Deep Learning School

Линейная регрессия

Почему линейные модели до сих пор используются?

- Очень простые, поэтому можно использовать там, где нужна интерпретируемость модели и надежность.
- Не переобучаются
- Легко применять

Постановка задачи

Датасет:

$$\mathbf{X} \in \mathbb{R}^{l \times n}, \mathbf{y} \in \mathbb{R}$$

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ & \ddots & & \\ & & \ddots & \\ x_{l1} & x_{l2} & \dots & x_{ln} \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_l \end{pmatrix}$$

Функция потерь:

$$L(\hat{f}) = \sum_{i=1}^{l} \left(\hat{f}\left(x_{i1}, \dots, x_{in}\right) - y_i \right)^2$$

Определение модели

Мы будем искать модель в следующем виде

$$\hat{f}(x_1, \ldots, x_n) = \mathbf{w}_0 + \mathbf{w}_1 x_1 + \ldots + \mathbf{w}_n x_n$$

Намного удобнее для записи внести 1 в вектор признаков $\mathbf{x} = (1, x_1, \ldots, x_n)$

$$\hat{f}(\mathbf{x}) = \sum_{i=0}^{n} \mathbf{w}_i \mathbf{x}_i = \mathbf{x} \cdot \mathbf{w}$$

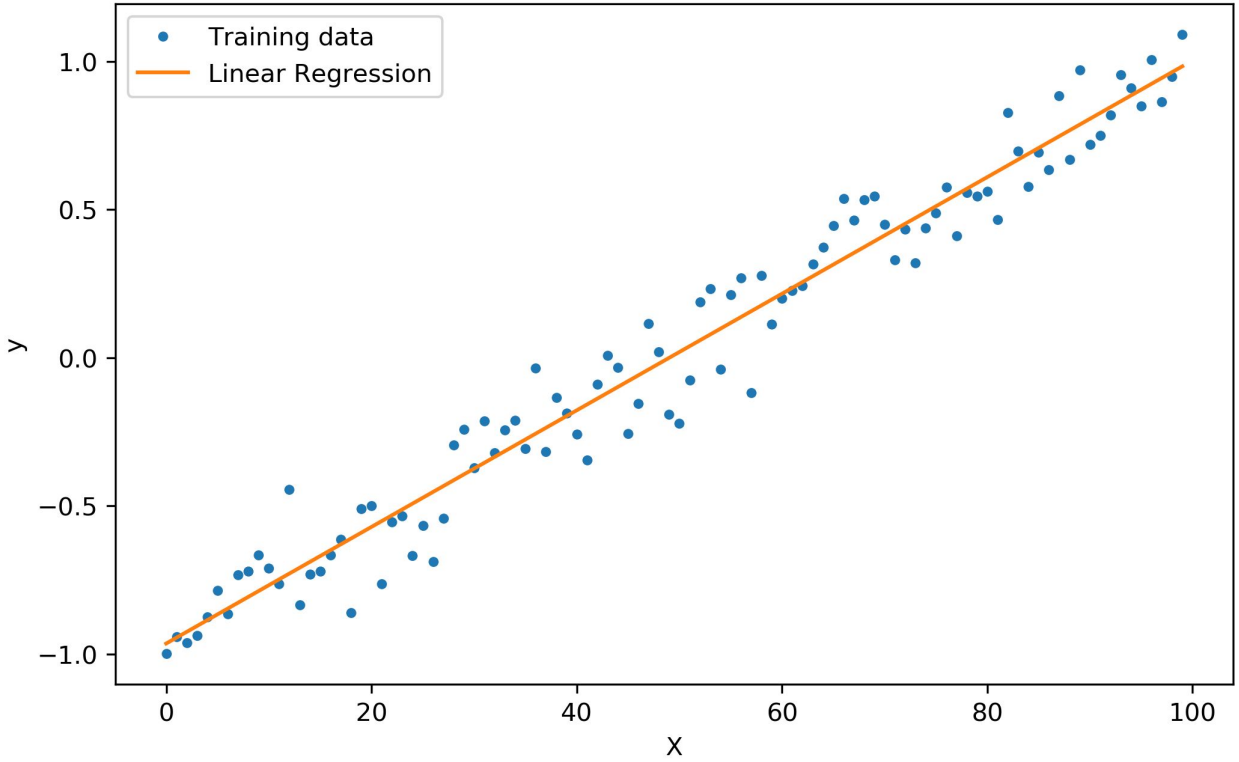
Линейность по параметрам

Какое может быть происхождение у признаков x_i?

- Просто численный признак
- Преобразования численных признаков (корень, логарифм, итд)
- Степени численного признака $x_2 = x_1^2, x_3 = x_1^3$
- Значения из One-Hot-Encoding
- Взаимодействия между разными признаками ($x_3 = x_1 x_2$)

Линейная модель линейна по параметрам, а не признакам.

Пример


Training data
Linear Regression
1.0
0.5
0.0
−0.5
−1.0
y
0
20
40
60
80
100
X

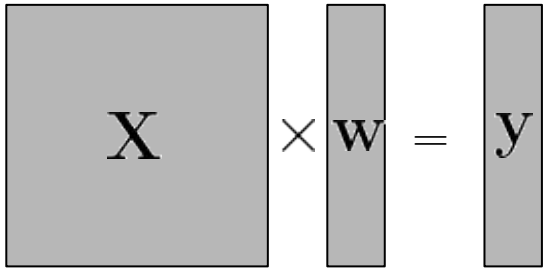
Точное решение

Запишем то, что мы хотим получить: $\mathbf{X}\mathbf{w} = \mathbf{y}$

$$\begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ & \ddots & & \\ & & \ddots & \\ & & & \ddots \\ x_{l1} & x_{l2} & \dots & x_{ln} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ \dots \\ \dots \\ y_l \end{pmatrix}$$

Точное решение

Случай $l = n$, тогда матрица $\mathbf{X}$ - квадратная и может иметь обратную.



Система линейных уравнений:

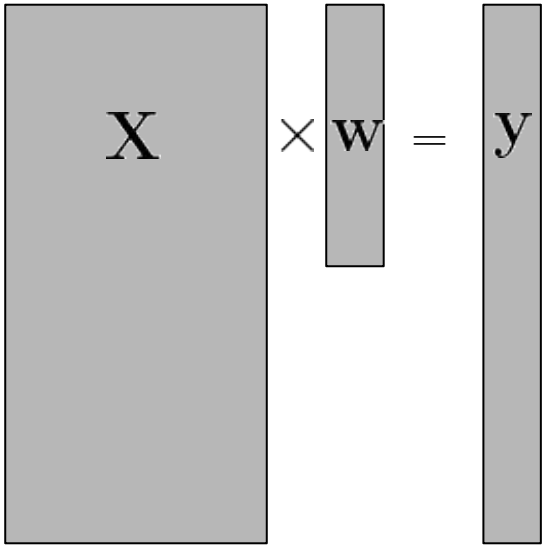
$$\mathbf{X}\mathbf{w} = \mathbf{y}$$

Решение:

$$\mathbf{w} = \mathbf{X}^{-1}\mathbf{y}$$

Pseudo-Inverse

Обычно $l \gg n$, тогда у нас *переопределенная система линейных уравнений*.



Система линейных уравнений:

$$\mathbf{X}\mathbf{w} = \mathbf{y}$$

Приближенное решение:

$$\mathbf{w} = \mathbf{X}^{+}\mathbf{y} = (\mathbf{X}^{\intercal}\mathbf{X})^{-1}\mathbf{X}^{\intercal}\mathbf{y}$$

Псевдообратная матрица дает решение с наименьшей квадратичной ошибкой.

Получение решения через производную

Подставим выражение для $\hat{f}(\mathbf{x})$ в функцию потерь и запишем в векторном виде:

$$L(\mathbf{w}) = \sum_{i=1}^{l} \left(\mathbf{x}_i^T \mathbf{w} - y_i\right)^2 = (\mathbf{X}\mathbf{w} - \mathbf{y})^\intercal (\mathbf{X}\mathbf{w} - \mathbf{y})$$

Возьмем производную

$$\frac{\partial \mathrm{L}}{\partial \mathbf{w}} = 2\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y})$$

Deep Learning School

Обучение классификаторов

Получение решения через производную

Возьмем производную

$$\frac{\partial \mathrm{L}}{\partial \mathbf{w}} = 2\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y})$$

Если у $\mathbf{X}$ линейно независимые столбцы, то можно приравнять производную к нулю.

$$\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

$$\mathbf{w} = (\mathbf{X}^\intercal\mathbf{X})^{-1}\mathbf{X}^\intercal\mathbf{y}$$

Постановка задачи

Датасет:

$\mathbf{X} \in \mathbb{R}^{l \times n}$, $\mathbf{y} \in \{0, 1\}^l$. То есть $\mathbf{y}$ это вектор из 0 и 1

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ & \ddots & & \\ & & \ddots & \\ x_{l1} & x_{l2} & \dots & x_{ln} \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_l \end{pmatrix}$$

Функция потерь:

Будет позже

Мы хотим выбрать функцию потерь, но какая лучше всего подойдет не знаем.

Попробуем искать лучшую модель с помощью теоремы из статистики.

Вероятностная модель

X- случайная величина вектор признаков.
Y- случайная величина целевая переменная.

Пример случайной модели (клики на рекламу):
X = (количество кликов раньше, время активности, уровень доходов)
Y = 1 если клик будет, 0 если клика не будет.
Тогда можно задать распределение вероятностей:

$$P_{X|Y}(Y = 1|X = (10\text{clicks}, 2\text{hours}, 10\text{dollars})) =$$

вероятность того, что человек с заданными характеристиками кликнет на рекламу.

Функция правдоподобия

Найдем способ для обучения любой модели, предсказывающей вероятность принадлежности к классу.

$\mathbf{x}$ - вектор признаков, $\hat{f}(\mathbf{x})$ - наша модель.

$$P(Y = 1|\mathbf{x}) = \hat{f}(\mathbf{x})$$

Назовем *правдоподобием* $\prod_{i=1}^{l} \mathrm{P}(Y = y_i|\mathbf{x}_i)$

Это вероятность получения нашей выборки согласно предсказаниям модели.

Обучение модели через максимальное правдоподобие

Теорема из статистики гарантирует, что если мы найдем параметры модели, которые максимизируют правдоподобие, то они будут хорошие.

$$\prod_{i=1}^{l} \mathrm{P}(Y = y_i | \mathbf{x}_i) \to \max_{\mathbf{w}}$$

$$ln(\prod_{i=1}^{l} \mathrm{P}(Y = y_i | \mathbf{x}_i)) \to \max_{\mathbf{w}}$$

Связь с минимизацией функции потерь

Преобразуем задачу максимизации в задачу минимизации.

$$L(\mathbf{w}) = -\sum_{i=1}^{l} ln(\mathrm{P}(Y = y_i|\mathbf{x}_i)) \to \min_{\mathbf{w}}$$

Мы видим что минимизация полученного выражения - то же самое, что минимизация эмпирического риска, где функция потерь - логарифм вероятности правильного класса.

Что мы сделали

Мы знаем, что максимизация правдоподобия дает хорошие веса из статистики. Изменив формулу, мы смогли найти такую функцию потерь, что ее минимизация и максимизация правдоподобия это одно и то же.

$$\prod_{i=1}^{l} \mathrm{P}(Y = y_i | \mathbf{x}_i) \to \max_{\mathbf{w}}$$

$$L(\mathbf{w}) = -\sum_{i=1}^{l} ln(\mathrm{P}(Y = y_i | \mathbf{x}_i)) \to \min_{\mathbf{w}}$$

Deep Learning School

Логистическая регрессия

Определение модели

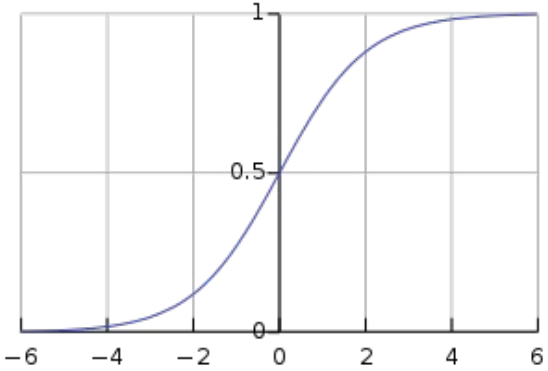
Мы будем искать модель в следующем виде.

$$\mathbf{x} = (1, x_1, \ldots, x_n)$$

$$\hat{f}(\mathbf{x}) = \sigma(\sum_{i=0}^{n} \mathbf{w}_i \mathbf{x}_i)$$

Определение сигмоиды:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Предсказание вероятности

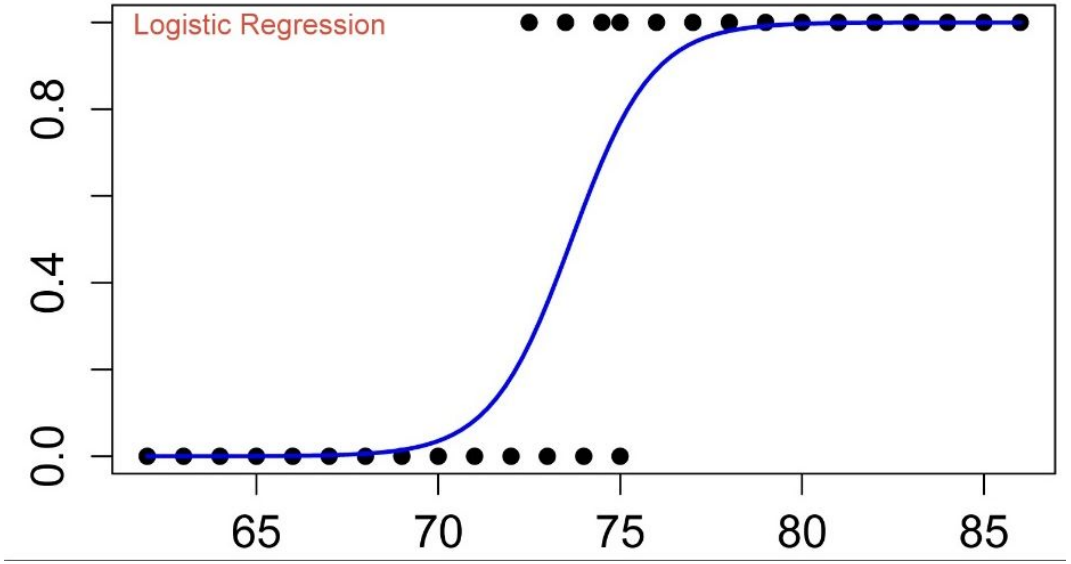
Будем считать, что наша модель предсказывает вероятности.
Именно поэтому она называется регрессией.

Вероятносвть для двух классов можно расписать так:

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

$$\ln \mathrm{P}\left(Y = y_i \mid \mathbf{x}_i\right) = y_i \ln\left(\hat{f}\left(\mathbf{x}_i\right)\right) + \left(1 - y_i\right) \ln\left(1 - \hat{f}\left(\mathbf{x}_i\right)\right)$$

Пример работы



Как выглядит обученная логистическая регрессия на данных с **одним признаком.**

Обучение логистической регрессии

В полученную ранее формулу функции потерь можно подставить вероятность, которую предсказывает логистическая регрессия.

Функция потерь для произвольного классификатора:

$$L(\mathbf{w}) = -\sum_{i=1}^{l} ln(\mathrm{P}(Y = y_i|\mathbf{x}_i)) \to \min_{\mathbf{w}}$$

Функция потерь для логистической регрессии (**LogLoss**):

$$L(\mathbf{w}) = -\sum_{i=1}^{l} y_i \ln(\hat{f}(\mathbf{x}_i)) + (1 - y_i) \ln(1 - \hat{f}(\mathbf{x}_i)) \to \min_{\mathbf{w}}$$

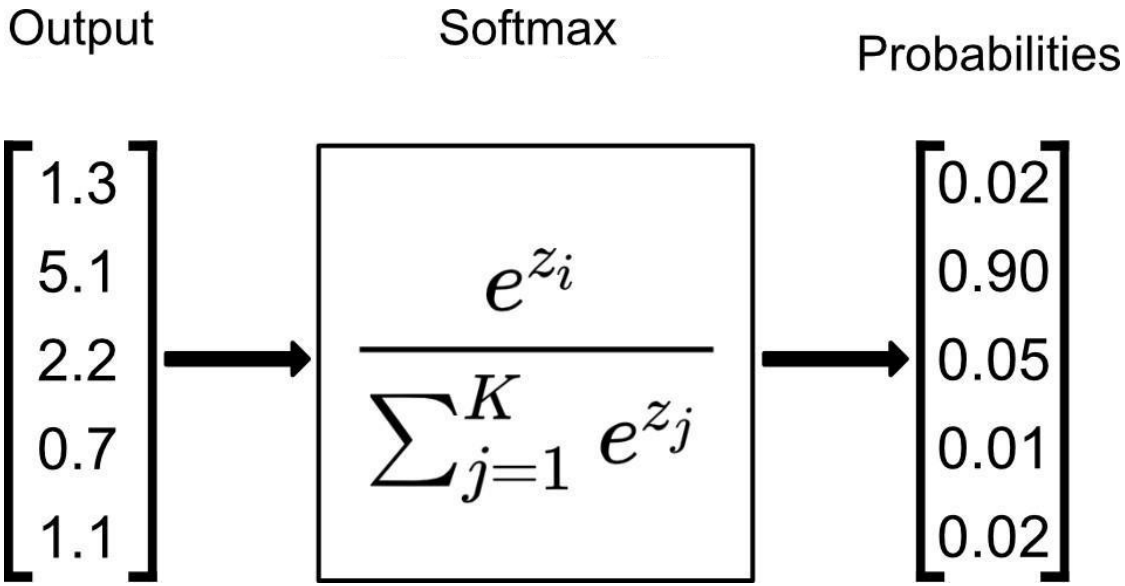
Обобщение на много классов

Пусть у нас есть m классов. Введем две новые функции:

$$logits(\mathbf{x}) = \begin{pmatrix} \mathbf{w}^1 \cdot \mathbf{x} \\ \mathbf{w}^2 \cdot \mathbf{x} \\ \dots \\ \mathbf{w}^m \cdot \mathbf{x} \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n} w_i^1 x_i \\ \sum_{i=0}^{n} w_i^2 x_i \\ \dots \\ \sum_{i=0}^{n} w_i^m x_i \end{pmatrix}$$

$$Softmax(\alpha) = \left(\frac{e^{\alpha_1}}{\sum_{i=0}^{m} e^{\alpha_i}}, \frac{e^{\alpha_2}}{\sum_{i=0}^{m} e^{\alpha_i}}, \dots . \frac{e^{\alpha_m}}{\sum_{i=0}^{m} e^{\alpha_i}} \right)$$

Пример работы Softmax



Много классов

$$\hat{f} = Softmax(logits(\mathbf{x}))$$

Выпишем предсказанную вероятность для к-го класса.
Ее можно подставить в функцию потерь для произвольного классификатора.

$$\mathrm{P}(Y = k|\mathbf{x}) = \hat{f}_k(\mathbf{x}) = \frac{e^{\sum_{i=0}^{n} w_i^k x_i}}{\sum_{k=1}^{m} e^{\sum_{i=0}^{n} w_i^k x_i}}$$

Deep Learning School

Градиентный спуск

Обучение логистической регрессии

В полученную ранее формулу функции потерь можно подставить вероятность, которую предсказывает логистическая регрессия.

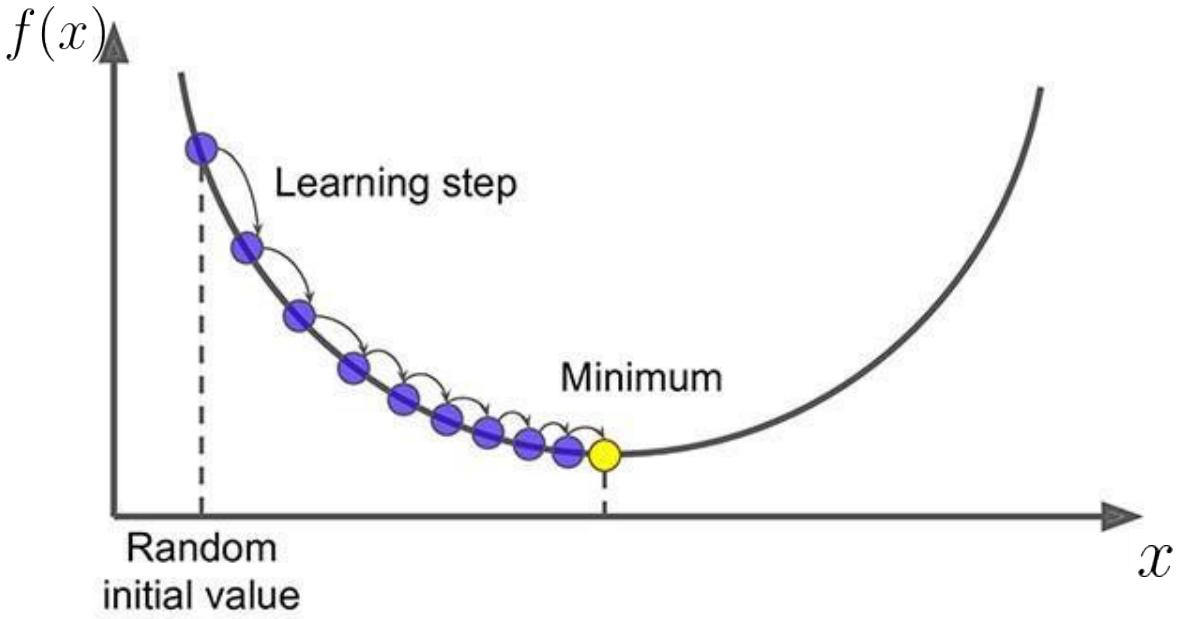
Функция потерь для произвольного классификатора:

$$L(\mathbf{w}) = -\sum_{i=1}^{l} ln(\mathrm{P}(Y = y_i|\mathbf{x}_i)) \to \min_{\mathbf{w}}$$

Функция потерь для логистической регрессии (**LogLoss**):

$$L(\mathbf{w}) = -\sum_{i=1}^{l} y_i \ln(\hat{f}(\mathbf{x}_i)) + (1 - y_i) \ln(1 - \hat{f}(\mathbf{x}_i)) \to \min_{\mathbf{w}}$$

Эвристика градиентного спуска



Градиентный спуск формализация

У нас стоит задача минимизации какой-то функции:

$$f(\mathbf{x}) \to \min_{\mathbf{x}}$$

Чтобы применять метод градиентного спуска нужно уметь вычислять градиент функции в точке:

$$\nabla f(x_1, \ldots, x_n) = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})$$

Заранее зададим некоторое число α, которое будет влиять на то, насколько большие шаги мы делаем. Оно называется *learning rate*

Шаг градиентного спуска

На каждом шаге будем менять все переменные, от которых зависит функция:

$$x_1 = x_1 - \alpha \frac{\partial f}{\partial x_1} \quad \cdots \quad x_n = x_n - \alpha \frac{\partial f}{\partial x_n}$$

Или в векторной форме:

$$\mathbf{x} = \mathbf{x} - \alpha \nabla f$$

Градиентный спуск

1) Выбираем точку, с которой начнем оптимизацию.
2) На каждом шаге будем менять все переменные, от которых зависит функция:

$$x_1 = x_1 - \alpha \frac{\partial f}{\partial x_1} \quad \cdots \quad x_n = x_n - \alpha \frac{\partial f}{\partial x_n}$$

Или в векторной форме:

$$\mathbf{x} = \mathbf{x} - \alpha \nabla f$$

Повторяем, пока изменение не будет достаточно маленьким
или пройдет много шагов.

Градиентный спуск для параболы

Будем минимизировать $f(x) = x^2, \;\; \nabla f = (2x)$

$\alpha = 0.1$

1) $x = 3$
2) Теперь делаем обновления:

$$x = x - 0.1 \cdot 2x$$

С каждым шагом мы будем приближаться к 0 - минимуму функции.

Градиентный спуск для линейной регрессии

Функция потерь (она зависит только от весов, потому что изменять мы будем их):

$$L(\mathbf{w}) = \sum_{i=1}^{l} \left(\mathbf{x}_i^T \mathbf{w} - y_i\right)^2$$

Производная функции потерь по весам:

Градиентный спуск для линейной регрессии

Пошагово возьмем производную лосса по параметрам:

$$\frac{\partial L}{\partial w_1} = \sum_{i=1}^{l} \frac{\partial (\mathbf{x}_i^T \mathbf{w} - y_i)^2}{\partial w_1}$$

$$\frac{\partial L}{\partial w_1} = \sum_{i=1}^{l} 2 \left(\mathbf{x}_i^T \mathbf{w} - y_i \right) \frac{\partial \mathbf{x}_i^T \mathbf{w}}{\partial w_1}$$

$$\frac{\partial L}{\partial w_1} = \sum_{i=1}^{l} 2 \left(\mathbf{x}_i^T \mathbf{w} - y_i \right) x_{i1}$$

Градиентный спуск для линейной регрессии

Функция потерь (она зависит только от весов, потому что изменять мы будем их):

$$L(\mathbf{w}) = \sum_{i=1}^{l} \left(\mathbf{x}_i^T \mathbf{w} - y_i\right)^2$$

Производная функции потерь по весам:

$$\frac{\partial L}{w_1} = \sum_{i=1}^{l} 2\left(\mathbf{x}_i^T \mathbf{w} - y_i\right) x_{i1}$$

Градиентный спуск для линейной регрессии

Будем минимизировать $L(\mathbf{w}) = \sum_{i=1}^{l} \left(\mathbf{x}_i^T \mathbf{w} - y_i\right)^2$

1) Как-то выберем начальные веса.
2) Теперь делаем обновления:

$$w_1 = w_1 - \alpha \frac{\partial L}{\partial w_1} \quad \cdots \quad w_n = w_n - \alpha \frac{\partial L}{\partial w_n}$$

Градиентный спуск для логистической регрессии

Функция потерь:

$$L(\mathbf{w}) = -\sum_{i=1}^{l} y_i \ln(\hat{f}(\mathbf{x}_i)) + (1 - y_i) \ln(1 - \hat{f}(\mathbf{x}_i))$$

$$\hat{f}(\mathbf{x}) = \sigma(\sum_{i=0}^{n} \mathbf{w}_i \mathbf{x}_i)$$

Градиентный спуск для логистической регрессии

Возьмем производную:

$$\frac{\partial L}{w_1} = -\sum_{i=1}^{l} \frac{y_i}{\hat{f}\left(\mathbf{x}_i\right)} \frac{\partial \hat{f}\left(\mathbf{x}_i\right)}{\partial w_1} - \frac{1-y_i}{1-\hat{f}\left(\mathbf{x}_i\right)} \frac{\partial \hat{f}\left(\mathbf{x}_i\right)}{\partial w_1}$$

$$\frac{\partial \hat{f}\left(\mathbf{x}_i\right)}{\partial w_1} = \hat{f}\left(\mathbf{x}_i\right)\left(1-\hat{f}\left(\mathbf{x}_i\right)\right) x_{i1}$$

Соединим:

$$\frac{\partial L}{w_1} = -\sum_{i=1}^{l} x_{i1}\left[y_i\left(1-\hat{f}\left(\mathbf{x}_i\right)\right) - \left(1-y_i\right)\hat{f}\left(\mathbf{x}_i\right)\right]$$

Deep Learning School

Регуляризация

Мультиколлинеарность для линейной регрессии

Вспомним определение линейной регрессии:

$$\hat{f}(\mathbf{x}) = \sum_{i=0}^{n} \mathbf{w}_i \mathbf{x}_i = \mathbf{x} \cdot \mathbf{w}$$

Если столбцы матрицы $\mathbf{X}$ линейно зависимы, то существуют такие коэффициенты $\beta = (\beta_1, \ldots, \beta_n)$, что $\mathbf{X}\beta = 0$.

Но тогда существует бесконечное количество весов, дающих одинаковые предсказания:

$$\forall c \in \mathbb{R} : \mathbf{x} \cdot (\mathbf{w} + c\beta) = \mathbf{x} \cdot \mathbf{w} + \mathbf{x} \cdot \beta = \mathbf{x} \cdot \mathbf{w}$$

Weight Decay

Мы можем предположить, что веса не должны быть большими по модулю.

Изменим функцию потерь, чтобы отражать это (β- некоторая константа):

L_2-регуляризация

$$L(\mathbf{w}) = \sum_{i=1}^{l} \left(\mathbf{x}_i^T \mathbf{w} - y_i\right)^2 + \beta \sum_{j=1}^{n} w_j^2$$

L_1-регуляризация

$$L(\mathbf{w}) = \sum_{i=1}^{l} \left(\mathbf{x}_i^T \mathbf{w} - y_i\right)^2 + \beta \sum_{j=1}^{n} |w_j|$$

Как изменится градиент

L_2-регуляризация

$$\frac{\partial L}{w_1} = \sum_{i=1}^{l} 2\left(\mathbf{x}_i^T \mathbf{w} - y_i\right) x_{i1} + 2\beta w_1$$

L_1-регуляризация

$$\frac{\partial L}{w_1} = \sum_{i=1}^{l} 2\left(\mathbf{x}_i^T \mathbf{w} - y_i\right) x_{i1} + \beta \operatorname{sign}\left(w_1\right)$$

Deep Learning School

Нормализация признаков

Что такое нормализация?

Мы изменяем признаки в датасете по правилу:

$$x_{ij}^{\text{new}} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

μ_j- среднее значение j-го признака в обучающей выборке
σ_j- стандартное отклонение j-го признака в обучающей выборке

Зачем?

- Градиентный спуск и другие методы плохо работают на признаках с очень большим или маленьким масштабом.
- Разный масштаб весов вредит регуляризации.
- Для нормированных данных веса говорят о важности признаков.

The End