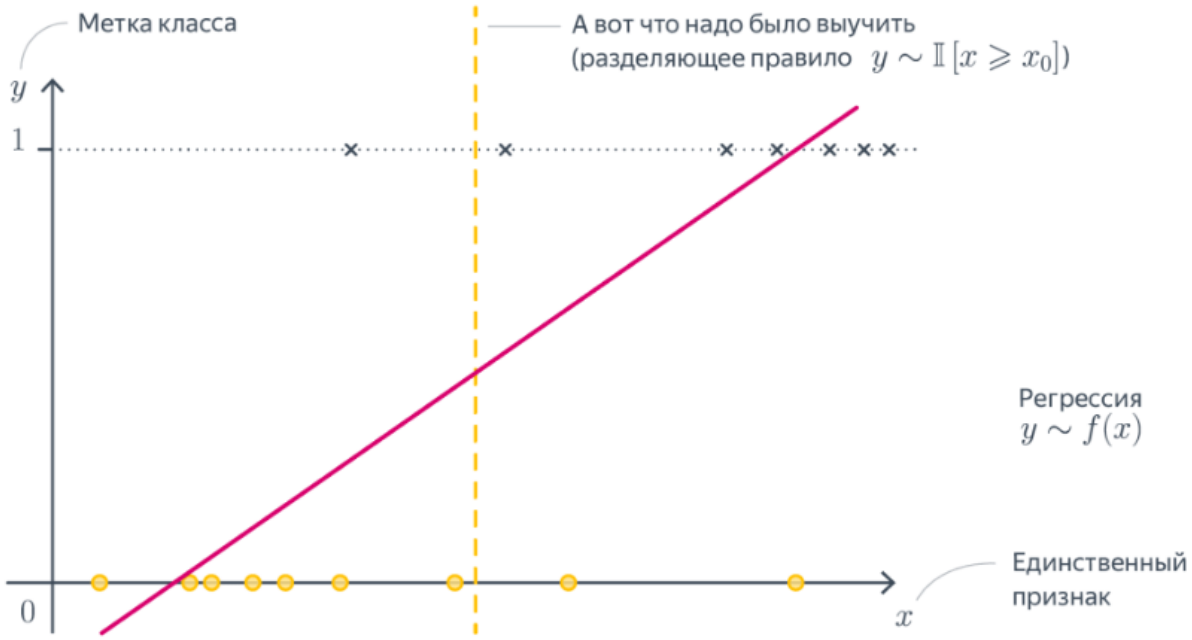
$$y = \text{sign}\langle w, x_i \rangle$$

бинарная классификация

То есть, какой знак у нашего выражения, такой и класс

Почему не решать задачу классификации, как задачу регресии (предсказывать 1 или -1, минимизируя какую-то функцию ошибки, и беря знак предсказания)? Потому что для регрессии нет асболютно никакой разницы,

по какую сторону от нашей прямой будет располагаться точка, плюс в регрессии мы обращаем внимание на модуль числа, а не его знак.



пример применения регрессии для бинарной классификации

Хорошо, если регрессию использовать нельзя, тогда нам нужно придумать свою функцию потерь для классификации - логично взять количество точек, которые мы классифицировали неверно.

$$\sum_i \mathbb{I}[y_i \neq sign\langle w, x_i \rangle] \longrightarrow \min_w$$

что хотим минимизировать в классификации

Если домножим нашу функцию на y*i*, то если знак < 0, то мы не угадали

$$\sum_i \mathbb{I}[y_i \langle w, x_i \rangle < 0] \longrightarrow \min_w$$

рабочая функция потерь классификации

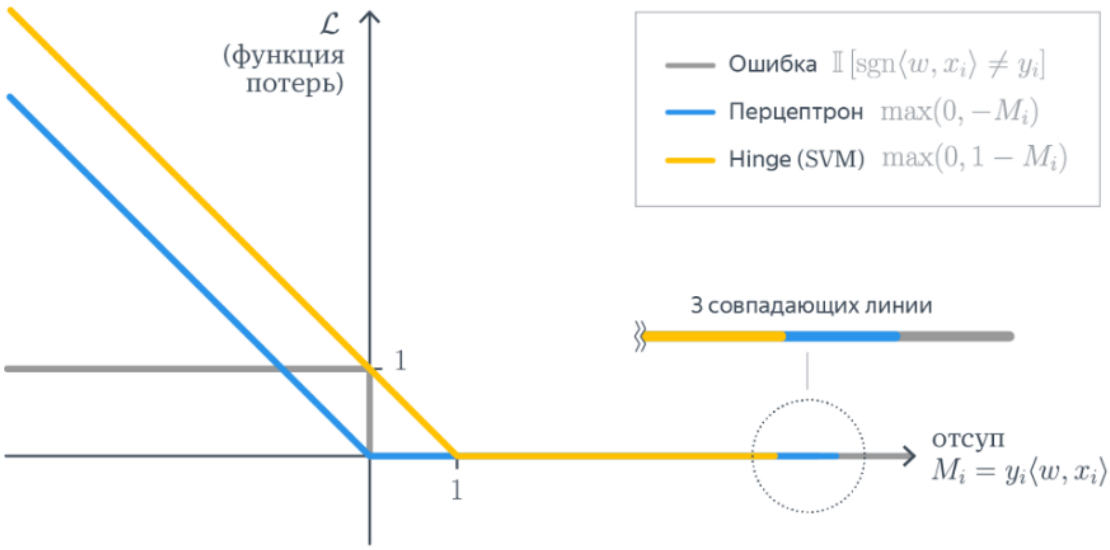
Чем дальше значение нашей функции от нуля, тем больше классификатор уверен в своём предсказании. Значение нашей функции M называют отступом (margin), а саму функцию - misclassification loss.

Что теперь делать в этой функцией, как минимизировать? Введём новую кусочно-постоянную функцию F(M), которая равна 1, если M < 0, 0, если M > 0

$$F(M) = \mathbb{I}[M < 0] = \begin{cases} 1, \ M < 0, \\ 0, \ M \geqslant 0 \end{cases}$$

функция, которую будем минимизировать

Но мы не можем минимизировать её градиентным спуском, так как она не является гладкой. Тогда будем рассматривать не F(M), а более гладкую, но аналогичную нашей, то есть мажорируем нашу функцию (Перцептрон, Hinge(SVM).



ошибка, перцептрон, Hinge(SVM)

P. S. если наш классификатор выдаёт большинство отрицательных значений, то возможно наш классификатор выдаёт классы противоположные искомых

P. S. не всегда случай, когда классификатор классифицирует очень уверенно (с большим margin), это хорошо, т.к. во многих случаях лучше, чтобы классификатор иногда честно говорил о своей неуверенности

В чём идея перцептрона? Будем считать margin только с случае неверного предсказания.

$$F(M) = \max(0, -M)$$

перцептрон

$$L(w, x, y) = \lambda||w||_2^2 + \sum_i \max(0, -y_i\langle w, x_i\rangle)$$

применили L2 регуляризацию к перцептрону

$$\nabla_w L(w, x, y) = 2\lambda w + \sum_i \begin{cases} 0, & y_i \langle w, x_i \rangle > 0 \\ -y_i x_i, & y_i \langle w, x_i \rangle \leq 0 \end{cases}$$

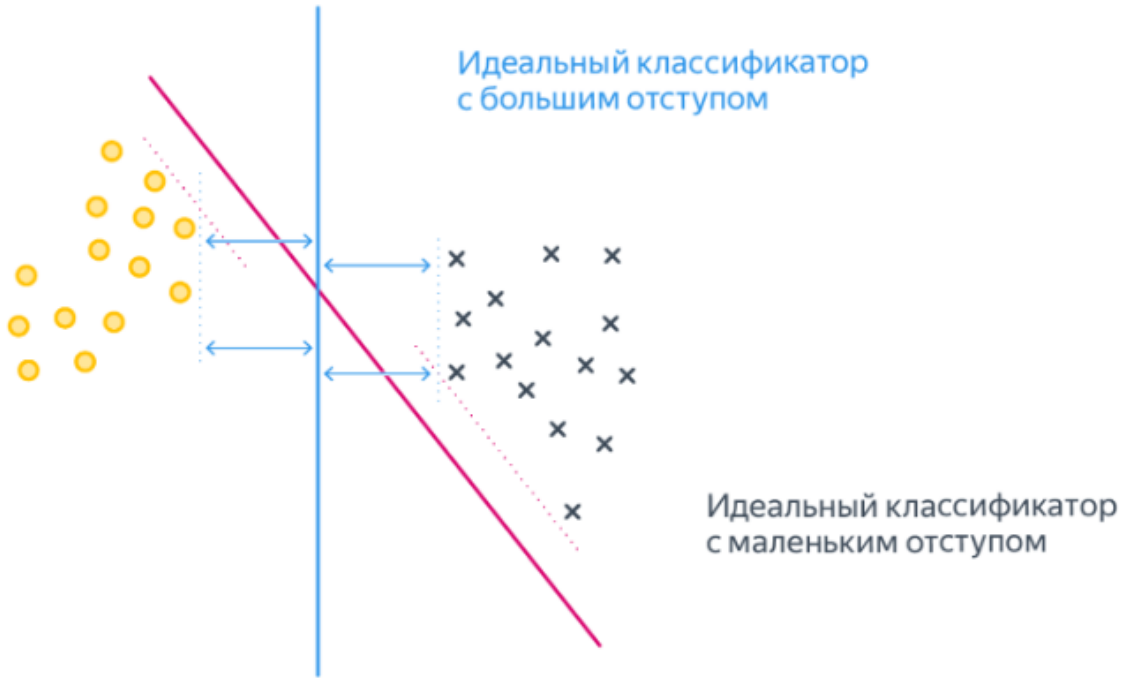
вычислим градиент для предыдущей функции

Ну а теперь смело применяем стахостический градиентный спуск!

Такое решение выдаёт неединственный ответ, он будет зависеть от изначальный параметров. Но это нормально, так как мы можем разными способами провести прямую между группами объектов.

примеры графического решения бинарной классификаци

Что по Hinge(SVM)? Допустим, мы хотим не просто корректно провести прямую между группами объектов, но и расположить её на равном растстоянии от этих групп, то есть максимизировать минимальный отступ.



чего бы нам хотелось

Это достигается путём прибавления единицы в формулу перцептрона:

$$F(M) = \max(0, 1 - M)$$

$$L(w, x, y) = \lambda ||w||_2^2 + \sum_i \max(0, 1 - y_i \langle w, x_i \rangle)$$

$$\nabla_w L(w, x, y) = 2\lambda w + \sum_i \begin{cases} 0, & 1 - y_i \langle w, x_i \rangle \leq 0 \\ -y_i x_i, & 1 - y_i \langle w, x_i \rangle > 0 \end{cases}$$

Hinge

Интуитивно: объекты, которые классифиуированны правильно, но не очень уверенно влияют на градиент, отодвигая разделяющую прямую от себя.

Строго: математика, в которой вводится ещё одна функция, минимизируя максимальную ширину половину ширины между группами классов.

Итоговое положение прямой зависит от ближайших к ней правильно классифицированных объектов, их называют опорными векторами (support vectors), а метод, который таится под строгой математикой из прошлого абзаца support vector machine (SVM). Раньше SVM был супер популярен, так как мало затрачивал вычислительные мощности и стабильно выдавал хороший результат.

Логистическая регрессия

Несмотря на название, это классификация. Та же бинарная классификация, только обозначим их теперь 0 и 1. Тогда можно подойти к задаче с точки зрения предсказания вероятностей принадлежности к тому или иному классу. Но т.к. вероятность всегда >= 0 и <=1, а наши функции потерь на выходе выдают практически неограниченные числа, то мы будем использовать логиты (logit)

$$\log\left(\frac{p}{1-p}\right)$$

логит

Логит - отношение логарифма вероятности положительного события к отрицательному

Если приравняем логит к нашей линейной функции, то можем вывести формулу вероятности принадлежности к классу:

$$\langle w, x_i \rangle = \log\left(\frac{p}{1-p}\right)$$

$$e^{\langle w, x_i \rangle} = \frac{p}{1-p}$$

$$p = \frac{1}{1 + e^{-\langle w, x_i \rangle}}$$

формула p

Функция в правой части является сигмоидой, поэтому

$$p = \sigma(\langle w, x_i \rangle)$$

p через сигмоиду

Хорошо, теперь вопрос в том, как оптимизировать веса, чтобы лучше предсказывать вероятность? Вспоминаем распределение Бернулли и применяем метод максимума правдоподобия(у нас событием будет принадлежность к классу). Далее идёт математика с теорвером...

$$\nabla_w L(y, X, w) = -\sum_i x_i \big(y_i - \sigma(\langle w, x_i \rangle)\big)$$

градиент логрега

После того, как мы определили вероятности и сказали, что например при p > 1/2 объекты будут принадлежать такомоту классу, то разделяющая

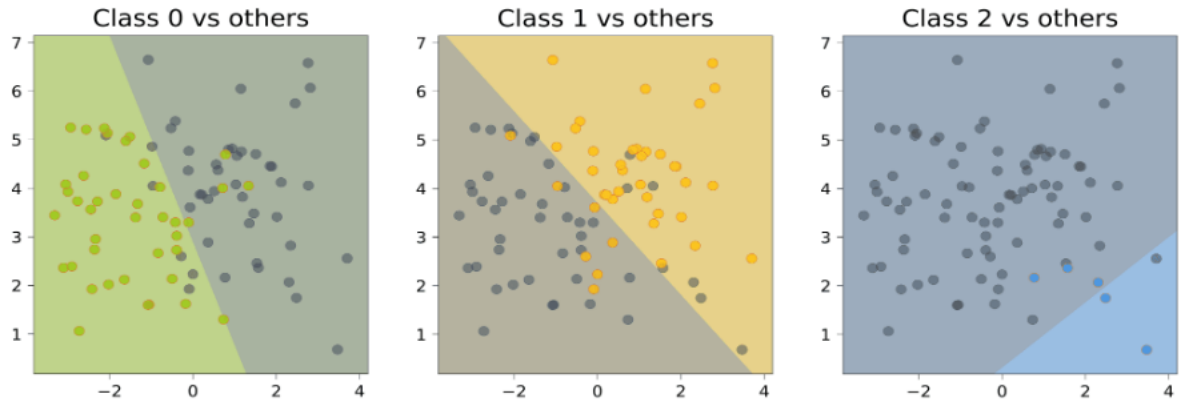
повернхность задаётся p = 1/2, а это равносильно приравнению к нулю линейной модели, то есть логрег также задаёт гиперплоскость.

Логрег выдаёт единственное решение, так как под капотом функция потерь имеет единственный глобальный минимум.

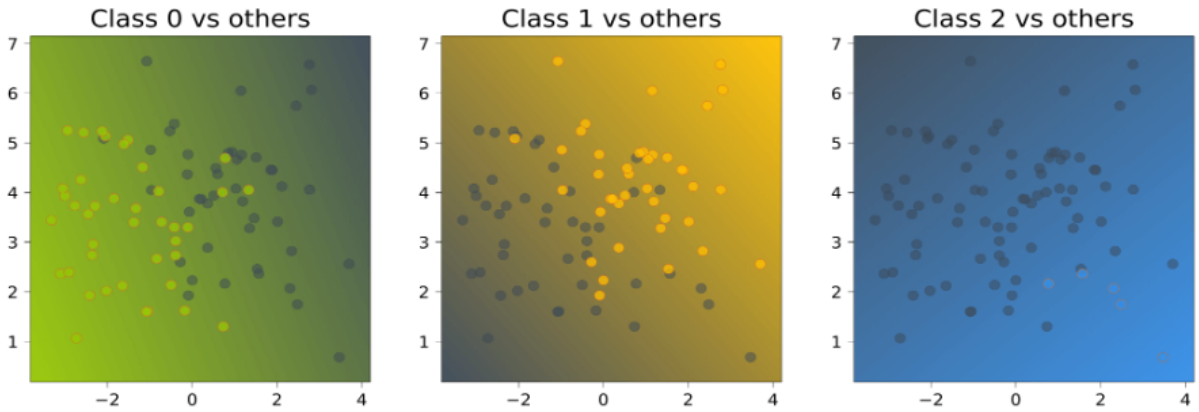
Многоклассовая классификация

Сведём эту задачу к бинарной. Два самых популярных способа: one-vs-many, all-vs-all.

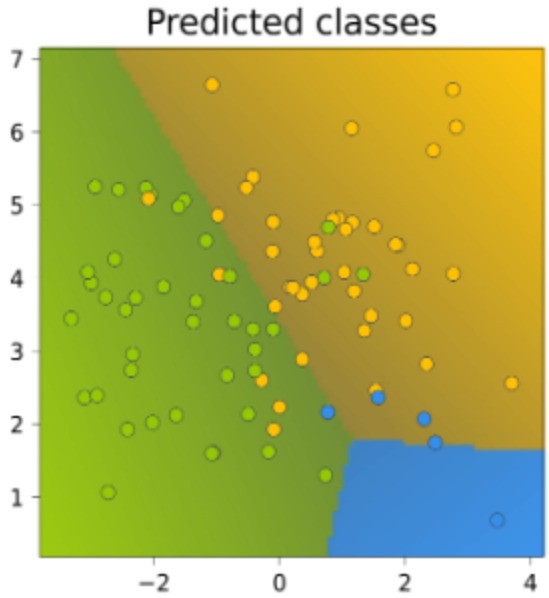
В случае одного против всех, обучим k линейных классификаторов, каждый из которых будет предсказывать соответствие своему классу(принадлежит этому классу другому). После чего скажем, что объект принадлежит тому классу, чей классификатор был самым уверенным среди остальных.



one-vs-many, pic.1



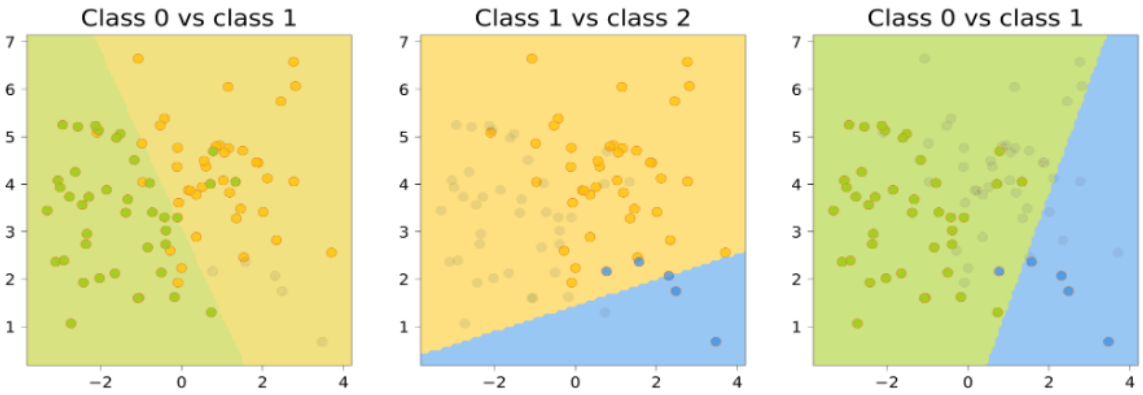
one-vs-many, pic.2



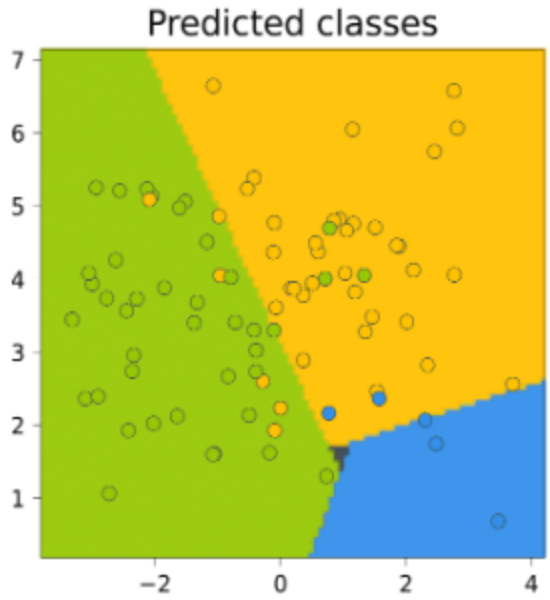
one-vs-many, pic.3

Можно заподозрить, что третий класс "обделили". Такое может происходить из-за того, что наши классификаторы имеют слишком разные веса на выходе, и далеко не всегда корректно будет их нормировать, чтобы избавиться от этого дизбаланска (пример когда нельзя нормировать веса - SVM).

В случае всех против всех, обучим число сочетаний из k по 2 бинарных классификаторов, каждый из которых будет предсказывать для каждого объекта принадлежность к классу i или j (i != j). В конечном итоге объекту присвоим тот класс, за который проголосовало наибольшее число классификаторов.



all-vs-all. pic.1



all-vs-all, pic.2

Серый треугольник - объекты, по которым нельзя дать обоснованное предсказание.

Многоклассовая логистическая регрессия

Также используем k бинарных линейных классификаторов, каждый из которых говорит о вероятности принадлежности к своему классу, затем применим оператор softmax, который отнормирует этот вектор вероятностей, тогда веротяность принадлежности объекта к k-ому классу будет:

$$P(y = k|x, w) = \frac{\exp\left(\langle w_k, x\rangle + w_{0k}\right)}{\sum_{j=1}^{K} \exp\left(\langle w_j, x\rangle + w_{0j}\right)}.$$

вероятность принадлежности к k-ому классу

Из этой формулы понятно, что нам нужно оптимизировать веса, будем делать это с помощью метода максимального правдоподобия:

$$\sum_{i=1}^{N} \log P(y = y_i|x_i, w) \to \max_{w_1,\ldots,w_K}$$

оптимизация весов

Масштабируемость линейных моделей

Во всез линейных моделях мы используем SGD, так что мы не загружаем все данные в оперативку, тем не менее фич может быть очень много:

1) Классификация текста: допустим мы хотим определить тональность текста, тогда мы должны завести несколько сотен тысяч фич (под каждое слово, чтобы проверять его наличие).

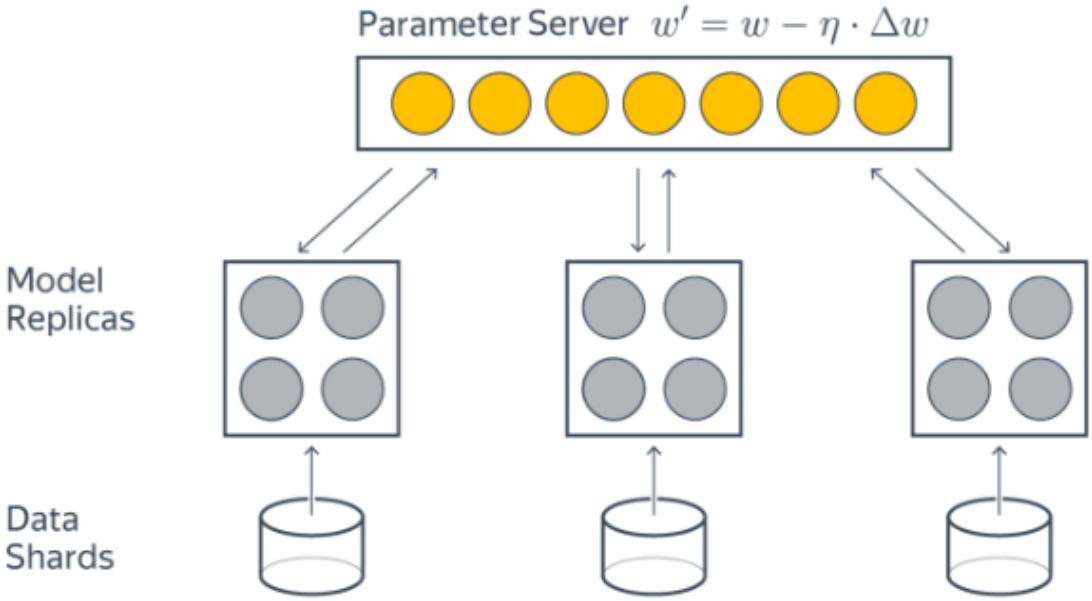
2) Предсказание кликабельности рекламы: нам нужно хранить количество пользователей * количество сайтов

Чтобы облегчить себе жизнь можно:

1) Несмотря на то, что выборки огромны, количество ненулевых объектов там небольшое, значит имеет смысл хранить вместо вектора словарь, в котором будут перечислены индексы и значения ненулевых элементов вектора.

2) Можно сжать вектор весов, используя hashing trick. Будем хранить веса в хеш-таблице и вычислять индекс по формуле hash(feature) % tablesize, таким оьразом мы совсем чуть-чуть теряем в качестве, но можем сократить веса на несколько порядков.

В случае, если мы совсем не можем жертвовать точностью, мы можем хранить фичи в шардированной хеш-таблице.



шардирования хеш-таблица

Кружок - отдельный сервер. Жёлтые - загружают данные, серые - хранят части модели. Для обучения жёлтые запрашивают данные у серых, вычисляют градиент и отправляет его обратно. Бесконечная масштабируемость у схемы.

Итог

Фактически, линейная модель - однослойная нейронная сеть. Сейчас мало где применяются, но как элемент более сложных моделей применяются много где, так что необходимо знать эти базовые вещи.