Linear Regression

Kairit Sirts

21.03.2014

Regression problem

- In **classification** problem we predict discrete class labels.
- In **clustering** problem we try to find hidden structure.
- In **regression** problem we predict **continuous values**.

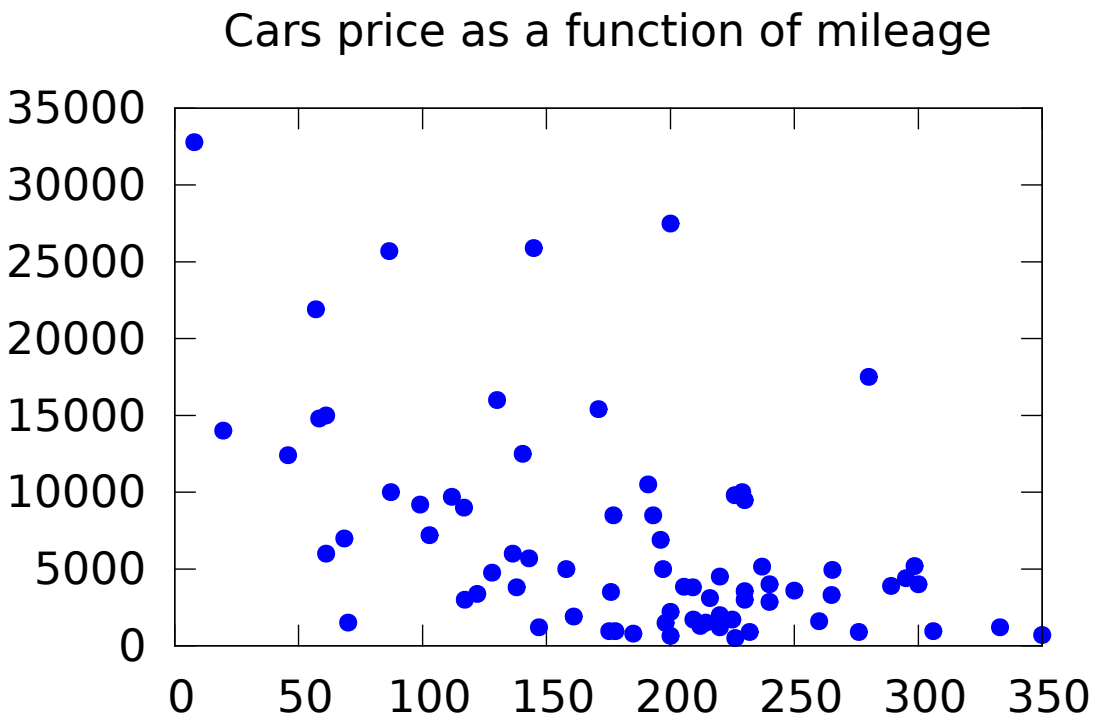
Example:

Set of cars data, recently put into sale on auto24.ee page.

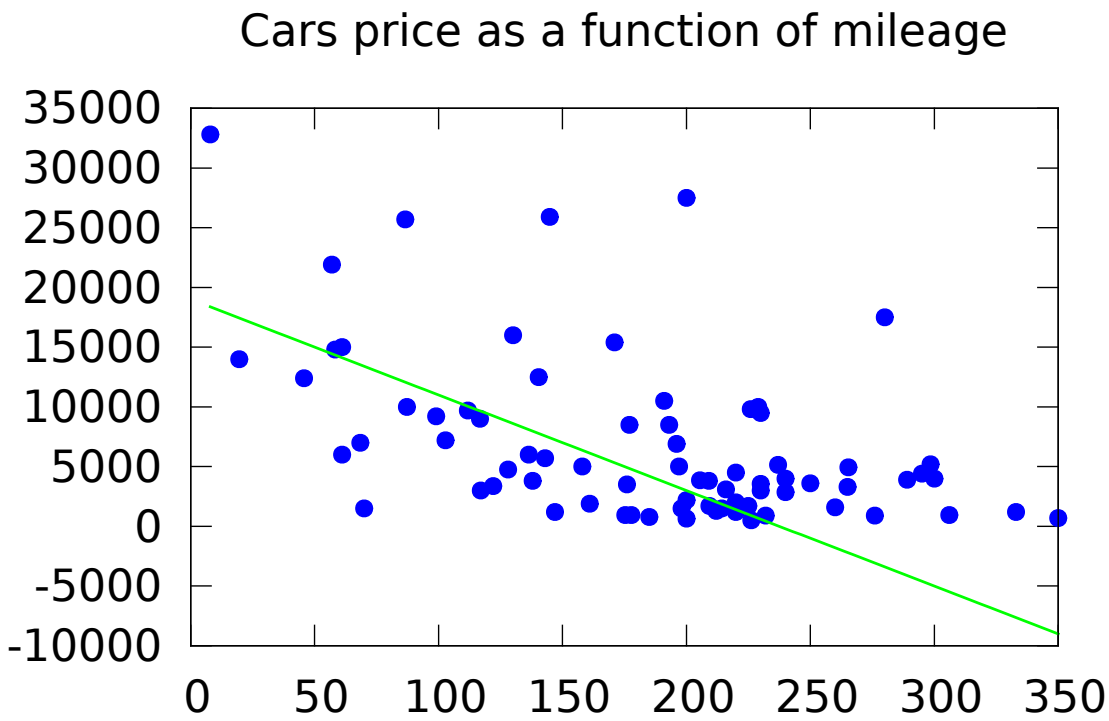
Year	Mileage	Power	Fuel	Transmission	Price
2009	144800	240	D	A	25900
2004	12800	96	P	M	4750
2003	229000	160	D	A	9999
2005	161000	51	D	M	1900
2008	220000	71	P	M	4500
...	...	...	...	...	...

Goal: Learn to predict the price of the car from its properties.

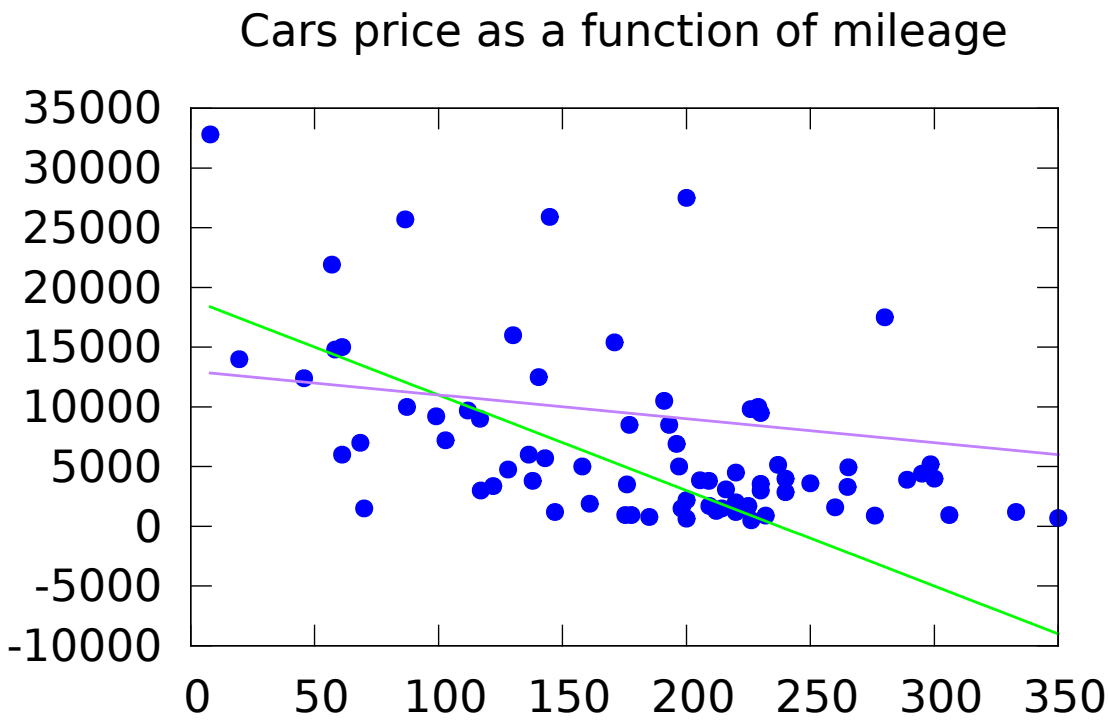
Example: Cars' prices



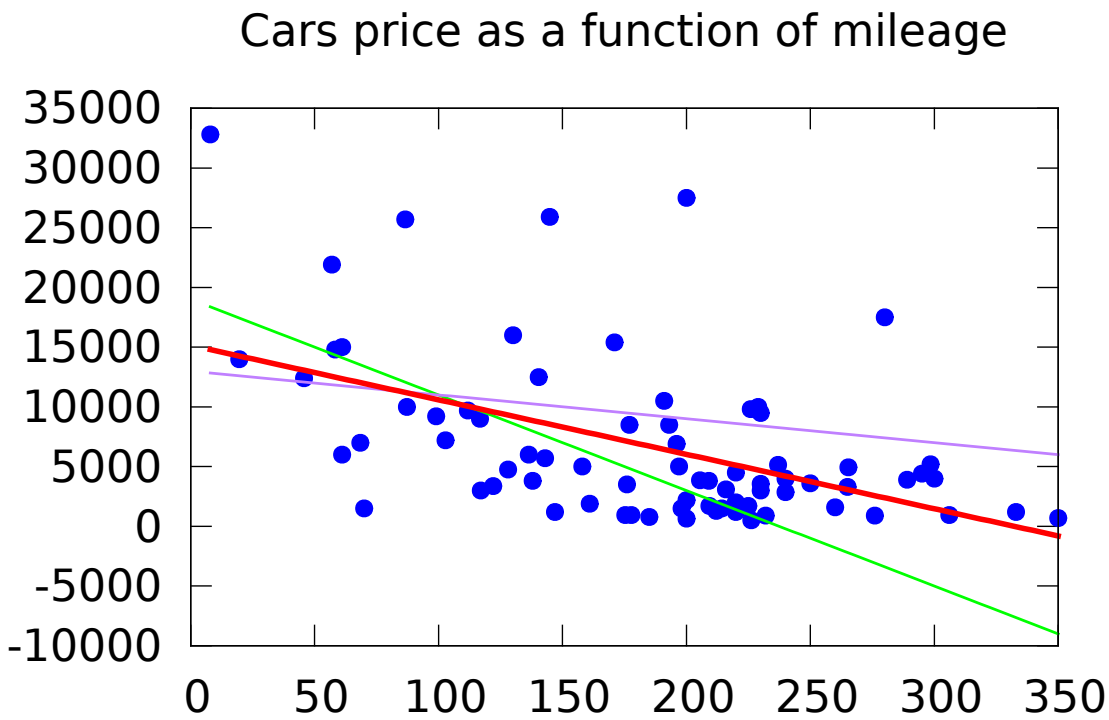
Example: Cars' prices



Example: Cars' prices



Example: Cars' prices



Linear hypothesis

The model for linear regression is a linear equation:

$$h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x$$

$\theta_0, \theta_1 \in R$ - model parameters

Our goal is to find a set of parameters that best fit to the data.

Linear hypothesis

We train the model to find the best parameters. The trained model can be used to make predictions on new data.

For example:

The trained model expressing the price of the car as a function of mileage is:

$$h_{\theta}(x) = 15150 - 45.6x$$

According to this model the car with the mileage 157600 km could be sold with the price:

$$h_{\theta}(x) = 15150 - 45.6 \times 157.6 \approx 7963$$

Generalization to n-dimensional data points

Usually the data points are real-valued vectors:

$$\mathbf{x} = [x_1, x_2, \cdots, x_n]^T, x_1 \cdots x_n \in \mathbb{R}$$

In this case the hypothesis is:

$$h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots \theta_n x_n$$

By defining $x_0 = 1$ we can write:

$$h_{\boldsymbol{\theta}}(x) = \sum_{j=0}^{n} \theta_j x_j = \boldsymbol{\theta}^T \mathbf{x},$$

where $\boldsymbol{\theta} = [\theta_0, \theta_1, \cdots, \theta_n]^T$

Notations

When we have m labelled training data points then they are denoted in pairs:

$$(\mathbf{x}_i, y_i), \quad i = 1 \cdots m$$

$\mathbf{x}_i$ are n-dimensional real-valued vectors.
$y_i \in \mathbb{R}$ is the true value or answer for the i-th data point.
$\mathbf{X}$ is called the **design matrix**, where each row is a data point.

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \cdots \\ \mathbf{x}_m^T \end{bmatrix}$$

Y is the column vector with correct answers.

$$Y = [y_1, y_2, \cdots, y_m]^T$$

Objective function

How to find the optimal parameters?

We wish to find parameters $\boldsymbol{\theta}$, such that the distances of all training data point from the line $\boldsymbol{\theta}^T \mathbf{x}$ would be minimal.

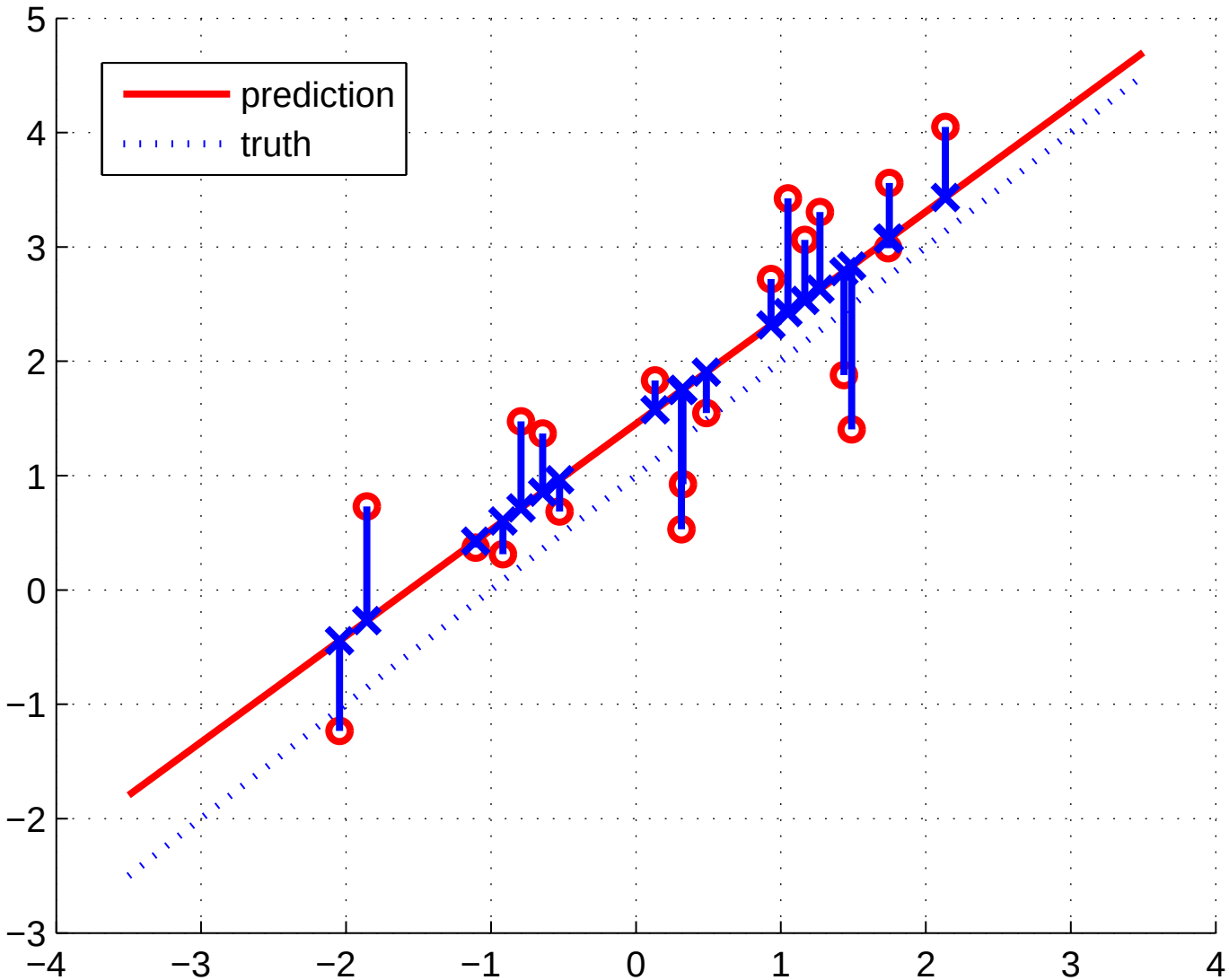
Define the objective function:

Objective function is the sum of the squares of the distances from each training point to the line determined by parameters $\boldsymbol{\theta}$:

$$J(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^{m} (y_i - \boldsymbol{\theta}^T \mathbf{x}_i)^2$$

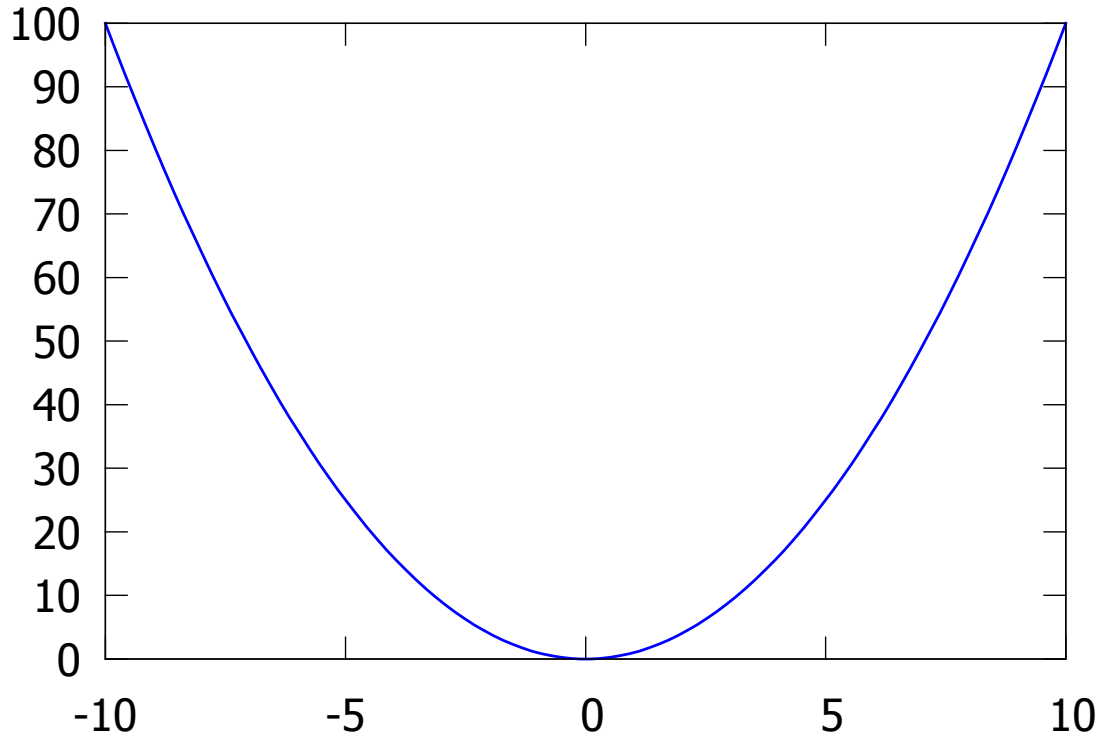
The linear regression with such an objective function is also called **Ordinary least squares regression**.

Ordinary least squares



Why squared cost-function?

We could have chosen just the absolute value of the distances or the absolute value of the cubes etc.



The squared function is convenient mathematically—it leads to a **convex** objective function that is guaranteed to have **only one** optimum, which is the **global optimum**.

Model vs fitting the parameters

Now we have defined the model and the cost function.

We need to decide, how to fit the parameters, meaning: how to find the parameters that minimize the cost function.

There are many ways for fitting the parameters. Most of the methods are general and not specific to the model we are discussing.

Fitting the parameters

- Iterative methods: Gradient Descent
- Analytical solution

Method of Gradient Descent

- Iteratively updates the parameters. Each parameter is updated in the direction of its negative gradient.
- Initialize the parameters randomly (for example, set to 0).
- For all θ_j (in parallel):

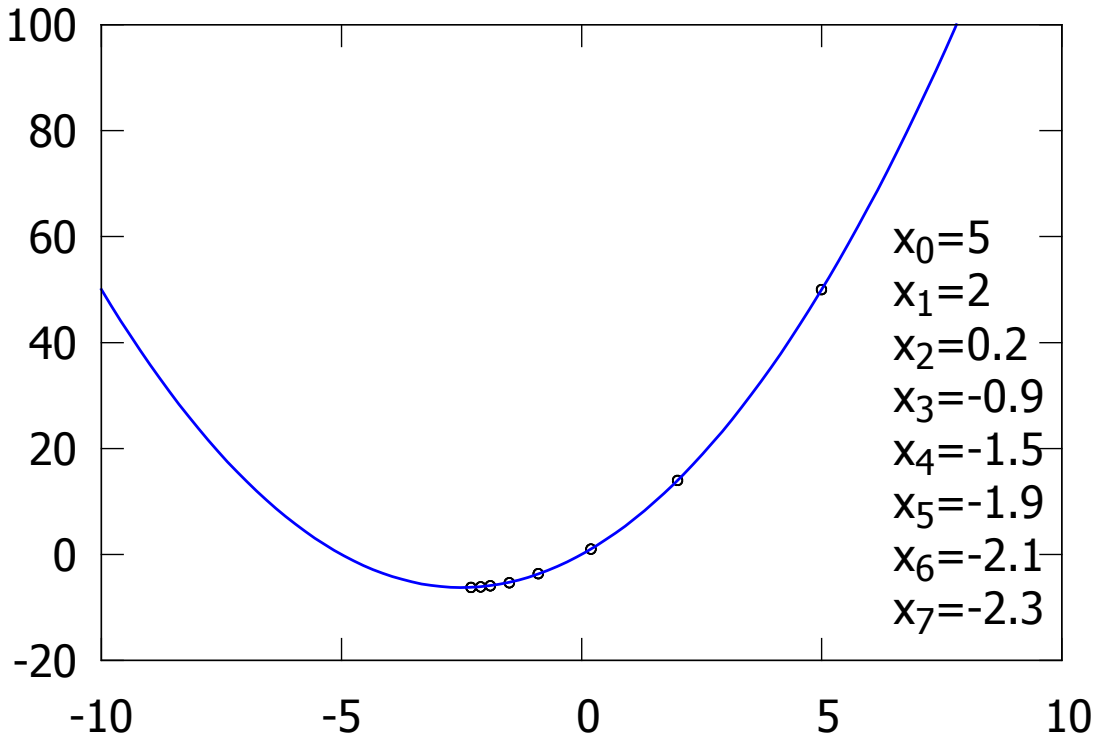
$$\theta_j^{(k+1)} = \theta_j^{(k)} - \alpha \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j}$$

- Continue until convergence
- α: learning rate, determines the length of the steps taken.
- Gradient descent is a **first order** method, because only uses first derivatives.

Gradient Descent in action

$$y = x^2 + 5x \qquad y' = 2x + 5$$

$$x_0 = 5 \qquad \alpha = 0.2$$



Gradient Descent for least squares

- Find the derivative of the objective function

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_{\boldsymbol{\theta}}(\mathbf{x}) - y)^2$$
$$= \frac{2}{2} (h_{\boldsymbol{\theta}}(\mathbf{x}) - y) \frac{\partial}{\partial \theta_j} h_{\boldsymbol{\theta}}(\mathbf{x}) = (h_{\boldsymbol{\theta}}(\mathbf{x}) - y) x_j$$

- For the whole data set

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = \sum_{i=1}^{m} (h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i) x_{ij}$$

- Update rule: for each θ_j simultaneously:

$$\theta_j^{k+1} = \theta_j^k - \alpha \sum_{i=1}^{m} (h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i) x_{ij}$$

Closed form solution

- Least squares linear regression can be also solved analytically (because the objective function is convex)
- Express the objective function in matrix notation:

$$
\begin{aligned}
J(\boldsymbol{\theta}) &= \frac{1}{2}\sum_{i=1}^{m}(h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i)^2 = \frac{1}{2}\sum_{i=1}^{m}(\boldsymbol{\theta}^T\mathbf{x}_i - y_i)^2 \\
&= \frac{1}{2}(\mathbf{X}\boldsymbol{\theta} - Y)^T(\mathbf{X}\boldsymbol{\theta} - Y) \\
&= \frac{1}{2}\boldsymbol{\theta}^T(\mathbf{X}^T\mathbf{X})\boldsymbol{\theta} - \boldsymbol{\theta}^T(\mathbf{X}^T Y) + \frac{1}{2}Y^T Y
\end{aligned}
$$

Closed form solution

- Find the derivative

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \left(\frac{1}{2} \boldsymbol{\theta}^T (\mathbf{X}^T \mathbf{X}) \boldsymbol{\theta} - \boldsymbol{\theta}^T (\mathbf{X}^T Y) + \frac{1}{2} Y^T Y \right)$$
$$= \mathbf{X}^T \mathbf{X} \boldsymbol{\theta} - \mathbf{X}^T Y$$

- Set it to 0

$$\mathbf{X}^T \mathbf{X} \boldsymbol{\theta} = \mathbf{X}^T Y$$

- Solve for $\boldsymbol{\theta}$

$$\boldsymbol{\theta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T Y$$

- This is called the **normal equation**.

Probabilistic interpretation of least squares

- ϵ is the **residual error** between predictions and true answers

$$y = h_{\boldsymbol{\theta}}(\mathbf{x}) + \epsilon$$

- Residual error is commonly assumed to be Gaussian:

$$\epsilon \sim \mathcal{N}(\mu, \sigma^2)$$

- Thus y can be interpreted as a random variable

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y|\mu(\mathbf{x}), \sigma^2(\mathbf{x}))$$

- In the simplest case $\mu = \boldsymbol{\theta}^T \mathbf{x}$ and noise σ^2 is fixed.

Probabilistic interpretation of least squares

- Use MLE and find the parameters that maximize the log-likelihood (minimize the negative log-likelihood)

$$\ell(\boldsymbol{\theta}) = \log p(\mathbf{X}|\boldsymbol{\theta}) = \sum_{i=1}^{m} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$$

- The log-likelihood of the defined model is:

$$\begin{aligned}\ell(\boldsymbol{\theta}) &= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - h_{\boldsymbol{\theta}}(\mathbf{x}_i))^2}{2\sigma^2}\right) \\ &= -\frac{1}{2\sigma^2} RSS(\boldsymbol{\theta}) - \frac{m}{2}\log(2\pi\sigma^2),\end{aligned}$$

- where **residual sum of squares** $RSS(\boldsymbol{\theta}) = \sum_{i=1}^{m} (y_i - \boldsymbol{\theta}^T \mathbf{x}_i)^2$

Non-linear functions with linear regression

- Replace $\mathbf{x}$ with some non-linear mapping of the inputs $\phi(\mathbf{x})$:

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \phi(\mathbf{x})$$

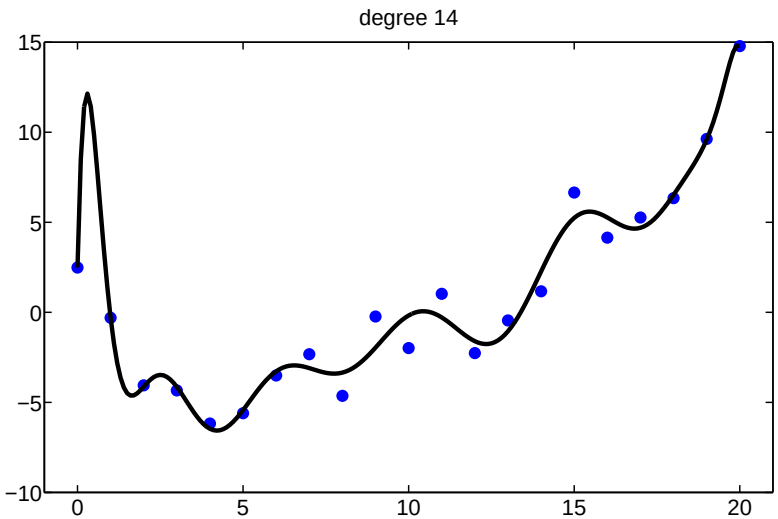
- This is called **basis function expansion**
- For example, **polyomial regression** has the basis function for some d:

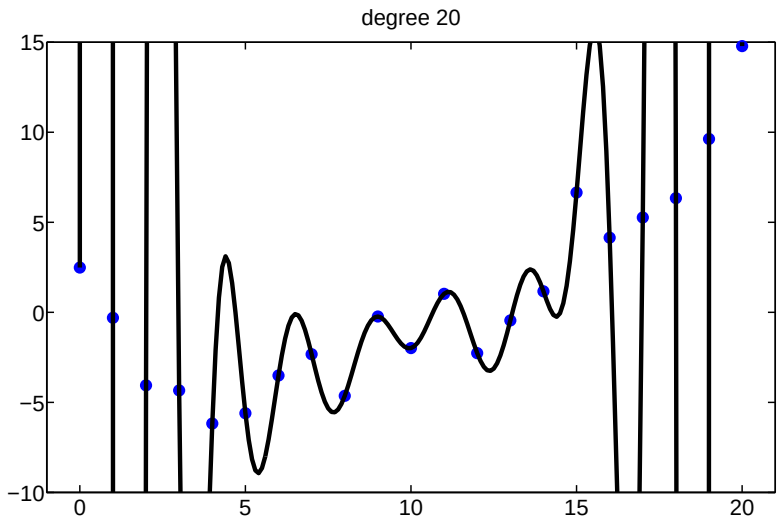
$$\phi(x) = [1, x, x^2, \ldots, x^d]$$

- The model is still linear in the parameters, so it is still called linear regression.

Polynomial regression

- Polynomial function has the global effect and thus with too large degree leads to overfitting.
- The parameters are big numbers that balance out exactly to fit the training data.





Encouraging small weights

- We can encourage small parameters by using a zero-mean Gaussian prior:

$$P(\boldsymbol{\theta}) = \prod_j \mathcal{N}(\theta_j | 0, \tau^2)$$

- Incorporating this into the log-likelihood yields:

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{m} \log \mathcal{N}(y_i | \boldsymbol{\theta}^T \mathbf{x}_i, \sigma^2) + \sum_{j=1}^{n} \log \mathcal{N}(\theta_j | 0, \tau^2)$$

Regularized linear regression

- Adding priors to weights leads to minimizing the **regularized** objective function:

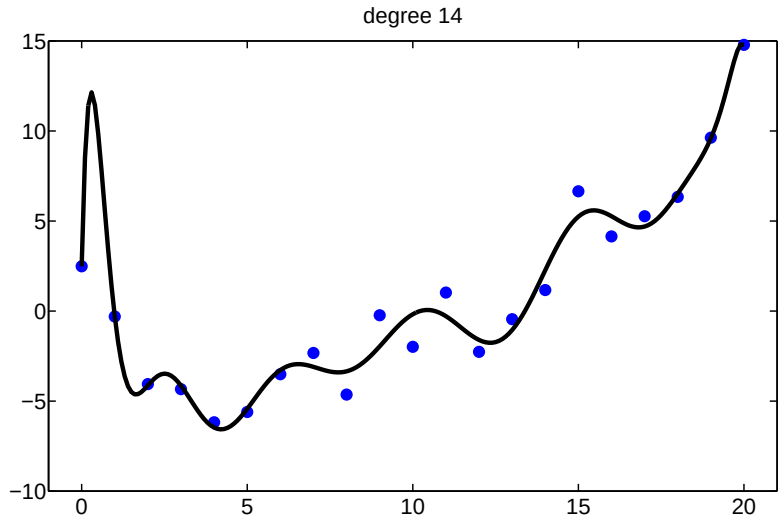
$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \boldsymbol{\theta}^T \mathbf{x}_i)^2 + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2,$$

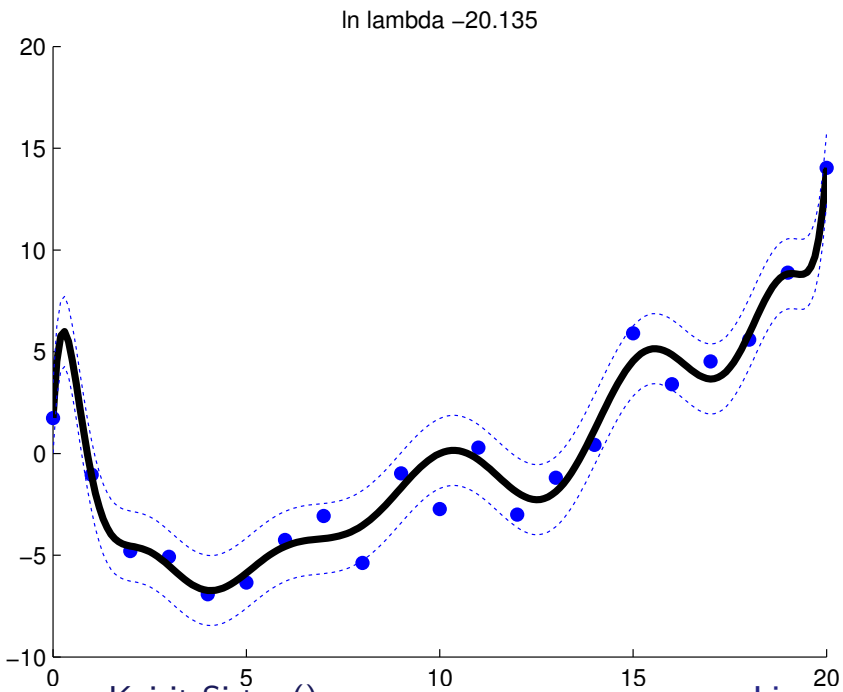
- where $\lambda = \sigma^2/\tau^2$.
- The regularized linear regression is called **ridge regression**.
- Ridge regression normal equation has the form:

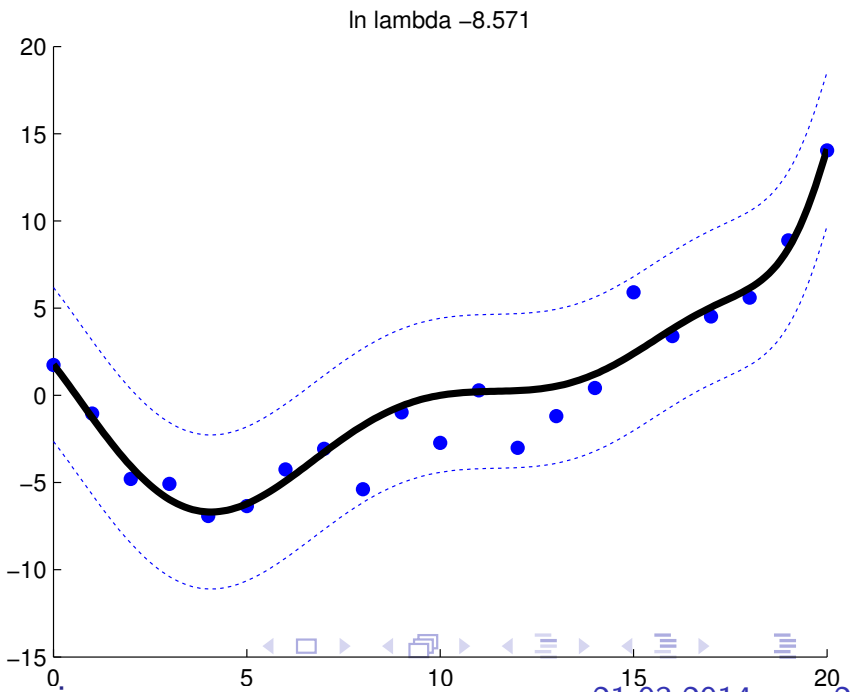
$$\boldsymbol{\theta}_{ridge} = (\lambda \mathbf{I} - \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T Y$$

- In general, adding Gaussian prior to parameters is called ℓ_2 regularization.

Effect of regularization

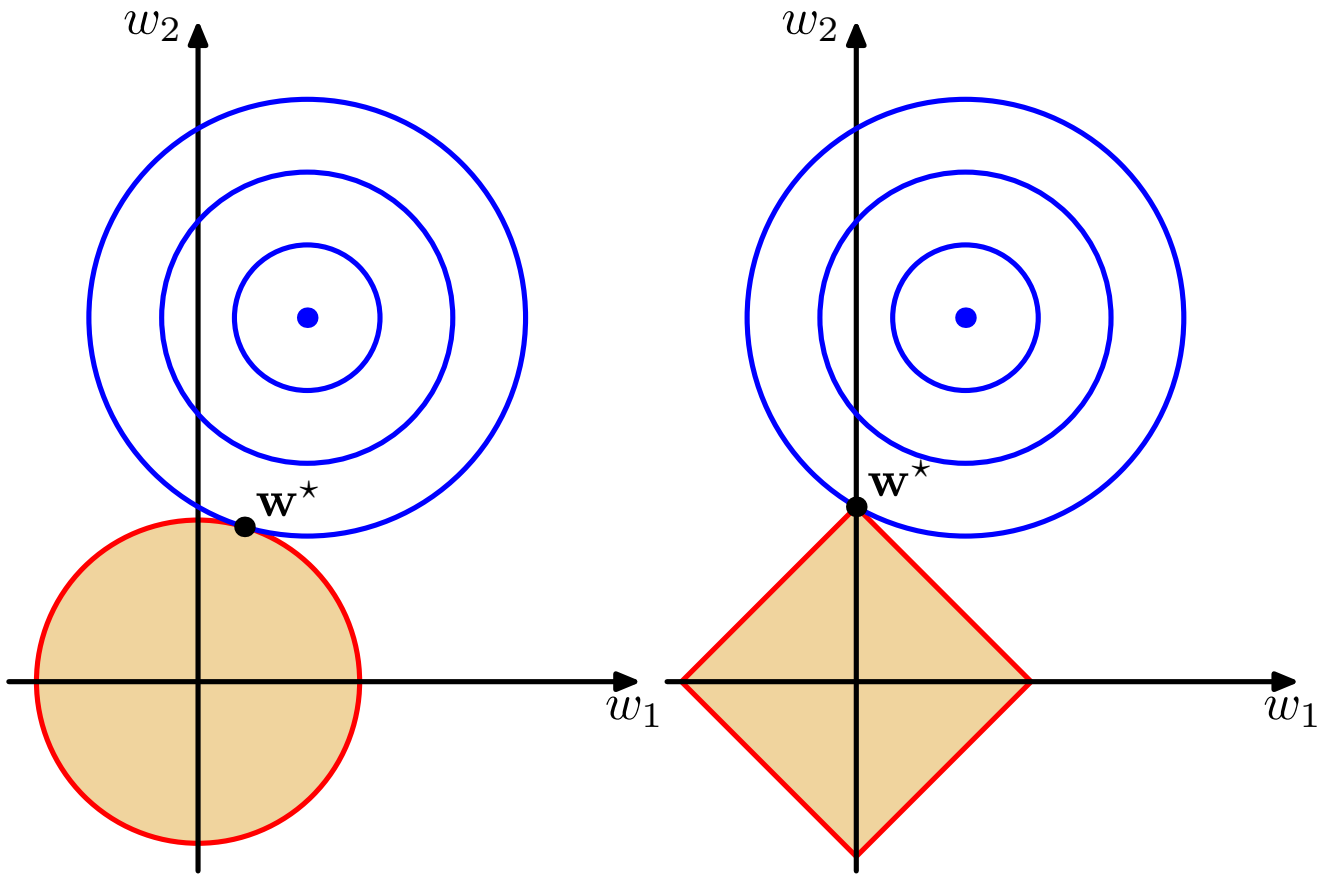






Other types of regularizations

- The regularization can also use other norms
- ℓ_1 regularization is called lasso and leads to sparse solutions.



Batch versus online learning

- Batch learning uses all the training data in each gradient descent iteration
- With large datasets this makes learning computationally very costly.
- An alternative is to use an **online** learning algorithm - **stochastic gradient descent**.
- With stochastic gradient update the parameters after observing every datapoint in turn.