MVE137
Probability and Statistical Learning Using Python

Supervised Learning

Alexandre Graell i Amat

alexandre.graell@chalmers.se

https://sites.google.com/site/agraellamat

September 28, 2021

Supervised learning

Statistical learning

Goal: Infer a predictive function (model), such that it can be used to predict the output for new, yet unseen data.

Examples: predicting housing or stock prices, image classification, ...

Supervised learning

Goal: Given a training set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, identify an algorithm to predict the outcome y for a new (yet unseen) data point $\boldsymbol{x}$.

- Learning a model from labeled data
- Predicting output of new data based on the learned model

- $\boldsymbol{x}_i$: free variables (features, predictors, covariates, domain points)
- y_i: target variables (dependent variables, labels, responses)

Supervised learning

Three classes of responses:

- Continuous (quantitative): take on numerical values
- Discrete (qualitative, categorical): a discrete set of categories
- Order categorical: the order is important

Two learning algorithms:

- Regression: predict a quantitative output
- Classification: predict a qualitative output

Statistical learning and decision theory

Goal: Given a training set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, predict y for a new (yet unseen) data point $\boldsymbol{x}$.

Impossible if no information on the mechanism relating $\boldsymbol{x}$ and y!

We may assume that $\boldsymbol{x}$ and y are related via a function

$$y = \tilde{f}(\boldsymbol{x})$$

Goal: Find best possible approximation of $\tilde{f}(\boldsymbol{x})$, $f(\boldsymbol{x})$ (prediction model). Predict the outcome y for $\boldsymbol{x}$ as $\hat{y} = f(\boldsymbol{x})$.

- Treat $\mathbf{x}$ and y as random variables, and

$$(\mathbf{x}_i, \mathrm{y}_i) \sim_{i.i.d.} p(\boldsymbol{x}, y), \quad i \in [N]\,.$$

Statistical learning and decision theory

How should we choose $f(\boldsymbol{x})$?

- Loss function $\ell(y, \hat{y}) = \ell(y, f(\boldsymbol{x}))$: cost (loss or risk) incurred when the correct value is y while the estimate is $\hat{y}$
- Quadratic loss function:

$$\ell(y, \hat{y}) = (y - \hat{y})^2 = (y - f(\boldsymbol{x}))^2,$$

Expected prediction error (expected generalization loss/error):

$$\begin{aligned} L(\hat{y}) &= \mathbb{E}_{\mathbf{x},\mathrm{y}\sim p(\boldsymbol{x},y)}[\ell(\mathrm{y}, f(\mathbf{x}))] \\ &= \int\int \ell(\mathrm{y}, f(\boldsymbol{x}))p(\boldsymbol{x}, y)\mathrm{d}\boldsymbol{x}\mathrm{d}y\,. \end{aligned}$$

Statistical learning and decision theory

How should we choose $f(\boldsymbol{x})$?

Optimal prediction $\hat{y}(\boldsymbol{x})$ obtained by minimizing generalization loss:

$$
\begin{aligned}
f^*(\boldsymbol{x}) &= \underset{f}{\operatorname{argmin}}\ L(\hat{y}) \\
&= \underset{f}{\operatorname{argmin}}\ \mathbb{E}_{\mathbf{x},\mathrm{y}\sim p(\boldsymbol{x},y)}[\ell(\mathrm{y}, f(\mathbf{x}))] \\
&= \underset{f}{\operatorname{argmin}}\, \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{x}}}\left[\mathbb{E}_{\mathrm{y}\sim p_{\mathrm{y}|\mathbf{x}}}[\ell(\mathrm{y}, f(\boldsymbol{x}))]\right]
\end{aligned}
$$

Hence, it suffices to solve

$$
f^*(\boldsymbol{x}) = \underset{f}{\operatorname{argmin}}\, \mathbb{E}_{\mathrm{y}\sim p_{\mathrm{y}|\mathbf{x}}}[\ell(\mathrm{y}, f(\boldsymbol{x}))]
$$

Statistical learning and decision theory

For the quadratic loss $\ell(y,\hat{y}) = (y - f(\boldsymbol{x}))^2$,

$$f^*(\boldsymbol{x}) = \underset{f}{\operatorname{argmin}}\ \mathbb{E}_{\mathsf{y}\sim p_{\mathsf{y}|\mathbf{x}}}[\ell(\mathsf{y}, f(\boldsymbol{x}))] =$$

Linear regression

Linear regression: Assumes linear model for $f(\boldsymbol{x})$,

$$\hat{y} = f(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{p} x_j \beta_j,$$

with $\boldsymbol{x} = (x_1, x_2, \ldots, x_p)^\mathsf{T}$.

β_0: intercept or bias

- For simplicity, we will write

$$\tilde{\boldsymbol{x}} = (1, x_1, \ldots, x_p)^\mathsf{T} \quad \text{and} \quad \boldsymbol{\beta} = (\beta_0, \ldots, \beta_p)^\mathsf{T}$$

so that

$$\hat{y} = \sum_{j=0}^{p} \tilde{x}_j \beta_j = \tilde{\boldsymbol{x}}^\mathsf{T} \boldsymbol{\beta}$$

Linear regression assumes $\tilde{f}(\boldsymbol{x}) \approx \boldsymbol{x}^\mathsf{T}\boldsymbol{\beta}$ or, equivalently, $\mathbb{E}_{\mathsf{y}|\mathbf{x}}[y|x] \approx \tilde{\boldsymbol{x}}^\mathsf{T}\boldsymbol{\beta}^*$.

Linear regression: Optimal $\boldsymbol{\beta}$

$$\begin{aligned}\boldsymbol{\beta}^* &= \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, L(\hat{y}) \\ &= \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, \mathbb{E}_{\mathsf{x},\mathsf{y}\sim p(\boldsymbol{x},y)}[\ell(\mathsf{y}, f(\mathbf{x}))] \\ &= \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, \mathbb{E}_{\mathsf{x},\mathsf{y}\sim p(\boldsymbol{x},y)}[(\mathsf{y} - \mathbf{x}^\mathsf{T}\boldsymbol{\beta})^2]\end{aligned}$$

Idea: Approximate the expectation by the empirical average over the N training points $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^N$,

$$\begin{aligned}\boldsymbol{\beta}^* &= \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, \frac{1}{N}\sum_{i=1}^N (y_i - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta})^2 \\ &= \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, \sum_{i=1}^N (y_i - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta})^2\end{aligned}$$

Least squares linear regression

$$
\begin{aligned}
\boldsymbol{\beta}^* &= \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{N} (y_i - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta})^2 \\
&= \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 \\
&= \underset{\boldsymbol{\beta}}{\operatorname{argmin}} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^\mathsf{T}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})
\end{aligned}
$$

with

$$
\boldsymbol{X} = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \dots & x_{1,p} \\ 1 & x_{2,1} & x_{2,2} & \dots & x_{2,p} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_{N,1} & x_{N,2} & \dots & x_{N,p} \end{pmatrix}
$$

and $\boldsymbol{y} = (y_1, \dots, y_N)^\mathsf{T}$

Least squares linear regression

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^\mathsf{T} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})$$

Least squares linear regression for classification

Binary classification: $y \in \{a, b\}$

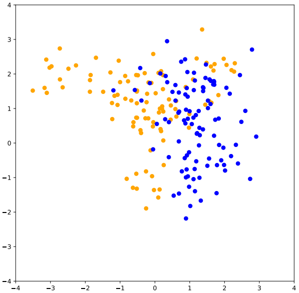
Idea: Encode $y = a \Longrightarrow y = 0$ and $y = b \Longrightarrow y = 1$ and apply plain linear regression plus thresholding:

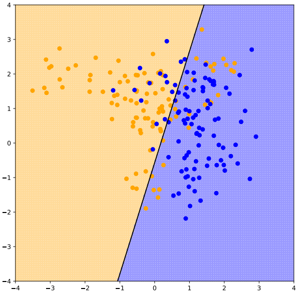
$$\hat{y}(\boldsymbol{x}) = \begin{cases} 0 & \text{if } f^*(\boldsymbol{x}) = \mathbb{E}_{\mathsf{y}|\mathbf{x}}[y|\boldsymbol{x}] = \boldsymbol{x}^\mathsf{T}\boldsymbol{\beta}^* \leq 0.5 \\ 1 & \text{otherwise} \end{cases}$$

Observations:

- The linear model determines the decision boundary $\{\boldsymbol{x} : \boldsymbol{x}^\mathsf{T}\boldsymbol{\beta}^* = 0.5\}$
- We can interpret $f^*(\boldsymbol{x}) = \mathbb{E}_{\mathsf{y}|\mathbf{x}}[y|x] = \boldsymbol{x}^\mathsf{T}\boldsymbol{\beta}$ as the probability $p(\mathsf{y} = 1|\boldsymbol{x})$

Least squares linear regression for classification





- 100 samples from two bivariate Gaussian distributions
- Blue: $\mathcal{N}((1,0)^{\mathsf{T}}, \boldsymbol{I})$
- Orange: $\mathcal{N}((0,1)^{\mathsf{T}}, \boldsymbol{I})$

Goal: For a new coordinate $\boldsymbol{x} = (x_1, x_2)$, determine to which of the Gaussians corresponds to.

k-Nearest neighbor regression

k-Nearest neighbor regression: Given $\boldsymbol{x}$, it predicts y as

$$\hat{y}(\boldsymbol{x}) = \frac{1}{k} \sum_{\boldsymbol{x}_i \in \mathcal{N}_k(\boldsymbol{x})} y_i(\boldsymbol{x}_i)$$

$\mathcal{N}_k(\boldsymbol{x})$: set of k nearest neighbors to $\boldsymbol{x}$ in the training set.

- For $\boldsymbol{x} \in \mathbb{R}^p$, we consider the Euclidean distance $d(\boldsymbol{x}, \boldsymbol{x}_i) = \|\boldsymbol{x} - \boldsymbol{x}_i\|^2 \longrightarrow$ $\mathcal{N}_k(\boldsymbol{x})$ is the set $\{\boldsymbol{x}_i\}$ closest (in Euclidean distance) to $\boldsymbol{x}$.

k-Nearest neighbor regression

Optimal solution:

$$\hat{y}(x) = f^*(\boldsymbol{x}) = \mathbb{E}_{\mathsf{y}|\mathbf{x}}[\mathsf{y}|\boldsymbol{x}]$$

k-nearest neighbors tries to accomplish this directly!

1. Replacing $\mathbf{x} = \boldsymbol{x}$ by neighborhood of $\boldsymbol{x}$ in the training data $\mathcal{N}_k(\boldsymbol{x})$
2. Replacing expectation by average over the k training neighbors

We make two approximations:

- Expectation approximated by sample average
- Conditioning at point $\boldsymbol{x}$ relaxed to conditioning on region close $\boldsymbol{x}$

k-Nearest neighbor regression

What's the role of k?

For a larger k ...

- Average is more accurate and stable (reduced variance)
- Neighborhood is bigger and less representative of $\mathbf{x} = \boldsymbol{x}$ (increased bias)

Under mild regularity conditions on $p(\boldsymbol{x}, y)$, as $N \to \infty$, $k(N) \to \infty$ and $k(N)/N \to 0$, $\hat{y}(\boldsymbol{x}) \to \tilde{f}(\boldsymbol{x}),\ \forall \boldsymbol{x}$.

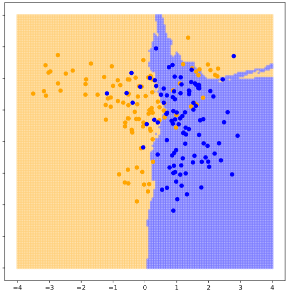
k-Nearest neighbors for classification

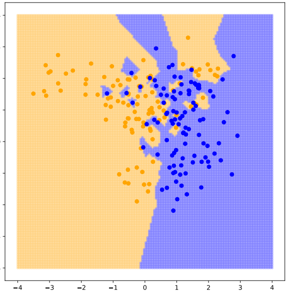
Idea: Replace averaging by a majority vote,

$$\hat{y}(\boldsymbol{x}) = \mathbb{1}\left\{\frac{1}{k}\sum_{\boldsymbol{x}_i \in \mathcal{N}_k(\boldsymbol{x})} \mathbb{1}\{y_i(\boldsymbol{x}_i) = 1\} > 0.5\right\}$$

1. Find $\mathcal{N}_k(\boldsymbol{x})$
2. Majority vote: Assign $\boldsymbol{x}$ to the class that most predictors in $\mathcal{N}_k(\boldsymbol{x})$ belong

k-Nearest neighbors for classification





- Left: 15-nearest neighbors
- Right: 1-nearest neighbor

Error on training data decreases with decreasing k, and is zero for $k = 1$.

Is $k = 1$ optimal?

Least squares regression vs k-nearest neighbors regression

Least squares regression:

- Assumes $\tilde{f}(\boldsymbol{x})$ well approximated by a globally linear function
- Very smooth boundary
- Stable to fit
- Linear decision boundary (strong assumption!)
- Low variance and (potentially) high bias

k-nearest neighbors regression:

- No stringent assumptions about underlying data
- Can adapt to any shape of the data
- Not smooth boundary (for small k)
- Unstable to fit (for small k)
- high variance and low bias

Parametric vs nonparametric models

Parametric models: Build $f(\boldsymbol{x})$ as a parametric model that applies to the whole space.

1. Select parametric model (hypothesis class), with fixed number of parameters
2. Learn parameters to fit the training data $\mathcal{D}$

Nonparametric models: Don't make explicit assumptions about form $f(\boldsymbol{x})$, but describe it in terms of local behavior of the training data in the region near $\boldsymbol{x}$.

1. Seek an estimate of f that gets as close to the data points as possible without being too wiggly
2. Advantage: accurately fit a wider range of possible shapes for f
3. Disadvantage: number parameters grows with amount of training data

Classification and statistical learning

- $y \in \mathcal{C} = \{C_1, \ldots, C_K\}$, $|\mathcal{C}| = K$
- We assume that $\boldsymbol{x}$ and y are related via function $\tilde{f}(\boldsymbol{x})$ (rule)

Goal: learn a rule $f(\boldsymbol{x})$ which maps $\boldsymbol{x}$ to one of the classes $\{C_1, \ldots, C_K\}$.

How should we choose $f(\boldsymbol{x})$?

- 0–1 loss function:

$$\ell(y, f(\boldsymbol{x})) = \begin{cases} 0 & y = f(\boldsymbol{x}) \\ 1 & y \neq f(\boldsymbol{x}) \end{cases}$$

Equivalently,

$$\ell(y, f(\boldsymbol{x})) = \mathbb{1}\{y \neq f(\boldsymbol{x})\}$$

Classification and statistical learning

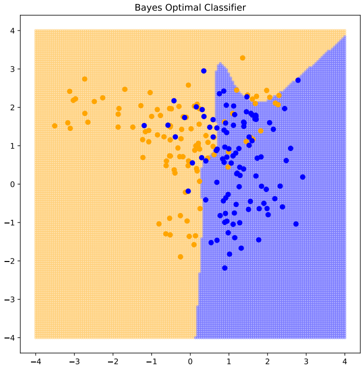
Optimal rule $f^*(\boldsymbol{x})$:

$$\begin{aligned} f^*(\boldsymbol{x}) &= \underset{f}{\operatorname{argmin}}\, \mathbb{E}_{\mathsf{x},\mathsf{y}\sim p(\boldsymbol{x},y)}[\ell(\mathsf{y}, f(\boldsymbol{x}))] \\ &= \underset{f}{\operatorname{argmin}}\, \mathbb{E}_{\mathsf{x}\sim p_{\mathsf{x}}}\left[\mathbb{E}_{\mathsf{y}\sim p_{\mathsf{y}|\mathsf{x}}}[\ell(\mathsf{y}, f(\boldsymbol{x}))]\right] \\ &= \underset{f}{\operatorname{argmin}}\, \mathbb{E}_{\mathsf{x}\sim p_{\mathsf{x}}}\left[\mathbb{E}_{\mathsf{y}\sim p_{\mathsf{y}|\mathsf{x}}}[\mathbb{1}\{\mathsf{y} \neq f(\boldsymbol{x})\}]\right] \end{aligned}$$

Classification and statistical learning

$$f^*(\boldsymbol{x}) = \operatorname*{argmin}_f \mathbb{E}_{\mathsf{x}\sim p_{\mathsf{x}}}\left[\mathbb{E}_{\mathsf{y}\sim p_{\mathsf{y}|\mathsf{x}}}[\mathbb{1}\{\mathsf{y} \neq f(\boldsymbol{x})\}]\right]$$

Classification and statistical learning



Classification and statistical learning

Optimal classification: Bayes' classifier

$$f^*(\boldsymbol{x}) = \underset{i \in [K]}{\operatorname{argmax}}\, p(\mathsf{y} = i | \boldsymbol{x})$$

k-nearest neighbors: Approximates the optimal solution as

$$f^*(\boldsymbol{x}) = \underset{j \in [K]}{\operatorname{argmax}}\, \frac{1}{k} \sum_{\boldsymbol{x}_i \in \mathcal{N}_k(\boldsymbol{x})} \mathbb{1}\{y_i = j\}$$

Curse of dimensionality (regression)

Optimal solution:

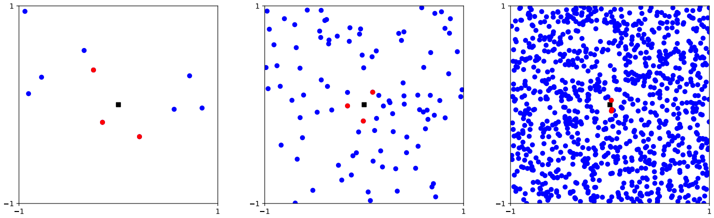
$$\hat{y}(x) = f^*(\boldsymbol{x}) = \mathbb{E}_{\mathsf{y}|\mathbf{x}}[\mathsf{y}|\boldsymbol{x}]$$

k-nearest neighbors: Approximates the optimal solution as

$$\hat{y}(\boldsymbol{x}) = \frac{1}{k} \sum_{\boldsymbol{x}_i \in \mathcal{N}_k(\boldsymbol{x})} y_i(\boldsymbol{x}_i)$$

Can we accurately approximate the optimal solution by considering a very large training set? How large?

Curse of dimensionality



3-nearest neighbors

- Blue and red: Training data points $\boldsymbol{x}_i \in \mathcal{X} = [-1,1]^2$ $(10, 100, 1000)$
- Black: New data point
- Red: 3 nearest neighbors

As N increases:

$$\hat{y}(\boldsymbol{x}) = \frac{1}{k} \sum_{\boldsymbol{x}_i \in \mathcal{N}_k(\boldsymbol{x})} y_i(\boldsymbol{x}_i) \to \mathbb{E}[\mathsf{y}|\boldsymbol{x}]$$

Curse of dimensionality

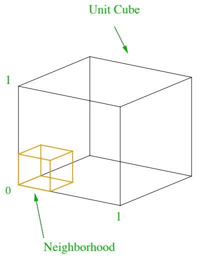
. . . but k-nearest neighbors (and other local methods) do not work well with high-dimensional inputs!

Curse of dimensionality: number of points exponential in number of dimensions!

1. Nearest neighbors not so close to $\boldsymbol{x}$
2. k-NNs of $\boldsymbol{x}$ closer to the boundary of $\mathcal{X}$
3. Need a prohibitive number of training samples to densely sample $\mathcal{X} \in \mathbb{R}^p$

(see “The elements of statistical learning,” Section 2.5, for 2 and 3)

For large p, the nearest neighbors are not so close

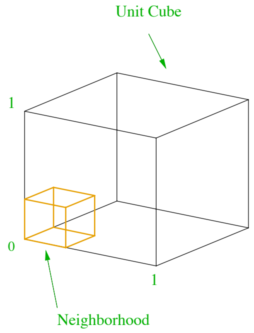


k-nearest neighbors to data with training points uniformly distributed in a p-dimensional unit hypercube, $\mathcal{X} = [0,1]^p$.

Want to estimate the density of class labels around a test point $\boldsymbol{x}$ by growing a hypercube around $\boldsymbol{x}$ until we capture a fraction ρ of the data points.

Expected length of the side of the smallest hypercube containing a fraction ρ of the data points:

For large p, the nearest neighbors are not so close



- Estimate based on 10% of the data ($\rho = 1/10$): $r_{10}(1/10) = 0.8$
- Estimate based on 1% of the data ($\rho = 1/100$): $r_{10}(1/100) = 0.63$

k-nearest neighbors is not local in higher dimensions! $\longrightarrow$ Far-away data points may not be good predictors for the behavior of the function at $\boldsymbol{x}$.

Probabilistic models for learning

Up to now we assumed

$$y = \tilde{f}(\boldsymbol{x})$$

Typically assume a probabilistic model of the form

$$y = \tilde{f}(\boldsymbol{x}) + \varepsilon$$

with ε independent of $\mathbf{x}$ and $\mathbb{E}[\varepsilon] = 0$.

Hence,

$$\mathbb{E}[\mathsf{y}|\mathbf{x} = \boldsymbol{x}] = \tilde{f}(\boldsymbol{x})$$

How do we effectively use the training data $\mathcal{D}$ to guide the learning of $f(\boldsymbol{x})$?

Probabilistic models for learning

The space of all possible regression functions $f(\boldsymbol{x})$ is enormous!

Idea: Consider a parametric form of $f(\boldsymbol{x})$, $f_{\boldsymbol{\theta}}(\boldsymbol{x})$, with parameters $\boldsymbol{\theta}$.

Example: Linear regression,

$$\begin{aligned}\hat{y} &= \sum_{j=0}^{p} x_j \beta_j + \varepsilon \\ &= \boldsymbol{x}^{\mathsf{T}} \boldsymbol{\beta} + \varepsilon\,.\end{aligned}$$

Linear function of the parameters $\beta_0, \ldots, \beta_p$ and of the input variables $x_1, \ldots, x_p$.

Probabilistic models for learning

More in general we may consider

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i=1}^{M} \theta_i \phi_i(\boldsymbol{x})$$

ϕ_j: basis functions or basis expansion
$f_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{w})$: linear basis expansion

Resulting model is much richer, but still a linear function in $\boldsymbol{\theta}$!

Maximum likelihood learning

- Assume $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ fixed
- We want to learn $\boldsymbol{\theta}$

Before (least-squares regression):

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{N} (y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2$$

Maximum likelihood: select $\boldsymbol{\theta}$ for which the training set $\mathcal{D}$ has the maximum probability of being observed.

Maximum likelihood learning

Choose $\boldsymbol{\theta}$ that maximizes

$$p(y_{\mathcal{D}}|\boldsymbol{x}_{\mathcal{D}}, \boldsymbol{\theta}) = \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})$$

or, equivalently, the log-likelihood (LL) function

$$\ln p(y_{\mathcal{D}}|\boldsymbol{x}_{\mathcal{D}}, \boldsymbol{\theta}) = \sum_{i=1}^{N} \ln p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})$$

Maximum likelihood learning

$$\begin{aligned}\boldsymbol{\theta}_{\mathsf{ML}} &= \operatorname*{argmax}_{\boldsymbol{\theta}} \ln p(y_{\mathcal{D}}|\boldsymbol{x}_{\mathcal{D}}, \boldsymbol{\theta}) \\ &= \operatorname*{argmin}_{\boldsymbol{\theta}} - \ln p(y_{\mathcal{D}}|\boldsymbol{x}_{\mathcal{D}}, \boldsymbol{\theta}) \\ &= \operatorname*{argmin}_{\boldsymbol{\theta}} - \sum_{i=1}^{N} \ln p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta}) \\ &= \operatorname*{argmin}_{\boldsymbol{\theta}} - \frac{1}{N} \sum_{i=1}^{N} \ln p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})\end{aligned}$$

If $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, from $y = f_{\boldsymbol{\theta}}(\boldsymbol{x}) + \varepsilon$:

$$\mathsf{y}_i|\boldsymbol{x}_i, \boldsymbol{\theta} \sim \mathcal{N}(y_i; f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \sigma^2)$$

Maximum likelihood learning

$$
\begin{aligned}
\boldsymbol{\theta}_{\mathsf{ML}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} -\frac{1}{N}\sum_{i=1}^{N} \ln p(y_i|\boldsymbol{x}_i,\boldsymbol{\theta}) \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} -\frac{1}{N}\sum_{i=1}^{N} \ln \mathcal{N}(y_i; f_{\boldsymbol{\theta}}(\boldsymbol{x}_i),\sigma^2)
\end{aligned}
$$

Maximum likelihood learning

The need of structured regression models

Goal: Choose a function $f \in \mathcal{F}$ that minimizes a given loss function $L(\hat{y}) = L(f; \mathcal{D})$ based on training set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}$, $i \in [N]$.

Example:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \ \text{RSS} \triangleq \sum_{i=1}^{N} (y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2 .$$

Observation: If $\mathcal{F}$ set of all possible functions, can make $\text{RSS}(f(\boldsymbol{x})) = 0$ (any $f(\boldsymbol{x})$ that passes through the training points)

... but not all will generalize well to new data.

Need to impose some constraints on $f(\boldsymbol{x})$!

The need of structured regression models

Which constraints should we impose?

- Restrict to parametric functions $f_{\boldsymbol{\theta}}$ (linear regression: $\mathcal{F}$ family of all linear functions)
- Smoother functions
- ...

Three classes of structured regression models:

- Roughness penalty
- Kernel methods
- Basis functions and dictionary methods

Class 1: Roughness penalty

Assuming a measure of "niceness" (e.g., smoothness) $J(f)$,

$$f(\boldsymbol{x}) = \underset{f \in \mathcal{F} : L(f;\mathcal{D})=0}{\operatorname{argmin}} \; J(f)$$

Example: smoothing splines
For one-dimensional data $x \in [0,1]$, $\mathcal{F}$ is the family of all twice-differentiable functions, and we choose $J(f)$ as

$$J(f) = \int_0^1 \left(f''(x)\right)^2 \mathrm{d}x$$

Can relax requirement that $L(f;\mathcal{D}) = 0$ via

$$f(\boldsymbol{x}) = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \; L(f;\mathcal{D}) + \lambda J(f)$$

Regularization methods: Trade-off between loss and smoothness

Class 2: Kernel methods

Estimate regression (or classification) function in a local neighborhood

- Need to specify nature of local neighborhood and class of functions used for local fit

Simplest form: Nadaraya-Watson weighted average,

$$f(\boldsymbol{x}) = \frac{\sum_{i=1}^{N} K_\lambda(\boldsymbol{x}, \boldsymbol{x}_i) y_i}{\sum_{i=1}^{N} K_\lambda(\boldsymbol{x}, \boldsymbol{x}_i)}$$

$K_\lambda(\boldsymbol{x}, \boldsymbol{b}_i)$: Kernel function; assigns weights to $\boldsymbol{x}_i$ depending on closeness to $\boldsymbol{x}$

Example 1: k-nearest neighbors

$$K_k(\boldsymbol{x}, \boldsymbol{x}_i) = \mathbb{1}\{\|\boldsymbol{x}_i - \boldsymbol{x}\| \leq \|\boldsymbol{x}_{(k)} - \boldsymbol{x}\|\}$$

$\boldsymbol{x}_{(k)}$: k-th closest input in data set to $\boldsymbol{x}$

Example 2: Gaussian kernel

$$K_\lambda(\boldsymbol{x}, \boldsymbol{x}_i) = \frac{1}{\lambda} \exp\left(\frac{\|\boldsymbol{x} - \boldsymbol{x}_i\|^2}{2\lambda}\right)$$

Class 2: Kernel methods

More in general,

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{N} K_\lambda(\boldsymbol{x}, \boldsymbol{x}_i)(y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2$$

Class 3: Basis functions and dictionary methods

f modeled as a linear expansion of basis functions:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i=1}^{M} \theta_i \phi_i(\boldsymbol{x})$$

The bias–variance trade-off

- k-NN method: k, the number of nearest neighbors
- Linear models: M, the model order (the number of basis functions)
- Roughness penalty methods: λ, the weight of the penalty term

Parameters control the capacity to fit data

Higher capacity (higher complexity) $\longrightarrow$ fit training data more accurately

. . . but unlikely to generalize well

Low capacity $\longrightarrow$ cannot capture all variations present on data and may generalize poorly

The bias–variance trade-off

Idea: Divide expected prediction error into its components (bias and variance for the squared error loss; approximation and estimation error for general case).

We consider:

$$y = \tilde{f}(\boldsymbol{x}) + \varepsilon,$$

with

- $\varepsilon \sim \mathcal{N}(0, \sigma^2) \longrightarrow \mathbb{E}[\mathsf{y}|\boldsymbol{x}] = \tilde{f}(\boldsymbol{x})$
- $\mathbf{x} = \boldsymbol{x}$ fixed
- square loss function $\ell(\mathsf{y}, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$
- Large number of datasets drawn from $p(y, \boldsymbol{x})$

Expected prediction (generalization) error:

$$\mathsf{Err}(y, f(\boldsymbol{x})) = \mathbb{E}_{\mathcal{D},\mathsf{x},\mathsf{y}}[\ell(\mathsf{y}, f(\mathbf{x}))]$$

The bias–variance trade-off

$$\mathsf{Err}(y, f(\boldsymbol{x})) = \mathbb{E}_{\mathcal{D},\mathbf{x},\mathsf{y}}[\ell(\mathsf{y}, f(\mathbf{x}))]$$

For $\mathbf{x} = \boldsymbol{x}$ fixed:

$$\begin{aligned}
\mathsf{Err}(\boldsymbol{x}) &= \mathbb{E}_{\mathsf{y}|\mathbf{x},\mathcal{D}}[\ell(\mathsf{y}, f(\boldsymbol{x}))|\mathbf{x} = \boldsymbol{x}] \\
&= \mathbb{E}_{\mathsf{y}|\mathbf{x}}\mathbb{E}_{\mathcal{D}}[\ell(\mathsf{y}, f(\boldsymbol{x}))|\mathbf{x} = \boldsymbol{x}] \\
&= \mathbb{E}_{\mathsf{y}|\mathbf{x}}\mathbb{E}_{\mathcal{D}}\left[(\mathsf{y} - f(\boldsymbol{x}))^2|\boldsymbol{x}\right] \\
&= \mathbb{E}_{\mathsf{y}|\mathbf{x}}\mathbb{E}_{\mathcal{D}}\left[(\mathsf{y} - \mathbb{E}_{\mathsf{y}|\mathbf{x}}[\mathsf{y}|\boldsymbol{x}] + \mathbb{E}_{\mathsf{y}|\mathbf{x}}[\mathsf{y}|\boldsymbol{x}] - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})] + \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})] - f(\boldsymbol{x}))^2\right] \\
&= \mathbb{E}_{\mathsf{y}|\mathbf{x}}\left[(\mathsf{y} - \mathbb{E}_{\mathsf{y}|\mathbf{x}}[\mathsf{y}|\boldsymbol{x}])^2\right] + (\mathbb{E}_{\mathsf{y}|\mathsf{x}}[\mathsf{y}|\boldsymbol{x}] - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})])^2 + \mathbb{E}_{\mathsf{y}|\mathbf{x}}\mathbb{E}_{\mathcal{D}}\left[(\mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})] - f(\boldsymbol{x}))^2\right] \\
&= \mathbb{E}_{\mathsf{y}|\mathbf{x}}\left[(\mathsf{y} - \tilde{f}(\boldsymbol{x}))^2\right] + (\tilde{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})])^2 + \mathbb{E}_{\mathsf{y}|\mathbf{x}}\mathbb{E}_{\mathcal{D}}\left[(\mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})] - f(\boldsymbol{x}))^2\right] \\
&= \mathsf{Var}[\mathsf{y}|\boldsymbol{x}] + (\tilde{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})])^2 + \mathbb{E}_{\mathcal{D}}\left[(\mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})] - f(\boldsymbol{x}))^2\right]
\end{aligned}$$

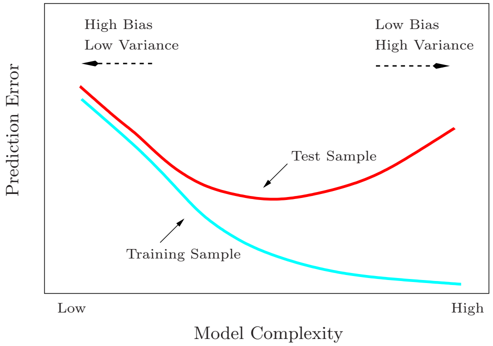
The bias–variance trade-off

$$\begin{aligned}\mathsf{Err}(\boldsymbol{x}) &= \mathsf{Var}[\mathsf{y}|\boldsymbol{x}] + (\tilde{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})])^2 + \mathbb{E}_{\mathcal{D}}\left[(\mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})] - f(\boldsymbol{x}))^2\right] \\ &= \sigma^2 + (\underbrace{\tilde{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})]}_{\text{bias}})^2 + \mathsf{Var}_{\mathcal{D}}[f(\boldsymbol{x})]\end{aligned}$$

Bias-variance decomposition

- σ^2: Irreducible error due to data randomness
- Bias: Approximation error due to the limited flexibility model
- Variance: Estimation error; sensitivity/variability of model due to randomness in $\mathcal{D}$

The bias–variance trade-off



Bias-variance trade-off:

- High capacity models: low bias, high variance (overfitting)
- Low capacity models: high bias, low variance (underfitting)

The bias–variance trade-off or least squares regression

$$\mathrm{Err}(\boldsymbol{x}) = \sigma^2 + \sigma^2 \boldsymbol{x}^{\mathsf{T}} \mathbb{E}_{\mathcal{D}}\left[(\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\right]\boldsymbol{x}$$

and

$$\begin{aligned}\mathrm{Err}(y, f(\boldsymbol{x})) &= \mathbb{E}_{\mathbf{x}}[\mathrm{Err}(\mathbf{x})] \\ &= \sigma^2 + \sigma^2 \mathbb{E}_{\mathbf{x},\mathcal{D}}\left[\mathbf{x}^{\mathsf{T}}(\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\mathbf{x}\right]\end{aligned}$$

Tutorial exercice!

Reading

“The elements of statistical learning,”
Chapters 1 and 2