Chapter 5

Machine Learning Basics

Learning Algorithms

- (Mitchell, 1997) A computer program is said to learn from *experience* E with respect to some class of *tasks* T and *performance measure* P, if its performance at tasks in T, as measured by P, improves with experience E
- Example: Linear Regression

$$\begin{tikzcd}
 & & & & & y \arrow[d] & \\
\boldsymbol{x} \arrow[r] & \boxed{\text{Rep.}} \arrow[r] & \phi(\boldsymbol{x}) \arrow[r] & \boxed{\text{Model}} \arrow[r] & \hat{y} \arrow[r] & \textcircled{-} \arrow[r] & \text{Cost}
\end{tikzcd}$$

  – Task T: To predict y from $\boldsymbol{x}$ by outputting

$$\hat{y} = \boldsymbol{w}^T \phi(\boldsymbol{x}) = \phi(\boldsymbol{x})^T \boldsymbol{w}$$

  – Experience E: To learn $\boldsymbol{w}$ by minimizing, over a training set

$(\boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})})$,

$$\text{MSE}^{(\text{train})} = \frac{1}{m^{(\text{train})}} \|\hat{\boldsymbol{y}}^{(\text{train})} - \boldsymbol{y}^{(\text{train})}\|_2^2$$

where

$$\hat{\boldsymbol{y}}^{(\text{train})} = \boldsymbol{\Phi}^{(\text{train})}\boldsymbol{w}, \quad \boldsymbol{\Phi}^{(\text{train})} = \begin{bmatrix} \phi(\boldsymbol{x}_0^{(\text{train})T}) \\ \phi(\boldsymbol{x}_1^{(\text{train})T}) \\ \vdots \\ \phi(\boldsymbol{x}_{m-1}^{(\text{train})T}) \end{bmatrix}$$

$$\boldsymbol{y}^{(\text{train})} = (y_0^{(\text{train})}, y_1^{(\text{train})}, \ldots, y_{m-1}^{(\text{train})})$$

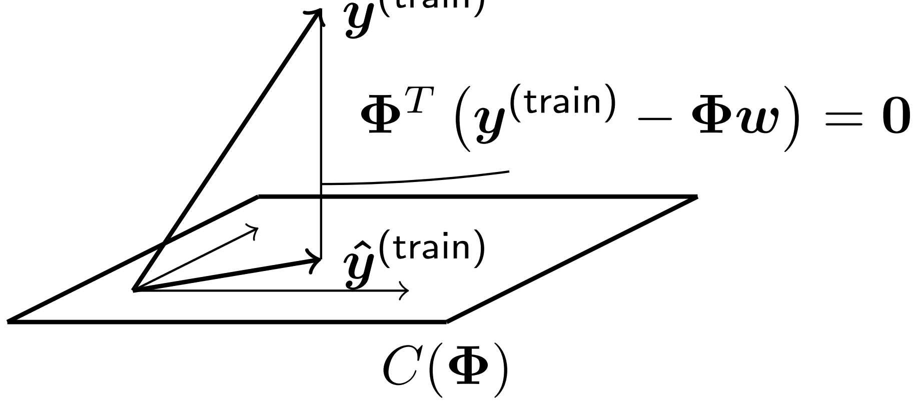
– Performance P: To measure mean squared error on a test set $(\boldsymbol{X}^{(\text{test})}, \boldsymbol{y}^{(\text{test})})$, i.e.,

$$\text{MSE}^{(\text{test})} = \frac{1}{m^{(\text{test})}} \|\hat{\boldsymbol{y}}^{(\text{test})} - \boldsymbol{y}^{(\text{test})}\|_2^2$$

- To minimize $\mathsf{MSE}_{\text{train}}$, $\boldsymbol{w}$ can be solved by setting

$$\nabla_{\boldsymbol{w}} \mathsf{MSE}^{(\text{train})} = \mathbf{0}$$

- A geometrical view is to solve $\hat{\boldsymbol{y}}^{(\text{train})}$ as the projection of $\boldsymbol{y}^{(\text{train})}$ onto the column space of $\boldsymbol{\Phi}^{(\text{train})}$


$\boldsymbol{y}^{(\text{train})}$
$\boldsymbol{\Phi}^T\left(\boldsymbol{y}^{(\text{train})} - \boldsymbol{\Phi}\boldsymbol{w}\right) = \mathbf{0}$
$\hat{\boldsymbol{y}}^{(\text{train})}$
$C(\boldsymbol{\Phi})$

- We then have

$$\boldsymbol{w} = \left(\boldsymbol{\Phi}^{(\text{train})T}\boldsymbol{\Phi}^{(\text{train})}\right)^{-1}\boldsymbol{\Phi}^{(\text{train})T}\boldsymbol{y}^{\text{train}}$$

- The present model can be extended to include a bias term b

$$\hat{y} = \boldsymbol{w}^T \phi(\boldsymbol{x}) + b = \tilde{\boldsymbol{w}}^T \tilde{\phi}(\boldsymbol{x}),$$

with

$$\tilde{\boldsymbol{w}} = \begin{bmatrix} \boldsymbol{w} \\ b \end{bmatrix}, \ \tilde{\phi}(\boldsymbol{x}) = \begin{bmatrix} \phi(\boldsymbol{x}) \\ 1 \end{bmatrix}$$

- Prediction with a polynominal of degree 2:

$$\hat{y} = w_2 x^2 + w_1 x^1 + b = \tilde{\boldsymbol{w}}^T \tilde{\phi}(x),$$

where

$$\tilde{\boldsymbol{w}} = \begin{bmatrix} w_2 \\ w_1 \\ b \end{bmatrix}, \ \tilde{\phi}(x) = \begin{bmatrix} \phi_2(x) \\ \phi_1(x) \\ 1 \end{bmatrix} = \begin{bmatrix} x^2 \\ x^1 \\ 1 \end{bmatrix}$$

Generalization

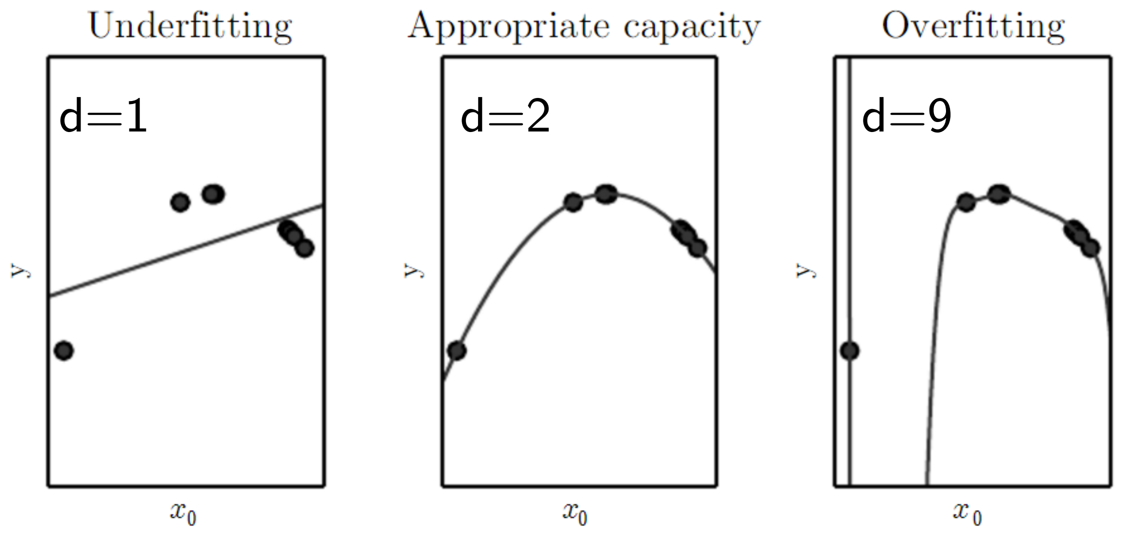
- The model's ability to perform well on *new, previously unseen* inputs
- Generalization error is defined to be the *expected value of the error* on a *new input* and is typically estimated by measuring the performance on a *test set* collected separately from the *training set*
- Examples $(\boldsymbol{x}, y)$ in the training and test sets are assumed to be drawn independently from the same distribution, $p_{\text{data}}(\boldsymbol{x}, y)$
- **Bayes error:** Minimum generalization error achieved by an oracle model having knowledge of $p_{\text{data}}(\boldsymbol{x}, y)$
- E.g.: if $y = \boldsymbol{w}^T \phi(\boldsymbol{x}) + \varepsilon$ and ε is Gaussian noise independent of $\boldsymbol{x}$, Bayes error $=$ Var(ε) with MSE measure

Training Error and Test Error

- (Pitfall) At first glance, the expected test error should be the same as the expected training error for a given model, because the data in these sets are drawn from the same distribution
- In practice, we sample the training set, use it to train the model, and then sample the test set to measure test error
- Generally, test error $\geq$ training error

Underfitting vs. Overfitting

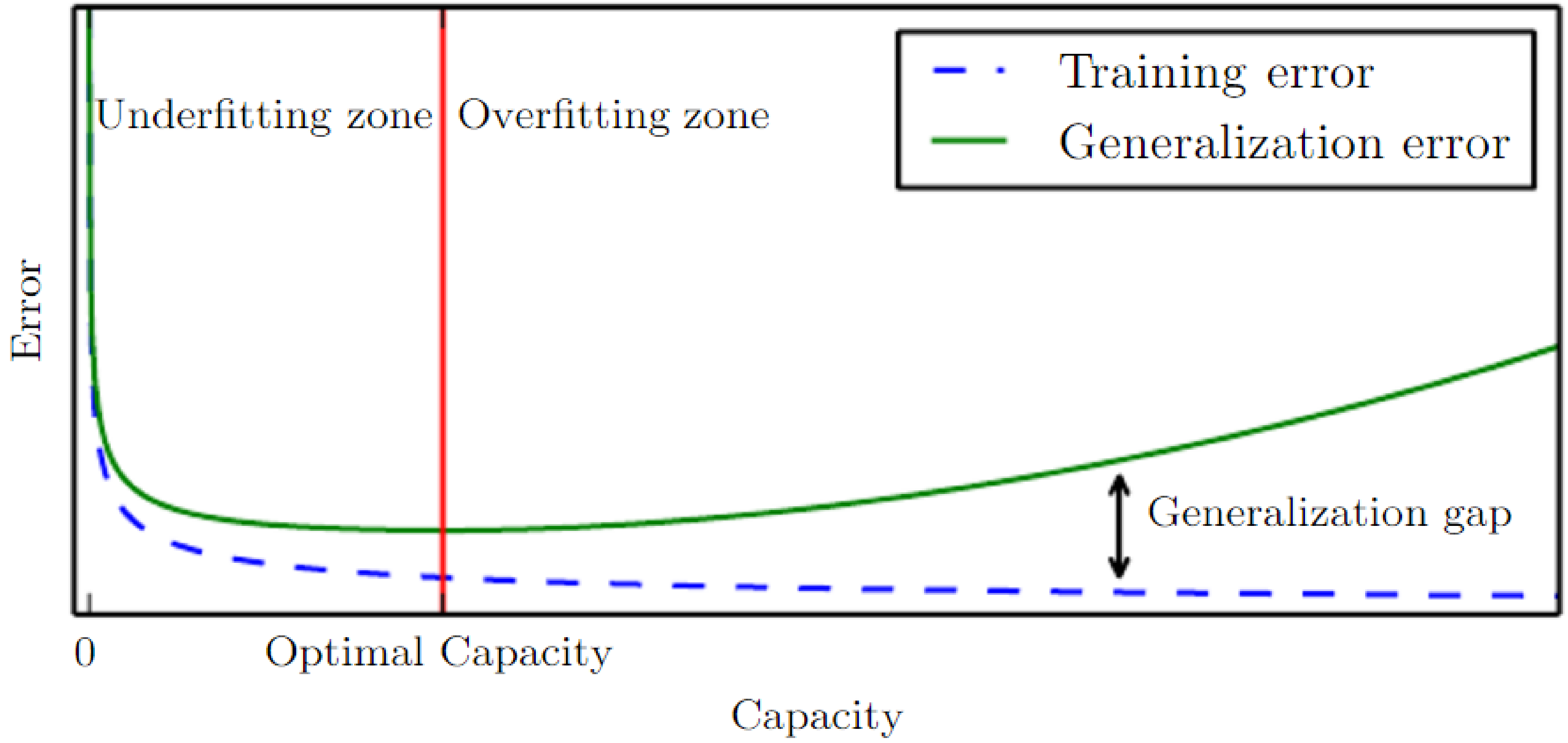
- Two objectives to achieve in designing a model
  1. Make training error small to avoid **underfitting**
  2. Make gap between training and test error small to avoid **overfitting**
- Trade-off can be made by altering the *model capacity*, which refers broadly to a model's ability to fit a wide varieties of functions
- Example: Fitting a polynomial model to quadratic data



$\hat{y} = \sum_{i=1}^{d} w_i x_0^i + b$

Capacity and Error

Typical relationship between capacity and error

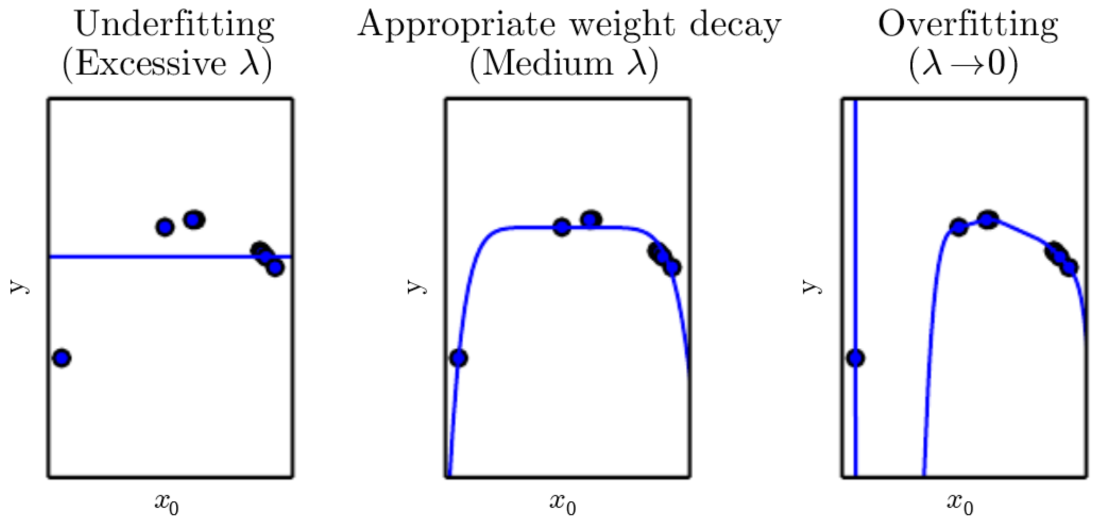


Regularization

- Modification made to the learning algorithm to reduce generalization error (usually at the cost of higher training error)
- Example: To include *weight decay* in the training criterion

$$J(\boldsymbol{w}) = \text{MSE}^{(\text{train})} + \lambda \boldsymbol{w}^T \boldsymbol{w}$$

where λ controls preference for small $\boldsymbol{w}$ and is determined a priori



A degree-9 polynomial model fitted to quadratic data

Estimators

- **Point estimation:** To provide a single estimate of some quantity from observing independent and identically distributed (i.i.d.) samples
  - Consider m i.i.d samples $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$ drawn from a Bernoulli distribution with mean θ

$$P(x^{(i)};\theta) = \theta^{x^{(i)}}(1-\theta)^{1-x^{(i)}},\ x^{(i)} = \{1,0\}$$

  - The *sample mean* can be used to give a point estimate of θ

$$\hat{\theta}_m = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

- A point estimator $\hat{\boldsymbol{\theta}}_m$ of a parameter $\boldsymbol{\theta}$ is any function of the observed samples $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(m)}\}$

$$\hat{\boldsymbol{\theta}}_m = g(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(m)})$$

- $\boldsymbol{\theta}$ is fixed but unknown from the **frequentist** viewpoint
- $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(m)}$ are seen samples of a random variable
- As a result, $\hat{\boldsymbol{\theta}}_m$ is a random variable

Bias

- The bias of the estimator $\hat{\boldsymbol{\theta}}_m$ is defined as

$$\text{bias}(\hat{\boldsymbol{\theta}}_m) = E(\hat{\boldsymbol{\theta}}_m) - \boldsymbol{\theta},$$

  where the expectation $E(\cdot)$ is taken w.r.t. $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(m)}$
  - $\hat{\boldsymbol{\theta}}_m$ is *unbiased* if bias($\hat{\boldsymbol{\theta}}_m$)=0
  - $\hat{\boldsymbol{\theta}}_m$ is *asymptotically unbiased* if $\lim_{m\to\infty} \text{bias}(\hat{\boldsymbol{\theta}}_m) = 0$

- In the Bernoulli example, the sample mean is an unbiased estimator

$$E\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right] = \frac{1}{m}\sum_{i=1}^{m} E\left[x^{(i)}\right] = \theta$$

Consistency

- An estimator $\hat{\boldsymbol{\theta}}_m$ is said to be consistent *in probability* if

$$\lim_{m \to \infty} P\left(\left|\hat{\boldsymbol{\theta}}_m - \boldsymbol{\theta}\right| > \varepsilon\right) = 0,\ \varepsilon > 0$$

- Consistency ensures that the bias of the estimator diminishes as the number of data samples grows
- In the Bernoulli example, the sample mean is consistent
  - Chebyshev's inequality

$$P(|X - \mu_X| > \varepsilon) \leq \frac{\sigma_x^2}{\varepsilon^2}$$

  - $\hat{\theta}_m$ has mean θ, variance $\theta(1-\theta)/m$

$$P(|\hat{\theta}_m - \theta| > \varepsilon) \leq \frac{\theta(1-\theta)}{m\varepsilon^2} \Rightarrow \lim_{m \to \infty} P(|\hat{\theta}_m - \theta| > \varepsilon) = 0$$

Estimators for Gaussian Distribution

- Gaussian probability density function

$$\mathcal{N}(x^{(i)}; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \text{exp} \left(-\frac{1}{2} \frac{(x^{(i)} - \mu)^2}{\sigma^2} \right)$$

- Sample mean (unbiased)

$$\hat{\mu}_m = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

- Sample variance (asymptotically unbiased)

$$\hat{\sigma}_m^2 = \frac{1}{m} \sum_{i=1}^{m} \left(x^{(i)} - \hat{\mu}_m \right)^2$$

  – It can be shown that $E[\hat{\sigma}_m^2] = (m-1)\sigma^2/m$
  – Unbiased sample variance $\tilde{\sigma}_m^2 = m\hat{\sigma}_m^2/(m-1)$

Variance of the Estimator

- Variance of the estimator indicates how much the estimator varies as a function of the samples; and its squared root is called standard error
- Example: Standard error of the sample mean $\hat{\mu}_m$

$$\mathsf{SE}(\hat{\mu}_m) = \sqrt{\mathsf{Var}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right]} = \frac{\sigma}{\sqrt{m}}$$

where σ is usually estimated by $\sqrt{\tilde{\sigma}_m^2}$

- By the central limit theorem,

$$\frac{\hat{\mu}_m - 0}{\mathsf{SE}(\hat{\mu}_m)} \sim \mathcal{N}(0,1)$$

- The 95 percent confidence interval can thus be derived as

$$(\hat{\mu}_m - 1.96\mathsf{SE}(\hat{\mu}_m), \hat{\mu}_m + 1.96\mathsf{SE}(\hat{\mu}_m))$$

- In experiments, it is common to say algorithm A performs better than B if its 95 percent upper bound of the test error is smaller than the lower bound of B's test error

Maximum Likelihood (ML) Estimation

- Consider examples $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(m)}$ drawn independently from a distribution $p_{\mathsf{model}}(\boldsymbol{x}; \boldsymbol{\theta})$, with parameter $\boldsymbol{\theta}$ being fixed but unknown

- The maximum likelihood estimator $\boldsymbol{\theta}_{\mathsf{ML}}$ for $\boldsymbol{\theta}$ is defined as

$$\begin{aligned}\boldsymbol{\theta}_{\mathsf{ML}} &= \arg\max_{\boldsymbol{\theta}} p_{\mathsf{model}}(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(m)}; \boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\mathsf{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\mathsf{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})\end{aligned}$$

  where $\sum_{i=1}^{m} \log p_{\mathsf{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$ is the *log-likelihood function* of $\boldsymbol{\theta}$

- The sum over examples can be written as expectation w.r.t. the empirical data distribution $\hat{p}_{\text{data}}$

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} E_{\boldsymbol{x}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x};\boldsymbol{\theta})$$

- Maximizing the log-likelihood amounts to minimizing the dissimilarity between the empirical data distribution $\hat{p}_{\text{data}}$ (defined by the training set) and the model distribution p_{model} in terms of KL divergence:

$$D_{KL}(\hat{p}_{\text{data}}\|p_{\text{model}}) = E_{\boldsymbol{x}\sim\hat{p}_{\text{data}}}[\log \hat{p}_{\text{data}}(\boldsymbol{x}) - \log p_{\text{model}}(\boldsymbol{x})]$$

- Remarks
  - In information theory, $D_{KL}(\hat{p}_{\text{data}}\|p_{\text{model}})$ denotes the extra amount of information (in bits when using $\log_2$ base) needed to send a message $\boldsymbol{x}$ drawn from $\hat{p}_{\text{data}}$ with a code optimized for messages drawn from p_{model}
  - $D_{KL}(\hat{p}_{\text{data}}\|p_{\text{model}})$ also corresponds exactly to the *cross-entropy* between $\hat{p}_{\text{data}}$ and p_{model} in machine learning parlance

Conditional Log-Likelihood Estimation

- The ML estimator generalized to learn a conditional probability $p_{\text{model}}(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$ in order to predict $\boldsymbol{y}$ given $\boldsymbol{x}$

$$\begin{aligned}\boldsymbol{\theta}_{\text{ML}} &= \arg\max_{\boldsymbol{\theta}} p_{\text{model}}(\boldsymbol{Y}|\boldsymbol{X};\boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} E_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{\text{data}}(\boldsymbol{x},\boldsymbol{y})} \log p_{\text{model}}(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})\end{aligned}$$

- Example: Linear Regression

$$y = \hat{y}(\boldsymbol{x};\boldsymbol{w}) + \varepsilon = \boldsymbol{w}^T\phi(\boldsymbol{x}) + \varepsilon$$

  where $\varepsilon \sim \mathcal{N}(0,\sigma^2)$ is independent of $\boldsymbol{x}$

  – It can be shown that $p(y|\boldsymbol{x};\boldsymbol{w}) = \mathcal{N}(y;\hat{y}(\boldsymbol{x};\boldsymbol{w}),\sigma^2)$

  – The conditional log-likelihood of $p_{\text{model}}(\boldsymbol{Y}|\boldsymbol{X};\boldsymbol{w})$ is given by

$$\sum_{i=1}^{m} \log p(y^{(i)}|\boldsymbol{x}^i;\boldsymbol{w}) = -m\log\sigma - \frac{m}{2}\log(2\pi) - \sum_{i=1}^{m}\frac{\|\hat{y}^i - y^{(i)}\|^2}{2\sigma^2}$$

– Maximizing the log-likelihood w.r.t. $\boldsymbol{w}$ leads to the same problem of minimizing

$$\text{MSE}^{(\text{train})} = \frac{1}{m^{(\text{train})}}\|\hat{\boldsymbol{y}}^{(\text{train})} - \boldsymbol{y}^{(\text{train})}\|_2^2$$

– Therefore, the data model $y = \hat{y}(\boldsymbol{x};\boldsymbol{w}) + \varepsilon$ provides an alternative view of the linear regression problem

Bayesian Statistics

- Assume the true parameter $\boldsymbol{\theta}$ is a random variable governed by a prior probability distribution $p(\boldsymbol{\theta})$
- The goal is to infer the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{X})$ of $\boldsymbol{\theta}$ by combining the prior $p(\boldsymbol{\theta})$ and the data likelihood $p(\boldsymbol{X}|\boldsymbol{\theta})$ via Bayes' rule:

$$p(\boldsymbol{\theta}|\boldsymbol{X}) = \frac{p(\boldsymbol{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{X})}$$

- Example: Linear Regression

$$y = \hat{y}(\boldsymbol{x};\boldsymbol{w}) + \varepsilon = \boldsymbol{w}^T\phi(\boldsymbol{x}) + \varepsilon$$

  – Again, $p(y|\boldsymbol{x},\boldsymbol{w}) = \mathcal{N}(y;\hat{y}(\boldsymbol{x};\boldsymbol{w}),\sigma^2)$, where presently $\sigma^2 = 1$
  – Assume $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w};\boldsymbol{\mu_0},\boldsymbol{\Lambda_0})$
  – Consider $\boldsymbol{x}$ to be deterministic data

– The posterior distribution $p(\boldsymbol{w}|\boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})})$ is given by

$$\begin{aligned}
&p(\boldsymbol{w}|\boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})})\\
&\propto p(\boldsymbol{w})p(\boldsymbol{y}^{(\text{train})}|\boldsymbol{X}^{(\text{train})}, \boldsymbol{w})\\
&\propto \text{exp}(-\frac{1}{2}(\boldsymbol{w}-\boldsymbol{u}_0)^T\boldsymbol{\Lambda}_0^{-1}(\boldsymbol{w}-\boldsymbol{u}_0))\text{exp}(-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\Phi}\boldsymbol{w})^T(\boldsymbol{y}-\boldsymbol{\Phi}\boldsymbol{w}))\\
&\propto \text{exp}(-\frac{1}{2}(\boldsymbol{w}-\boldsymbol{u}_m)^T\boldsymbol{\Lambda}_m^{-1}(\boldsymbol{w}-\boldsymbol{u}_m))\\
&= \mathcal{N}(\boldsymbol{w}; \boldsymbol{\mu}_m, \boldsymbol{\Lambda}_m)
\end{aligned}$$

where

$$\begin{aligned}
\boldsymbol{\Phi} &= \boldsymbol{\Phi}^{(\text{train})}\\
\boldsymbol{\Lambda}_m &= (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \boldsymbol{\Lambda}_0^{-1})^{-1}\\
\boldsymbol{u}_m &= \boldsymbol{\Lambda}_m(\boldsymbol{\Phi}^T\boldsymbol{y}^{(\text{train})} + \boldsymbol{\Lambda}_0^{-1}\boldsymbol{u}_0)
\end{aligned}$$

– When $\boldsymbol{\mu}_0 = \mathbf{0}$ and $\boldsymbol{\Lambda}_0 = \frac{1}{\lambda}\boldsymbol{I}$, $\boldsymbol{\mu}_m$ leads to the same solution as minimizing

$$J(\boldsymbol{w}) = \text{MSE}^{(\text{train})} + \lambda \boldsymbol{w}^T \boldsymbol{w}$$

- Given $p(\boldsymbol{w}|\boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})})$, one can infer the probability distribution of $y^{(\text{new})}$ at unseen $\boldsymbol{x}^{(\text{new})}$ by

$$\begin{aligned}
& p(y^{(\text{new})}|\boldsymbol{x}^{(\text{new})}, \boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})}) \\
&= \int p(y^{(\text{new})}, \boldsymbol{w}|\boldsymbol{x}^{(\text{new})}, \boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})}) d\boldsymbol{w} \\
&= \int p(\boldsymbol{w}|\boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})}) p(y^{(\text{new})}|\boldsymbol{x}^{(\text{new})}, \boldsymbol{w}) d\boldsymbol{w} \\
&= \int \mathcal{N}(\boldsymbol{w}; \boldsymbol{\mu}_m, \boldsymbol{\Lambda}_m) \mathcal{N}(y^{(\text{new})}; \hat{y}(\boldsymbol{x}^{(\text{new})}; \boldsymbol{w}), 1) d\boldsymbol{w}
\end{aligned}$$

- Equivalently, this is to ask about the distribution of

$$y^{(\text{new})} = \hat{y}(\boldsymbol{x}^{(\text{new})}; \boldsymbol{w}) + \varepsilon = \boldsymbol{w}^T \phi(\boldsymbol{x}^{(\text{new})}) + \varepsilon$$

given $(\boldsymbol{X}^{(\text{train})}, \boldsymbol{Y}^{(\text{train})})$, with

$$p(\boldsymbol{w}|\boldsymbol{X}^{(\text{train})}, \boldsymbol{Y}^{(\text{train})}) = \mathcal{N}(\boldsymbol{w}; \boldsymbol{\mu}_m, \boldsymbol{\Lambda}_m)$$
$$p(\varepsilon|\boldsymbol{X}^{(\text{train})}, \boldsymbol{Y}^{(\text{train})}) = \mathcal{N}(\varepsilon; 0, 1)$$

and $\boldsymbol{w}$, ε being conditionally independent

$$\begin{aligned} & p(\boldsymbol{w}, \varepsilon|\boldsymbol{X}^{(\text{train})}, \boldsymbol{Y}^{(\text{train})}) \\ & = p(\boldsymbol{w}|\boldsymbol{X}^{(\text{train})}, \boldsymbol{Y}^{(\text{train})}) p(\varepsilon|\boldsymbol{X}^{(\text{train})}, \boldsymbol{Y}^{(\text{train})}) \end{aligned}$$

- We further recognize that

1. $\boldsymbol{w}^T \phi(\boldsymbol{x}^{(\text{new})})|\boldsymbol{X}^{(\text{train})}, \boldsymbol{Y}^{(\text{train})}$ is a Gaussian

$$\begin{aligned} & \boldsymbol{w}^T \phi(\boldsymbol{x}^{(\text{new})})|\boldsymbol{X}^{(\text{train})}, \boldsymbol{Y}^{(\text{train})} \\ & \sim \mathcal{N}(\boldsymbol{\mu}_m^T \phi(\boldsymbol{x}^{(\text{new})}), \phi(\boldsymbol{x}^{(\text{new})})^T \boldsymbol{\Lambda}_m \phi(\boldsymbol{x}^{(\text{new})})) \end{aligned}$$

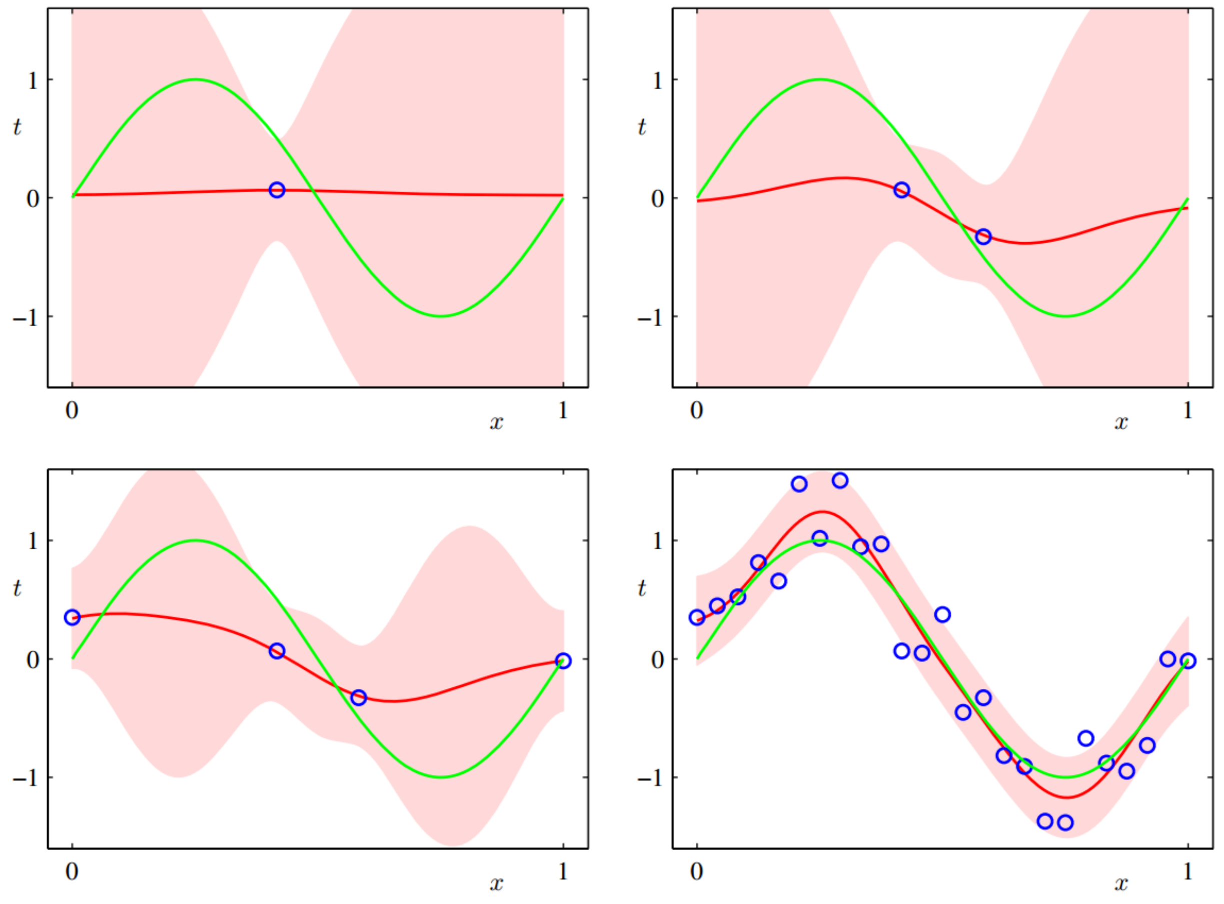
2. $\boldsymbol{w}^T \phi(\boldsymbol{x}^{(\text{new})}) + \varepsilon|\boldsymbol{X}^{(\text{train})}, \boldsymbol{Y}^{(\text{train})}$ (sum of two conditionally

independent Gaussian's) is another Gaussian

$$\begin{aligned} &\boldsymbol{w}^T\phi(\boldsymbol{x}^{(\text{new})}) + \varepsilon | \boldsymbol{X}^{(\text{train})}, \boldsymbol{Y}^{(\text{train})} \\ &\sim \mathcal{N}(\boldsymbol{\mu}_m^T\phi(\boldsymbol{x}^{(\text{new})}), \phi(\boldsymbol{x}^{(\text{new})})^T\boldsymbol{\Lambda}_m\phi(\boldsymbol{x}^{(\text{new})}) + 1) \end{aligned}$$

- This concludes our evaluation for $p(y^{(\text{new})}|\boldsymbol{x}^{(\text{new})}, \boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})})$

Bayesian Statistics



Ground truth (Green); $\mu_{y^{(\text{new})}}$ (Red); Data (Blue); $\sigma_{y^{(\text{new})}}$ (Pink)

Maximum A Posteriori (MAP) Estimation

- The full Bayesian treatment may sometimes be intractable
- To offer a point estimate in the Bayesian framework, we usually choose

$$\begin{aligned}\boldsymbol{\theta}_{\mathsf{MAP}} &= \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\boldsymbol{X}) \\ &= \arg\max_{\boldsymbol{\theta}} \frac{p(\boldsymbol{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{X})} \\ &= \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{X}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})\end{aligned}$$

- Many regularized estimation strategies (e.g. ML+weight decay) can be interpreted as making the MAP inference, if the regularization term (e.g. weight decay) admits an explanation of $\log p(\boldsymbol{\theta})$
- Example: Linear Regression

$$\boldsymbol{w}_{\mathsf{MAP}} = \arg\max_{\boldsymbol{w}} p(\boldsymbol{w}|\boldsymbol{X}^{(\mathsf{train})}, \boldsymbol{y}^{(\mathsf{train})})$$

$$= \arg\max_{\boldsymbol{w}} \mathcal{N}(\boldsymbol{w}; \boldsymbol{\mu}_m, \boldsymbol{\Lambda}_m)$$
$$= \boldsymbol{\mu}_m$$

- We may then choose the prediction model to be

$$\hat{y}(\boldsymbol{x}^{(\mathsf{new})}; \boldsymbol{w}) = \boldsymbol{w}_{\mathsf{MAP}}^T \phi(\boldsymbol{x}^{(\mathsf{new})}) = \boldsymbol{\mu}_m^T \phi(\boldsymbol{x}^{(\mathsf{new})}),$$

  which in the present case coincides with the mean of the posterior distribution $p(y^{(\mathsf{new})}|\boldsymbol{x}^{(\mathsf{new})}, \boldsymbol{X}^{(\mathsf{train})}, \boldsymbol{y}^{(\mathsf{train})})$

- From the earlier derivation and definitions,

$$\boldsymbol{\mu}_m = \boldsymbol{\Lambda}_m(\boldsymbol{\Phi}^T \boldsymbol{y}^{(\mathsf{train})} + \boldsymbol{\Lambda}_0^{-1} \boldsymbol{u}_0)$$

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi(\boldsymbol{x}_0^{(\mathsf{train})T}) \\ \phi(\boldsymbol{x}_1^{(\mathsf{train})T}) \\ \vdots \\ \phi(\boldsymbol{x}_{m-1}^{(\mathsf{train})T}) \end{bmatrix}, \ \boldsymbol{y}^{(\mathsf{train})} = \begin{bmatrix} y_0^{(\mathsf{train})} \\ y_1^{(\mathsf{train})} \\ \vdots \\ y_{m-1}^{(\mathsf{train})} \end{bmatrix}$$

- Assuming $\boldsymbol{\mu}_0 = 0$, we have

$$\begin{aligned}\hat{y}(\boldsymbol{x}^{(\text{new})}; \boldsymbol{w}) &= \phi(\boldsymbol{x}^{(\text{new})})^T \boldsymbol{\Lambda}_m \boldsymbol{\Phi}^T \boldsymbol{y}^{(\text{train})} \\ &= \sum_{i=0}^{m-1} \underbrace{\phi(\boldsymbol{x}^{(\text{new})})^T \boldsymbol{\Lambda}_m \phi(\boldsymbol{x}_i^{(\text{train})})} y_i^{(\text{train})} \\ &= \sum_{i=0}^{m-1} \underbrace{k(\boldsymbol{x}^{(\text{new})}, \boldsymbol{x}_i^{(\text{train})})}_{\text{Kernel}} y_i^{(\text{train})}\end{aligned}$$

where

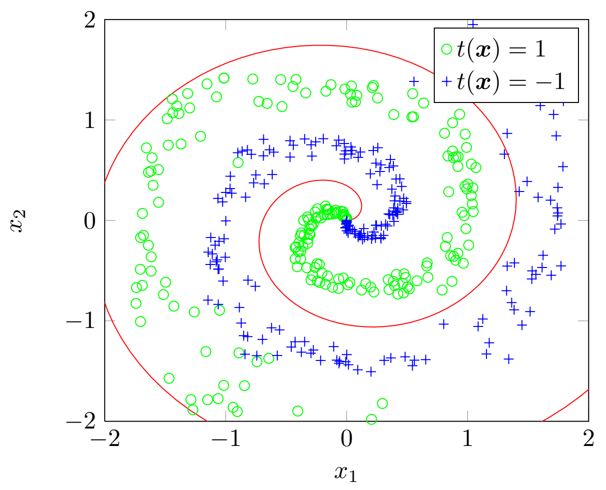
$$k(\boldsymbol{x}^{(\text{new})}, \boldsymbol{x}_i^{(\text{train})}) = \phi(\boldsymbol{x}^{(\text{new})})^T \boldsymbol{\Lambda}_m \phi(\boldsymbol{x}_i^{(\text{train})})$$

- It is seen that the prediction $\hat{y}(\boldsymbol{x}^{(\text{new})}; \boldsymbol{w})$ is a weighted combination of the training data $y_i^{(\text{train})}$ with weights determined by the kernel function $k(\cdot)$ measuring a certain type of distance between $\boldsymbol{x}^{(\text{new})}$ and $\boldsymbol{x}_i^{(\text{train})}$

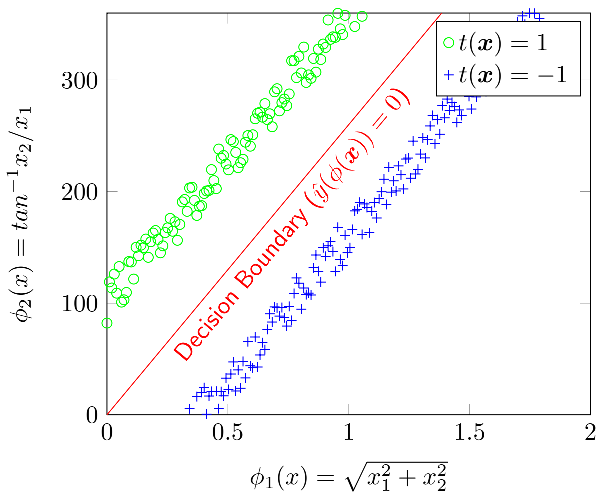
Support Vector Machines (SVM)

- One of the most influential approaches to supervised learning
- **Idea:** To find a hyperplane (in feature space) for classifying linearly separable training data according to the sign of $\hat{y}(\phi(\boldsymbol{x}))$

$$\hat{y}(\phi(\boldsymbol{x})) = \boldsymbol{w}^T \phi(\boldsymbol{x}) + b$$



raw data domain



feature domain

- The hyperplane, known as the decision boundary, is defined by

$$\hat{y}(\phi(\boldsymbol{x})) = \boldsymbol{w}^T\phi(\boldsymbol{x}) + b = 0$$

- $\boldsymbol{w}$ is a vector orthogonal to every vector in the decision boundary
- Any point $\phi(\boldsymbol{x})$ to the decision boundary has a distance

$$\frac{|\hat{y}(\phi(\boldsymbol{x}))|}{\|\boldsymbol{w}\|}$$

- **Maximum margin classifiers**: Maximizing the smallest distance between the decision boundary and any of the training samples $\phi(\boldsymbol{x}_n)$

$$\arg\max_{\boldsymbol{w},b}\min_n \frac{t_n\hat{y}(\phi(\boldsymbol{x}_n))}{\|\boldsymbol{w}\|} = \arg\max_{\boldsymbol{w},b}\left\{\frac{1}{\|\boldsymbol{w}\|}\min_n[t_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n)+b)]\right\}$$

  where $t_n \in \{-1, 1\}$ is the ground-truth label associated with $\phi(\boldsymbol{x}_n)$

- Noting that $\boldsymbol{w}, b$ can be scaled simultaneously $(\boldsymbol{w} \to \kappa\boldsymbol{w}, b \to \kappa b)$ without changing the resulting distance, we can choose a κ such that

$t_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n)+b)=1$ for the closest point $\phi(\boldsymbol{x}_n)$ to the boundary

- This allows us to reformulate the problem as a *constrained optimization problem*

$$\min \frac{1}{2}\|\boldsymbol{w}\|_2^2, \text{ subject to}$$

$$t_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n)+b)\geq 1, \forall n$$

- Using the method of Lagrange multipliers, which will be introduced next, the solution for $\hat{y}(\phi(\boldsymbol{x}))$ can be solved as

$$\hat{y}(\phi(\boldsymbol{x}))=\sum_{i=1}^{N} a_n t_n \underbrace{\phi(\boldsymbol{x}_n)^T\phi(\boldsymbol{x})} + b = \sum_{i=1}^{N} a_n t_n \underbrace{k(\boldsymbol{x}_n,\boldsymbol{x})}_{\text{Kernel}} + b$$

where $a_n \geq 0, \forall n$ and most of them are zero

- *Support vectors* refer to those $\phi(\boldsymbol{x}_n)$'s whose $a_n \neq 0$

Constrained Optimization

- To find the maximal/minimal value of $f(\boldsymbol{x})$ for $\boldsymbol{x}$ (known as feasible points) in some set $\mathbb{S}$, e.g.

$$\begin{aligned}&\arg\min_{\boldsymbol{x}} f(\boldsymbol{x}),\ \text{subject to}\\ &g^{(i)}(x) = 0,\ i = 1, \ldots, m\\ &h^{(j)}(x) \le 0,\ j = 1, \ldots, n\end{aligned}$$

  where $\mathbb{S} = \{\boldsymbol{x} | \forall i, g^{(i)}(x) = 0,\ \forall j, h^{(j)} \le 0\}$ are defined by m *equality constraints* and n *inequality constraints*

- The **Karush-Kuhn-Tucker (KKT)** approach obtains a solution by solving the unconstrained optimization of the Lagrangian function:

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \ge \mathbf{0}} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha})$$

where

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_{j=1}^{n} \alpha_j h^{(j)}(\boldsymbol{x})$$

and $\boldsymbol{\lambda}$ and $\boldsymbol{\alpha}$ are termed Lagrange multipliers

- The optimal solution satisfies (necessary conditions)
  1. $\nabla L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = \mathbf{0}$
  2. All constraints on $\boldsymbol{x}$ and Lagrange multipliers are met
  3. Complementary slackness: $\boldsymbol{\alpha} \odot \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0}$, i.e. $\alpha_j = 0$ for $h^{(j)}(\boldsymbol{x}) < 0$ (inactive) and $\alpha_j \geq 0$ for $h^{(j)}(\boldsymbol{x}) = 0$ (active)
- It is easy to see that when any constraint is violated,

$$\max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq \mathbf{0}} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = \infty,$$

which excludes infeasible points from being considered

- On the other hand, when the constraints are all satisfied,

$$\max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq \mathbf{0}} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\boldsymbol{x}),$$

which ensures the optimum within feasible points is unchanged

- Example: To solve $\boldsymbol{w}, b$

$$\min \frac{1}{2} \|\boldsymbol{w}\|_2^2, \text{ subject to}$$

$$t_n(\boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b) \geq 1, \forall n$$

- We set to zero the gradient w.r.t.

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\boldsymbol{w}\|_2^2 + \sum_{j=1}^{n} \alpha_j (1 - t_n(\boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b))$$

and arrive at

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \mathbf{0} \Rightarrow \boldsymbol{w} = \sum_{j=1}^{n} \alpha_j t_n \phi(\boldsymbol{x}_n)$$

$$\nabla_b L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = 0 \Rightarrow \sum_{j=1}^{n} \alpha_j t_n = \mathbf{0}$$

- At this point, we already have the form of the optimal $\boldsymbol{w}$
- To solve for $\boldsymbol{\alpha}$, we can substitute this $\boldsymbol{w}$ back to the Lagrangian

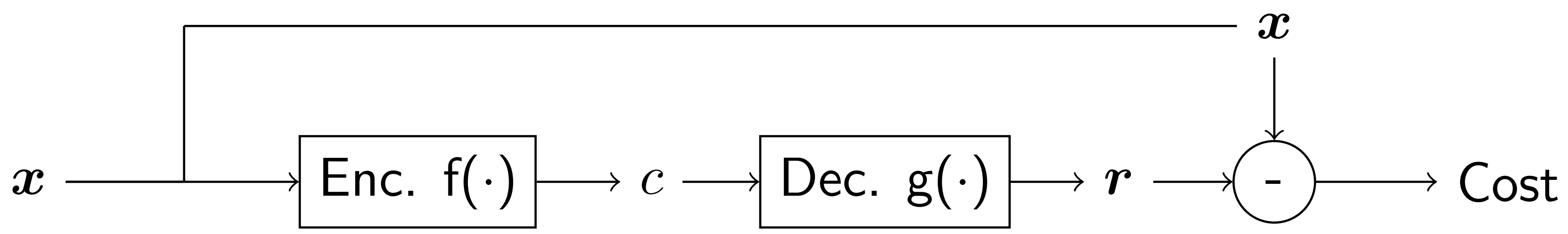
$$\max_{\boldsymbol{\alpha}} \sum_{j=1}^{n} \alpha_j - \frac{1}{2} \sum_{p=1}^{m} \sum_{q=1}^{m} \alpha_p \alpha_q t_p t_q \phi(\boldsymbol{x}_p)^T \phi(\boldsymbol{x}_q), \text{ s.t.}$$

$$\alpha_i \geq 0, \forall i \text{ and } \sum_{j=1}^{n} \alpha_j t_n = \mathbf{0}$$

- How to solve b? (To be continued)

Principle Component Analysis (PCA)

- An *unsupervised learning* algorithm that learns a data representation



  - Input: $\boldsymbol{x} \in \mathbb{R}^n$
  - Representation of input: $\boldsymbol{c} \in \mathbb{R}^l$
  - Decoder: $g(\boldsymbol{c}) = \boldsymbol{D}\boldsymbol{c}$ with $\boldsymbol{D}$ composed of orthonormal columns
  - Cost: $\|\boldsymbol{x} - g(\boldsymbol{c})\|_2^2$
  - Encoder has the form $f(\boldsymbol{x}) = \boldsymbol{D}^T\boldsymbol{x}$ when Cost is minimized

- Objective: Given training data $\boldsymbol{X}^{(\text{train})}$, we wish to find $\boldsymbol{D}$

$$\arg\min_{\boldsymbol{D}} \|\boldsymbol{X}^{(\text{train})} - \boldsymbol{X}^{(\text{train})}\boldsymbol{D}\boldsymbol{D}^T\|_F^2, \text{ s.t. } \boldsymbol{D}^T\boldsymbol{D} = \boldsymbol{I}$$

- Recall that

$$\boldsymbol{X}^{(\text{train})} = \begin{bmatrix} \boldsymbol{x}_0^{(\text{train})T} \\ \boldsymbol{x}_1^{(\text{train})T} \\ \vdots \\ \boldsymbol{x}_{m-1}^{(\text{train})T} \end{bmatrix}, \ \|\boldsymbol{A}\|_F^2 = \text{Tr}\{\boldsymbol{A}\boldsymbol{A}^T\} = \sum_{i,j} A_{i,j}^2$$

- By simple algebra, we have

$$\begin{aligned} &\|\boldsymbol{X}^{(\text{train})} - \boldsymbol{X}^{(\text{train})}\boldsymbol{D}\boldsymbol{D}^T\|_F^2 \\ &= \text{Tr}(\boldsymbol{X}^{(\text{train})}\boldsymbol{X}^{(\text{train})T}) - \text{Tr}(\boldsymbol{X}^{(\text{train})}\boldsymbol{D}\boldsymbol{D}^T\boldsymbol{X}^{(\text{train})T}) \end{aligned}$$

where the first term has nothing to do with $\boldsymbol{D}$

- The initial problem then simplifies to

$$\arg\max_{\boldsymbol{D}} \text{Tr}(\boldsymbol{X}^{(\text{train})}\boldsymbol{D}\boldsymbol{D}^T\boldsymbol{X}^{(\text{train})T}), \ \text{s.t.} \ \ \boldsymbol{D}^T\boldsymbol{D} = \boldsymbol{I}$$

- Observing that the Trace operator has the property

$$\mathsf{Tr}(\boldsymbol{ABC}) = \mathsf{Tr}(\boldsymbol{BCA}) = \mathsf{Tr}(\boldsymbol{CAB})$$

(as long as all matrix multiplications are allowed), we arrive at

$$\arg\max_{\boldsymbol{D}} \mathsf{Tr}(\boldsymbol{D}^T \boldsymbol{X}^{(\mathsf{train})T} \boldsymbol{X}^{(\mathsf{train})} \boldsymbol{D}), \text{ s.t. } \boldsymbol{D}^T \boldsymbol{D} = \boldsymbol{I}$$

- By a further application of Singular Value Decomposition to

$$\boldsymbol{X}^{(\mathsf{train})}_{m \times n} = \boldsymbol{U}_{m \times m} \boldsymbol{\Sigma}_{m \times n} \boldsymbol{V}^T_{n \times n},$$

  – $\boldsymbol{U}$ is the eigenvector matrix of $\boldsymbol{X}^{(\mathsf{train})}_{m \times n} \boldsymbol{X}^{(\mathsf{train})T}_{m \times n}$ and satisfies

$$\boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{U}^T \boldsymbol{U} = \boldsymbol{I}_{m \times m}$$

  – $\boldsymbol{V}$ is the eigenvector matrix of $\boldsymbol{X}^{(\mathsf{train})T}_{m \times n} \boldsymbol{X}^{(\mathsf{train})}_{m \times n}$ and satisfies

$$\boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{V}^T \boldsymbol{V} = \boldsymbol{I}_{n \times n}$$

– $\mathbf{\Sigma}$ is the singular matrix given by

$$\left[\begin{array}{cccc:c} \sigma_1 & 0 & \dots & 0 & \\ 0 & \sigma_2 & \dots & 0 & \\ \vdots & \vdots & \ddots & 0 & \mathbf{0} \\ 0 & 0 & 0 & \sigma_r & \\ \hdashline & & \mathbf{0} & & \mathbf{0} \end{array}\right]_{m\times n}$$

- We see that the columns of $\boldsymbol{D}$ have an obvious choice of the l columns in $\boldsymbol{V}$ which corresponds to the largest l singular values

$$\arg\max_{\boldsymbol{D}} \mathsf{Tr}(\boldsymbol{D}^T\boldsymbol{V}\mathbf{\Sigma}^T\mathbf{\Sigma}\boldsymbol{V}^T\boldsymbol{D}), \text{ s.t. } \boldsymbol{D}^T\boldsymbol{D} = \boldsymbol{I}$$

where

$$\boldsymbol{\Sigma}^T\boldsymbol{\Sigma} = \left[\begin{array}{cccc:c} \sigma_1^2 & 0 & \dots & 0 & \\ 0 & \sigma_2^2 & \dots & 0 & \\ \vdots & \vdots & \ddots & 0 & \mathbf{0} \\ 0 & 0 & 0 & \sigma_r^2 & \\ \hdashline & & \mathbf{0} & & \mathbf{0} \end{array}\right]_{n\times n}$$

Gradient-based Learning

- Learning algorithms often seek to minimize/maximize some objective function w.r.t. the model parameter, e.g.

$$\arg\min_{\boldsymbol{w}} J(\boldsymbol{w}) = -E_{\boldsymbol{x},y\sim\hat{p}_{\mathsf{data}}}[p_{\mathsf{model}}(y|\boldsymbol{x})]$$

- Very often, there is no closed-form solution
- Gradient-based learning algorithms are thus called for to update estimates of the solution via an iterative procedure

Steepest Descent

- To decrease $J(\boldsymbol{w})$ in the direction in which it decreases the fastest

$$\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} - \epsilon \nabla_{\boldsymbol{w}} J(\boldsymbol{w}^{(n)})$$

  where ϵ controls the step size for each update

- The negative gradient $-\nabla_{\boldsymbol{w}} J(\boldsymbol{w}^{(n)})$ points to the direction in which $J(\boldsymbol{w})$ decreases the fastest at $\boldsymbol{w}^{(n)}$
  - To see this, we define the directional directive at $\boldsymbol{w}_0$ to be

$$\frac{\partial}{\partial \alpha} J(\boldsymbol{w}_0 + \alpha \boldsymbol{u})$$

  - It can then be evaluated as

$$\frac{\partial}{\partial \alpha} J(\boldsymbol{w}_0 + \alpha \boldsymbol{u}) = \boldsymbol{u}^T \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_0)$$

using the Taylor-1 approximation at $\boldsymbol{w}_0$

$$J(\boldsymbol{w}) \approx J(\boldsymbol{w}_0) + (\boldsymbol{w} - \boldsymbol{w}_0)^T \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_0)$$

- The unit vector $\boldsymbol{u}$ that points in the direction $-\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_0)$ yields a minimal directive among other unit vectors

- Instead of using a fixed ϵ, we can use line search to adapt its value to the curvature of $J(\boldsymbol{w})$ at $\boldsymbol{w}_0$ along $-\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_0)$
- This relies on approximating $J(\boldsymbol{w})$ at $\boldsymbol{w}_0$ with Taylor-2 approximation

$$J(\boldsymbol{w}) \approx J(\boldsymbol{w}_0) + (\boldsymbol{w} - \boldsymbol{w}_0)^T \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_0) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_0)^T \boldsymbol{H} (\boldsymbol{w} - \boldsymbol{w}_0)$$

where $\boldsymbol{H}$ is the Hessian matrix defined as

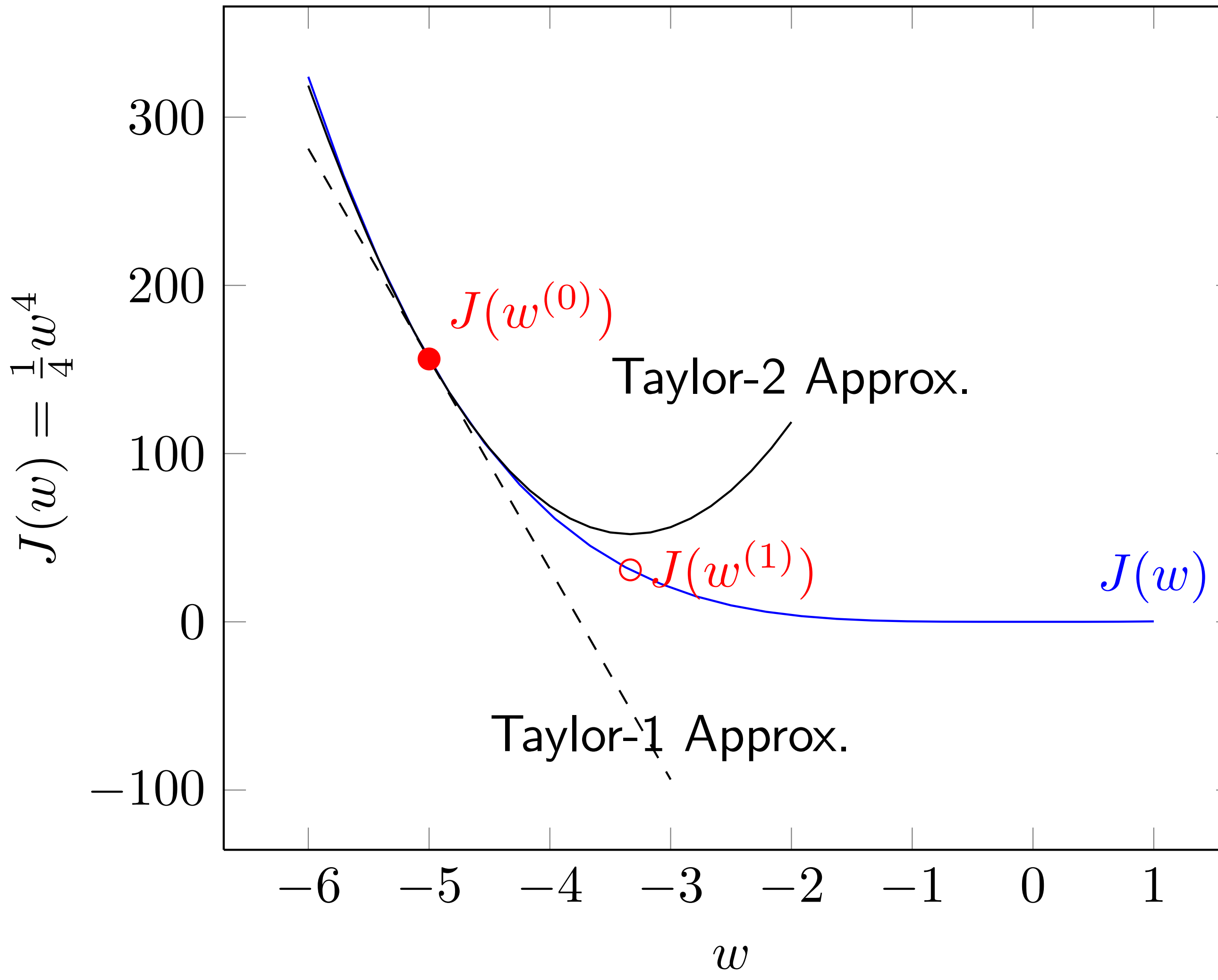
$$\boldsymbol{H}(J(\boldsymbol{w}_0))_{i,j} = \frac{\partial^2}{\partial w_i \partial w_j} J(\boldsymbol{w}_0)$$

- We wish to find an ϵ that maximizes

$$J(\boldsymbol{w}_0) - J(\boldsymbol{w}_0 - \epsilon \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_0))$$

- The optimal ϵ is given by

$$\epsilon^* = \frac{\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_0)^T \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_0)}{\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_0)^T \boldsymbol{H} \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_0)}$$


$J(w) = \frac{1}{4}w^4$
w
$J(w^{(0)})$
$J(w^{(1)})$
$J(w)$
Taylor-2 Approx.
Taylor-1 Approx.
300
200
100
0
-100
-6
-5
-4
-3
-2
-1
0
1

Newton's Method

- To solve for the critical point that minimizes $J(\boldsymbol{w})$ approximated by the Taylor-2 expansion at $\boldsymbol{w_0}$ as the new estimate

$$\arg\min_{\boldsymbol{w}} J(\boldsymbol{w}_0) + (\boldsymbol{w} - \boldsymbol{w}_0)^T \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_0) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_0)^T \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}_0)$$

- By setting the gradient w.r.t. $\boldsymbol{w}$ to zero, we have

$$\boldsymbol{w}^* = \boldsymbol{w}_0 - \boldsymbol{H}(J(\boldsymbol{w}_0))^{-1} \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_0)$$

- The iterative update of $\boldsymbol{w}$ then becomes

$$\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} - \boldsymbol{H}(J(\boldsymbol{w}^{(n)}))^{-1} \nabla_{\boldsymbol{w}} J(\boldsymbol{w}^{(n)})$$

Example

- We wish to find $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$, with

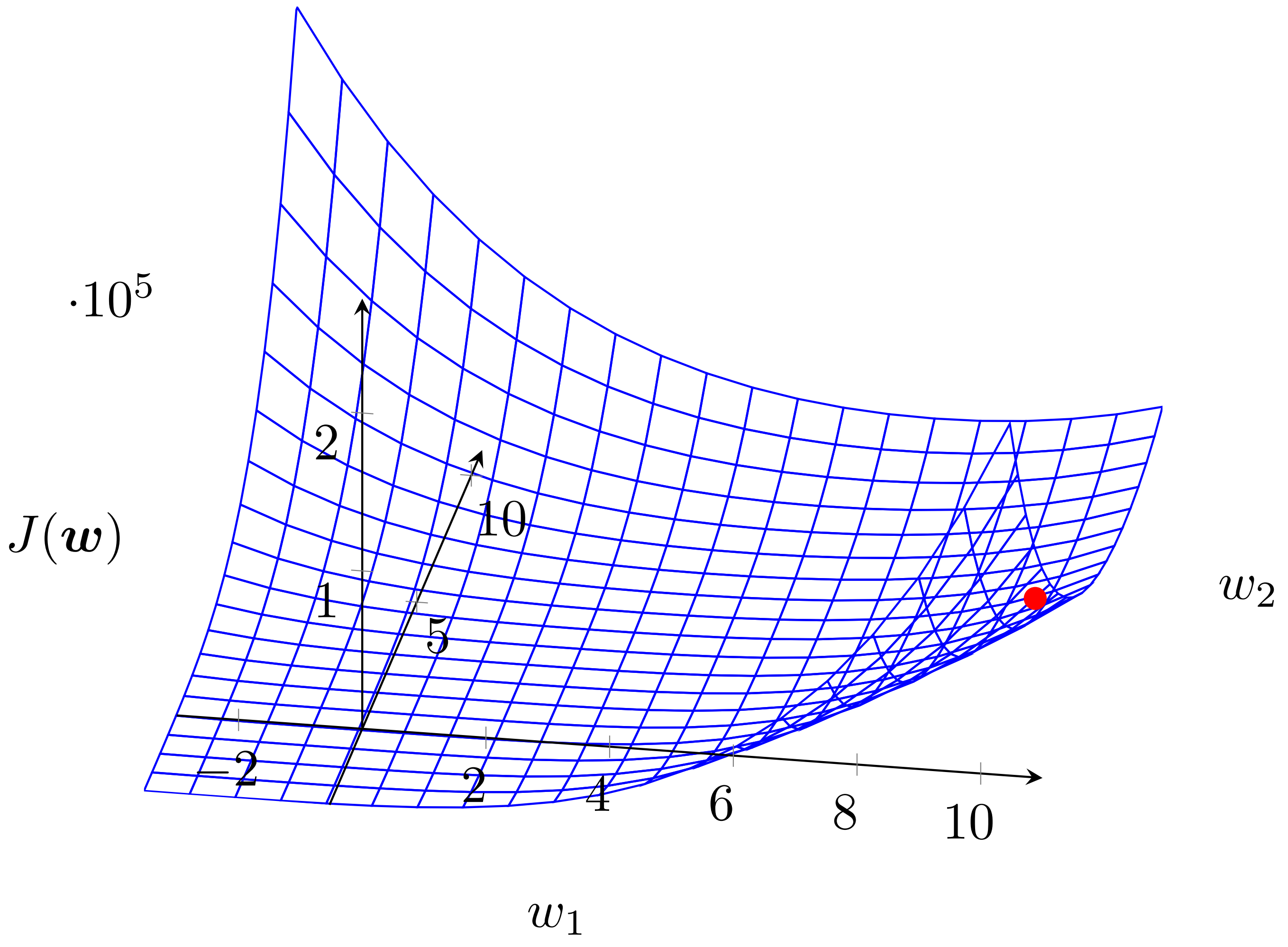
$$J(\boldsymbol{w}) = \frac{1}{4}(\boldsymbol{w}^T \boldsymbol{M} \boldsymbol{w})^2, \ \boldsymbol{M} = \begin{bmatrix} 6 & -4 \\ -4 & 6 \end{bmatrix}$$

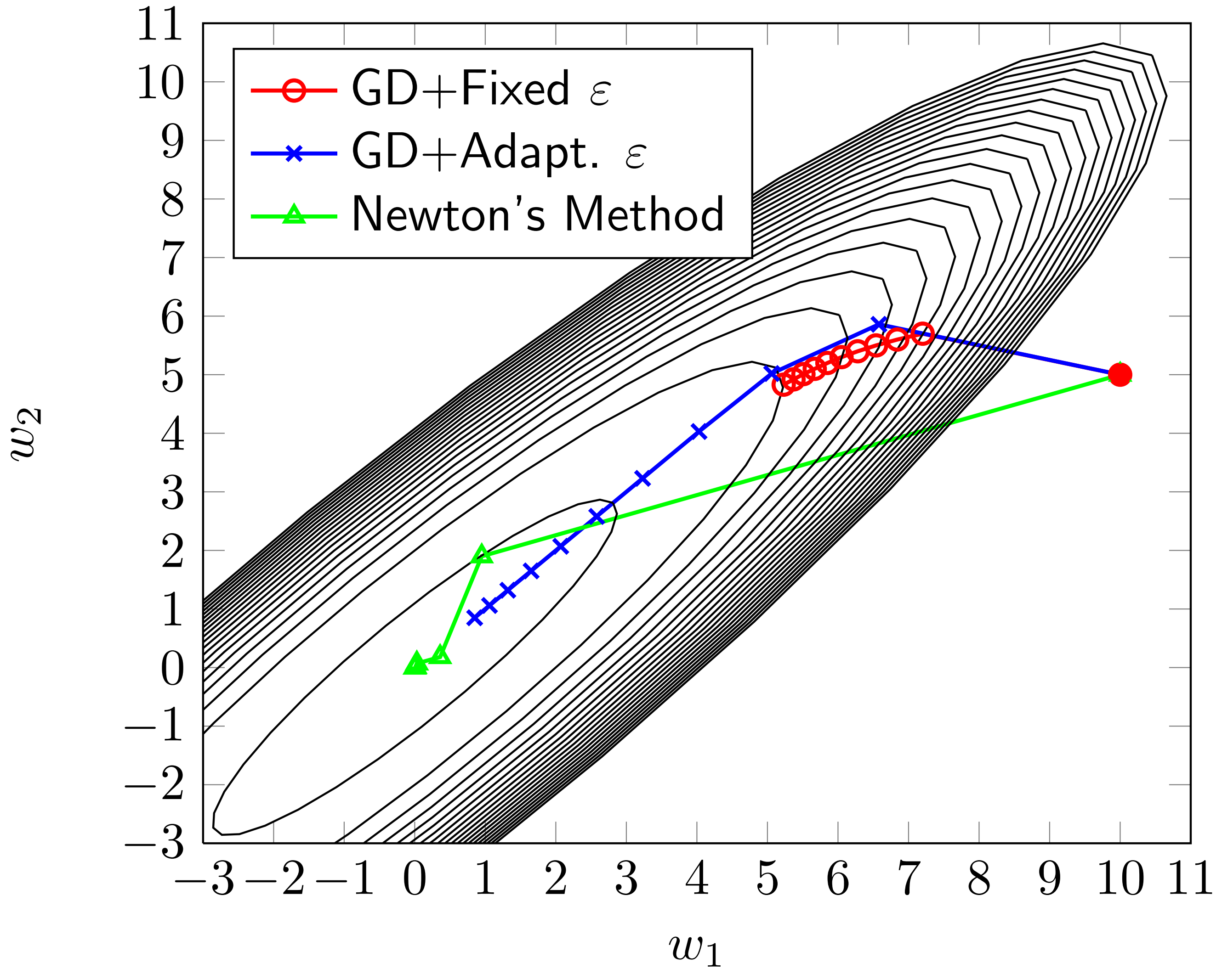
- $\boldsymbol{M}$ has the following pairs of eigenvalues and eigenvectors

$$\lambda_1 = 1 \rightarrow \boldsymbol{v}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \ \lambda_2 = 5 \rightarrow \boldsymbol{v}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

- The gradient and the Hessian matrix are given by

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = (\boldsymbol{w}^T \boldsymbol{M} \boldsymbol{w}) \boldsymbol{M} \boldsymbol{w}$$
$$\boldsymbol{H}\{J(\boldsymbol{w})\} = 2\boldsymbol{M}\boldsymbol{w}(\boldsymbol{M}\boldsymbol{w})^T + \boldsymbol{w}^T \boldsymbol{M} \boldsymbol{w} \boldsymbol{M}$$


$\cdot 10^5$
$J(\boldsymbol{w})$
2
1
10
5
w_2
−2
2
4
6
8
10
w_1



GD+Fixed ε
GD+Adapt. ε
Newton's Method
w_1
w_2

Review

- Overfitting vs. Underfitting
- Generalization
- Regularization
- Estimators (ML vs. MAP)
- Support Vector Machines (SVM)
- Constrained Optimization
- Principle Component Analysis (PCA)
- Gradient-based Learning