universität wien

Introduction to Machine Learning

Linear Regression

Nils M. Kriege

WS 2023

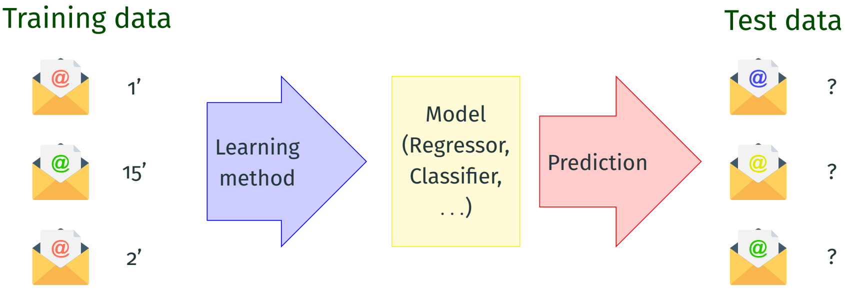
Data Mining and Machine Learning
Faculty of Computer Science
University of Vienna

Credit: Slides based on the IML Lectures by Sebastian Tschiatschek and Andreas Krause

Summary so far

- Two basic forms of learning:
  - Supervised vs. Unsupervised learning
- Key challenge in ML:
  - Trading goodness of fit and model complexity
- Representation of data is of key importance

Supervised Learning Pipeline



$\mathcal{X}$ $\mathcal{Y}$

$$f: \mathcal{X} \to \mathcal{Y}$$

Representation

$\boxed{\text{Model fitting}}$

Prediction and Generalization

Regression

- Instance of supervised learning
- **Goal**: Predict real valued labels (possibly vectors)
- Examples:

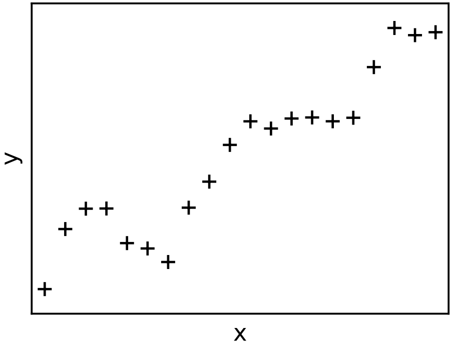
$\mathcal{X}$	$\mathcal{Y}$
Flight routes	delay (minutes)
Real estate objects	price
Patient & drug	treatment effectiveness
$\ldots$	

More concrete real-world example: Diabetes

[Efron et al '04]

- Features $\mathcal{X}$:
  - Age
  - Sex
  - Body mass index
  - Average blood pressure
  - Six blood serum measurements (S1-S6)
- Label (target) $\mathcal{Y}$:
  - Quantitative measure of disease progression

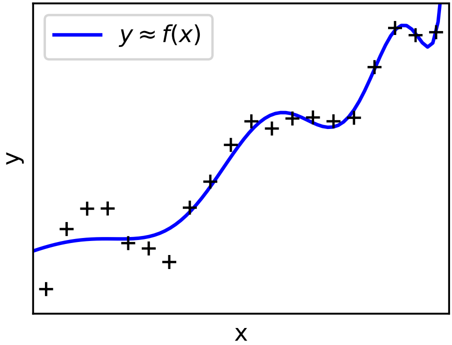
Regression



- *Goal*: learn real valued mapping $f : \mathbb{R}^d \to \mathbb{R}$

[regression/regression-goal.py]

Regression

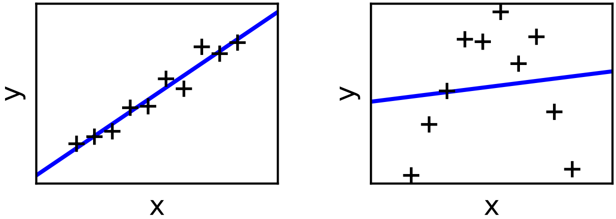


- *Goal*: learn real valued mapping $f : \mathbb{R}^d \to \mathbb{R}$

[regression/regression-goal.py]

Important choices in regression

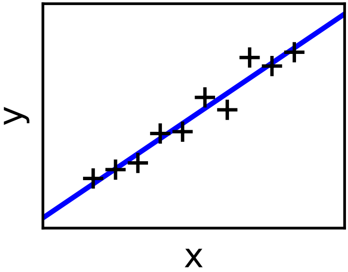
1. What types of functions f should we consider? Examples:



2. How should we measure goodness of fit?

[regression/regression-types-of-functions.py]

Example: Linear Regression



Goal: $y \approx f(x)$

Linear functions:

- 1-dim: $f(x) = ax + b$
- 2-dim: $f(x_1, x_2) = ax_1 + bx_2 + c$
- $\dots$
- d-dim: $f(\mathbf{x}) = w_1x_1 + w_2x_2 + \cdots + w_dx_d + w_0 = \mathbf{w}^T\mathbf{x} + w_0$

Homogeneous Representation

Goal: Simplify $\mathbf{w}^T\mathbf{x} + w_0$ by removing the term w_0

Idea: Augment input data $\mathbf{x}$ by adding a component $x_0 = 1$

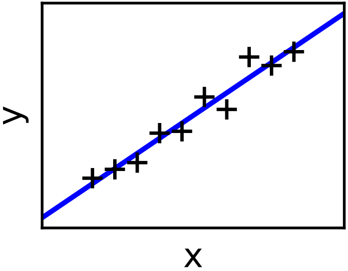
$$\mathbf{w}^T\mathbf{x} + w_0 = \widetilde{\mathbf{w}}^T\widetilde{\mathbf{x}}$$

$$\mathbf{x} = (x_1, \ldots, x_d)^T \qquad \widetilde{\mathbf{x}} = (1, x_1, \ldots, x_d)^T$$

$$\mathbf{w} = (w_1, \ldots, w_d)^T \qquad \widetilde{\mathbf{w}} = (w_0, w_1, \ldots, w_d)^T$$

Quantifying goodness of fit

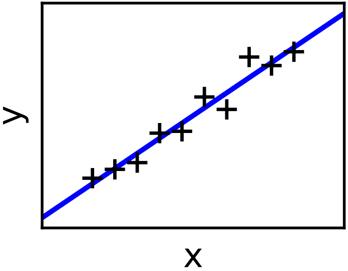
$$\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \qquad \mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}$$



Error at point *i*:
$r_i = y_i - \mathbf{w}^T \mathbf{x}_i$

Quantifying goodness of fit

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \qquad \mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}$$



Error at point i:
$r_i = y_i - \mathbf{w}^T\mathbf{x}_i$

- Measure squared error per data point: $(y_i - \mathbf{w}^T\mathbf{x}_i)^2$
- Sum of errors

$$\hat{R}(\mathbf{w}) = \sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

Least-squares linear regression optimization

[Legendre 1805, Gauss 1809]

- Given a dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \ldots$
- $\ldots$ how do we find the optimal weight vector?

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

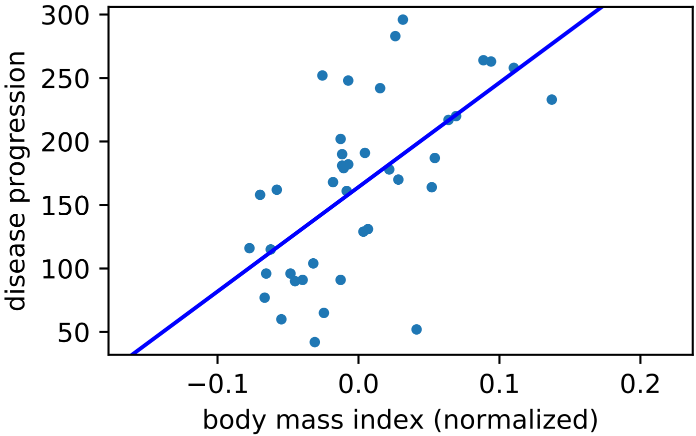
How to solve? Example: Scikit Learn

```
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training set
regr.fit(X_train, Y_train)

# Make predictions on the testing set
Y_pred = regr.predict(X_test)
```

Demo



[regression/diabetes.py]

Method 1: Closed form solution

- Setting: $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ $\qquad \mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}$
- The problem

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

can be solved in closed form:

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y},$$

where:

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix} = \begin{pmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,d} \\ x_{2,1} & x_{2,2} & \ldots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \ldots & x_{n,d} \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

Method 1: Closed form solution - Derivation

How to derive the closed form solution?

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

$$\begin{aligned}
\hat{R}(\mathbf{w}) &= \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \\
&= \sum_{i=1}^{n} (\mathbf{y} - \mathbf{X}\mathbf{w})_i^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) \\
&= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} \\
\nabla_{\mathbf{w}}\hat{R}(\mathbf{w}) &= 0 - 2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\mathbf{w} \\
\nabla_{\mathbf{w}}\hat{R}(\mathbf{w}) = 0 &\iff \mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y} \\
&\implies \hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}
\end{aligned}$$

Method 2: Optimization

- The objective function

$$\hat{R}(\mathbf{w}) = \sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

  is convex!
- A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex iff $\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^d, \lambda \in [0, 1]$, it holds that

$$f(\lambda\mathbf{x} + (1-\lambda)\mathbf{x}') \leq \lambda f(\mathbf{x}) + (1-\lambda)f(\mathbf{x}').$$

Gradient Descent

Gradient Descent

- Start at an arbitrary $\mathbf{w}_0 \in \mathbb{R}^d$
- For $t = 0, 1, 2, \ldots$, do
  - $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla \hat{R}(\mathbf{w_t})$

- Hereby, η_t is called learning rate or step rate

Convergence of gradient descent

- Under mild assumptions, if step size sufficiently small, gradient descent converges to a stationary point (gradient = 0)
- For convex objectives, it therefore finds the optimal solution!
- In the case of the squared loss, linear convergence for properly chosen constant stepsize.

Computing the gradient

$$\nabla \hat{R}(\mathbf{w}) = \left[\frac{\partial}{\partial w_1} \hat{R}(\mathbf{w}), \frac{\partial}{\partial w_2} \hat{R}(\mathbf{w}), \ldots, \frac{\partial}{\partial w_d} \hat{R}(\mathbf{w}) \right]$$

1-dim: $$\nabla \hat{R}(w) = \frac{\partial}{\partial w} \hat{R}(w) = \frac{\partial}{\partial w} \sum_{i=1}^{n} (y_i - wx_i)^2$$

$$= \sum_{i=1}^{n} \frac{\partial}{\partial w} (y_i - wx_i)^2 \qquad \text{e.g., via chain rule}$$

$$= \sum_{i=1}^{n} 2\underbrace{(y_i - wx_i)}_{=r_i}(-x_i) = -2 \sum_{i=1}^{n} r_i x_i$$

d-dim: $$\nabla \hat{R}(\mathbf{w}) = \cdots = \sum_{i=1}^{n} 2(y_i - \mathbf{w}^T \mathbf{x}_i)(-\mathbf{x}_i) = -2 \sum_{i=1}^{n} r_i \mathbf{x}_i$$

Demo: Gradient descent

[regression/GD.ipynb]

Choosing a stepsize

What happens if we choose a poor stepsize?

Adaptive step size

- Can update the step size adaptively. For example:
  - (a) Via line search (optimizing step size every step):

    Suppose that at iteration t, we have $\mathbf{w}_t$, $\mathbf{g}_t = \nabla \hat{R}(\mathbf{w}_t)$
    Define: $\eta_t^* = \arg\min_{\eta \in [0,\infty]} \hat{R}(\mathbf{w}_t - \eta \mathbf{g}_t)$

  - (b) "Bold driver" heuristic:
    - Initial learning rate η_0.
    - If function decreases, increase step size:

      If $\hat{R}(\mathbf{w}_{t+1}) < \hat{R}(\mathbf{w}_t)$ then $\eta_{t+1} = \eta_t c_+$ with $c_+ > 1$.

    - If function increases, decrease step size:

      If $\hat{R}(\mathbf{w}_{t+1}) > \hat{R}(\mathbf{w}_t)$ then $\eta_{t+1} = \eta_t c_-$ with $c_- < 1$.

Demo: Gradient Descent for Linear Regression

[regression/LR.ipynb]

Gradient Descent vs Closed Form

Why would one ever consider performing gradient descent, when it is possible to find closed form solution?

Closed form:

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

Gradient descent:

$$\nabla\hat{R}(\mathbf{w}) = \sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i$$

Gradient Descent vs Closed Form

Why would one ever consider performing gradient descent, when it is possible to find closed form solution?

Closed form:

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

Gradient descent:

$$\nabla\hat{R}(\mathbf{w}) = \sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i$$

- Computational complexity

Gradient Descent vs Closed Form

Why would one ever consider performing gradient descent, when it is possible to find closed form solution?

Closed form:

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

Gradient descent:

$$\nabla \hat{R}(\mathbf{w}) = \sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i$$

- Computational complexity
- May not need an optimal solution

Gradient Descent vs Closed Form

Why would one ever consider performing gradient descent, when it is possible to find closed form solution?

Closed form:

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

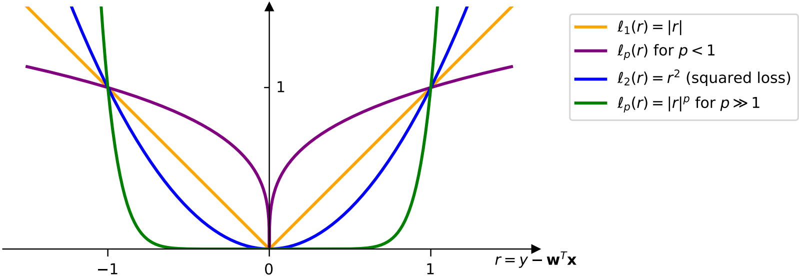
Gradient descent:

$$\nabla \hat{R}(\mathbf{w}) = \sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i$$

- Computational complexity
- May not need an optimal solution
- Many problems don't admit closed form solution

Other Loss Functions

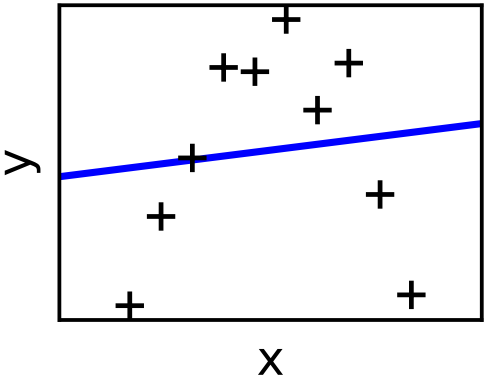
- So far: Measure goodness of fit via squared error
- Many other loss functions possible (and sensible!)



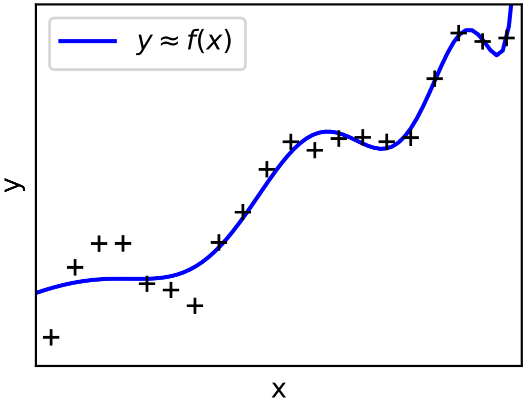
Supervised learning summary so far

Representation/ features	Linear hypotheses
Model/ objective	Loss-function (squared loss, ℓ_p loss)
Method	Exact solution, Gradient Descent
Evaluation metric	Empirical risk = (mean) squared error

Outlook: Fitting nonlinear functions



Outlook: Fitting nonlinear functions



Outlook: Linear regression for polynomials

We can fit non-linear functions via linear regression, using nonlinear features of our data (basis functions):

$$f(\mathbf{x}) = \sum_{i=1}^{D} w_i \phi_i(\mathbf{x})$$