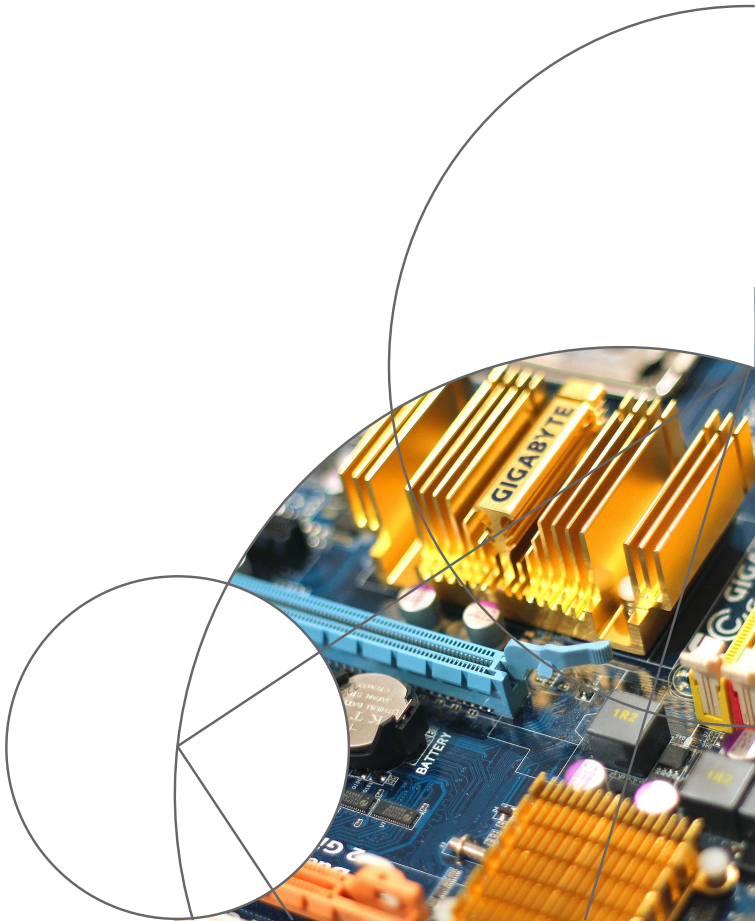
L4 – Large-Scale Least-Squares

Large-Scale Data Analysis

Fabian Gieseke

Image Group
Department of Computer Science
University of Copenhagen

Universitetsparken 1, Room 1-1-N110
fabian.gieseke@di.ku.dk

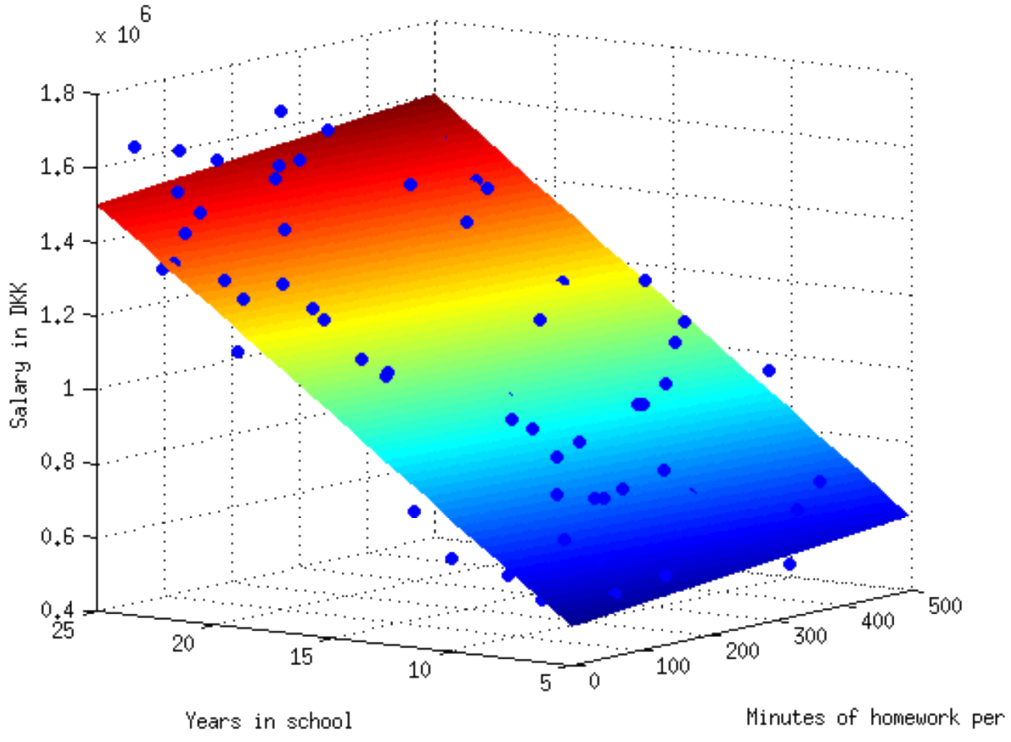


2 May 2019

Outline

Outline

Recap: Multivariate Linear Regression



General Form

- Given: Training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$.
- Goal: Linear model $f(\mathbf{z}; \mathbf{w}) = w_0 + w_1 z_1 + w_2 z_2 + \ldots + w_d z_d$ that is a good "fit".

Recap: Multivariate Linear Regression

- Given: Training set of the form $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$.
- Let's "augment" all patterns $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$. This yields an augmented data matrix $\mathbf{X} \in \mathbb{R}^{n \times (d+1)}$ and an associated target vector $\mathbf{y} \in \mathbb{R}^n$:

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \ldots & x_{1,d} \\ 1 & x_{2,1} & x_{2,2} & \ldots & x_{2,d} \\ \vdots & & & & \\ 1 & x_{n,1} & x_{n,2} & \ldots & x_{n,d} \end{bmatrix} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Recap: Multivariate Linear Regression

- Given: Training set of the form $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$.
- Let's "augment" all patterns $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$. This yields an augmented data matrix $\mathbf{X} \in \mathbb{R}^{n \times (d+1)}$ and an associated target vector $\mathbf{y} \in \mathbb{R}^n$:

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \ldots & x_{1,d} \\ 1 & x_{2,1} & x_{2,2} & \ldots & x_{2,d} \\ \vdots & & & & \\ 1 & x_{n,1} & x_{n,2} & \ldots & x_{n,d} \end{bmatrix} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

- Linear regression aims at minimizing the following loss:

Squared Loss

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i; \mathbf{w}) - y_i)^2 = \frac{1}{n} (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y})$$

Simplifying the Objective

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y})$$

$$\begin{aligned} \mathcal{L}(\mathbf{w}) &= \frac{1}{n}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) \end{aligned}$$

Simplifying the Objective

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y})$$

$$\begin{aligned}
\mathcal{L}(\mathbf{w}) &= \frac{1}{n}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) \\
&= \frac{1}{n}((\mathbf{X}\mathbf{w})^T - \mathbf{y}^T)(\mathbf{X}\mathbf{w} - \mathbf{y})
\end{aligned}$$

Simplifying the Objective

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y})$$

$$\begin{aligned}
\mathcal{L}(\mathbf{w}) &= \frac{1}{n}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) \\
&= \frac{1}{n}((\mathbf{X}\mathbf{w})^T - \mathbf{y}^T)(\mathbf{X}\mathbf{w} - \mathbf{y}) \\
&= \frac{1}{n}(\mathbf{X}\mathbf{w})^T\mathbf{X}\mathbf{w} - \frac{1}{n}\mathbf{y}^T\mathbf{X}\mathbf{w} - \frac{1}{n}(\mathbf{X}\mathbf{w})^T\mathbf{y} + \frac{1}{n}\mathbf{y}^T\mathbf{y}
\end{aligned}$$

Simplifying the Objective

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y})$$

$$\begin{aligned}
\mathcal{L}(\mathbf{w}) &= \frac{1}{n}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) \\
&= \frac{1}{n}((\mathbf{X}\mathbf{w})^T - \mathbf{y}^T)(\mathbf{X}\mathbf{w} - \mathbf{y}) \\
&= \frac{1}{n}(\mathbf{X}\mathbf{w})^T\mathbf{X}\mathbf{w} - \frac{1}{n}\mathbf{y}^T\mathbf{X}\mathbf{w} - \frac{1}{n}(\mathbf{X}\mathbf{w})^T\mathbf{y} + \frac{1}{n}\mathbf{y}^T\mathbf{y} \\
&= \frac{1}{n}\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - \frac{2}{n}\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \frac{1}{n}\mathbf{y}^T\mathbf{y}
\end{aligned}$$

Simplifying the Objective

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y})$$

$$\begin{aligned}
\mathcal{L}(\mathbf{w}) &= \frac{1}{n}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) \\
&= \frac{1}{n}((\mathbf{X}\mathbf{w})^T - \mathbf{y}^T)(\mathbf{X}\mathbf{w} - \mathbf{y}) \\
&= \frac{1}{n}(\mathbf{X}\mathbf{w})^T\mathbf{X}\mathbf{w} - \frac{1}{n}\mathbf{y}^T\mathbf{X}\mathbf{w} - \frac{1}{n}(\mathbf{X}\mathbf{w})^T\mathbf{y} + \frac{1}{n}\mathbf{y}^T\mathbf{y} \\
&= \frac{1}{n}\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - \frac{2}{n}\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \frac{1}{n}\mathbf{y}^T\mathbf{y}
\end{aligned}$$

with $\mathbf{y}^T\mathbf{X}\mathbf{w} = ((\mathbf{X}\mathbf{w})^T\mathbf{y})^T \in \mathbb{R}$

Gradient & Stationary Point

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n}\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - \frac{2}{n}\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \frac{1}{n}\mathbf{y}^T\mathbf{y}$$

Gradient & Stationary Point

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n}\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - \frac{2}{n}\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \frac{1}{n}\mathbf{y}^T\mathbf{y}$$

Toolbox (Table 1.4 in Rogers & Girolami)

1. $f(\mathbf{w}) = \mathbf{w}^T\mathbf{x} \Rightarrow \nabla f(\mathbf{w}) = \mathbf{x}$
2. $f(\mathbf{w}) = \mathbf{x}^T\mathbf{w} \Rightarrow \nabla f(\mathbf{w}) = \mathbf{x}$
3. $f(\mathbf{w}) = \mathbf{w}^T\mathbf{w} \Rightarrow \nabla f(\mathbf{w}) = 2\mathbf{w}$
4. $f(\mathbf{w}) = \mathbf{w}^T\mathbf{C}\mathbf{w} \Rightarrow \nabla f(\mathbf{w}) = 2\mathbf{C}\mathbf{w}$

Gradient & Stationary Point

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n}\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - \frac{2}{n}\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \frac{1}{n}\mathbf{y}^T\mathbf{y}$$

Toolbox (Table 1.4 in Rogers & Girolami)

1. $f(\mathbf{w}) = \mathbf{w}^T\mathbf{x} \Rightarrow \nabla f(\mathbf{w}) = \mathbf{x}$
2. $f(\mathbf{w}) = \mathbf{x}^T\mathbf{w} \Rightarrow \nabla f(\mathbf{w}) = \mathbf{x}$
3. $f(\mathbf{w}) = \mathbf{w}^T\mathbf{w} \Rightarrow \nabla f(\mathbf{w}) = 2\mathbf{w}$
4. $f(\mathbf{w}) = \mathbf{w}^T\mathbf{C}\mathbf{w} \Rightarrow \nabla f(\mathbf{w}) = 2\mathbf{C}\mathbf{w}$

We have $\nabla\mathcal{L}(\mathbf{w}) = \frac{2}{n}\mathbf{X}^T\mathbf{X}\mathbf{w} - \frac{2}{n}\mathbf{X}^T\mathbf{y}$ and therefore:

$$\nabla\mathcal{L}(\mathbf{w}) = \mathbf{0}$$

Gradient & Stationary Point

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n}\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - \frac{2}{n}\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \frac{1}{n}\mathbf{y}^T\mathbf{y}$$

Toolbox (Table 1.4 in Rogers & Girolami)

1. $f(\mathbf{w}) = \mathbf{w}^T\mathbf{x} \Rightarrow \nabla f(\mathbf{w}) = \mathbf{x}$
2. $f(\mathbf{w}) = \mathbf{x}^T\mathbf{w} \Rightarrow \nabla f(\mathbf{w}) = \mathbf{x}$
3. $f(\mathbf{w}) = \mathbf{w}^T\mathbf{w} \Rightarrow \nabla f(\mathbf{w}) = 2\mathbf{w}$
4. $f(\mathbf{w}) = \mathbf{w}^T\mathbf{C}\mathbf{w} \Rightarrow \nabla f(\mathbf{w}) = 2\mathbf{C}\mathbf{w}$

We have $\nabla\mathcal{L}(\mathbf{w}) = \frac{2}{n}\mathbf{X}^T\mathbf{X}\mathbf{w} - \frac{2}{n}\mathbf{X}^T\mathbf{y}$ and therefore:

$$\begin{aligned} & \nabla\mathcal{L}(\mathbf{w}) && = \mathbf{0} \\ \Leftrightarrow \quad & \tfrac{2}{n}\mathbf{X}^T\mathbf{X}\mathbf{w} - \tfrac{2}{n}\mathbf{X}^T\mathbf{y} && = \mathbf{0} \end{aligned}$$

Gradient & Stationary Point

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n}\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - \frac{2}{n}\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \frac{1}{n}\mathbf{y}^T\mathbf{y}$$

Toolbox (Table 1.4 in Rogers & Girolami)

1. $f(\mathbf{w}) = \mathbf{w}^T\mathbf{x} \Rightarrow \nabla f(\mathbf{w}) = \mathbf{x}$
2. $f(\mathbf{w}) = \mathbf{x}^T\mathbf{w} \Rightarrow \nabla f(\mathbf{w}) = \mathbf{x}$
3. $f(\mathbf{w}) = \mathbf{w}^T\mathbf{w} \Rightarrow \nabla f(\mathbf{w}) = 2\mathbf{w}$
4. $f(\mathbf{w}) = \mathbf{w}^T\mathbf{C}\mathbf{w} \Rightarrow \nabla f(\mathbf{w}) = 2\mathbf{C}\mathbf{w}$

We have $\nabla\mathcal{L}(\mathbf{w}) = \frac{2}{n}\mathbf{X}^T\mathbf{X}\mathbf{w} - \frac{2}{n}\mathbf{X}^T\mathbf{y}$ and therefore:

$$\begin{aligned} & \nabla\mathcal{L}(\mathbf{w}) && = \mathbf{0} \\ \Leftrightarrow \quad & \tfrac{2}{n}\mathbf{X}^T\mathbf{X}\mathbf{w} - \tfrac{2}{n}\mathbf{X}^T\mathbf{y} && = \mathbf{0} \\ \Leftrightarrow \quad & \mathbf{X}^T\mathbf{X}\mathbf{w} && = \mathbf{X}^T\mathbf{y} \end{aligned}$$

Recap: Multivariate Linear Regression

- Given: Training set of the form $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$.
- Goal: Find $(d+1)$-dimensional weight vector $\hat{\mathbf{w}} = [\hat{w}_0, \hat{w}_1, \ldots, \hat{w}_d]^T$ that minimizes $\mathcal{L}(\mathbf{w}) = \frac{1}{n}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y})$, i.e., which is a solution for

$$\begin{aligned} & \nabla \mathcal{L}(\mathbf{w}) && = \mathbf{0} \\ \Leftrightarrow \quad & \mathbf{X}^T\mathbf{X}\mathbf{w} && = \mathbf{X}^T\mathbf{y} \end{aligned} \tag{1}$$

Recap: Multivariate Linear Regression

- Given: Training set of the form $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$.
- Goal: Find $(d+1)$-dimensional weight vector $\hat{\mathbf{w}} = [\hat{w}_0, \hat{w}_1, \ldots, \hat{w}_d]^T$ that minimizes $\mathcal{L}(\mathbf{w}) = \frac{1}{n}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y})$, i.e., which is a solution for

$$\begin{aligned} & \nabla \mathcal{L}(\mathbf{w}) && = \mathbf{0} \\ \Leftrightarrow \quad & \mathbf{X}^T\mathbf{X}\mathbf{w} && = \mathbf{X}^T\mathbf{y} \end{aligned} \tag{1}$$

Computation

1. Definition of data matrix $\mathbf{X} \in \mathbb{R}^{n \times (d+1)}$
2. There are different ways to compute an optimal weight vector $\hat{\mathbf{w}}$:
   1. Compute $\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ (e.g., via `numpy.linalg.inv`)
   2. Directly solve system of equations (1) (e.g., via `numpy.linalg.solve`)
   3. ...
3. For new point $\mathbf{x}_{new} \in \mathbb{R}^d$: Compute $y_{new} = [1, \mathbf{x}_{new}^T]\hat{\mathbf{w}}$

Recap: Multivariate Linear Regression

- Given: Training set of the form $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$.
- Goal: Find $(d+1)$-dimensional weight vector $\hat{\mathbf{w}} = [\hat{w}_0, \hat{w}_1, \ldots, \hat{w}_d]^T$ that minimizes $\mathcal{L}(\mathbf{w}) = \frac{1}{n}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y})$, i.e., which is a solution for

$$\begin{aligned} & \nabla \mathcal{L}(\mathbf{w}) && = \mathbf{0} \\ \Leftrightarrow \quad & \mathbf{X}^T\mathbf{X}\mathbf{w} && = \mathbf{X}^T\mathbf{y} \end{aligned} \tag{1}$$

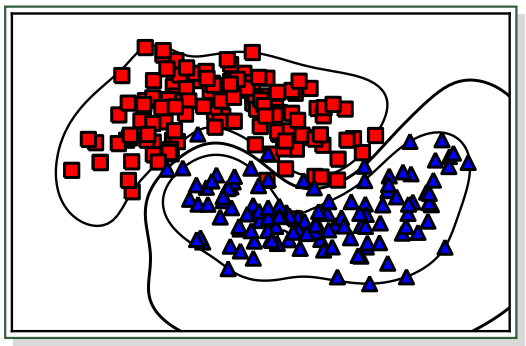
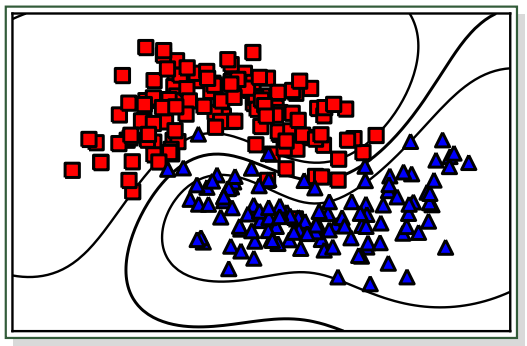
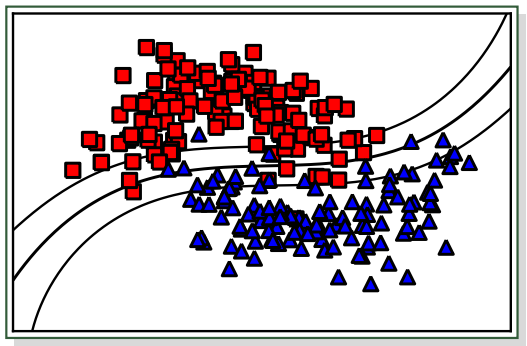
Computation

1. Definition of data matrix $\mathbf{X} \in \mathbb{R}^{n \times (d+1)}$
2. There are different ways to compute an optimal weight vector $\hat{\mathbf{w}}$:
   1. Compute $\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ (e.g., via `numpy.linalg.inv`)
   2. Directly solve system of equations (1) (e.g., via `numpy.linalg.solve`)
   3. ...
3. For new point $\mathbf{x}_{new} \in \mathbb{R}^d$: Compute $y_{new} = [1, \mathbf{x}_{new}^T]\hat{\mathbf{w}}$

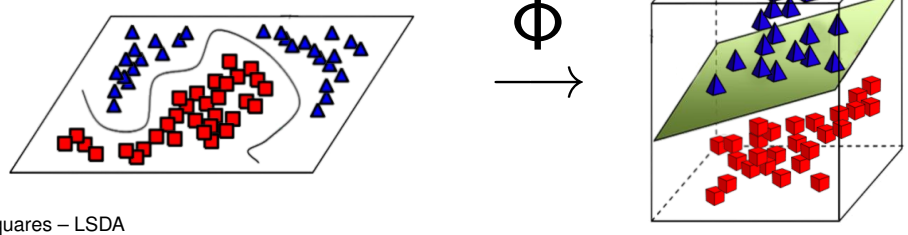
Question: What is the asymptotic runtime for approach (1)?

Outline

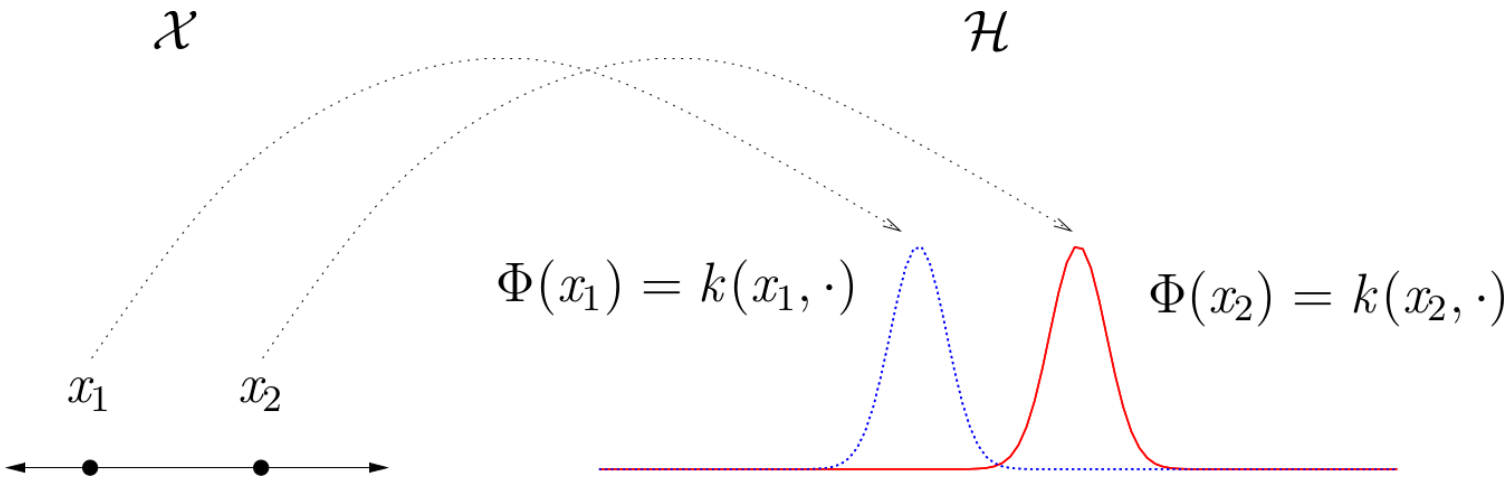
Recap: Kernel Methods

- Make use of kernel functions $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Examples:
  - Linear kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$.
  - Polynomial kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle^p$ for $p \in \mathbb{N}$.
  - RBF kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$ with $\gamma > 0$.
- The kernel k is called positive semidefinite in case the associated kernel matrix $\mathbf{K}$ is positive semidefinite. In this case, the kernel gives rise to a feature map $\Phi : \mathcal{X} \to \mathcal{H}$ into a Hilbert Space $\mathcal{H}$.

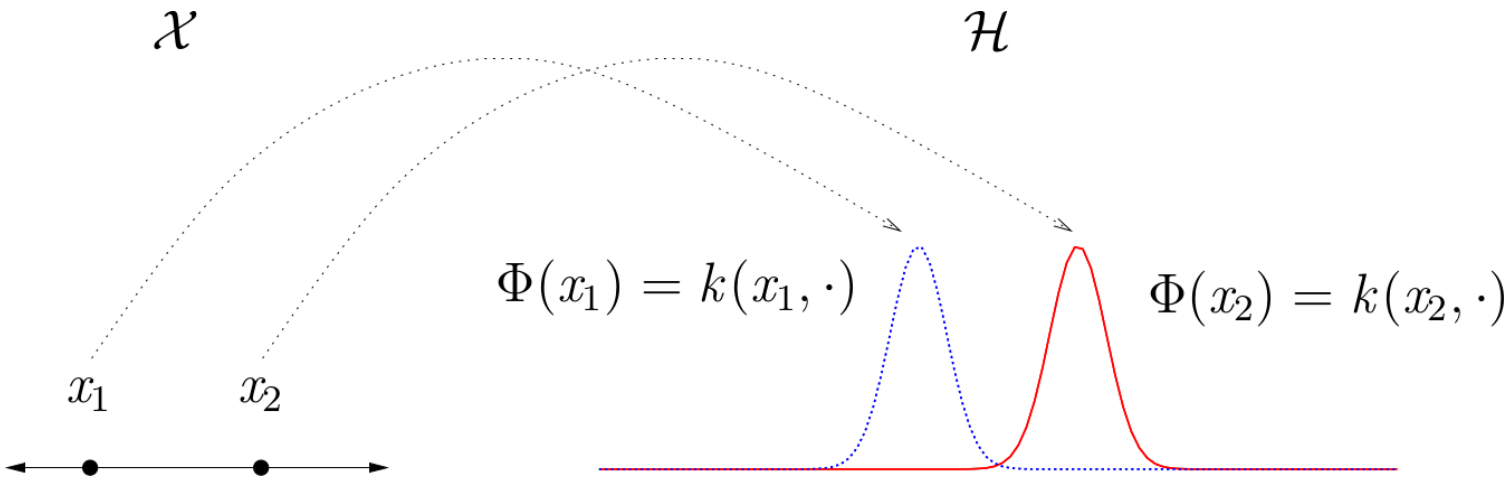


Recap: Kernel to Feature Map



1. Feature map: $\Phi : \mathcal{X} \to \mathbb{R}^{\mathcal{X}} := \{f : \mathcal{X} \to \mathbb{R}\}$ with $\Phi(\mathbf{x})(\cdot) = k(\cdot, \mathbf{x})$

Recap: Kernel to Feature Map

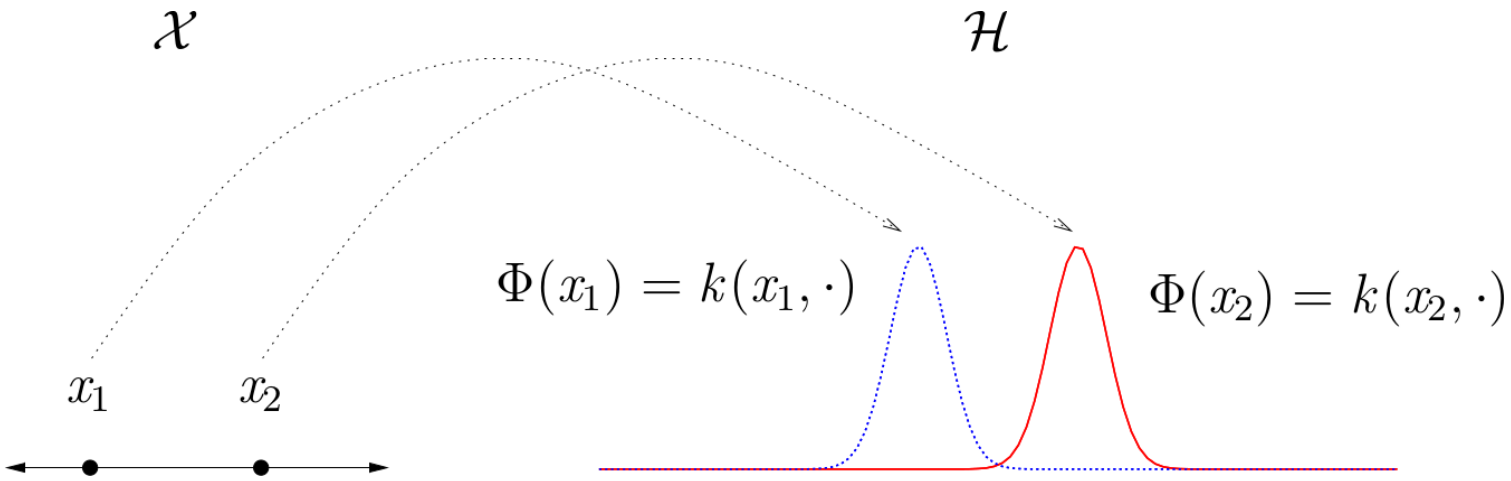


1. Feature map: $\Phi : \mathcal{X} \to \mathbb{R}^{\mathcal{X}} := \{f : \mathcal{X} \to \mathbb{R}\}$ with $\Phi(\mathbf{x})(\cdot) = k(\cdot, \mathbf{x})$
2. Vector space: $\text{span}\{k(\cdot, \mathbf{x}) \mid \mathbf{x} \in \mathcal{X}\}$ consisting of all functions of the form

$$f(\cdot) = \sum_{i=1}^{m} \alpha_i k(\cdot, \mathbf{x}_i)$$

for any $m \in \mathbb{N}$, $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{X}$, and $\alpha_1, \ldots, \alpha_m \in \mathbb{R}$.

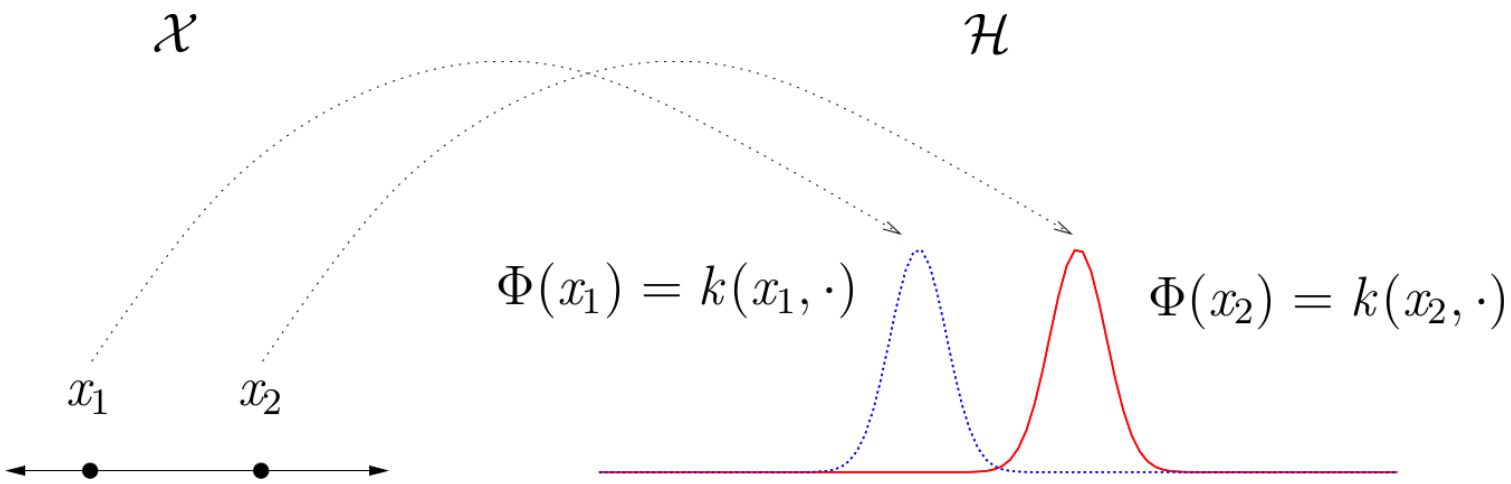
Recap: Kernel to Feature Map



3 Dot product: For $f(\cdot) = \sum_{i=1}^{m} \alpha_i k(\cdot, \mathbf{x}_i)$ and $g(\cdot) = \sum_{j=1}^{m'} \beta_j k(\cdot, \mathbf{x}'_j)$, let us define the following dot product:

$$\langle f, g \rangle := \sum_{i=1}^{m} \sum_{j=1}^{m'} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{x}'_j)$$

Recap: Kernel to Feature Map



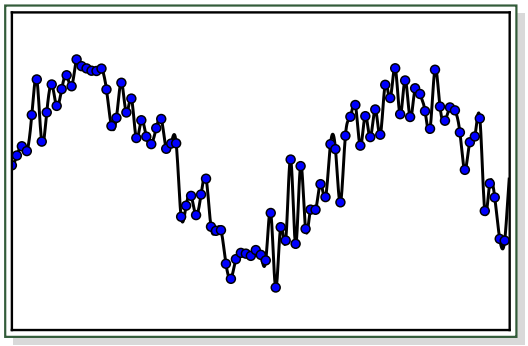
3 Dot product: For $f(\cdot) = \sum_{i=1}^{m} \alpha_i k(\cdot, \mathbf{x}_i)$ and $g(\cdot) = \sum_{j=1}^{m'} \beta_j k(\cdot, \mathbf{x}'_j)$, let us define the following dot product:

$$\langle f, g \rangle := \sum_{i=1}^{m} \sum_{j=1}^{m'} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{x}'_j)$$

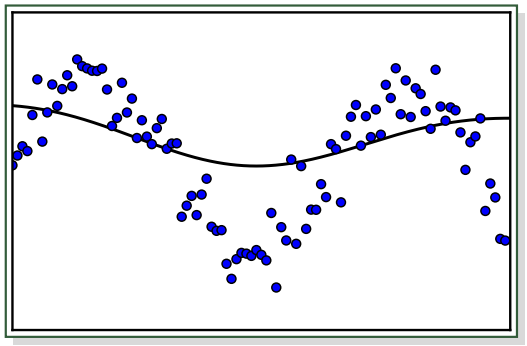
4 Reproducing kernel property: We have

$$\left\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \right\rangle = \left\langle k(\cdot, \mathbf{x}), k(\cdot, \mathbf{x}') \right\rangle = k(\mathbf{x}, \mathbf{x}')$$

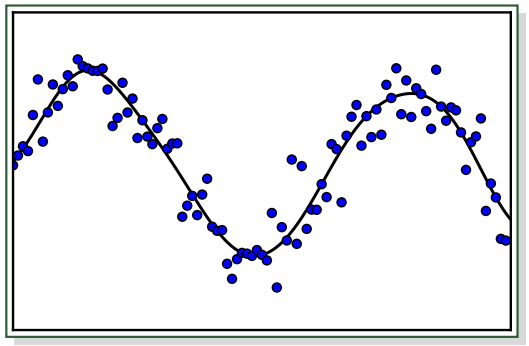
Recap: Regularization Networks



$\lambda = \text{small}$



$\lambda = \text{large}$



$\lambda = \text{middle}$

Regularized Least-Squares

Let $\mathcal{X} = \mathbb{R}^d$, $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$, and $\lambda > 0$. Goal: Find a model $f \in \mathcal{H}$ in the hypothesis space $\mathcal{H}$ that minimizes

$$\underset{f \in \mathcal{H}}{\text{minimize}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2}_{\text{Small Loss}} + \underbrace{\lambda \|f\|^2}_{\text{Small Complexity}} \tag{2}$$

Here: Models of the form $f(\cdot) = \sum_{j=1}^{m} \alpha_j k(\cdot, \mathbf{z}_j)$ with arbitrary m, arbitrary $\alpha \in \mathbb{R}^m$, and arbitrary $\mathbf{z}_1, \ldots, \mathbf{z}_m \in \mathcal{X}$. The norm $\|f\|^2$ measures the complexity of a model.

Optimal Solution

Regularized Least-Squares

Let $\mathcal{X} = \mathbb{R}^d$, $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$, and $\lambda > 0$. Goal: Find a model $f \in \mathcal{H}$ in the hypothesis space $\mathcal{H}$ that minimizes

$$\underset{f \in \mathcal{H}}{\text{minimize}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2}_{\text{Small Loss}} + \underbrace{\lambda \|f\|^2}_{\text{Small Complexity}}$$

Here: Models of the form $f(\cdot) = \sum_{j=1}^{m} \alpha_j k(\cdot, \mathbf{z}_j)$ with arbitrary m, arbitrary $\alpha \in \mathbb{R}^m$, and arbitrary $\mathbf{z}_1, \ldots, \mathbf{z}_m \in \mathcal{X}$. The norm $\|f\|^2$ measures the complexity of a model.

Optimal Solution

Regularized Least-Squares

Let $\mathcal{X} = \mathbb{R}^d$, $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$, and $\lambda > 0$. Goal: Find a model $f \in \mathcal{H}$ in the hypothesis space $\mathcal{H}$ that minimizes

$$\underset{f \in \mathcal{H}}{\text{minimize}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2}_{\text{Small Loss}} + \underbrace{\lambda \|f\|^2}_{\text{Small Complexity}}$$

Here: Models of the form $f(\cdot) = \sum_{j=1}^{m} \alpha_j k(\cdot, \mathbf{z}_j)$ with arbitrary m, arbitrary $\alpha \in \mathbb{R}^m$, and arbitrary $\mathbf{z}_1, \ldots, \mathbf{z}_m \in \mathcal{X}$. The norm $\|f\|^2$ measures the complexity of a model.

The representer theorem guarantees that the solution for this problem can be written as

$$f(\cdot) = \sum_{i=1}^{n} c_i k(\cdot, \mathbf{x}_i)$$

for some $\mathbf{c} \in \mathbb{R}^n$. Thus: We only need the training patterns $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to represent an optimal solution!

Schölkopf, Herbrich, and Smola.
A Generalized Representer Theorem. COLT, 2001.

Optimal Solution

The complexity term $\|f\|^2$ can be rewritten in the following way:

$$\begin{aligned}
\|f\|^2 &= \langle f, f \rangle \\
&= \left\langle \sum_{i=1}^{n} c_i k(\cdot, x_i), \sum_{i=1}^{n} c_i k(\cdot, x_i) \right\rangle \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) \\
&= \mathbf{c}^T \mathbf{K} \mathbf{c}
\end{aligned}$$

Optimal Solution

- We can therefore rewrite (2) in the following way:

$$\underset{\mathbf{c}\in\mathbb{R}^n}{\text{minimize}} \quad \frac{1}{n}(\mathbf{y}-\mathbf{K}\mathbf{c})^T(\mathbf{y}-\mathbf{K}\mathbf{c})+\lambda\mathbf{c}^t\mathbf{K}\mathbf{c}$$

Optimal Solution

- We can therefore rewrite (2) in the following way:

$$\underset{\mathbf{c}\in\mathbb{R}^n}{\text{minimize}} \quad \frac{1}{n}(\mathbf{y}-\mathbf{K}\mathbf{c})^T(\mathbf{y}-\mathbf{K}\mathbf{c})+\lambda\mathbf{c}^t\mathbf{K}\mathbf{c}$$

- The gradient of $F(\mathbf{c}) = \frac{1}{n}(\mathbf{y}-\mathbf{K}\mathbf{c})^T(\mathbf{y}-\mathbf{K}\mathbf{c})+\lambda\mathbf{c}^t\mathbf{K}\mathbf{c}$ is given by

$$\nabla F(\mathbf{c}) = -\frac{2}{n}\mathbf{K}^T(\mathbf{y}-\mathbf{K}\mathbf{c})+2\lambda\mathbf{K}\mathbf{c}$$

  and its Hessian by

$$\nabla^2 F(\mathbf{c}) = \frac{2}{n}\mathbf{K}^T\mathbf{K}+2\lambda\mathbf{K}$$

Optimal Solution

- We can therefore rewrite (2) in the following way:

$$\underset{\mathbf{c}\in\mathbb{R}^n}{\text{minimize}} \quad \frac{1}{n}(\mathbf{y}-\mathbf{K}\mathbf{c})^T(\mathbf{y}-\mathbf{K}\mathbf{c})+\lambda\mathbf{c}^t\mathbf{K}\mathbf{c}$$

- The gradient of $F(\mathbf{c}) = \frac{1}{n}(\mathbf{y}-\mathbf{K}\mathbf{c})^T(\mathbf{y}-\mathbf{K}\mathbf{c})+\lambda\mathbf{c}^t\mathbf{K}\mathbf{c}$ is given by

$$\nabla F(\mathbf{c}) = -\frac{2}{n}\mathbf{K}^T(\mathbf{y}-\mathbf{K}\mathbf{c})+2\lambda\mathbf{K}\mathbf{c}$$

  and its Hessian by

$$\nabla^2 F(\mathbf{c}) = \frac{2}{n}\mathbf{K}^T\mathbf{K}+2\lambda\mathbf{K}$$

- $\nabla^2 F(\mathbf{c})$ is positive semidefinite. Why?

Optimal Solution

- Thus, F is convex in $\mathbf{c}$. We can find a global minimum by setting the gradient to $\mathbf{0}$:

$$\begin{aligned} & \nabla F(\mathbf{c}) && = \mathbf{0} \\ \Leftrightarrow \quad & -\tfrac{2}{n}\mathbf{K}^T(\mathbf{y}-\mathbf{K}\mathbf{c}) + 2\lambda\mathbf{K}\mathbf{c} && = \mathbf{0} \\ \Leftrightarrow \quad & (\mathbf{K}^T\mathbf{K} + n\lambda\mathbf{K})\mathbf{c} && = \mathbf{K}^T\mathbf{y} \\ \Leftrightarrow \quad & (\mathbf{K} + n\lambda\mathbf{I})\mathbf{c} && = \mathbf{y} \end{aligned}$$

For the last step, we assumed that $\mathbf{K}$ is invertible. If $\mathbf{K}$ is not invertible, then the following solution still depicts one of the optimal solutions.

Optimal Solution

- Thus, F is convex in $\mathbf{c}$. We can find a global minimum by setting the gradient to $\mathbf{0}$:

$$\begin{aligned} & \nabla F(\mathbf{c}) && = \mathbf{0} \\ \Leftrightarrow \quad & -\tfrac{2}{n}\mathbf{K}^T(\mathbf{y}-\mathbf{K}\mathbf{c})+2\lambda\mathbf{K}\mathbf{c} && = \mathbf{0} \\ \Leftrightarrow \quad & (\mathbf{K}^T\mathbf{K}+n\lambda\mathbf{K})\mathbf{c} && = \mathbf{K}^T\mathbf{y} \\ \Leftrightarrow \quad & (\mathbf{K}+n\lambda\mathbf{I})\mathbf{c} && = \mathbf{y} \end{aligned}$$

  For the last step, we assumed that $\mathbf{K}$ is invertible. If $\mathbf{K}$ is not invertible, then the following solution still depicts one of the optimal solutions.

- Thus, an optimal solution can be found by computing

$$\mathbf{c}^* = (\mathbf{K}+n\lambda\mathbf{I})^{-1}\mathbf{y}$$

Runtime and Space Analysis

Regularized Least-Squares

Let $\mathcal{X} = \mathbb{R}^d$, $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$, and $\lambda > 0$. Goal: Find a model $f \in \mathcal{H}$ in the hypothesis space $\mathcal{H}$ that minimizes

$$\underset{f \in \mathcal{H}}{\text{minimize}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2}_{\text{Small Loss}} + \underbrace{\lambda \|f\|^2}_{\text{Small Complexity}}$$

Here: Models of the form $f(\cdot) = \sum_{j=1}^{m} \alpha_j k(\cdot, \mathbf{z}_j)$ with arbitrary m, arbitrary $\alpha \in \mathbb{R}^m$, and arbitrary $\mathbf{z}_1, \ldots, \mathbf{z}_m \in \mathcal{X}$. The norm $\|f\|^2$ measures the complexity of a model.

Runtime and Space Analysis

Regularized Least-Squares

Let $\mathcal{X} = \mathbb{R}^d$, $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$, and $\lambda > 0$. Goal: Find a model $f \in \mathcal{H}$ in the hypothesis space $\mathcal{H}$ that minimizes

$$\underset{f \in \mathcal{H}}{\text{minimize}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2}_{\text{Small Loss}} + \underbrace{\lambda \|f\|^2}_{\text{Small Complexity}}$$

Here: Models of the form $f(\cdot) = \sum_{j=1}^{m} \alpha_j k(\cdot, \mathbf{z}_j)$ with arbitrary m, arbitrary $\alpha \in \mathbb{R}^m$, and arbitrary $\mathbf{z}_1, \ldots, \mathbf{z}_m \in \mathcal{X}$. The norm $\|f\|^2$ measures the complexity of a model.

- The optimal model is given by $f(\cdot) = \sum_{i=1}^{n} c_i^* k(\cdot, \mathbf{x}_i)$ with $\mathbf{c}^* = (\mathbf{K} + n\lambda \mathbf{I})^{-1} \mathbf{y}$
- What is the runtime and space complexity for this approach?

Runtime and Space Analysis

Regularized Least-Squares

Let $\mathcal{X} = \mathbb{R}^d$, $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$, and $\lambda > 0$. Goal: Find a model $f \in \mathcal{H}$ in the hypothesis space $\mathcal{H}$ that minimizes

$$\underset{f \in \mathcal{H}}{\text{minimize}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2}_{\text{Small Loss}} + \underbrace{\lambda \|f\|^2}_{\text{Small Complexity}}$$

Here: Models of the form $f(\cdot) = \sum_{j=1}^{m} \alpha_j k(\cdot, \mathbf{z}_j)$ with arbitrary m, arbitrary $\alpha \in \mathbb{R}^m$, and arbitrary $\mathbf{z}_1, \ldots, \mathbf{z}_m \in \mathcal{X}$. The norm $\|f\|^2$ measures the complexity of a model.

- The optimal model is given by $f(\cdot) = \sum_{i=1}^{n} c_i^* k(\cdot, \mathbf{x}_i)$ with $\mathbf{c}^* = (\mathbf{K} + n\lambda \mathbf{I})^{-1} \mathbf{y}$
- What is the runtime and space complexity for this approach?
  - Space: $O(n^2)$ space for $\mathbf{K} + n\lambda \mathbf{I}$
  - Runtime: $O(n^3)$ for computing the inverse of this matrix (plus runtime needed for the computation of the kernel matrix).
    Note that solving such a system of linear equations generally takes $O(n^3)$ time.

Trick I: Linear Kernel

- For a linear kernel, we $\mathbf{K} = \mathbf{X}\mathbf{X}^T$ with $\mathbf{X} \in \mathbb{R}^{n \times d}$ containing the patterns as rows. We can therefore rewrite the task:

$$\begin{aligned}
&\underset{\mathbf{c} \in \mathbb{R}^n}{\text{minimize}} && \frac{1}{n}(\mathbf{y} - \mathbf{K}\mathbf{c})^T(\mathbf{y} - \mathbf{K}\mathbf{c}) + \lambda \mathbf{c}^t \mathbf{K}\mathbf{c} \\
&\underset{\mathbf{c} \in \mathbb{R}^n}{\text{minimize}} && \frac{1}{n}\left(\mathbf{y} - \mathbf{X}\mathbf{X}^T\mathbf{c}\right)^T\left(\mathbf{y} - \mathbf{X}\mathbf{X}^T\mathbf{c}\right) + \lambda \mathbf{c}^T \mathbf{X}\mathbf{X}^T\mathbf{c} \\
&\underset{\mathbf{w} \in \mathbb{R}^d}{\text{minimize}} && \frac{1}{n}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w}
\end{aligned}$$

Trick I: Linear Kernel

- For a linear kernel, we $\mathbf{K} = \mathbf{X}\mathbf{X}^T$ with $\mathbf{X} \in \mathbb{R}^{n \times d}$ containing the patterns as rows. We can therefore rewrite the task:

$$\begin{aligned}
\underset{\mathbf{c} \in \mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{n}(\mathbf{y} - \mathbf{K}\mathbf{c})^T(\mathbf{y} - \mathbf{K}\mathbf{c}) + \lambda \mathbf{c}^t \mathbf{K}\mathbf{c} \\
\underset{\mathbf{c} \in \mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{n}\left(\mathbf{y} - \mathbf{X}\mathbf{X}^T\mathbf{c}\right)^T\left(\mathbf{y} - \mathbf{X}\mathbf{X}^T\mathbf{c}\right) + \lambda \mathbf{c}^T \mathbf{X}\mathbf{X}^T\mathbf{c} \\
\underset{\mathbf{w} \in \mathbb{R}^d}{\text{minimize}} \quad & \frac{1}{n}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w}
\end{aligned}$$

- The gradient w.r.t. $\mathbf{w}$ is given by (almost as before):

$$\nabla G(\mathbf{w}) = \frac{2}{n}\mathbf{X}^T\mathbf{X}\mathbf{w} - \frac{2}{n}\mathbf{X}^T\mathbf{y} + 2\lambda\mathbf{w}$$

Trick I: Linear Kernel

- For a linear kernel, we $\mathbf{K} = \mathbf{X}\mathbf{X}^T$ with $\mathbf{X} \in \mathbb{R}^{n \times d}$ containing the patterns as rows. We can therefore rewrite the task:

$$\begin{aligned}
\underset{\mathbf{c} \in \mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{n}(\mathbf{y} - \mathbf{K}\mathbf{c})^T(\mathbf{y} - \mathbf{K}\mathbf{c}) + \lambda \mathbf{c}^t \mathbf{K}\mathbf{c} \\
\underset{\mathbf{c} \in \mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{n}\left(\mathbf{y} - \mathbf{X}\mathbf{X}^T\mathbf{c}\right)^T\left(\mathbf{y} - \mathbf{X}\mathbf{X}^T\mathbf{c}\right) + \lambda \mathbf{c}^T \mathbf{X}\mathbf{X}^T\mathbf{c} \\
\underset{\mathbf{w} \in \mathbb{R}^d}{\text{minimize}} \quad & \frac{1}{n}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w}
\end{aligned}$$

- The gradient w.r.t. $\mathbf{w}$ is given by (almost as before):

$$\nabla G(\mathbf{w}) = \frac{2}{n}\mathbf{X}^T\mathbf{X}\mathbf{w} - \frac{2}{n}\mathbf{X}^T\mathbf{y} + 2\lambda\mathbf{w}$$

- Enforcing $\nabla G(\mathbf{w}) ! = \mathbf{0}$ yields: $\mathbf{w}^* = (\mathbf{X}^T\mathbf{X} + n\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$

Trick I: Linear Kernel

- For a linear kernel, we $\mathbf{K} = \mathbf{X}\mathbf{X}^T$ with $\mathbf{X} \in \mathbb{R}^{n \times d}$ containing the patterns as rows. We can therefore rewrite the task:

$$\begin{aligned}
\underset{\mathbf{c} \in \mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{n}(\mathbf{y} - \mathbf{K}\mathbf{c})^T(\mathbf{y} - \mathbf{K}\mathbf{c}) + \lambda \mathbf{c}^t \mathbf{K}\mathbf{c} \\
\underset{\mathbf{c} \in \mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{n}\left(\mathbf{y} - \mathbf{X}\mathbf{X}^T\mathbf{c}\right)^T\left(\mathbf{y} - \mathbf{X}\mathbf{X}^T\mathbf{c}\right) + \lambda \mathbf{c}^T \mathbf{X}\mathbf{X}^T\mathbf{c} \\
\underset{\mathbf{w} \in \mathbb{R}^d}{\text{minimize}} \quad & \frac{1}{n}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w}
\end{aligned}$$

- The gradient w.r.t. $\mathbf{w}$ is given by (almost as before):

$$\nabla G(\mathbf{w}) = \frac{2}{n}\mathbf{X}^T\mathbf{X}\mathbf{w} - \frac{2}{n}\mathbf{X}^T\mathbf{y} + 2\lambda\mathbf{w}$$

- Enforcing $\nabla G(\mathbf{w}) ! = \mathbf{0}$ yields: $\mathbf{w}^* = (\mathbf{X}^T\mathbf{X} + n\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$

Note: No $n \times n$ kernel matrix is computed! An offset parameter can be added by prepending a column of ones to $\mathbf{X}$, see before!

Trick II: Fast Search for λ

- Problem: Time-consuming grid-search for λ!

Trick II: Fast Search for λ

- Problem: Time-consuming grid-search for λ!
- Compute the eigendecomposition $\mathbf{K} = \mathbf{Q}\Lambda\mathbf{Q}^T$ of the (real-valued) kernel matrix $\mathbf{K}$, where $\mathbf{Q}$ is an orthogonal matrix and Λ a diagonal matrix containing the eigenvalues. The computation takes $O(n^3)$ time.

Trick II: Fast Search for λ

- Problem: Time-consuming grid-search for λ!
- Compute the eigendecomposition $\mathbf{K} = \mathbf{Q}\Lambda\mathbf{Q}^T$ of the (real-valued) kernel matrix $\mathbf{K}$, where $\mathbf{Q}$ is an orthogonal matrix and Λ a diagonal matrix containing the eigenvalues. The computation takes $O(n^3)$ time.
- We can then rewrite the optimal solution in the following way:

$$\mathbf{c}^* = (\mathbf{Q}\Lambda\mathbf{Q}^T + n\lambda\mathbf{Q}\mathbf{Q}^T)^{-1}\mathbf{y} = (\mathbf{Q}(\Lambda + n\lambda\mathbf{I})\mathbf{Q}^T)^{-1}\mathbf{y} = \mathbf{Q}(\Lambda + n\lambda\mathbf{I})^{-1}\mathbf{Q}^T\mathbf{y}$$

Trick II: Fast Search for λ

- Problem: Time-consuming grid-search for λ!
- Compute the eigendecomposition $\mathbf{K} = \mathbf{Q}\Lambda\mathbf{Q}^T$ of the (real-valued) kernel matrix $\mathbf{K}$, where $\mathbf{Q}$ is an orthogonal matrix and Λ a diagonal matrix containing the eigenvalues. The computation takes $\mathcal{O}(n^3)$ time.
- We can then rewrite the optimal solution in the following way:

$$\mathbf{c}^* = (\mathbf{Q}\Lambda\mathbf{Q}^T + n\lambda\mathbf{Q}\mathbf{Q}^T)^{-1}\mathbf{y} = (\mathbf{Q}(\Lambda + n\lambda\mathbf{I})\mathbf{Q}^T)^{-1}\mathbf{y} = \mathbf{Q}(\Lambda + n\lambda\mathbf{I})^{-1}\mathbf{Q}^T\mathbf{y}$$

- Fast search: Fix other parameters (e.g., kernel width γ). For each λ do:
  1. Compute $\mathbf{B}_1 = \mathbf{Q}^T\mathbf{y} \in \mathbb{R}^{n\times 1}$ in $\mathcal{O}(n^2)$ time. (matrix-vector product)
  2. Compute $\mathbf{B}_2 = (\Lambda + n\lambda\mathbf{I})^{-1}$ in $\mathcal{O}(n)$ time.
     (it's a simple diagonal matrix containing only strictly positive real values!)
  3. Compute $\mathbf{B}_3 = \mathbf{B}_2\mathbf{B}_1$ in $\mathcal{O}(n)$ time. (diagonal matrix-vector product)
  4. Compute $\mathbf{c}^* = \mathbf{Q}\mathbf{B}_3$ in $\mathcal{O}(n^2)$ time. (matrix-vector product)

Trick II: Fast Search for λ

- Problem: Time-consuming grid-search for λ!
- Compute the eigendecomposition $\mathbf{K} = \mathbf{Q}\Lambda\mathbf{Q}^T$ of the (real-valued) kernel matrix $\mathbf{K}$, where $\mathbf{Q}$ is an orthogonal matrix and Λ a diagonal matrix containing the eigenvalues. The computation takes $\mathcal{O}(n^3)$ time.
- We can then rewrite the optimal solution in the following way:

$$\mathbf{c}^* = (\mathbf{Q}\Lambda\mathbf{Q}^T + n\lambda\mathbf{Q}\mathbf{Q}^T)^{-1}\mathbf{y} = (\mathbf{Q}(\Lambda + n\lambda\mathbf{I})\mathbf{Q}^T)^{-1}\mathbf{y} = \mathbf{Q}(\Lambda + n\lambda\mathbf{I})^{-1}\mathbf{Q}^T\mathbf{y}$$

- Fast search: Fix other parameters (e.g., kernel width γ). For each λ do:
  1. Compute $\mathbf{B}_1 = \mathbf{Q}^T\mathbf{y} \in \mathbb{R}^{n\times 1}$ in $\mathcal{O}(n^2)$ time. (matrix-vector product)
  2. Compute $\mathbf{B}_2 = (\Lambda + n\lambda\mathbf{I})^{-1}$ in $\mathcal{O}(n)$ time.
     (it's a simple diagonal matrix containing only strictly positive real values!)
  3. Compute $\mathbf{B}_3 = \mathbf{B}_2\mathbf{B}_1$ in $\mathcal{O}(n)$ time. (diagonal matrix-vector product)
  4. Compute $\mathbf{c}^* = \mathbf{Q}\mathbf{B}_3$ in $\mathcal{O}(n^2)$ time. (matrix-vector product)
- Hence: Instead of spending $\mathcal{O}(n^3)$ for each λ, we only need to spend $\mathcal{O}(n^3)$ once and can then test each λ in only $\mathcal{O}(n^2)$ time!

Trick III: Large-Scale Non-Linear Models

- Problem: For large n, the cubic runtime and the quadratic space requirements become a bottleneck!

Trick III: Large-Scale Non-Linear Models

- Problem: For large n, the cubic runtime and the quadratic space requirements become a bottleneck!
- Any ideas how we can still compute such models for non-linear kernels?

Trick III: Large-Scale Non-Linear Models

- Problem: For large n, the cubic runtime and the quadratic space requirements become a bottleneck!
- Any ideas how we can still compute such models for non-linear kernels?
- So far, we have considered models of the form

$$f(\cdot) = \sum_{i=1}^{n} c_i k(\cdot, \mathbf{x}_i)$$

with $\mathbf{c} \in \mathbb{R}^n$. Idea: Only allow a subset of the coefficients to be non-zero!

Trick III: Large-Scale Non-Linear Models

- Problem: For large n, the cubic runtime and the quadratic space requirements become a bottleneck!
- Any ideas how we can still compute such models for non-linear kernels?
- So far, we have considered models of the form

$$f(\cdot) = \sum_{i=1}^{n} c_i k(\cdot, \mathbf{x}_i)$$

  with $\mathbf{c} \in \mathbb{R}^n$. Idea: Only allow a subset of the coefficients to be non-zero!
- Let $R = \{i_1, \ldots, i_r\} \subseteq \{1, \ldots, n\}$ a subset of indices. Then, search for models of the form

$$f(\cdot) = \sum_{v=1}^{r} c_{i_v} k(\cdot, \mathbf{x}_{i_v})$$

Trick III: Large-Scale Non-Linear Least-Squares

- Let $R = \{i_1, \ldots, i_r\} \subseteq \{1, \ldots, n\}$ a subset of indices. Then, search for models of the form

$$f(\cdot) = \sum_{v=1}^{r} c_{i_v} k(\cdot, \mathbf{x}_{i_v})$$

- Let $\mathbf{K}_R$ denote the submatrix of $\mathbf{K}$ containing only the rows indexed by R. Further, let $\mathbf{K}_{R,R}$ denote the submatrix containing only the rows and columns indexed by R.

Trick III: Large-Scale Non-Linear Least-Squares

- Let $R = \{i_1, \ldots, i_r\} \subseteq \{1, \ldots, n\}$ a subset of indices. Then, search for models of the form

$$f(\cdot) = \sum_{v=1}^{r} c_{i_v} k(\cdot, \mathbf{x}_{i_v})$$

- Let $\mathbf{K}_R$ denote the submatrix of $\mathbf{K}$ containing only the rows indexed by R. Further, let $\mathbf{K}_{R,R}$ denote the submatrix containing only the rows and columns indexed by R.
- The approximation leads to the following task (check on your own!):

$$\underset{\hat{\mathbf{c}} \in \mathbb{R}^r}{\text{minimize}} \quad \frac{1}{n} \left(\mathbf{y} - (\mathbf{K}_R)^T \hat{\mathbf{c}} \right)^T \left(\mathbf{y} - (\mathbf{K}_R)^T \hat{\mathbf{c}} \right) + \lambda \hat{\mathbf{c}}^T \mathbf{K}_{R,R} \hat{\mathbf{c}}$$

Trick III: Large-Scale Non-Linear Least-Squares

- Let $R = \{i_1, \ldots, i_r\} \subseteq \{1, \ldots, n\}$ a subset of indices. Then, search for models of the form

$$f(\cdot) = \sum_{v=1}^{r} c_{i_v} k(\cdot, \mathbf{x}_{i_v})$$

- Let $\mathbf{K}_R$ denote the submatrix of $\mathbf{K}$ containing only the rows indexed by R. Further, let $\mathbf{K}_{R,R}$ denote the submatrix containing only the rows and columns indexed by R.
- The approximation leads to the following task (check on your own!):

$$\underset{\hat{\mathbf{c}} \in \mathbb{R}^r}{\text{minimize}} \quad \frac{1}{n} \left(\mathbf{y} - (\mathbf{K}_R)^T \hat{\mathbf{c}} \right)^T \left(\mathbf{y} - (\mathbf{K}_R)^T \hat{\mathbf{c}} \right) + \lambda \hat{\mathbf{c}}^T \mathbf{K}_{R,R} \hat{\mathbf{c}}$$

- Similar derivations as before yield (check on your own!):

$$\hat{\mathbf{c}}^* = \left(\mathbf{K}_R (\mathbf{K}_R)^T + n\lambda \mathbf{K}_{R,R} \right)^{-1} \mathbf{K}_R \mathbf{y}$$

Trick III: Large-Scale Non-Linear Least-Squares

- Let $R = \{i_1, \ldots, i_r\} \subseteq \{1, \ldots, n\}$ a subset of indices. Then, search for models of the form

$$f(\cdot) = \sum_{v=1}^{r} c_{i_v} k(\cdot, \mathbf{x}_{i_v})$$

- Let $\mathbf{K}_R$ denote the submatrix of $\mathbf{K}$ containing only the rows indexed by R. Further, let $\mathbf{K}_{R,R}$ denote the submatrix containing only the rows and columns indexed by R.
- The approximation leads to the following task (check on your own!):

$$\underset{\hat{\mathbf{c}} \in \mathbb{R}^r}{\text{minimize}} \quad \frac{1}{n} \left(\mathbf{y} - (\mathbf{K}_R)^T \hat{\mathbf{c}} \right)^T \left(\mathbf{y} - (\mathbf{K}_R)^T \hat{\mathbf{c}} \right) + \lambda \hat{\mathbf{c}}^T \mathbf{K}_{R,R} \hat{\mathbf{c}}$$

- Similar derivations as before yield (check on your own!):

$$\hat{\mathbf{c}}^* = \left(\mathbf{K}_R (\mathbf{K}_R)^T + n\lambda \mathbf{K}_{R,R} \right)^{-1} \mathbf{K}_R \mathbf{y}$$

- This can be computed in $O(nr^2)$ time using $O(nr)$ space.

Trick III: Large-Scale Non-Linear Least-Squares

- Let $R = \{i_1, \ldots, i_r\} \subseteq \{1, \ldots, n\}$ a subset of indices. Then, search for models of the form

$$f(\cdot) = \sum_{v=1}^{r} c_{i_v} k(\cdot, \mathbf{x}_{i_v})$$

- Let $\mathbf{K}_R$ denote the submatrix of $\mathbf{K}$ containing only the rows indexed by R. Further, let $\mathbf{K}_{R,R}$ denote the submatrix containing only the rows and columns indexed by R.
- The approximation leads to the following task (check on your own!):

$$\underset{\hat{\mathbf{c}} \in \mathbb{R}^r}{\text{minimize}} \quad \frac{1}{n} \left(\mathbf{y} - (\mathbf{K}_R)^T \hat{\mathbf{c}} \right)^T \left(\mathbf{y} - (\mathbf{K}_R)^T \hat{\mathbf{c}} \right) + \lambda \hat{\mathbf{c}}^T \mathbf{K}_{R,R} \hat{\mathbf{c}}$$

- Similar derivations as before yield (check on your own!):

$$\hat{\mathbf{c}}^* = \left(\mathbf{K}_R (\mathbf{K}_R)^T + n\lambda \mathbf{K}_{R,R} \right)^{-1} \mathbf{K}_R \mathbf{y}$$

- This can be computed in $O(nr^2)$ time using $O(nr)$ space.

WARNING: In practice, do not invert the matrices (numerical issues!). Instead, directly solve the linear system of equations (e.g., via `numpy.linalg.solve`).

Nyström Approximation

- The matrix

$$\widetilde{\mathbf{K}} = (\mathbf{K}_R)^T (\mathbf{K}_{R,R})^{-1} \mathbf{K}_R \in \mathbb{R}^{n \times n} \tag{3}$$

  is called the Nyström approximation of the kernel matrix $\mathbf{K}$.

Nyström Approximation

- The matrix
$$\widetilde{\mathbf{K}} = (\mathbf{K}_R)^T (\mathbf{K}_{R,R})^{-1} \mathbf{K}_R \in \mathbb{R}^{n \times n} \tag{3}$$
is called the Nyström approximation of the kernel matrix $\mathbf{K}$.
- A good assignment for r depends on the dataset! The larger, the better.

Nyström Approximation

- The matrix
$$\widetilde{\mathbf{K}} = (\mathbf{K}_R)^T (\mathbf{K}_{R,R})^{-1} \mathbf{K}_R \in \mathbb{R}^{n \times n} \tag{3}$$
is called the Nyström approximation of the kernel matrix $\mathbf{K}$.
- A good assignment for r depends on the dataset! The larger, the better.
- How to select a subset R such that $\widetilde{\mathbf{K}}$ is a good approximation of the kernel matrix $\mathbf{K}$ is subject of ongoing research.

Nyström Approximation

- The matrix
$$\widetilde{\mathbf{K}} = (\mathbf{K}_R)^T (\mathbf{K}_{R,R})^{-1} \mathbf{K}_R \in \mathbb{R}^{n \times n} \tag{3}$$
is called the Nyström approximation of the kernel matrix $\mathbf{K}$.
- A good assignment for r depends on the dataset! The larger, the better.
- How to select a subset R such that $\widetilde{\mathbf{K}}$ is a good approximation of the kernel matrix $\mathbf{K}$ is subject of ongoing research.

Question: Difference of this approach compared to using only a subset of r training instances?

Demo Time!

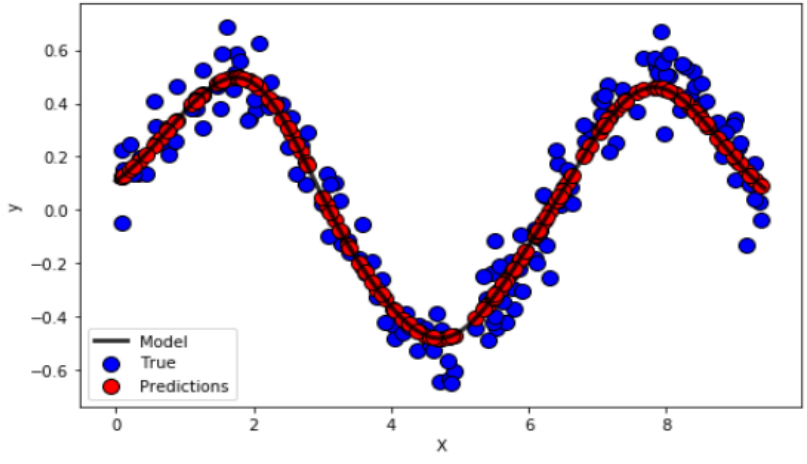
jupyter Least_Squares (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

```
from sklearn.model_selection import train_test_split

X, y = get_dataset(n=500)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)

plt.figure(figsize=(8,5))

model = LeastSquaresRegression(kernel="fast_rbf", lam=0.01, gamma=1, r=None)
model.fit(X_train, y_train)
preds = model.predict(X_test)

# plot test points and predictions
plt.scatter(X_test, y_test, s=100, c="b", edgecolor="k", linewidths=1, label="True")
plt.scatter(X_test, preds, s=100, c="r", edgecolor="k", linewidths=1, label="Predictions")

# plot model
x = numpy.linspace(X.min(), X.max(), 500)
plt.plot(x, model.predict(x), "-k", alpha=0.8, linewidth=3.0, label="Model")

plt.xlabel("X")
plt.ylabel("y")
plt.legend(loc=0)

plt.show()
```



Outline

Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?

Manuel Fernández-Delgado MANUEL.FERNANDEZ.DELGADO@USC.ES
Eva Cernadas EVA.CERNADAS@USC.ES
Senén Barro SENEN.BARRO@USC.ES
CITIUS: Centro de Investigación en Tecnoloxías da Información da USC
University of Santiago de Compostela
Campus Vida, 15872, Santiago de Compostela, Spain

Dinani Amorim DINANIAMORIM@GMAIL.COM
Departamento de Tecnologia e Ciências Sociais- DTCS
Universidade do Estado da Bahia
Av. Edgard Chastinet S/N - São Geraldo - Juazeiro-BA, CEP: 48.305-680, Brasil

Editor: Russ Greiner

Abstract

We evaluate **179 classifiers** arising from **17 families** (discriminant analysis, Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, bagging, stacking, random forests and other ensembles, generalized linear models, nearest-neighbors, partial least squares and principal component regression, logistic and multinomial regression, multiple adaptive regression splines and other methods), implemented in Weka, R (with and without the caret package), C and Matlab, including all the relevant classifiers available today. We use **121 data sets**, which represent **the whole UCI** data base (excluding the large-scale problems) and other own real problems, in order to achieve significant conclusions about the classifier behavior, not dependent on the data set collection. **The classifiers most likely to be the bests are the random forest** (RF) versions, the best of which (implemented in R and accessed via caret) achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets. However, the difference is not statistically significant with the second best, the SVM with Gaussian kernel

Now!

http://jmlr.csail.mit.edu/papers/volume15/delgado14a/delgado14a.pdf

Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?

Manuel Fernández-Delgado MANUEL.FERNANDEZ.DELGADO@USC.ES
Eva Cernadas EVA.CERNADAS@USC.ES
Senén Barro SENEN.BARRO@USC.ES
CITIUS: Centro de Investigación en Tecnoloxías da Información da USC
University of Santiago de Compostela
Campus Vida, 15872, Santiago de Compostela, Spain

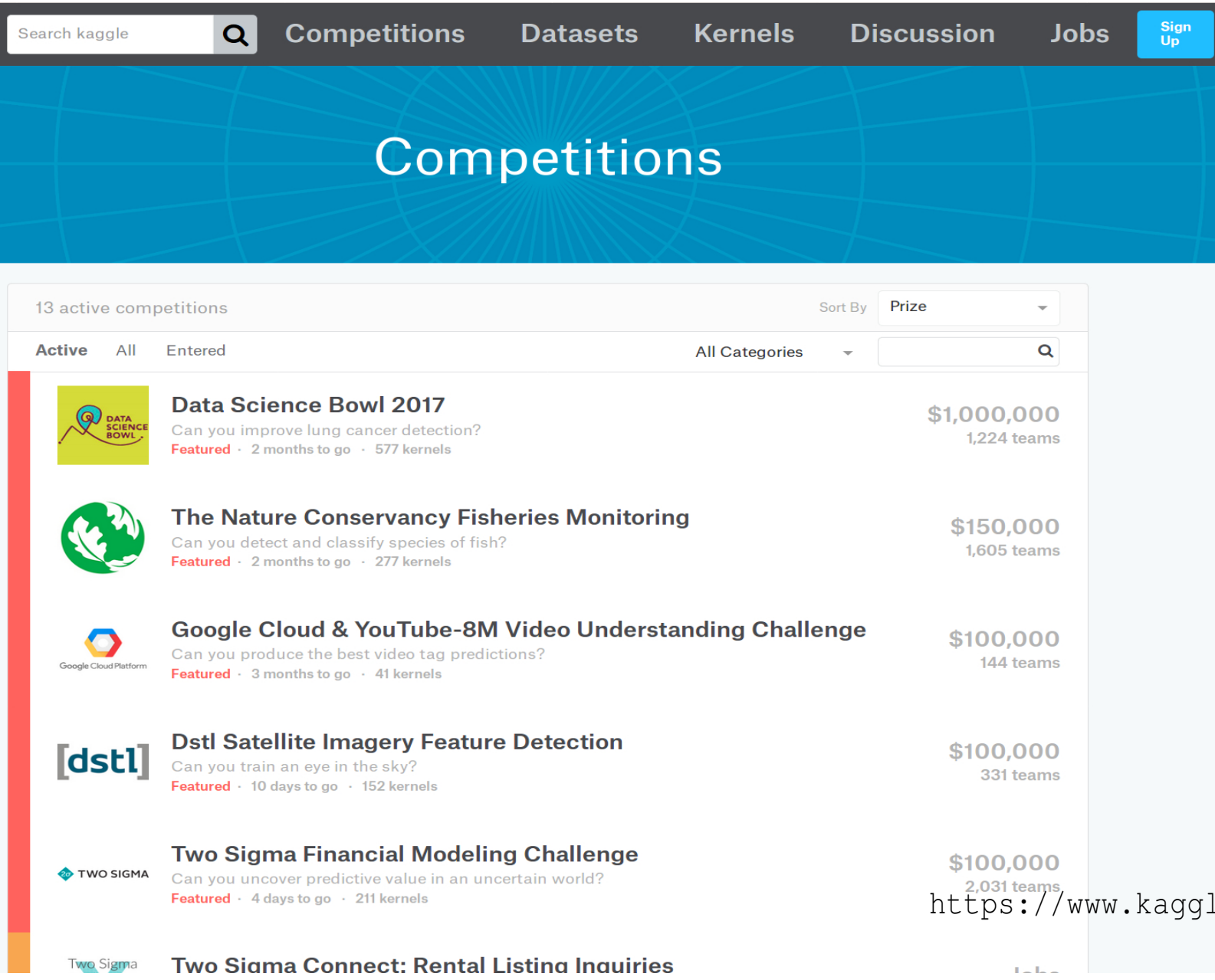
Dinani Amorim DINANIAMORIM@GMAIL.COM
Departamento de Tecnologia e Ciências Sociais- DTCS
Universidade do Estado da Bahia
Av. Edgard Chastinet S/N - São Geraldo - Juazeiro-BA, CEP: 48.305-680, Brasil

Editor: Russ Greiner

Abstract

We evaluate **179 classifiers** arising from **17 families** (discriminant analysis, Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, bagging, stacking, random forests and other ensembles, generalized linear models, nearest-neighbors, partial least squares and principal component regression, logistic and multinomial regression, multiple adaptive regression splines and other methods), implemented in Weka, R (with and without the caret package), C and Matlab, including all the relevant classifiers available today. We use **121 data sets**, which represent **the whole UCI** data base (excluding the large-scale problems) and other own real problems, in order to achieve significant conclusions about the classifier behavior, not dependent on the data set collection. **The classifiers most likely to be the bests are the random forest** (RF) versions, the best of which (implemented in R and accessed via caret) achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets. However, the difference is not statistically significant with the second best, the SVM with Gaussian kernel

Data Analysis: Real-World Competitions



https://www.kaggle.com

Now!

https://dnc1994.com/2016/05/rank-10-percent-in-first-kaggle-competition-en/

Wille
最後に辿り着いた場所

How to Rank 10% in Your First Kaggle Competition

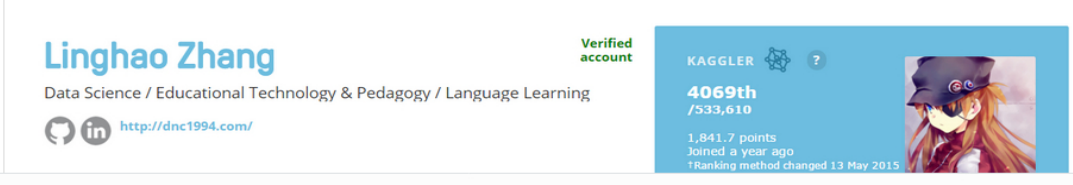
Posted on 2016-05-10 | In Data Science | 22 Comments

Introduction

Kaggle is the best place to learn from other data scientists. Many companies provide data and prize money to set up data science competitions on Kaggle. Recently I had my first shot on Kaggle and **ranked 98th (~ 5%) among 2125 teams**. Being my Kaggle debut, I feel quite satisfied with the result. Since many Kaggle beginners set 10% as their first goal, I want to share my two cents on how to achieve that.

This post is also available in Chinese.

Updated on Oct 28th, 2016: I made many wording changes and added several updates to this post. Note that Kaggle has went through some major changes since I published this post, especially with its ranking system. Therefore some descriptions here might not apply anymore.



Now!

https://dnc1994.com/2016/05/rank-10-percent-in-first-kaggle-competition-en/

Model Selection

When the features are set, we can start training models. Kaggle competitions usually favor **tree-based models**:

- **Gradient Boosted Trees**
- Random Forest
- Extra Randomized Trees

The following models are slightly worse in terms of general performance, but are suitable as base models in ensemble learning (will be discussed later):

- SVM
- Linear Regression
- Logistic Regression
- Neural Networks

Note that this does not apply to computer vision competitions which are pretty much dominated by neural network models.

All these models are implemented in **Sklearn**.

Here I want to emphasize the greatness of **Xgboost**. The outstanding performance of gradient boosted trees and Xgboost's efficient implementation makes it very popular in Kaggle competitions. Nowadays almost every winner uses Xgboost in one way or another.

Updated on Oct 28th, 2016: Recently Microsoft open sourced **LightGBM**, a potentially better

Now!

https://dnc1994.com/2016/05/rank-10-percent-in-first-kaggle-competition-en/

Model Selection

When the features are set, we can start training models. Kaggle competitions usually favor **tree-based models**:

- **Gradient Boosted Trees**
- Random Forest
- Extra Randomized Trees

The following models are slightly worse in terms of general performance, but are suitable as base models in ensemble learning (will be discussed later):

- SVM
- Linear Regression
- Logistic Regression
- Neural Networks

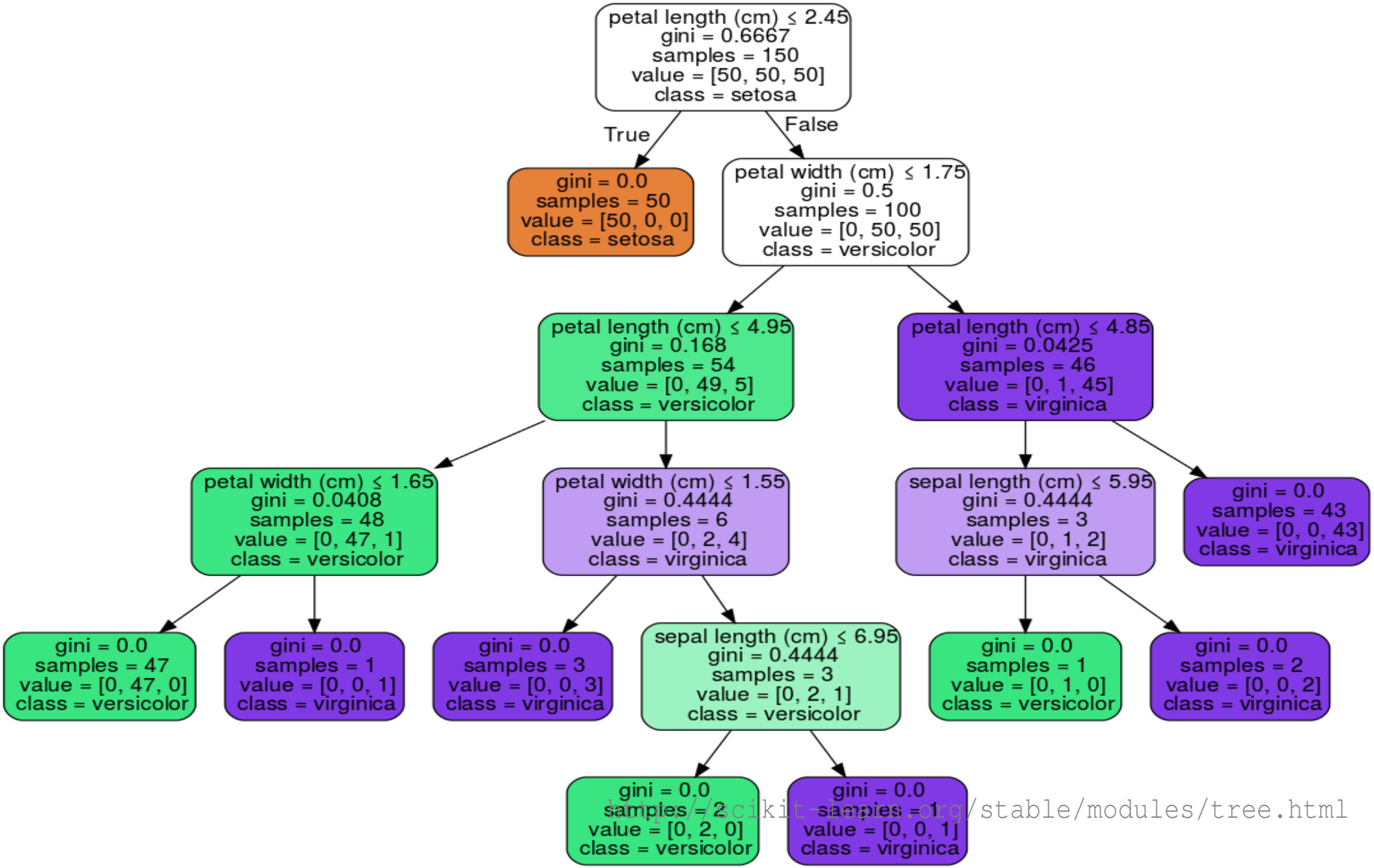
Note that this does not apply to computer vision competitions which are pretty much dominated by neural network models.

All these models are implemented in **Sklearn**.

Here I want to emphasize the greatness of **Xgboost**. The outstanding performance of gradient boosted trees and Xgboost's efficient implementation makes it very popular in Kaggle competitions. Nowadays almost every winner uses Xgboost in one way or another.

Updated on Oct 28th, 2016: Recently Microsoft open sourced **LightGBM**, a potentially better

Example: Iris Flower Classification Tree

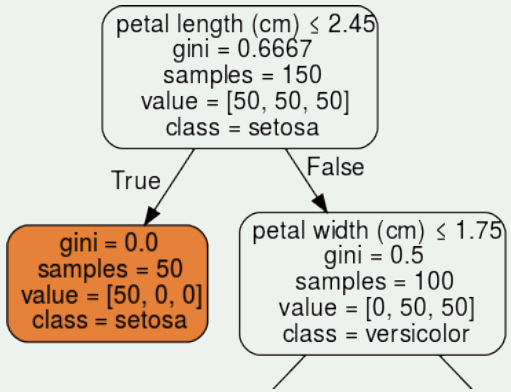


http://scikit-learn.org/stable/modules/tree.html

Classification and Regression Trees (CARTs)

Structure & Training

- Every internal node is associated with one coordinate $i \in \{1, \ldots, d\}$ and a threshold θ.
- At each inner node, the training data $S = \{(\mathbf{x}_1, y_1), \ldots\}$ at that node are split into



$$L_{i,\theta} = \{(\mathbf{x}, y) \in S \,|\, x_i \leq \theta\} \text{ and } R_{i,\theta} = \{(\mathbf{x}, y) \in S \,|\, x_i > \theta\},$$

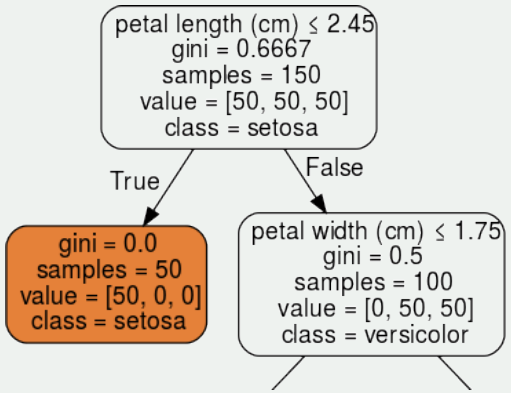
which are passed to the left/right child for further processing.

- Construction: The optimal tree cannot be found efficiently. Therefore, trees are built using a heuristic.
- The leaf nodes, indexed by $\tau = 1, \ldots, M$, define regions $\mathcal{R}_\tau \subseteq \mathbb{R}^d$.

Building CARTs

Basic Idea

- Every inner node is associated with one coordinate $i \in \{1, \ldots, d\}$ and a threshold θ.
- At each inner node, the associated data $S = \{(\mathbf{x}_1, y_1), \ldots\}$ are split into $L_{i,\theta}$ and $R_{i,\theta}$ such that the information gain



$$G_{i,\theta}(S) = Q(S) - \frac{|L_{i,\theta}|}{|S|} Q(L_{i,\theta}) - \frac{|R_{i,\theta}|}{|S|} Q(R_{i,\theta})$$

  is maximized, where Q is some impurity measure.
- If the number $|S|$ of points at a node is smaller than a user-defined value or if S is pure (i.e., all point have the same label), the node becomes a leaf (further stopping rules exist!).

Simplified Objective

Note that

$$\underset{i,\theta}{\text{maximize}}\ G_{i,\theta}(S) = Q(S) - \frac{|L_{i,\theta}|}{|S|}Q(L_{i,\theta}) - \frac{|R_{i,\theta}|}{|S|}Q(R_{i,\theta})$$

is equivalent to

$$\underset{i,\theta}{\text{minimize}}\ |L_{i,\theta}|Q(L_{i,\theta}) + |R_{i,\theta}|Q(R_{i,\theta})$$

Let's build a tree!

Recursive Tree Construction

Procedure: BUILDTREE(S,m)

Require: $S = \{(\mathbf{x}_1, y_1), \ldots\}$ and number m.

Ensure: Tree $\mathcal{T}$ built for the input patterns in S.

1: **if** $|S| \leq m$ **then**
2: **return** leaf node storing the labels $\{y_1, \ldots, y_{|S|}\}$
3: **end if**
4: **if** $y_i = y_j$ for all $(\mathbf{x}_i, y_i), (\mathbf{x}_j, y_j) \in S$ **then**
5: **return** leaf node storing the labels $\{y_1, \ldots, y_{|S|}\}$
6: **end if**
7: Find $(i^*, \theta^*) = \text{argmax}_{i,\theta}\, G_{i,\theta}(S)$
8: $\mathcal{T}_l$ = BUILDTREE($L_{i,\theta}$, m)
9: $\mathcal{T}_r$ = BUILDTREE($R_{i,\theta}$, m)
10: Generate node that stores the splitting information (i^*, θ^*) and pointers to its subtrees $\mathcal{T}_l$ and $\mathcal{T}_r$. Let $\mathcal{T}$ denote the resulting tree.
11: **return** $\mathcal{T}$

Regression Trees

Construction: What are "optimal" splits?

- Impurity measure: For regression problems, a common choice is

$$Q(S) = \sum_{(\mathbf{x},y)\in S} (y - \hat{y})^2,$$

  where $\bar{y} = \frac{1}{|S|} \sum_{(\mathbf{x},y)\in S} y$

- Find optimal splitting dimension i^* and threshold θ^* w.r.t.

$$\underset{i,\theta}{\text{minimize}} \; |L_{i,\theta}| Q(L_{i,\theta}) + |R_{i,\theta}| Q(R_{i,\theta})$$

Regression Trees

Construction: What are "optimal" splits?

- Impurity measure: For regression problems, a common choice is

$$Q(S) = \sum_{(\mathbf{x},y)\in S} (y - \hat{y})^2,$$

  where $\bar{y} = \frac{1}{|S|} \sum_{(\mathbf{x},y)\in S} y$

- Find optimal splitting dimension i^* and threshold θ^* w.r.t.

$$\underset{i,\theta}{\text{minimize}} \; |L_{i,\theta}| Q(L_{i,\theta}) + |R_{i,\theta}| Q(R_{i,\theta})$$

How to compute the optimal dimension and threshold?

Regression Trees

Construction: What are "optimal" splits?

- Impurity measure: For regression problems, a common choice is

$$Q(S) = \sum_{(\mathbf{x},y)\in S} (y - \hat{y})^2,$$

  where $\bar{y} = \frac{1}{|S|} \sum_{(\mathbf{x},y)\in S} y$

- Find optimal splitting dimension i^* and threshold θ^* w.r.t.

$$\underset{i,\theta}{\text{minimize}} \; |L_{i,\theta}| Q(L_{i,\theta}) + |R_{i,\theta}| Q(R_{i,\theta})$$

How to compute the optimal dimension and threshold?

1. Naive approach, $\mathcal{O}(d \cdot |S|^2)$ time: For each i, do:
   - Consider all possible thresholds (e.g., $\{\mathbf{x}_i | \mathbf{x} \in S\}$).
   - For each threshold θ: Compute $|L_{i,\theta}| Q(L_{i,\theta})$ and $|R_{i,\theta}| Q(R_{i,\theta})$.

Regression Trees

Construction: What are "optimal" splits?

- Impurity measure: For regression problems, a common choice is

$$Q(S) = \sum_{(\mathbf{x},y)\in S} (y - \hat{y})^2,$$

  where $\bar{y} = \frac{1}{|S|} \sum_{(\mathbf{x},y)\in S} y$

- Find optimal splitting dimension i^* and threshold θ^* w.r.t.

$$\underset{i,\theta}{\text{minimize}} \; |L_{i,\theta}| Q(L_{i,\theta}) + |R_{i,\theta}| Q(R_{i,\theta})$$

How to compute the optimal dimension and threshold?

1. Naive approach, $\mathcal{O}(d \cdot |S|^2)$ time: For each i, do:
   - Consider all possible thresholds (e.g., $\{\mathbf{x}_i | \mathbf{x} \in S\}$).
   - For each threshold θ: Compute $|L_{i,\theta}| Q(L_{i,\theta})$ and $|R_{i,\theta}| Q(R_{i,\theta})$.
2. Better approach, $\mathcal{O}(d \cdot |S| \log |S|)$ time: For each i, do:
   - Sort all instances $(\mathbf{x}, y)$ in S according to the patterns i-th dimension.
   - Scan this sorted list and update these values incrementally!

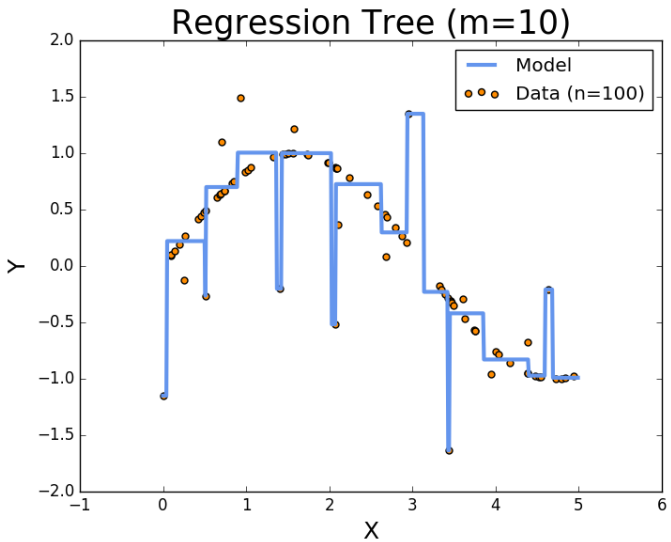
Regression Trees

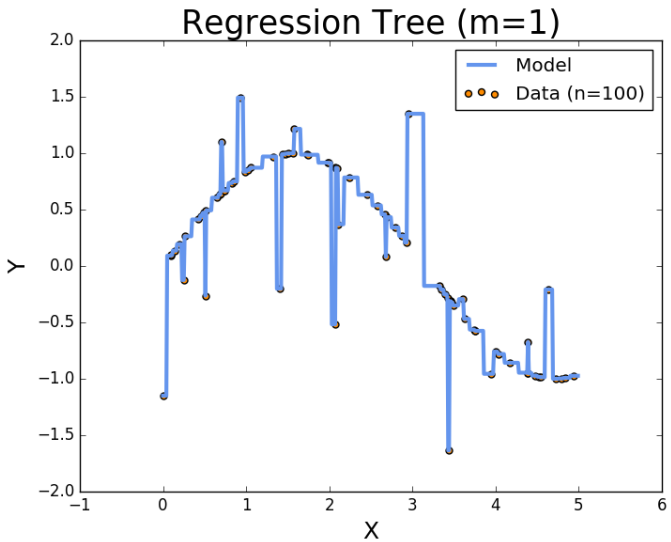
Prediction

- Training data $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \in \mathbb{R}^d \times \mathbb{R}$
- Each leaf node $\tau = 1, \ldots, M$ of the tree $\mathcal{T}$ corresponds to a region $R_\tau \subseteq \mathbb{R}^d$
- The output $\hat{y} = \mathcal{T}(\mathbf{x})$ given a new input instance $\mathbf{x}$ is

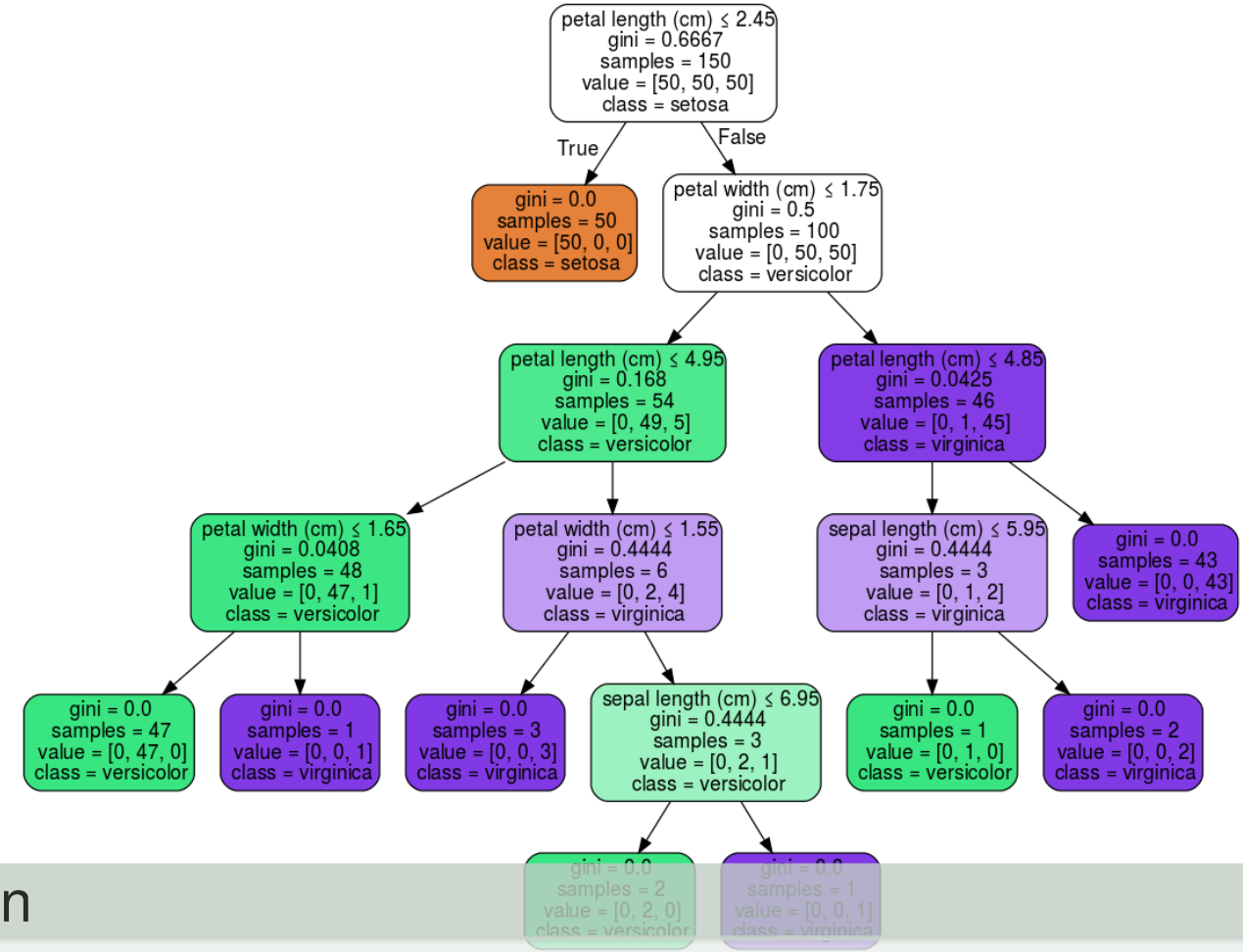
$$\mathcal{T}(\mathbf{x}) = \sum_{\tau=1}^{M} c_\tau \mathbb{I}\{\mathbf{x} \in \mathcal{R}_\tau\} \ ,$$

where $c_\tau = \frac{1}{|R_\tau \cap S|} \sum_{\mathbf{x}_j \in \mathcal{R}_\tau \cap S} y_j$ (thus: mean of all labels stored in leaf).





Classification Trees



Prediction

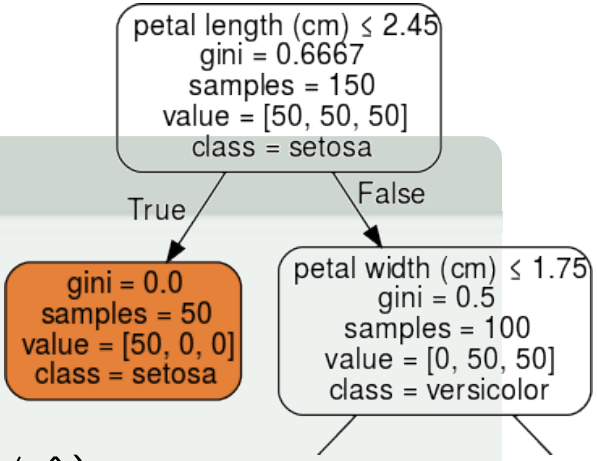
1. Traverse the tree $\mathcal{T}$ to find the leaf that contains the query $\mathbf{x}$.
2. Each leaf node $\tau = 1, \ldots, M$ of the tree $\mathcal{T}$ corresponds to a region $R_\tau \subset \mathbb{R}^d$
3. Use majority class in leaf as prediction $\mathcal{T}(\mathbf{x})$.

Classification Trees

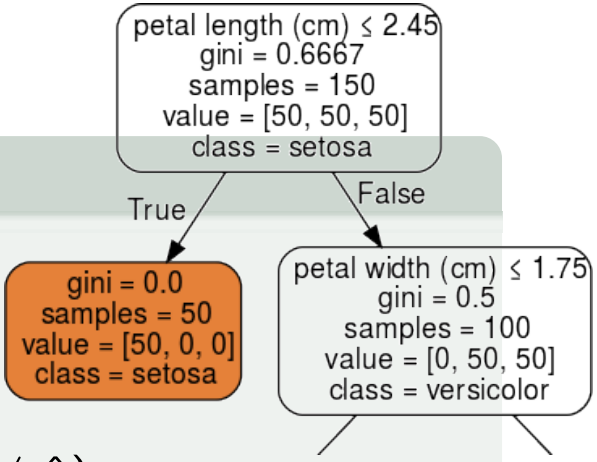
Construction

As before: We need to define an impurity measure $Q(S)$.
Let p_{Sk} be the fraction of points belonging to class k in S.

1. Misclassification error: Let $\hat{y} = \text{argmax}_k\, p_{Sk}$ be the dominant class in S. Then $Q(S) = \frac{1}{|S|} \sum_{(\mathbf{x}_j, y_j) \in S} \mathbb{I}(y_j \neq \hat{y})$
2. Gini index: $Q(S) = \sum_{k=1}^{K} p_{Sk}(1 - p_{Sk})$
3. ...



Classification Trees



Construction

As before: We need to define an impurity measure $Q(S)$.
Let p_{Sk} be the fraction of points belonging to class k in S.

1. Misclassification error: Let $\hat{y} = \text{argmax}_k\, p_{Sk}$ be the dominant class in S. Then $Q(S) = \frac{1}{|S|} \sum_{(\mathbf{x}_j, y_j) \in S} \mathbb{I}(y_j \neq \hat{y})$
2. Gini index: $Q(S) = \sum_{k=1}^{K} p_{Sk}(1 - p_{Sk})$
3. ...

At home: Check Gini indices for all three nodes.

Outline

Summary & Outlook

Today

1. Large-Scale Least-Squares

Outlook

- Today (Practical Sessions): VirtualBox, Google Colab, HW1/HW2
- Tuesday (Lecture): Large-Scale Random Forests & Boosted Trees I (FG)
- Thursday (Lecture): Neural Networks (CI)

References I

J. L Bentley.

Multidimensional binary search trees used for associative searching.
Communications of the ACM, 18(9):509–517, 1975.

V. Garcia, É. Debreuve, F. Nielsen, and M. Barlaud.

K-nearest neighbor search: Fast gpu-based implementations and application to high-dimensional feature matching.
In *2010 IEEE International Conference on Image Processing*, pages 3757–3760, 2010.

Fabian Gieseke, Justin Heinermann, Cosmin Oancea, and Christian Igel.

Buffer k-d trees: Processing massive nearest neighbor queries on GPUs.
In *Proceedings of the 31st International Conference on Machine Learning. JMLR W&CP*, volume 32, pages 172–180, 2014.

Simon Rogers and Mark Girolami.

A First Course in Machine Learning.
Chapman & Hall/CRC, 1st edition, 2011.

P. Ram and A. G. Gray.

Which space partitioning tree to use for search?
In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, pages 656–664, USA, 2013.

Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola.

A generalized representer theorem.
In *Proceedings of the 14th Annual Conference on Computational Learning Theory and and 5th European Conference on Computational Learning Theory*, pages 416–426, London, UK, UK, 2001. Springer-Verlag.