Regresja liniowa jednej i wielu zmiennych

Joanna Jaworek-Korjakowska

WEAIIB, Katedra Automatyki i Robotyki, ISS

2019

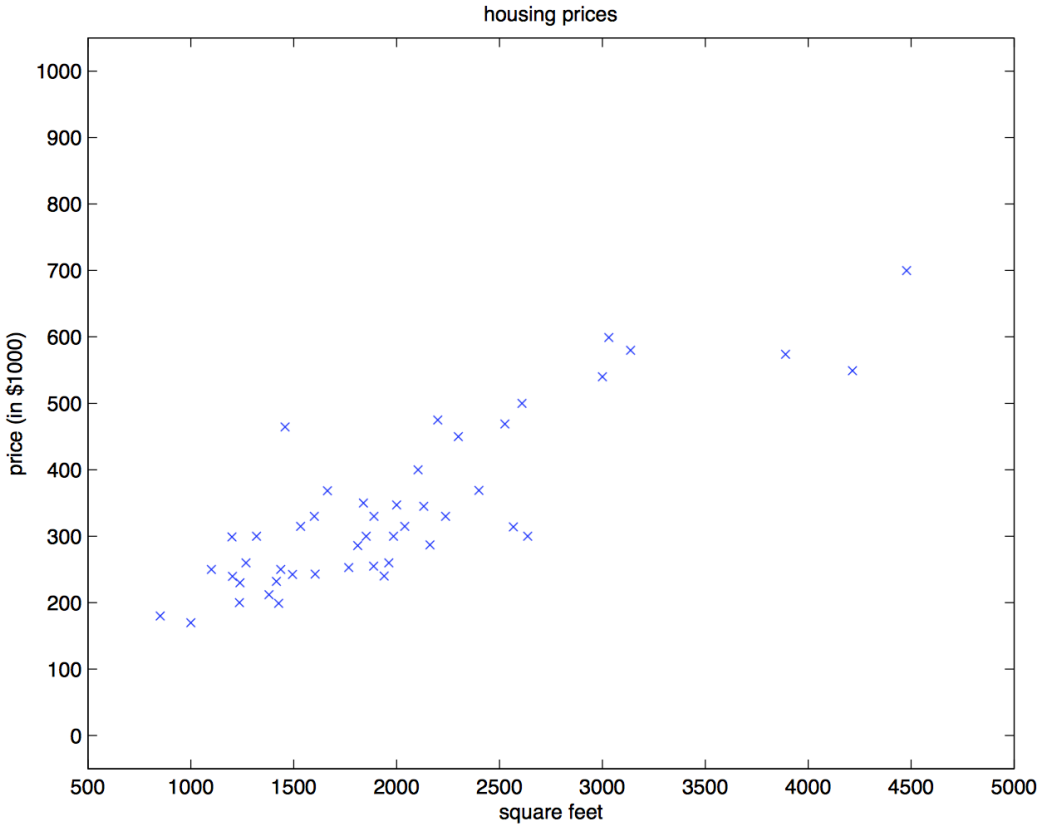
Regresja liniowa - wprowadznie

Zestaw danych dotyczących powierzchni mieszkalnej i ceny domów w mieście

Living area (feet^2)	Price (1000\$s)
2104	400
1600	330
2400	369
1416	232
3000	540
$\vdots$	$\vdots$

Jak możemy przewidywać ceny domów w zależności od ich powierzchni mieszkalnej?

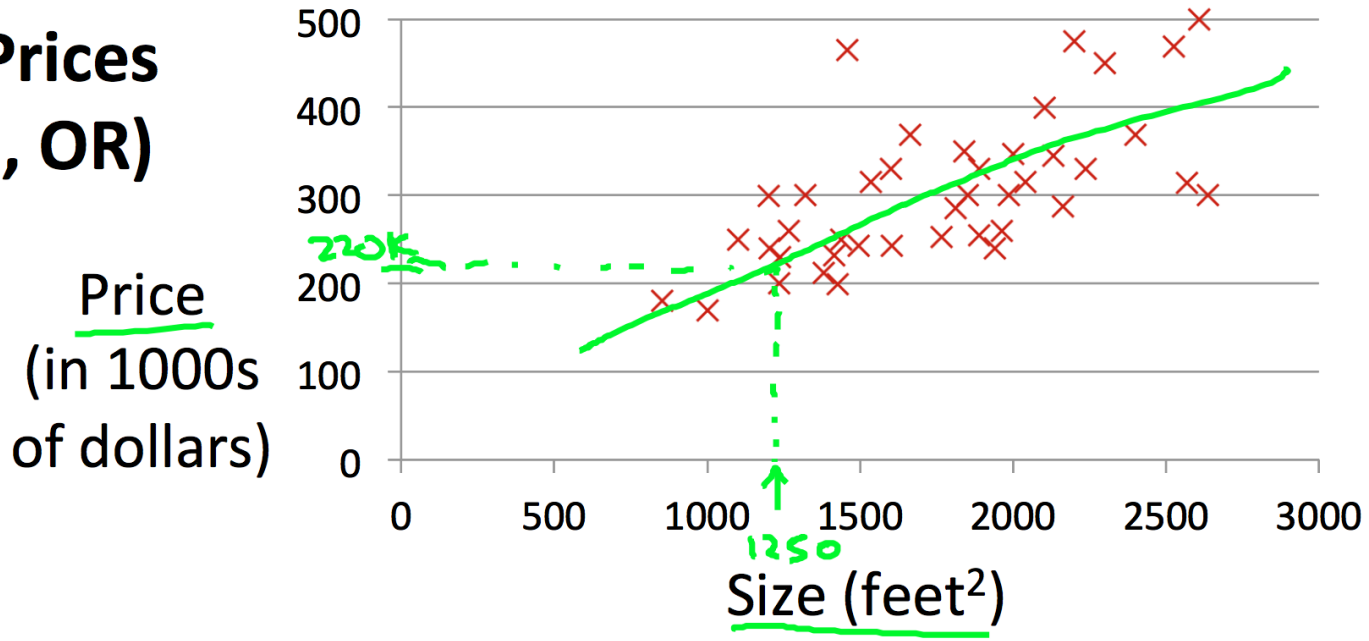
Regresja liniowa - wprowadzenie



Problem regresji - gdy zmienna docelowa, którą próbujemy przewidzieć, jest ciągła.

Regresja liniowa

Housing Prices (Portland, OR)



Supervised Learning

Given the "right answer" for each example in the data.

Regression Problem

Predict real-valued output

Classification: Discrete-valued output

Regresja liniowa

Training set of housing prices (Portland, OR)

Size in feet^2 (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

$m = 47$

Notation:

- **m** = Number of training examples
- **x**'s = "input" variable / features
- **y**'s = "output" variable / "target" variable

(x, y) - one training example

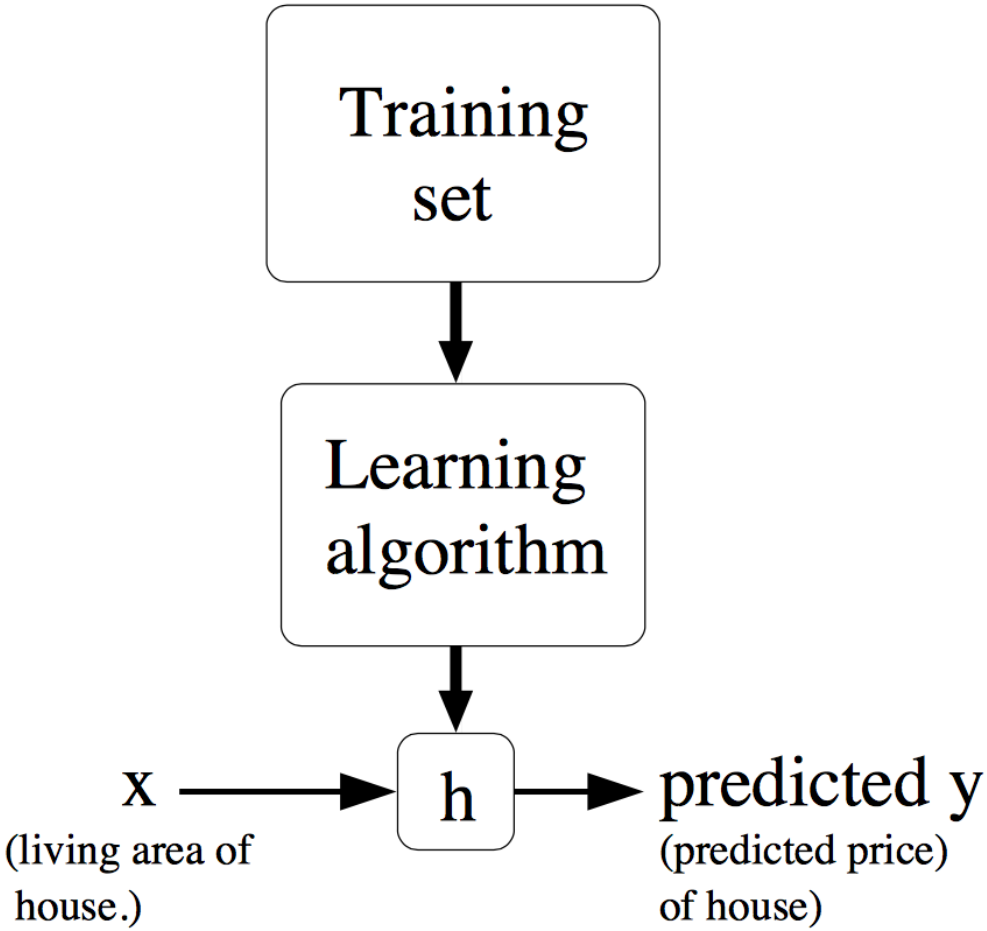
$(x^{(i)}, y^{(i)})$ - i^{th} training example

$x^{(1)} = 2104$

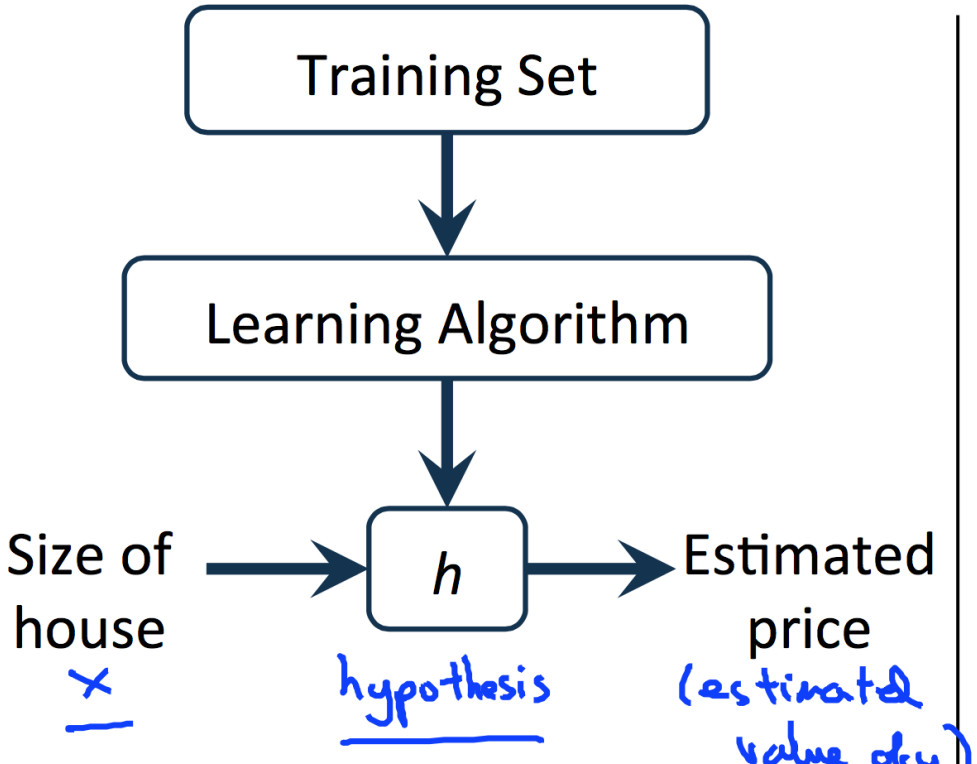
$x^{(2)} = 1416$

$y^{(1)} = 460$

Regresja liniowa



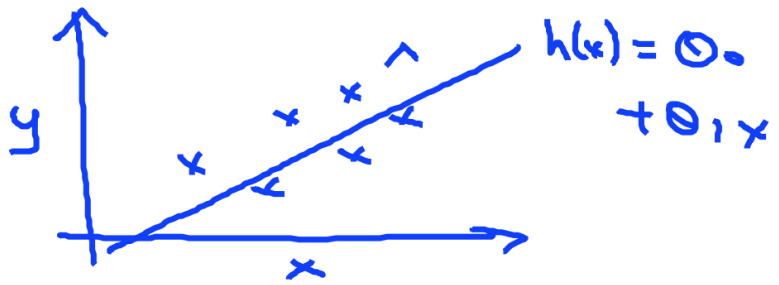
Regresja liniowa



How do we represent *h* ?

$h_\theta(x) = \theta_0 + \theta_1 x$

Shorthand: $h(x)$



Linear regression with one variable. (x)

Univariate linear regression.

└ one variable

Sformułowanie problemu

Regresja liniowa

Rozpatrujemy najprostszy przypadek relacji pomiędzy **zmienną objaśniającą** x oraz **zmienną objaśnianą** y

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i,$$

gdzie $i = 1, \ldots, n$, przy czym β_0, β_1 to nieznane parametry, natomiast ϵ_i to (nieobserwowalne) wartości losowe (błędy), wyjaśniające różnice pomiędzy zaobserwowanymi danymi a wartościami przewidywanymi.

Sformułowanie problemu

Regresja liniowa

Rozpatrujemy najprostszy przypadek relacji pomiędzy **zmienną objaśniającą** x oraz **zmienną objaśnianą** y

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i,$$

gdzie $i = 1, \ldots, n$, przy czym β_0, β_1 to nieznane parametry, natomiast ϵ_i to (nieobserwowalne) wartości losowe (błędy), wyjaśniające różnice pomiędzy zaobserwowanymi danymi a wartościami przewidywanymi.

Jest to oczywiście układ n równań następującej postaci

$$\begin{aligned} y_1 &= \beta_0 + \beta_1 x_1 + \epsilon_1 \\ y_2 &= \beta_0 + \beta_1 x_2 + \epsilon_2 \\ &\vdots \\ y_n &= \beta_0 + \beta_1 x_n + \epsilon_n, \end{aligned}$$

w którym zdaniem jest wyznaczenie (estymacja) nieznanych parametrów β_0, β_1.

Sformułowanie problemu

Training Set

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

$m = 47$

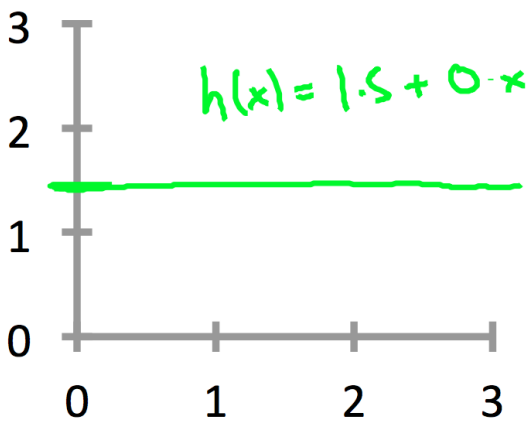
Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

θ_i's: Parameters

How to choose θ_i's ?

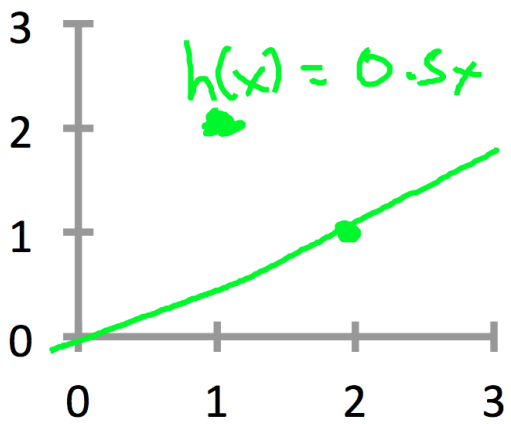
Sformułowanie problemu

$$h_\theta(x) = \theta_0 + \theta_1 x$$



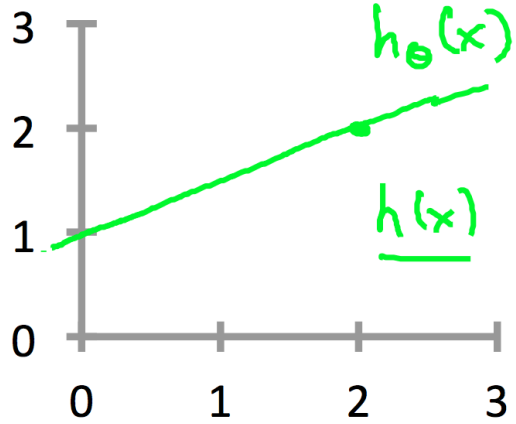
$\rightarrow \theta_0 = 1.5$

$\rightarrow \theta_1 = 0$



$\rightarrow \theta_0 = 0$

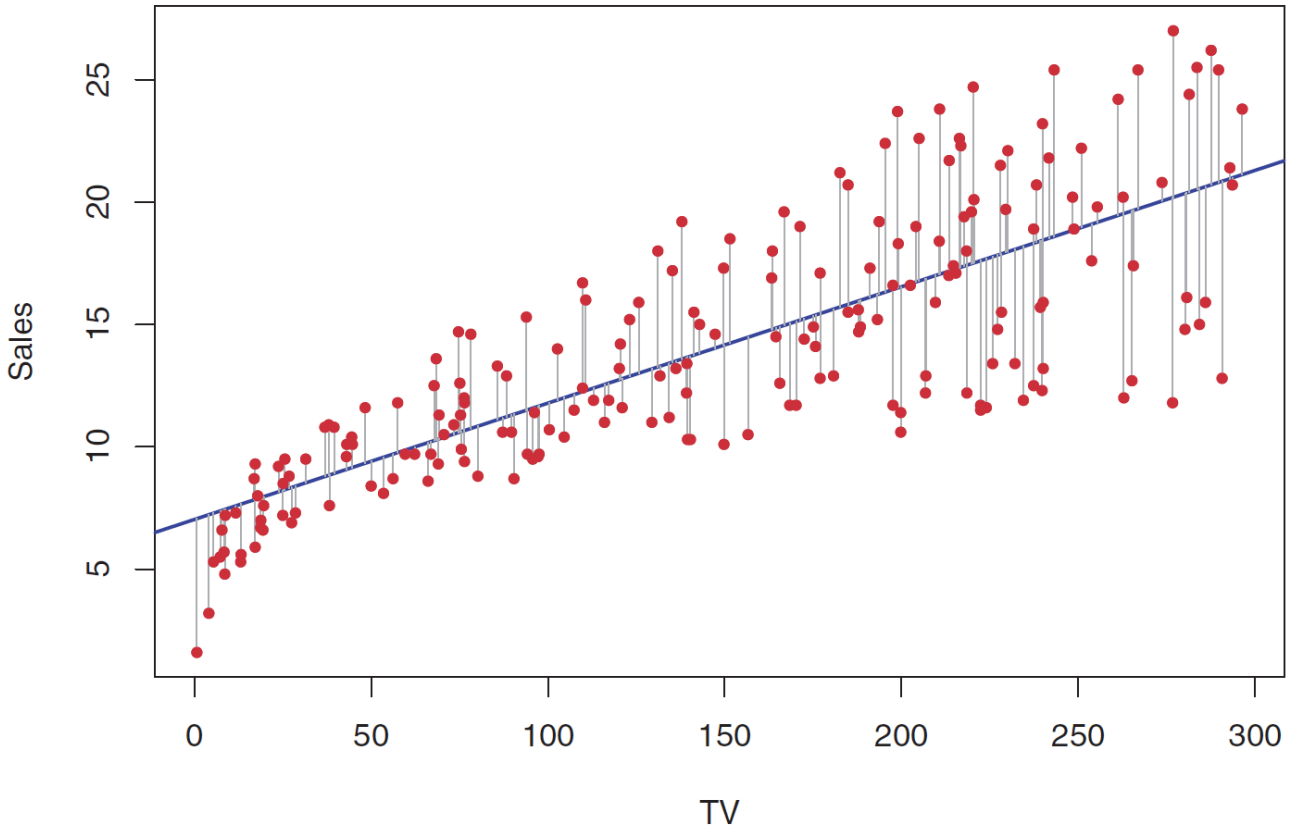
$\rightarrow \theta_1 = 0.5$



$\rightarrow \theta_0 = 1$

$\rightarrow \theta_1 = 0.5$

Sformułowanie problemu



Idea: jest dobranie θ_0 , θ_1, aby $h_\theta(x)$ jak najlepiej pasowało do y, dla przykładów treningowych (x, y)

Funkcja kosztu

Problem uczenia maszynowego polega na tym: jak mając zbiór uczący znaleźć "dobre"parametry? Aby sformalizować ten problem wyprowadzimy funkcję kosztu.

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

Teraz możemy powiedzieć, że "dobre"parametry to takie, które minimalizują funkcję kosztu.

Funkcja kosztu

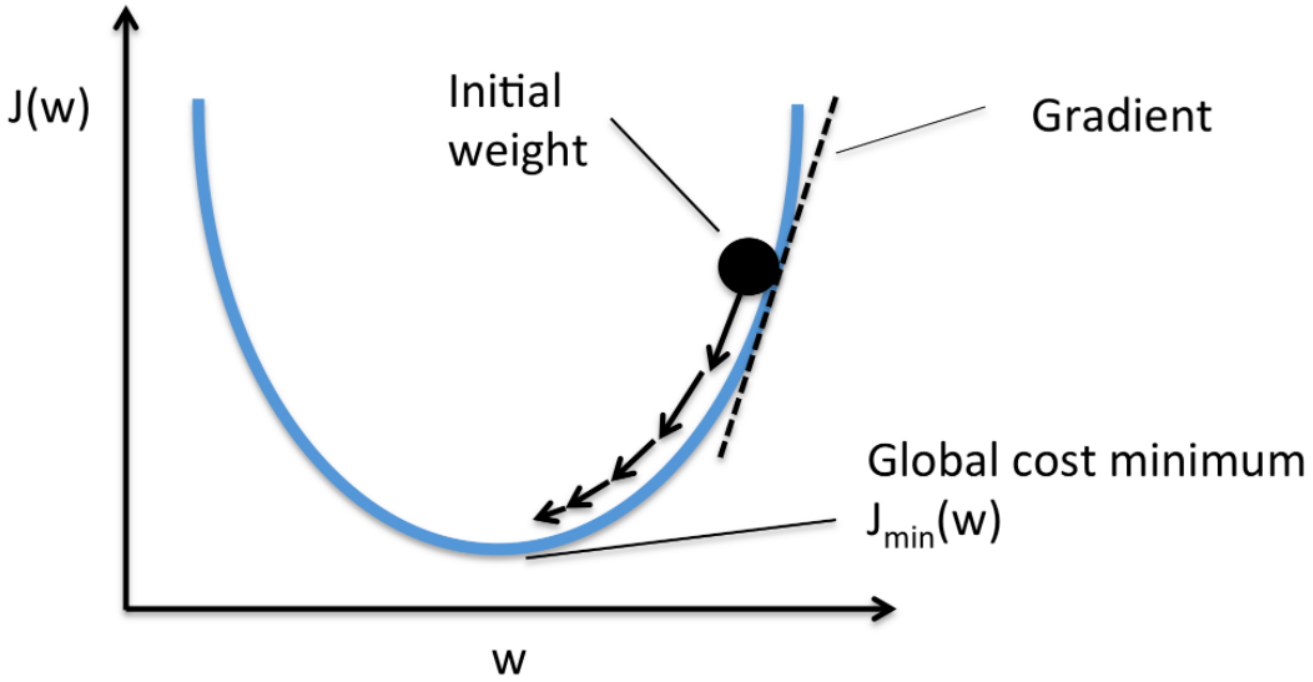
$(h_\theta(x) - y)^2$ - chcemy, aby kwadratowa różnica między wartościami treningowymi, a wartościami hipotezy h była jak najmniejsza.
Dokładniejsza specyfikacja naszej funkcji kosztu, którą chcemy zminimalizować wygląda następująco:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

- oznaczamy funkcję kosztu jako $J(\theta_0, \theta_1)$
- iterujemy po wszystkich przykładach treningowym, sumując błąd na każdym z nich
- bierzemy średnią, dzieląc ją dodatkowo przez 2, aby dalsze obliczenia były łatwiejsze

To, co nas interesuje to: $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$ Oznacza to: znajdź mi takie θ_0, θ_1, żeby wartość $J(\theta_0, \theta_1)$ była jak najmniejsza.
Tego typu funkcję kosztu nazywamy błędem kwadratowym, jest ona często używana w problemie minimalizacji. Są oczywiście inne funkcje, ale ta działa bardzo dobrze i jest popularna.

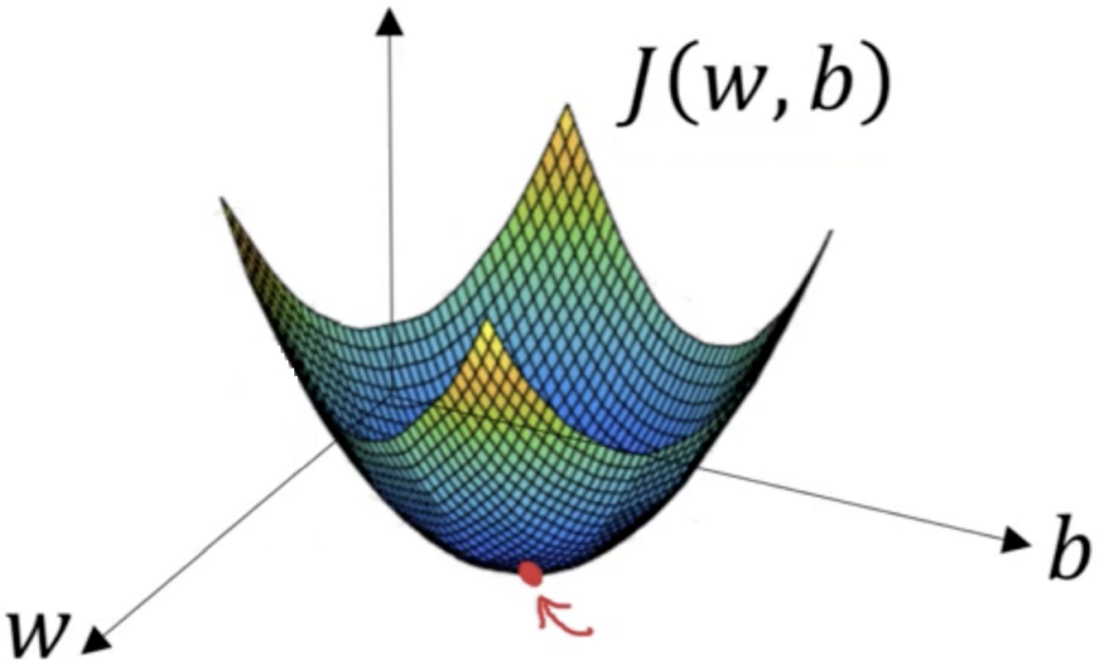
Przykład I



Wykres przedstawia zależność wartości funkcji kosztu w zależności od θ.

Przykład II

Jak będzie wyglądał teraz wykres funkcji kosztu? Ponieważ funkcja kosztu zależy teraz od dwóch zmiennych, θ_0 i θ_1 wykres będzie dwuwymiarowy. Okazuje się że jest on paraboloidą. Jednak w dalszej części kursu, kiedy parametrów θ może być więcej, narysowanie wykresu funkcji kosztu staje się niemożliwe. Pomocna za to okazuje się rzut konturu, czyli poniższy wykres:



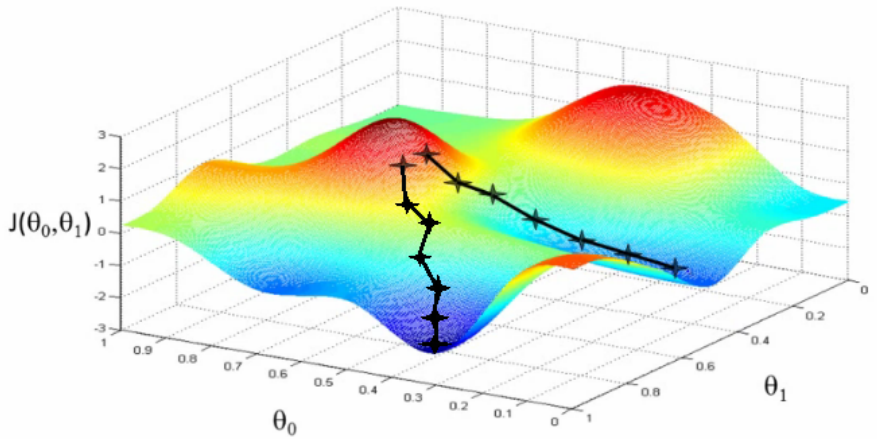
Gradient prosty - gradient descent

Mamy naszą hipotezę oraz funkcję, która określa jak dobra jest hipoteza - funkcję kosztu. Stoimy przed problemem ulepszenia hipotezy, czyli minimalizacji funkcji kosztu $J(\theta_0, ..., \theta_n)$.

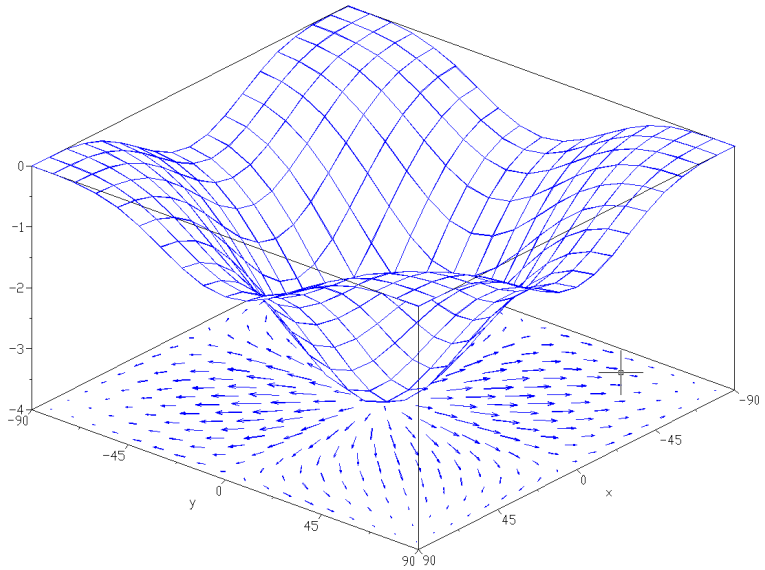
Rozwiązaniem jest zastosowanie metody gradientu prostego (gradient descent).
Agorytm:

- ustawiamy parametry $\theta_0 = 0, ..., \theta_n = 0$
- zmieniamy parametry $\theta_0, ..., \theta_n$, aż osiągniemy minimum funkcji

Przykład dla $n = 1$.



Gradient prosty



Gradient funkcji w danym punkcie przestrzeni to wektor, który wskazuje kierunek w jakim powinniśmy się poruszać, aby wartości funkcji rosły najszybciej. Zatem taki wektor wzięty z ujemnym znakiem reprezentuje kierunek, w którym powinniśmy się poruszać, aby osiągnąć minimum funkcji.
Gradient funkcji zapisujemy:
$\nabla J = (\frac{\partial}{\partial \theta_0} J, ..., \frac{\partial}{\partial \theta_n} J)$

Gradient prosty dla regresji liniowej

Chcemy zastosować gradient prosty dla regresji liniowej.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Jak widać, we wzorze na gradient (1) występuje pochodna: $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$.
Musimy ją więc wyznaczyć:

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$$

Potrzebujemy tych pochodnych dla $j = 0$ i $j = 1$:

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\frac{\partial}{\partial} \frac{1}{} \sum^{m}$$

Gradient prosty dla regresji liniowej

Chcemy zastosować gradient prosty dla regresji liniowej.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Jak widać, we wzorze na gradient (1) występuje pochodna: $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$.
Musimy ją więc wyznaczyć:

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$$

Gradient prosty dla regresji liniowej

Potrzebujemy tych pochodnych dla $j = 0$ i $j = 1$:

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

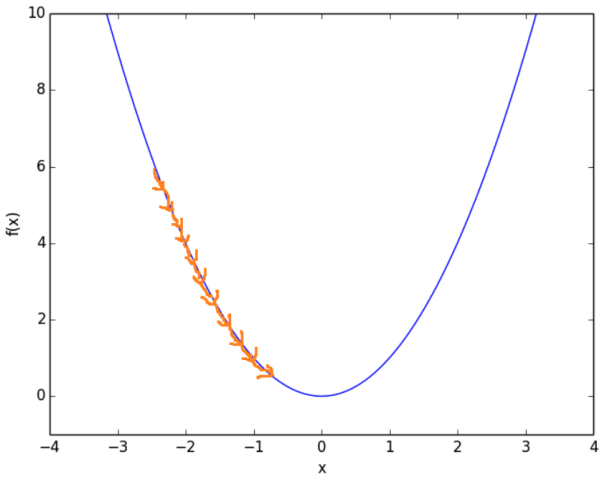
Wstawmy otrzymane wartości do wzoru na gradient:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

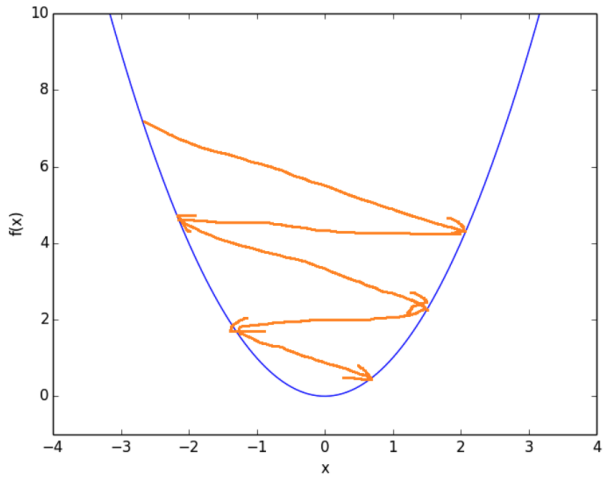
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

Powyższe parametry poprawiamy oczywiście równocześnie.

α - learning rate



za mały:
uczenie jest wolne



za duży:
algorytm nie trafia w minimum,
a nawet nie zbiega się

Metoda gradientu dla wielu zmiennych

Teraz będziemy musieli korzystać z innych cech danego zjawiska takich jak ilość sypialni, okien, pięter etc.

$$ozn\ x_1, x_2, ..., x_n$$

Ilość cech będziemy oznaczać przez n. Funkcja hipotezy h jako argument będzie przyjmowała wektor x zawierający wartości poszczególnych cech.

$$h_\Theta(x) = \Theta_0 + x_1\Theta_1 + x_2\Theta_2 + ... + x_n\Theta_n$$

Dla uproszczenia zapisu zdefiniujemy

$$\forall i \epsilon \mathbb{N} : x_0^i = 1$$

teraz:

$$x \epsilon \Re^{n+1},\ x^{(i)} = \begin{bmatrix} x_0^{(i)} & x_1^{(i)} & ... & x_n^{(i)} \end{bmatrix}$$

$$h_\Theta(x) = \Theta_0 x_0 + \Theta_1 x_1 + ... + \Theta_n x_n$$

Współczynniki theta również zgrupujemy w n+1 wymiarowy wektor theta.

$$\Theta = \begin{bmatrix} \Theta_0 & \Theta_0 & ... & \Theta_n \end{bmatrix}$$

Metoda gradientu dla wielu zmiennych

Jak używać metody gradientu prostego dla regresji liniowej z wieloma własnościami?
Dla naszej nowej hipotezy funkcja kosztu wygląda w następujacy sposób:

$$J(\Theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})^2$$

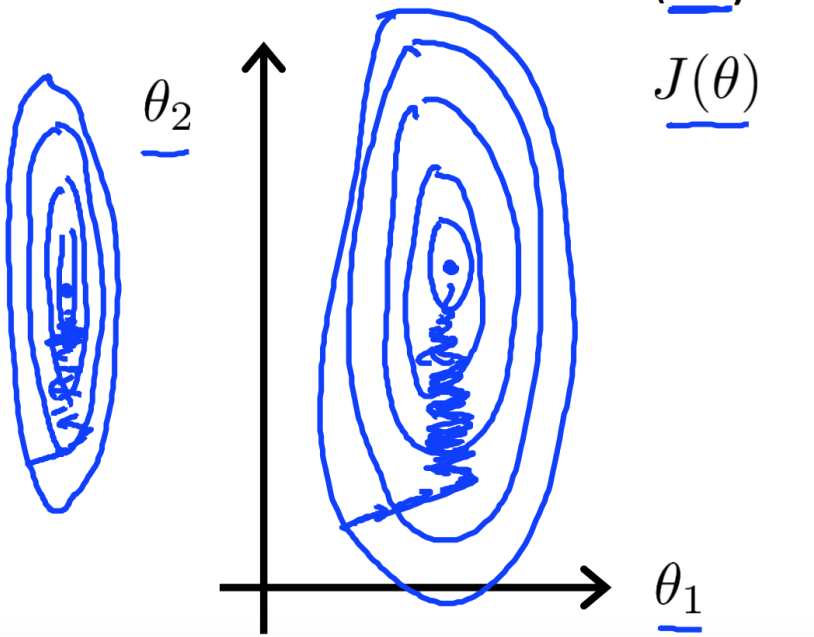
Algorytm gradientu prostego:
Powtarzaj do zbieżności
$dla\ j = \{1, ..., n\}:\ \Theta_j := \Theta_j - \alpha\frac{\partial}{\partial\Theta_j}J(\Theta)$ przypisuj jednocześnie!
Warto zauważyć, że nasz algorytm zbytnio się nie zmienił i wciąż działa w taki sam sposób dla przypadku w którym korzystamy z tylko jednej właściwości.

Skalowanie danych

Idea: Make sure features are on a similar scale.

E.g. x_1 = size (0-2000 feet^2)

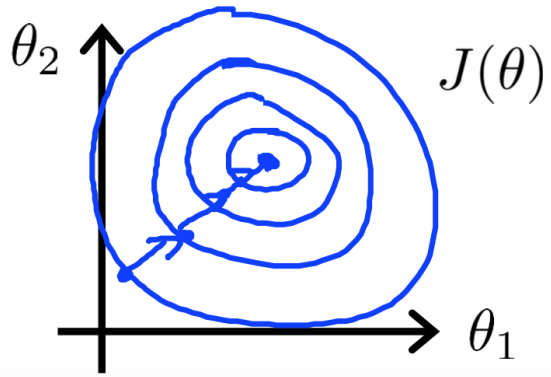
x_2 = number of bedrooms (1-5)



$$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$

$0 \le x_1 \le 1$ $\quad 0 \le x_2 \le 1$



W rzeczywistości elipsy będą jeszcze węższe niż są tutaj przedstawione. Czerwone strzałki przedstawiają drogę jaką pokonuje nasz gradient. Im węższa elipsa tym nasz gradient będzie działał dłużej.

Równanie normalne

Równanie normalne, dla niektórych problemów regresji liniowej, pozwoli nam w znacznie lepszy sposób rozwiązać problem obliczając optymalne rozwiązanie.

Algorytm, którego używaliśmy do tej pory, gradient prosty, jest algorytmem iteracyjnym, który potrzebuje wielu kroków do osiągnięcia minimum.

W przeciwieństwie do niego, równanie normalne pozwoli nam rozwiązać ten problem dla θ analitycznie, w jednym kroku. Ma ono pewne zalety, ale także wady, lecz zanim do tego przejdziemy, spróbujmy poznać pewną intuicję.

Możemy obliczyć optymalne wartości θ, rozwiązując poniższe równanie:

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

.

Gradient prosty - równanie normalne

Kiedy powinniśmy używać gradientu prostego, a kiedy równania normalnego?
m - liczba przykładów treningowych, *n* - liczba własności

Gradient prosty	**Równanie normalne**		
Musimy wybrać $\alpha $	Nie musimy wybierać $\alpha $		
Potrzebuje wielu iteracji	Nie musimy iterować		
Działa bardzo dobrze nawet gdy n jest bardzo duże	Musimy policzyć $(X^T X)^{-1} $, co jest bardzo wolne, gdy n jest duże.		
Bardziej uniwersalny, działa dla wielu problemów	Tylko dla regresji liniowej		