Introduction to AI for postgraduate students

Lecture Note 4

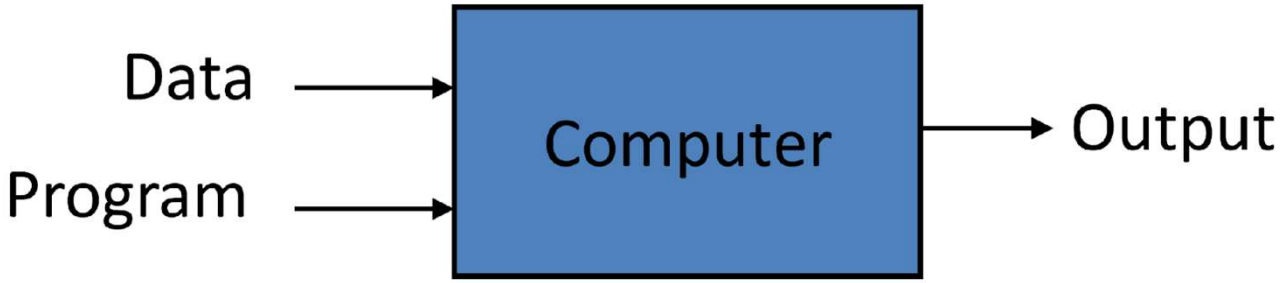
Machine Learning Basics

POSTECH

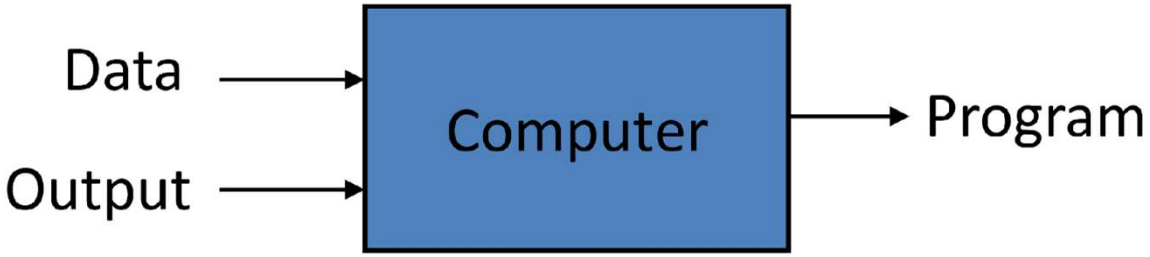
AiSLab

Approach in Machine Learning

- Traditional Programming



- Machine Learning
  - The ability to perform a task in a situation which has never been encountered before (**learning = generalization**)



Magic?

No, more like gardening

- **Seeds** = Algorithms
- **Nutrients** = Data
- **Gardener** = You
- **Plants** = Programs

Machine Learning Problem

- Learning = Improving with experience at some task
  - Improve over task T,
  - with respect to performance measure P,
  - based on experience E.
- E.g., Learn to play checkers
  - T : Play checkers
  - P : % of games won in world tournament
  - E : opportunity to play against self

- Tens of thousands of machine learning algorithms are existing

- Every machine learning algorithm has three components:
  - **Representation**
  - **Evaluation**
  - **Optimization**

Three Components of ML

- Representations
  - Decision trees
  - Sets of rules / Logic programs
  - Instances
  - Graphical models (Bayes/Markov nets)
  - Neural networks
  - Support vector machines
  - Model ensembles

- Evaluation
  - Accuracy
  - Precision and recall
  - Squared error
  - Likelihood
  - Posterior probability
  - Cost / Utility
  - Margin
  - Entropy
  - K-L divergence

- Optimization
  - Combinatorial optimization
    - ➢ E.g.: Greedy search
  - Convex optimization
    - ➢ E.g.: Gradient descent
  - Constrained optimization
    - ➢ E.g.: Linear programming

Types of Learning

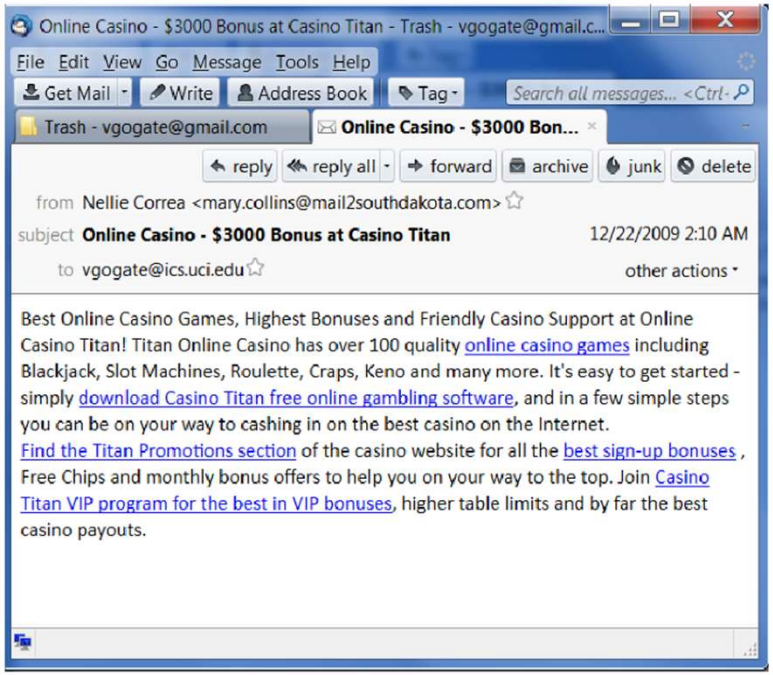
- Based on information available
  - Supervised–true labels provided
  - Reinforcement–Only indirect labels provided (reward/punishment)
  - Unsupervised–No feedback & no labels

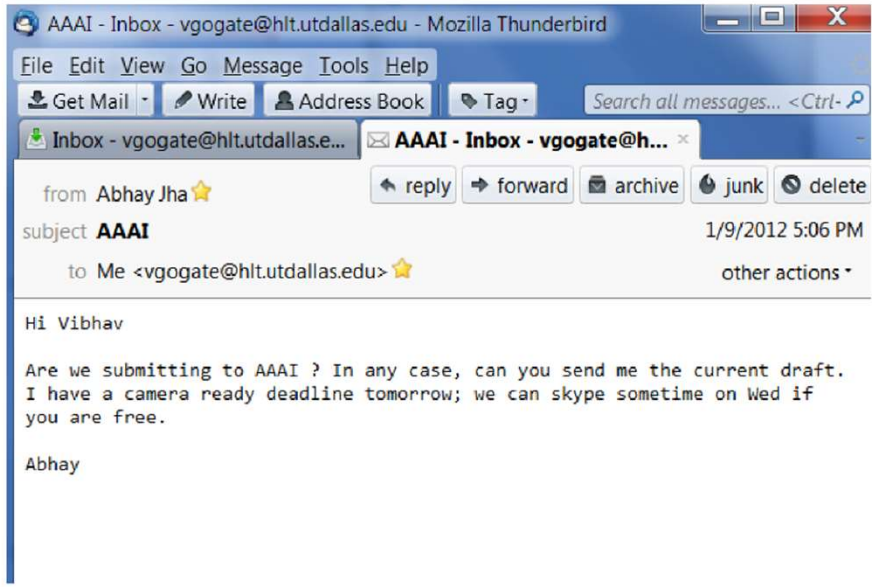
- Based on the role of the learner
  - Passive–given a set of data, produce a model
  - Online–given one data point at a time, update model
  - Active–ask for specific data points to improve mode

- Based on type of output
  - Concept Learning–Binary output based on +ve/-ve examples
  - Classification–Classifying into one among many classes
  - Regression–Numeric, ordered output

Classification Example

- Spam Filtering
  - Classify as “spam or “not spam”





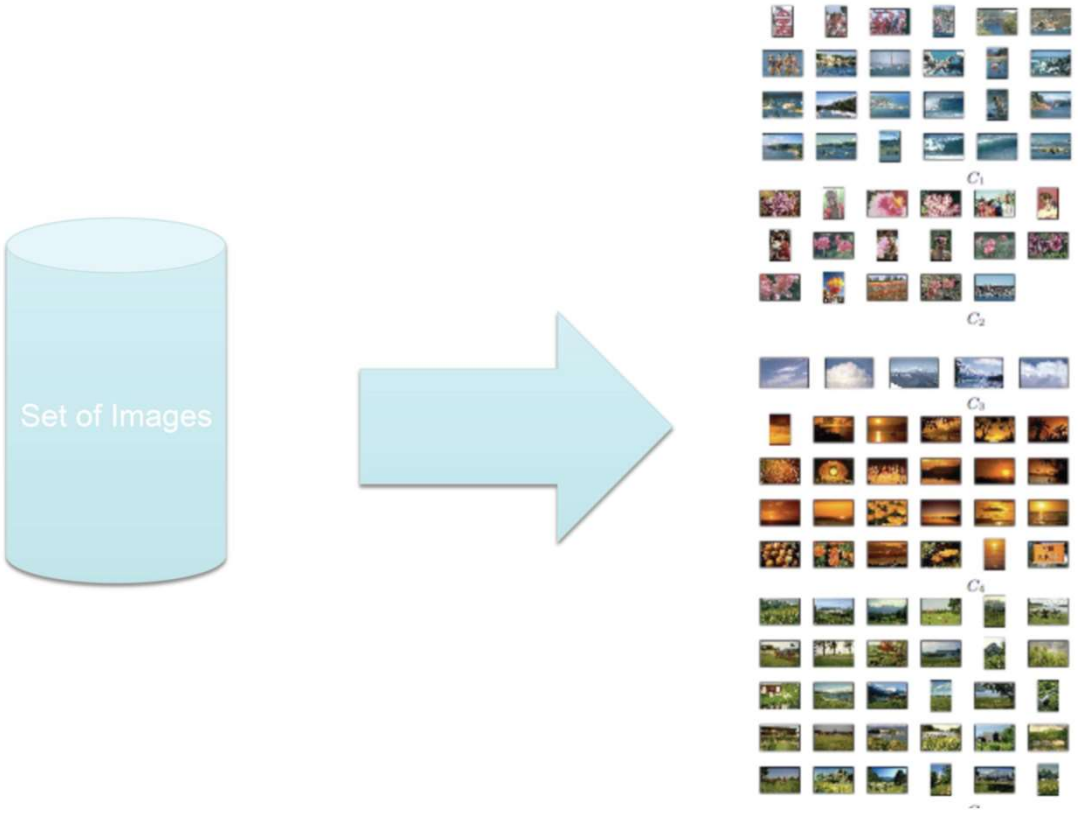
Regression Example

- Predicting Gold/Stock Prices
  - Given historical data on Gold prices, predict tomorrow's price

Good ML can make you rich (but there is still some risk involved).

Example of Unsupervised Learning

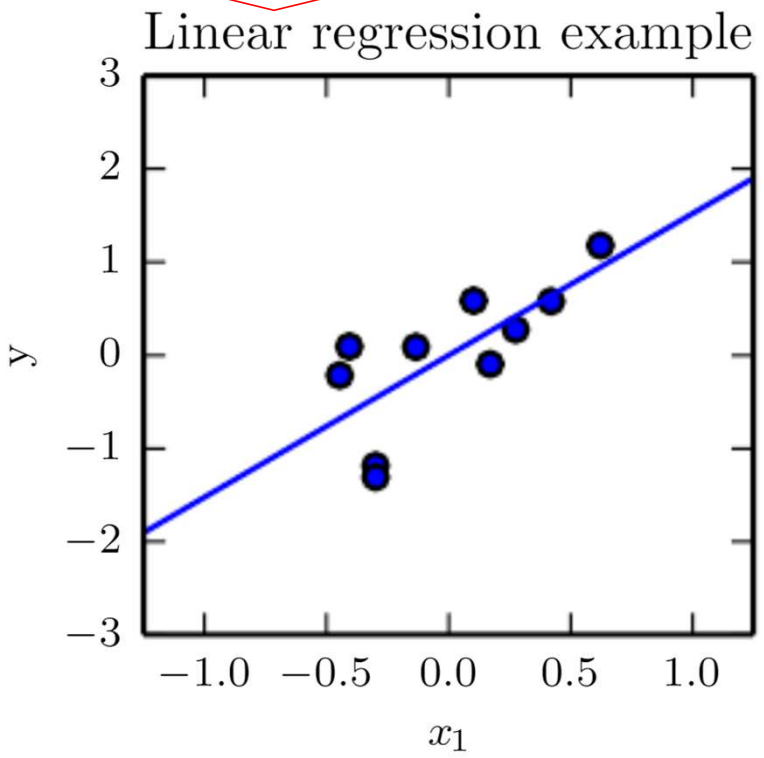
- Clustering of Images
  - Discover structure in data



Linear Regression

Want to find the best linear approximate using the parameters $\mathbf{w} \in \mathbb{R}^n$

$$\hat{y} = \mathbf{w}^T \mathbf{x} + b$$



General notations

the vector of training data samples

$$\mathbf{X}^{(\text{train})} = \begin{bmatrix} \left(\mathbf{x}_1^{(\text{train})}\right)^T \\ \vdots \\ \left(\mathbf{x}_{m^{(\text{train})}}^{(\text{train})}\right)^T \end{bmatrix}$$

number of training data samples

$$\mathbf{y}^{(\text{train})} = \begin{bmatrix} \left(y_1^{(\text{train})}\right)^T \\ \vdots \\ \left(y_m^{(\text{train})}\right)^T \end{bmatrix}$$

outputs (labels) for the training data samples

Linear Regression

Similarly we define the test data and the corresponding outputs:

$$\mathbf{X}^{(\text{test})} = \begin{bmatrix} \left(\mathbf{x}_1^{(\text{test})}\right)^T \\ \vdots \\ \left(\mathbf{x}_{m^{(\text{test})}}^{(\text{test})}\right)^T \end{bmatrix} \qquad \mathbf{y}^{(\text{test})} = \begin{bmatrix} \left(y_1^{(\text{test})}\right)^T \\ \vdots \\ \left(y_{m^{(\text{test})}}^{(\text{test})}\right)^T \end{bmatrix}$$

Augmented approximates for the training data:

$$\hat{\mathbf{y}}^{(\text{train})} = \begin{bmatrix} \hat{y}_1^{(\text{train})} \\ \vdots \\ \hat{y}_m^{(\text{train})} \end{bmatrix} = \begin{bmatrix} (\mathbf{x}_1^{(\text{train})})^T \mathbf{w} \\ \vdots \\ (\mathbf{x}_m^{(\text{train})})^T \mathbf{w} \end{bmatrix} = \mathbf{X}^{(\text{train})} \mathbf{w}$$

Linear Regression

Our goal is to minimize the mean square error:

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

The above function is a quadratic form (convex) w.r.t. the parameter **w**.

Using the vector calculus, we can find the optimal **w**, by making the gradient zero:

$$\nabla_{\boldsymbol{w}}\text{MSE}_{\text{train}} = 0$$

$$\Rightarrow \nabla_{\boldsymbol{w}}\frac{1}{m}||\hat{\boldsymbol{y}}^{(\text{train})} - \boldsymbol{y}^{(\text{train})}||_2^2 = 0$$

Linear Regression

$$\mathbf{y}^T\mathbf{X}\mathbf{w} = \mathbf{w}^T\mathbf{X}^T\mathbf{y}$$

$$\Rightarrow \frac{1}{m}\nabla_{\boldsymbol{w}}||\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}||_2^2 = 0$$

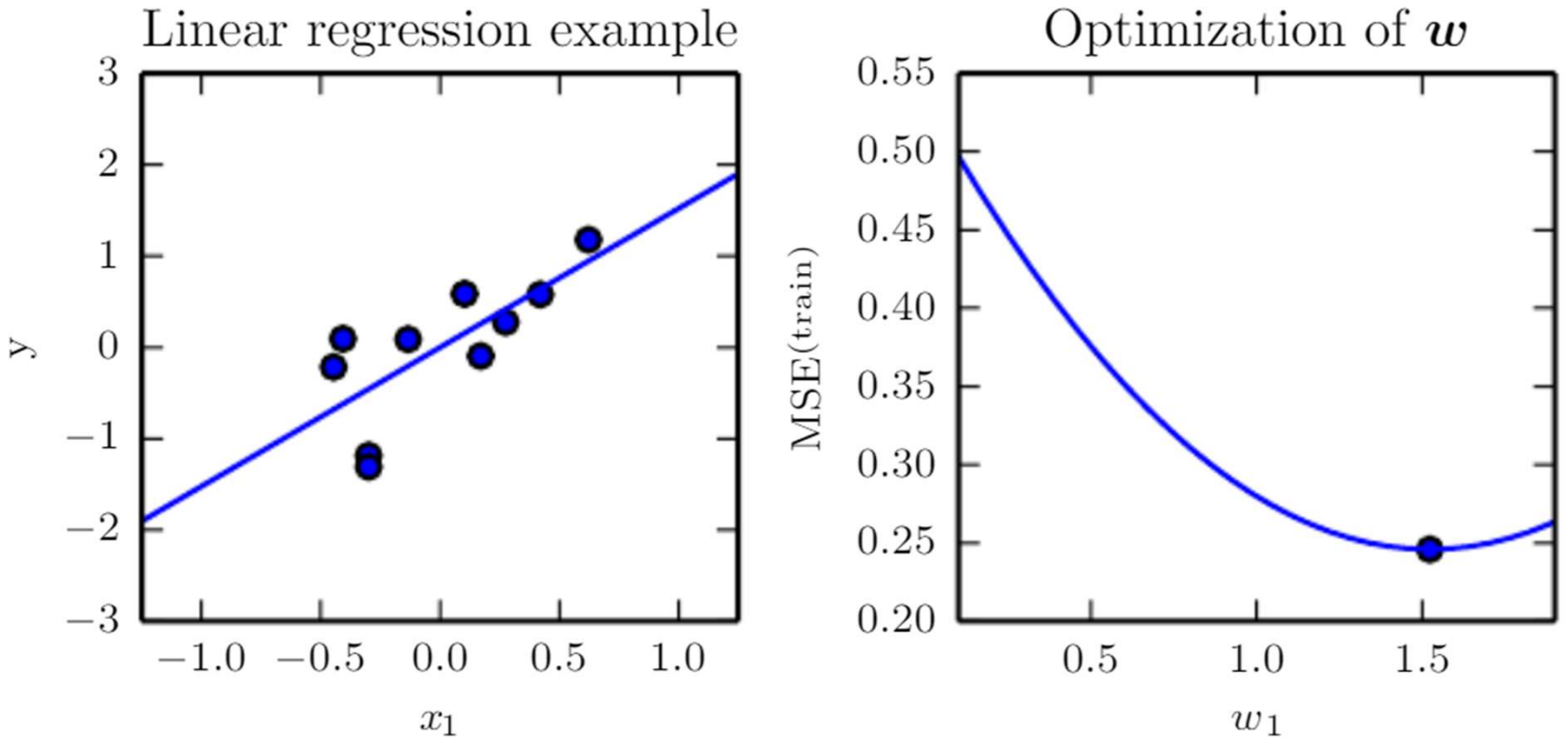
$$\Rightarrow \nabla_{\boldsymbol{w}}\left(\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}\right)^\top\left(\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}\right) = 0$$

$$\Rightarrow \nabla_{\boldsymbol{w}}\left(\boldsymbol{w}^\top\boldsymbol{X}^{(\text{train})\top}\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - 2\boldsymbol{w}^\top\boldsymbol{X}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})} + \boldsymbol{y}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})}\right) = 0$$

$$\Rightarrow 2\boldsymbol{X}^{(\text{train})\top}\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - 2\boldsymbol{X}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})} = 0$$

$$\Rightarrow \boldsymbol{w} = \left(\boldsymbol{X}^{(\text{train})\top}\boldsymbol{X}^{(\text{train})}\right)^{-1}\boldsymbol{X}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})}$$

Linear Regression



Generalization

Generalization: ability to perform well on previously unobserved inputs (test data)

Test error (generalization error): expected value of the error on the test data (new data)

$$\frac{1}{m}\left\|\boldsymbol{X}^{(\text{test})}\boldsymbol{w}-\boldsymbol{y}^{(\text{test})}\right\|^2$$

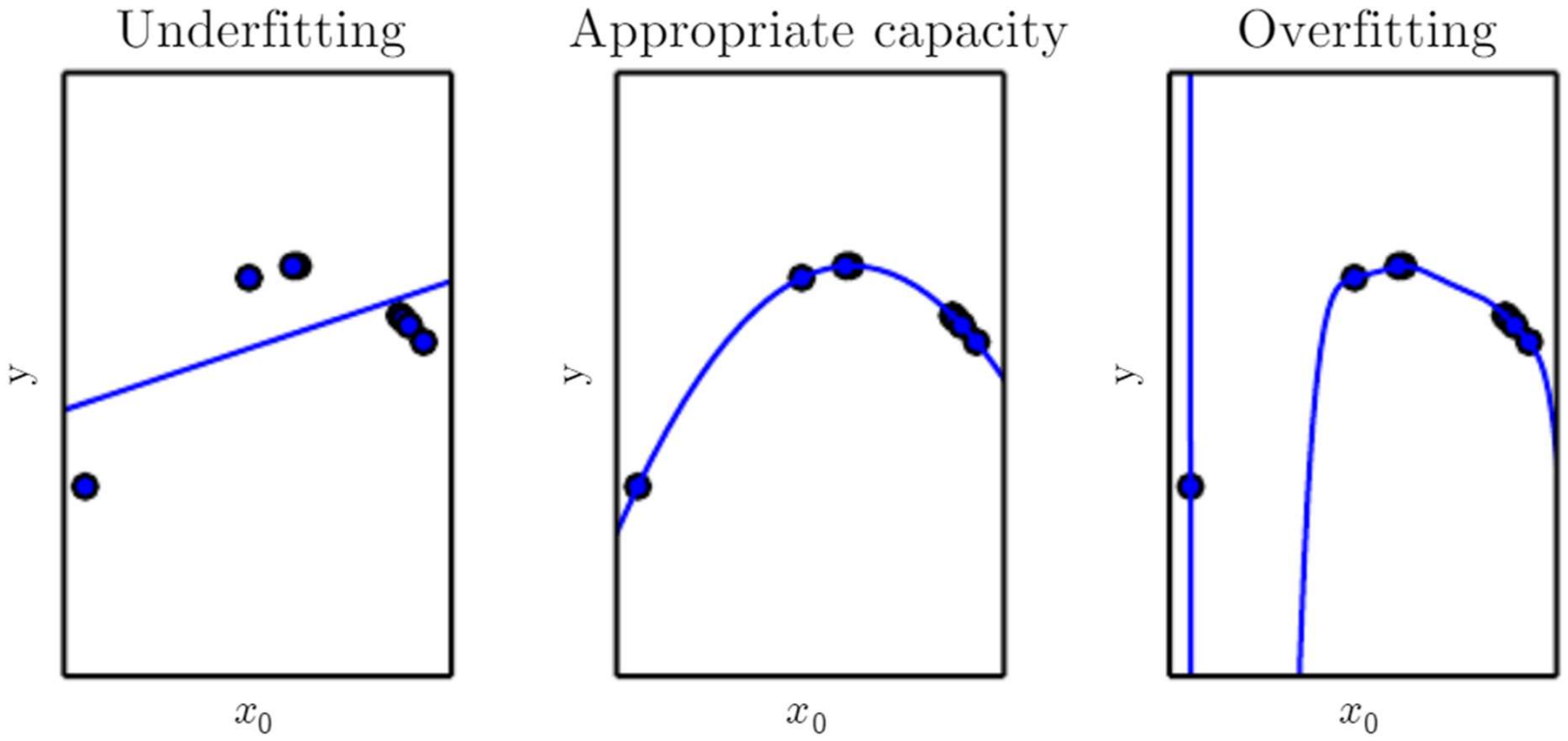
Goal of machine learning

- Make the training error small
- Make the gap between training and test error small

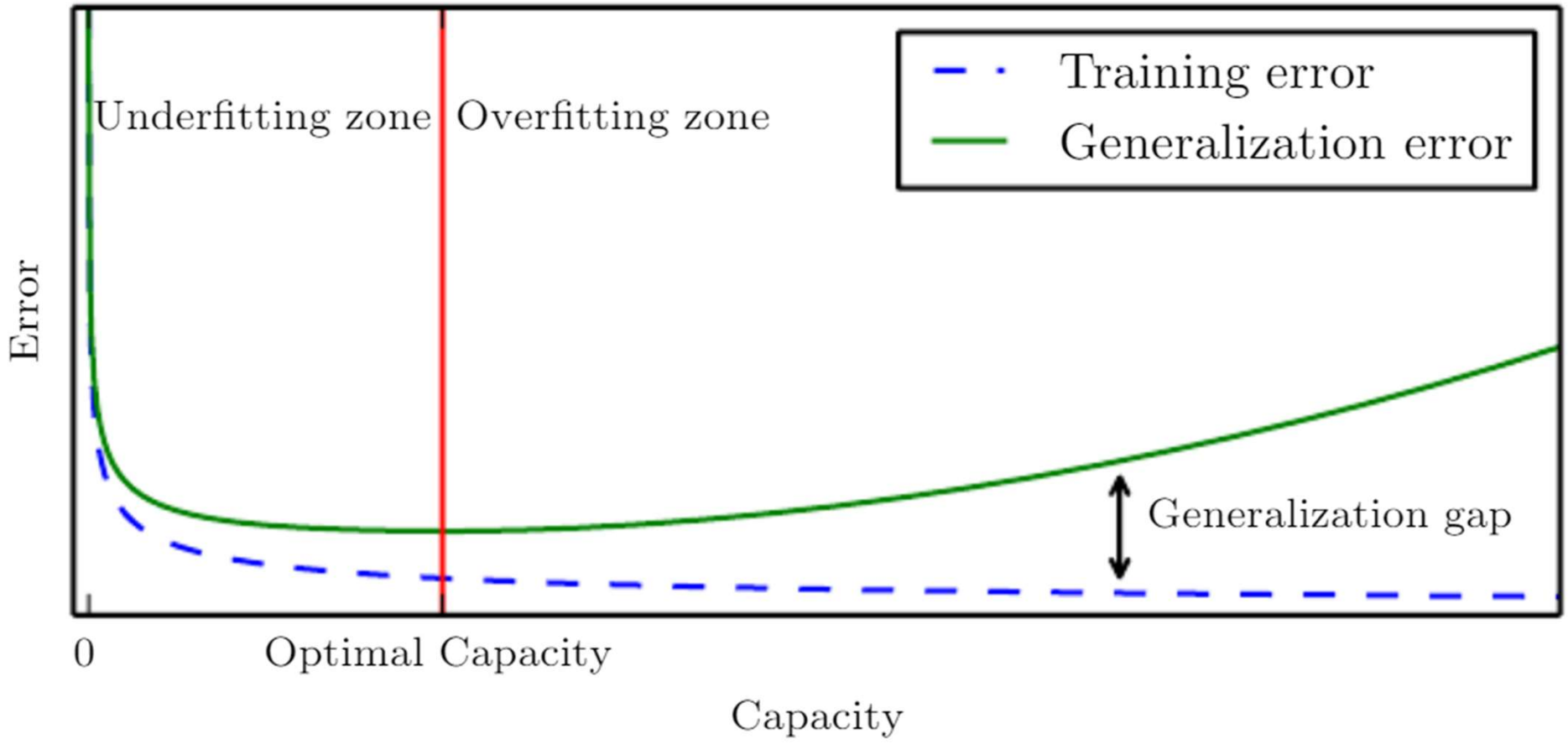
Model's capacity: ability to fit a wide variety of functions

Underfitting and Overfitting

We need to choose a proper model: limiting model's capacity to some extent helps us avoid overfitting



Underfitting and Overfitting



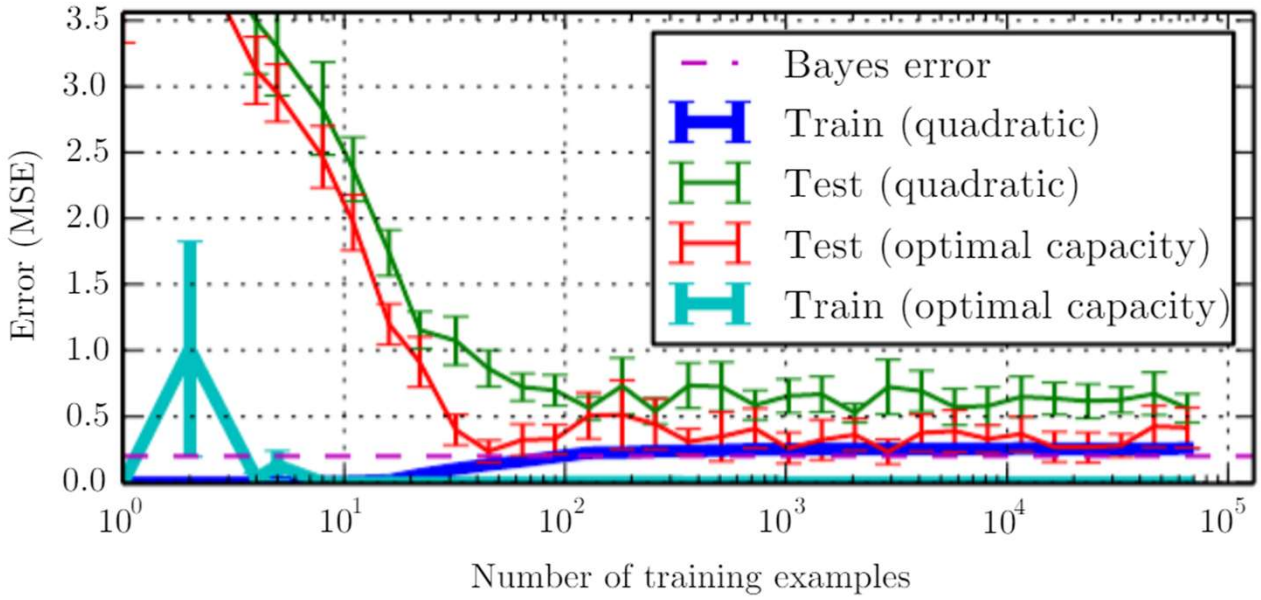
Effect of the Training Dataset Size

Bayes error

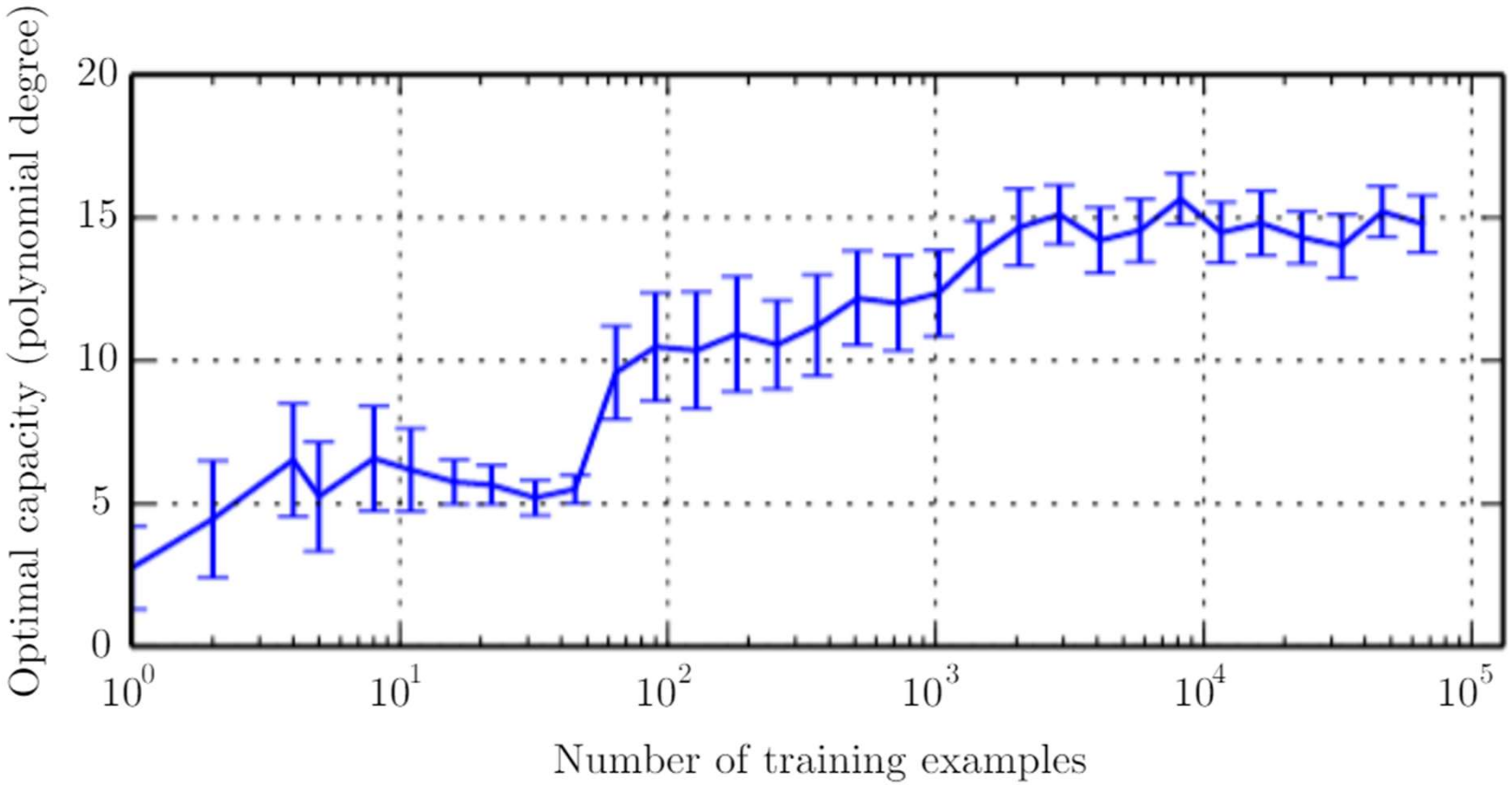
- Error incurred by assuming the ideal model (i.e., making predictions from the true distribution $p(\mathbf{x},y)$
- kind of a lower-bound of error

Experiment setup

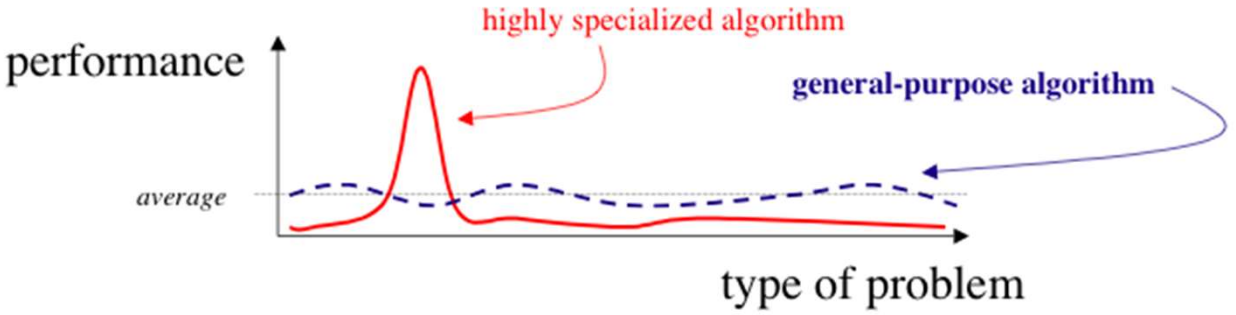
- **noise** is added, generated **40** different training sets for each training dataset size to show **95-percentile** confidence level.



Effect of the Training Dataset Size



No Free Lunch Theorem



"Averaged over all possible data-generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points."

= "**no** machine learning algorithm is **universally any better** than any other"

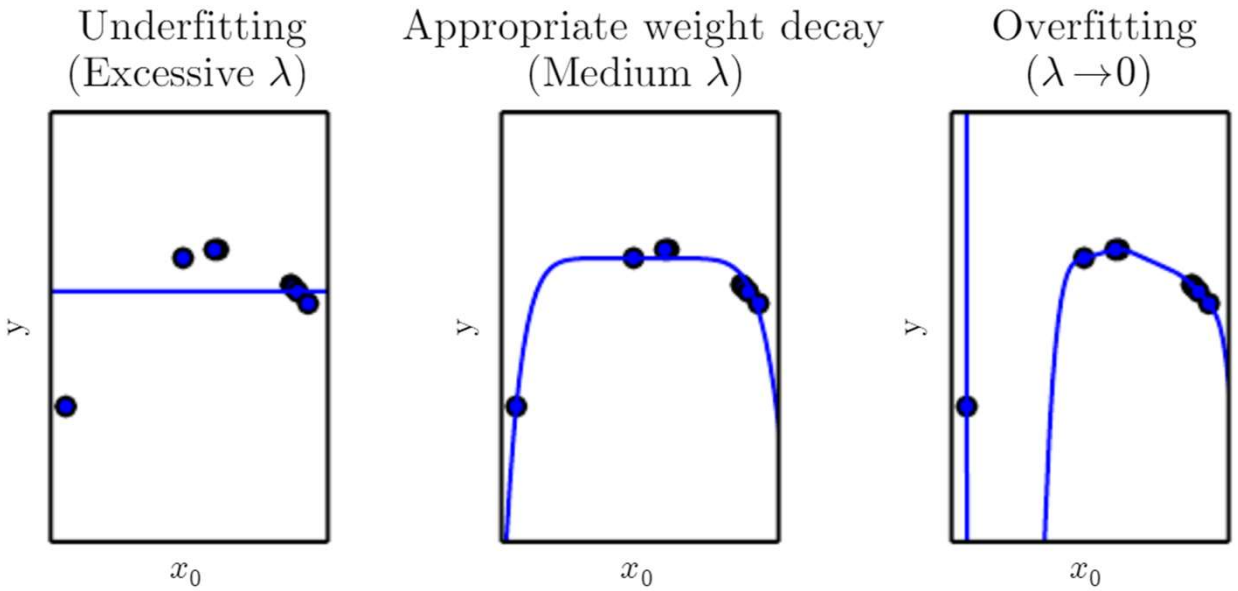
Regularization

Regularization: any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training data

value chosen ahead of time that controls the strength of our preference for smaller weights

Example: **weight decay** in the MSE loss function

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



Hyperparameters and Validation Sets

Hyperparameters: settings that we can use to control the algorithm's behavior

- Example: λ in the linear regression

Validation set

- used to update the hyperparameters during or after training.
- test data are not used in any way to make choices about the model, including its hyperparameters
  - So, no example from the test set can be used in the validation set.
- Typically, we split the training data into two disjoint subsets
  - one is used to learn the parameters of the algorithm.
  - the other one is used for the validation set.

k-fold Cross-Validation

Used when the dataset is too small

$\boldsymbol{z}^{(i)} = (\boldsymbol{x}^{(i)}, y^{(i)})$ in supervised learning
$\boldsymbol{z}^{(i)} = \boldsymbol{x}^{(i)}$ in unsupervised learning

Define KFoldXV($\mathbb{D}, A, L, k$):
Require: $\mathbb{D}$, the given dataset, with elements $\boldsymbol{z}^{(i)}$
Require: A, the learning algorithm, seen as a function that takes a dataset as input and outputs a learned function
Require: L, the loss function, seen as a function from a learned function f and an example $\boldsymbol{z}^{(i)} \in \mathbb{D}$ to a scalar $\in \mathbb{R}$
Require: k, the number of folds
  Split $\mathbb{D}$ into k mutually exclusive subsets $\mathbb{D}_i$, whose union is $\mathbb{D}$
  for i from 1 to k **do**
    $f_i = A(\mathbb{D} \backslash \mathbb{D}_i)$
    for $\boldsymbol{z}^{(j)}$ in $\mathbb{D}_i$ **do**
      $e_j = L(f_i, \boldsymbol{z}^{(j)})$
    end for
  end for
  Return $\boldsymbol{e}$

Estimators

Point estimation

- Attempt to provide the single "best" prediction of some quantity of interest.
- The quantity of interest can be a single parameter, a vector of parameters, or a whole function.

Point estimator or statistic for estimating the true value $\boldsymbol{\theta}$ for m i.i.d. data points $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$:

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

Function estimator

- Assuming $\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ stands for the part of $\boldsymbol{y}$ that is not predictable from $\boldsymbol{x}$
- The function estimator finds the approximator $\hat{f}$, generally with parameters $\boldsymbol{\theta}$.

Bias

Bias of an estimator:

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

where $\boldsymbol{\theta}$ is the true underlying value

Unbiased estimator satisfies:

$$\text{bias}(\hat{\boldsymbol{\theta}}_m) = \mathbf{0} \longleftrightarrow \mathbb{E}(\hat{\boldsymbol{\theta}}_m) = \boldsymbol{\theta}$$

Asymptotically unbiased estimator satisfies:

$$\lim_{m\to\infty} \text{bias}(\hat{\boldsymbol{\theta}}_m) = \mathbf{0} \longleftrightarrow \lim_{m\to\infty} \mathbb{E}(\hat{\boldsymbol{\theta}}_m) = \boldsymbol{\theta}$$

Example of Unbiased Estimator

Gaussian distribution

Consider a Gaussian distribution with mean μ and variance σ^2:

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

i.i.d. samples generated from the above distribution: $\{x^{(1)}, \ldots, x^{(m)}\}$

Sample mean estimator for μ:

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

Example of Unbiased Estimator (Cont'd)

Bias of the sample mean estimator:

$$
\begin{aligned}
\text{bias}(\hat{\mu}_m) &= \mathbb{E}[\hat{\mu}_m] - \mu \\
&= \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right] - \mu \\
&= \left(\frac{1}{m}\sum_{i=1}^{m} \mathbb{E}\left[x^{(i)}\right]\right) - \mu \\
&= \left(\frac{1}{m}\sum_{i=1}^{m} \mu\right) - \mu \\
&= \mu - \mu = 0
\end{aligned}
$$

Example of Biased Estimator

Consider the **Gaussian distribution** $\mathcal{N}\left(x^{(i)}; \mu, \sigma^2\right)$.

Sample variance estimator:

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

where $\hat{\mu}_m$ is the sample mean.

Bias of the estimator:

$$\text{bias}(\hat{\sigma}_m^2) = \mathbb{E}[\hat{\sigma}_m^2] - \sigma^2$$

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

$$= \frac{m-1}{m}\sigma^2$$

Example of Biased Estimator (Cont'd)

Unbiased sample variance estimator would be:

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

$$\begin{aligned}
\mathbb{E}[\tilde{\sigma}_m^2] &= \mathbb{E}\left[\frac{1}{m-1}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2\right] \\
&= \frac{m}{m-1}\mathbb{E}[\hat{\sigma}_m^2] \\
&= \frac{m}{m-1}\left(\frac{m-1}{m}\sigma^2\right) \\
&= \sigma^2.
\end{aligned}$$

- While unbiased estimators are clearly desirable, they are not always the "best" estimators.
- As we will see we often use biased estimators that possess other important properties

Variance and Standard Error

Variance of an estimator is simply the variance:

$$\mathrm{Var}(\hat{\theta})$$

Standard error is the square root of the variance.

For example,

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

Variance and Standard Error (Cont'd)

Example: Bernoulli distribution

Consider the data samples $\{x^{(1)}, \ldots, x^{(m)}\}$ drawn i.i.d. from a Bernoulli distribution

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

Mean estimator:

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

Variance of the estimator:

$$\begin{aligned}
\mathrm{Var}\left(\hat{\theta}_m\right) = \mathrm{Var}\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right) &= \frac{1}{m^2}\sum_{i=1}^{m}\mathrm{Var}\left(x^{(i)}\right) \\
&= \frac{1}{m^2}\sum_{i=1}^{m}\theta(1-\theta) \\
&= \frac{1}{m^2}m\theta(1-\theta) \\
&= \frac{1}{m}\theta(1-\theta)
\end{aligned}$$

Trading off Bias and Variance to Minimize MSE

There exists a **trade-off relationship between bias and variance**.

We **cannot tell which one is always better**.

One best way is to **minimize MSE**, which incorporates **both the measures**:

$$\begin{aligned}\mathrm{MSE} &= \mathbb{E}[(\hat{\theta}_m - \theta)^2] \\ &= \mathrm{Bias}(\hat{\theta}_m)^2 + \mathrm{Var}(\hat{\theta}_m)\end{aligned}$$

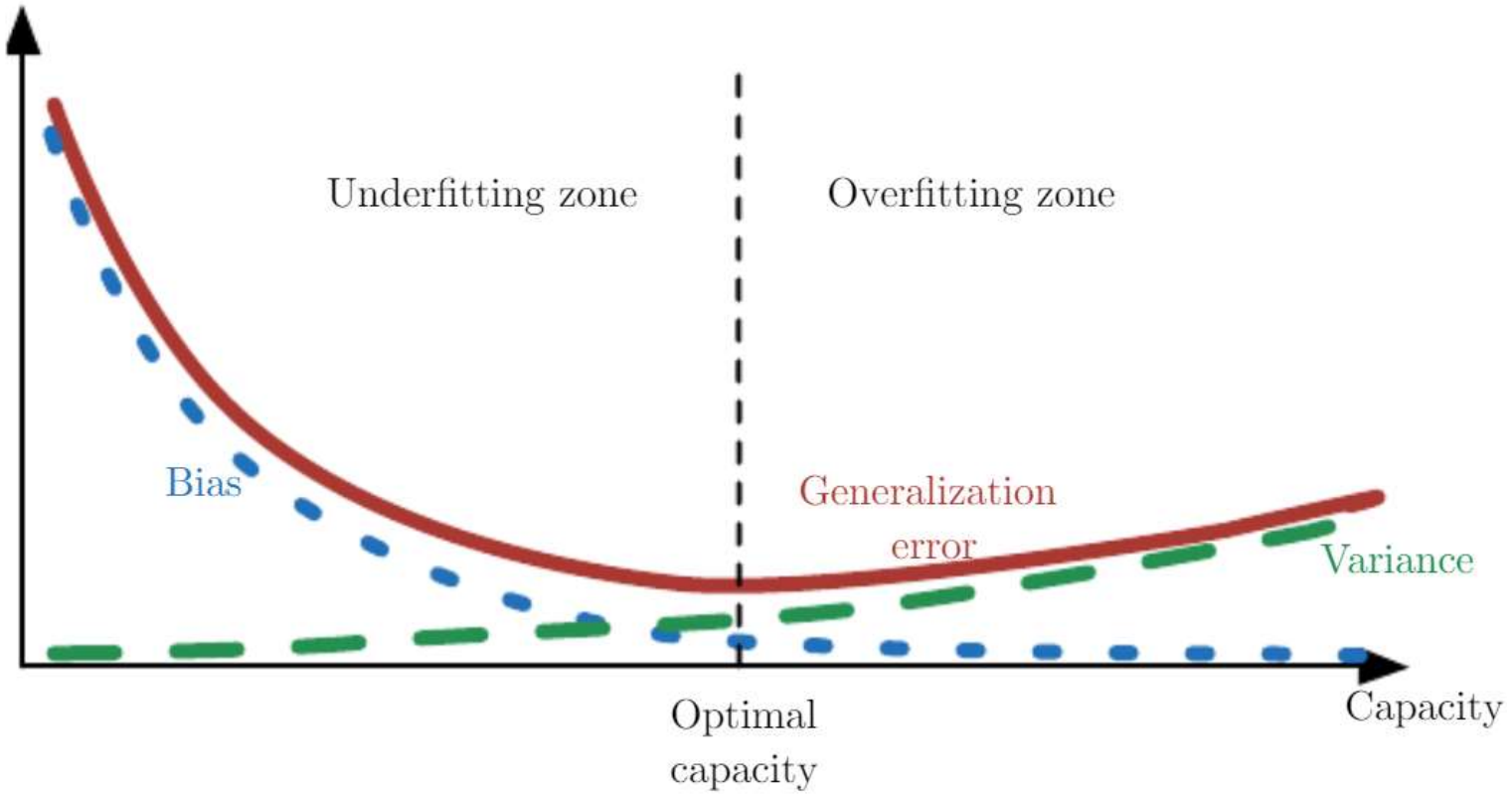
$$\boldsymbol{MSE} = E\left(\hat{\theta}_m^2 - 2\hat{\theta}_m\theta + \theta^2\right) = E\left(\hat{\theta}_m^2\right) - 2E\left(\hat{\theta}_m\right)\theta + \theta^2$$

$$\boldsymbol{Bias}\left(\widehat{\boldsymbol{\theta}}_{\boldsymbol{m}}\right)^{\boldsymbol{2}} = \left(E\left(\hat{\theta}_m\right) - \theta\right)^2 = E\left(\hat{\theta}_m\right)^2 - 2\theta E\left(\hat{\theta}_m\right) + \theta^2$$

$$\boldsymbol{Var}\left(\widehat{\boldsymbol{\theta}}_{\boldsymbol{m}}\right) = E\left(\hat{\theta}_m - E(\hat{\theta}_m)\right)^2 = E(\hat{\theta}_m^2) - 2E\left(\hat{\theta}_m\right)^2 + E\left(\hat{\theta}_m\right)^2 = E(\hat{\theta}_m^2) - E\left(\hat{\theta}_m\right)^2$$

$$\boldsymbol{Bias}\left(\widehat{\boldsymbol{\theta}}_{\boldsymbol{m}}\right)^{\boldsymbol{2}} + \boldsymbol{Var}\left(\widehat{\boldsymbol{\theta}}_{\boldsymbol{m}}\right) = E(\hat{\theta}_m^2) - 2\theta E\left(\hat{\theta}_m\right) + \theta^2$$

Trading off Bias and Variance to Minimize MSE



Consistency

Consistency

- Defines the behavior of an estimator as the amount of training data grows.

Weak consistency:

$$\text{plim}_{m\to\infty}\,\hat{\theta}_m = \theta$$

Convergence in probability: for any $\epsilon > 0$, $P\left(\left|\hat{\theta}_m - \theta\right| > \epsilon\right) \to 0$ as $m \to \infty$

Strong consistency:

$$p\left(\lim_{m\to\infty}\,\mathbf{x}^{(m)} = \boldsymbol{x}\right) = 1$$

Maximum Likelihood Estimation

What is the **best** estimator for a stochastic problem?

- **Maximum likelihood** estimator
- Can be optimal, but generally not optimal.
- But still, the performance is descent, and is easy to use.

Consider examples $\mathbb{X} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ drawn independently from the true but unknown data-generating distribution $p_{\text{data}}(\boldsymbol{x})$.

We will construct the probability model with parameter $\boldsymbol{\theta}$ as $p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$.

The ML estimator for $\boldsymbol{\theta}$ is then defined by

$$\begin{aligned}\boldsymbol{\theta}_{\text{ML}} &= \arg\max_{\boldsymbol{\theta}} p_{\text{model}}(\mathbb{X}; \boldsymbol{\theta}), \\ &= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})\end{aligned}$$

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \boldsymbol{x}) = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{x} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$$

Maximum Likelihood Estimation (Cont'd)

Alternative form:

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\mathrm{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

Multiplying $\frac{1}{m}$ to the cost function, and assuming we have many samples,

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x}; \boldsymbol{\theta})$$

Interpretation of the ML estimation

- Minimizing the dissimilarity between the empirical distribution $\hat{p}_{\mathrm{data}}(\boldsymbol{x})$ and the model distribution $p_{\mathrm{model}}(\boldsymbol{x})$.
- Of course, it would be best if we know $p_{\mathrm{data}}(\boldsymbol{x})$ instead of $\hat{p}_{\mathrm{data}}(\boldsymbol{x})$. But $p_{\mathrm{data}}(\boldsymbol{x})$ is not available.

Maximum Likelihood Estimation (Cont'd)

Resemblance to the **KL divergence**:

- Minimizing the KL divergence

$$D_{\mathrm{KL}}\left(\hat{p}_{\mathrm{data}} \| p_{\mathrm{model}}\right) = \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\mathrm{data}}}\left[\log \hat{p}_{\mathrm{data}}(\boldsymbol{x}) - \log p_{\mathrm{model}}(\boldsymbol{x})\right]$$

is the **same** as maximizing

$$\mathbb{E}_{\mathbf{x}\sim\hat{p}_{\mathrm{data}}}\left[\log p_{\mathrm{model}}(\boldsymbol{x})\right]$$

So, we get the **same** cost function:

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x};\boldsymbol{\theta})$$

Conditional Log-Likelihood

In machine learning, let's say we want to predict $\boldsymbol{Y}$ stocastically for given input $\boldsymbol{X}$.

We want to form the estimate of the conditional probability $P(\boldsymbol{Y}|\boldsymbol{X})$ as $P(\boldsymbol{Y}|\boldsymbol{X};\boldsymbol{\theta})$ by optimizing the parameters $\boldsymbol{\theta}$.

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{Y} \mid \boldsymbol{X};\boldsymbol{\theta})$$

If the examples are i.i.d., then we have a decomposed version:

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log P(\boldsymbol{y}^{(i)} \mid \boldsymbol{x}^{(i)};\boldsymbol{\theta})$$

Linear Regression as ML

Consider the **input-output relationship**:

$$x = y + z,$$

where $z \sim \mathcal{N}(0, \sigma^2)$.

So, for given x, we get the **conditional probability**:

$$p(y|x) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(\frac{(x-y)^2}{2\sigma^2}\right)$$

The objective of the machine learning is to **estimate $\boldsymbol{y}$ (mean)** stocastically **for given $\boldsymbol{x}$.**

Suppose that $\hat{\boldsymbol{y}}^{(i)}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$ gives the prediction of the mean for given the i-th example $\boldsymbol{x}^{(i)}$.

Linear Regression as ML (Cont'd)

Assuming the examples are i.i.d., the **ML cost function** gives us

$$\sum_{i=1}^{m} \log p(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

$$= -m \log \sigma - \frac{m}{2} \log(2\pi) - \sum_{i=1}^{m} \frac{\left\| \hat{y}^{(i)} - y^{(i)} \right\|^2}{2\sigma^2}$$

Maximizing the above cost function is the same as **minimizing**

$$\sum_{i=1}^{m} \frac{\left\| \hat{y}^{(i)} - y^{(i)} \right\|^2}{2\sigma^2}$$

On the other hand, the **MSE** is defined by

$$\text{MSE}_{\text{train}} = \frac{1}{m} \sum_{i=1}^{m} ||\hat{y}^{(i)} - y^{(i)}||^2$$

So, in this case, **maximizing ML** is equivalent to **minimizing MSE**

Bayesian Statistics

In **ML**, the parameter $\boldsymbol{\theta}$ is treated as a deterministic unknown variable.

But in **Bayesian statistics**, $\boldsymbol{\theta}$ is a unknown random variable.

Before observing the data, we need to represent our knowledge of $\boldsymbol{\theta}$ using the prior probability distribution $p(\boldsymbol{\theta})$.

Generally, we select a prior distribution that is quite broad to reflect a high degree of uncertainty in the value of $\boldsymbol{\theta}$ before observing any data.

We can recover the effect of data on our belief about $\boldsymbol{\theta}$ by

$$p(\boldsymbol{\theta} \mid x^{(1)}, \ldots, x^{(m)}) = \frac{p(x^{(1)}, \ldots, x^{(m)} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(x^{(1)}, \ldots, x^{(m)})}$$

Our goal would be finding $\boldsymbol{\theta}$ that maximizes the above conditional probability.

We may want to make a prediction for the next data sample from

$$p(x^{(m+1)} \mid x^{(1)}, \ldots, x^{(m)}) = \int p(x^{(m+1)} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid x^{(1)}, \ldots, x^{(m)}) \, d\boldsymbol{\theta}$$

Bayesian Linear Regression

Consider the linear regression

$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$$

Given a set of m training samples $\left(\boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})}\right)$, we get the prediction

$$\hat{\boldsymbol{y}}^{(\text{train})} = \boldsymbol{X}^{(\text{train})}\boldsymbol{w}$$

Assuming the Gaussian-noisy modeling, $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} = y +$ Gaussian noise(zero mean, unit var), we get

$$p(\boldsymbol{y}^{(\text{train})} \mid \boldsymbol{X}^{(\text{train})}, \boldsymbol{w}) = \mathcal{N}(\boldsymbol{y}^{(\text{train})}; \boldsymbol{X}^{(\text{train})}\boldsymbol{w}, \boldsymbol{I})$$

$$\propto \exp\left(-\frac{1}{2}(\boldsymbol{y}^{(\text{train})} - \boldsymbol{X}^{(\text{train})}\boldsymbol{w})^\top(\boldsymbol{y}^{(\text{train})} - \boldsymbol{X}^{(\text{train})}\boldsymbol{w})\right)$$

To assume a fairly broad prior distribution, we use

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}; \boldsymbol{\mu}_0, \boldsymbol{\Lambda}_0) \propto \exp\left(-\frac{1}{2}(\boldsymbol{w} - \boldsymbol{\mu}_0)^\top \boldsymbol{\Lambda}_0^{-1}(\boldsymbol{w} - \boldsymbol{\mu}_0)\right)$$

Bayesian Linear Regression (Cont'd)

Then, we can determine the **posterior** distribution over the model parameters:

$$p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) \propto p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}) p(\boldsymbol{w})$$

This is due to

$$p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y}) = \frac{p(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{y})}{p(\boldsymbol{X}, \boldsymbol{y})} = \frac{p(\boldsymbol{w}, \boldsymbol{y}|\boldsymbol{X}) p(\boldsymbol{X})}{p(\boldsymbol{y}|\boldsymbol{X}) p(\boldsymbol{X})} = \frac{p(\boldsymbol{w}, \boldsymbol{y}|\boldsymbol{X})}{p(\boldsymbol{y}|\boldsymbol{X})} = \frac{p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w}) p(\boldsymbol{w})}{p(\boldsymbol{y}|\boldsymbol{X})} \propto p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w}) p(\boldsymbol{w})$$

Thus, we further have

$$p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) \propto \exp\left(-\frac{1}{2}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})\right) \exp\left(-\frac{1}{2}(\boldsymbol{w} - \boldsymbol{\mu}_0)^\top \boldsymbol{\Lambda}_0^{-1} (\boldsymbol{w} - \boldsymbol{\mu}_0)\right)$$

Due to $p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})$

Due to $p(\boldsymbol{w})$

$$\propto \exp\left(-\frac{1}{2}\left(-2\boldsymbol{y}^\top \boldsymbol{X}\boldsymbol{w} + \boldsymbol{w}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{w} + \boldsymbol{w}^\top \boldsymbol{\Lambda}_0^{-1} \boldsymbol{w} - 2\boldsymbol{\mu}_0^\top \boldsymbol{\Lambda}_0^{-1} \boldsymbol{w}\right)\right)$$

Bayesian Linear Regression (Cont'd)

We now define $\boldsymbol{\Lambda}_m = \left(\boldsymbol{X}^\top \boldsymbol{X} + \boldsymbol{\Lambda}_0^{-1}\right)^{-1}$ and $\boldsymbol{\mu}_m = \boldsymbol{\Lambda}_m \left(\boldsymbol{X}^\top \boldsymbol{y} + \boldsymbol{\Lambda}_0^{-1}\boldsymbol{\mu}_0\right)$

With this choice, we get

$$p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) \propto \exp\left(-\frac{1}{2}(\boldsymbol{w} - \boldsymbol{\mu}_m)^\top \boldsymbol{\Lambda}_m^{-1}(\boldsymbol{w} - \boldsymbol{\mu}_m) + \frac{1}{2}\boldsymbol{\mu}_m^\top \boldsymbol{\Lambda}_m^{-1}\boldsymbol{\mu}_m\right)$$

$$\propto \exp\left(-\frac{1}{2}(\boldsymbol{w} - \boldsymbol{\mu}_m)^\top \boldsymbol{\Lambda}_m^{-1}(\boldsymbol{w} - \boldsymbol{\mu}_m)\right).$$

With $\boldsymbol{\mu}_0 = \mathbf{0}$ and $\boldsymbol{\Lambda}_0 = \frac{1}{\alpha}\boldsymbol{I}$, $\boldsymbol{\Lambda}_m = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \alpha\boldsymbol{I}\right)^{-1}$ and $\boldsymbol{\mu}_m = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \alpha\boldsymbol{I}\right)^{-1}\left(X^T y\right)$.

So, maximizing $p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y})$ gives us

$$\boldsymbol{w}^*_{Bayes} = \boldsymbol{\mu}_m = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \alpha\boldsymbol{I}\right)^{-1}\left(\boldsymbol{X}^T \boldsymbol{y}\right)$$

Bayesian Linear Regression (Cont'd)

On the other hand, consider the ML problem:

$$w_{ML}^{*} = \arg\max_{\boldsymbol{w}} p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w}\|^2$$

Then, with the regularization of weight decaying, we have

$$w_{RegML}^{*} = \arg\min_{\boldsymbol{w}} \|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w}\|^2 + \alpha \boldsymbol{w}^T\boldsymbol{w}$$

Since, the above problem is convex, we need to the point with zero gradient:

$$\nabla\left(\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w}\|^2 + \alpha \boldsymbol{w}^T\boldsymbol{w}\right) = \nabla\left(\boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - 2\boldsymbol{y}^T\boldsymbol{X}\boldsymbol{w} + \boldsymbol{w}^T\left(\boldsymbol{X}^T\boldsymbol{X}+\alpha\boldsymbol{I}\right)\boldsymbol{w}\right) = -2\boldsymbol{X}^T\boldsymbol{y} + 2\left(\boldsymbol{X}^T\boldsymbol{X}+\alpha\boldsymbol{I}\right)\boldsymbol{w}$$

Which gives us the optimal point:

$$w_{RegML}^{*} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\alpha\boldsymbol{I}\right)^{-1}\left(\boldsymbol{X}^T\boldsymbol{y}\right)$$

So, we get the same results.

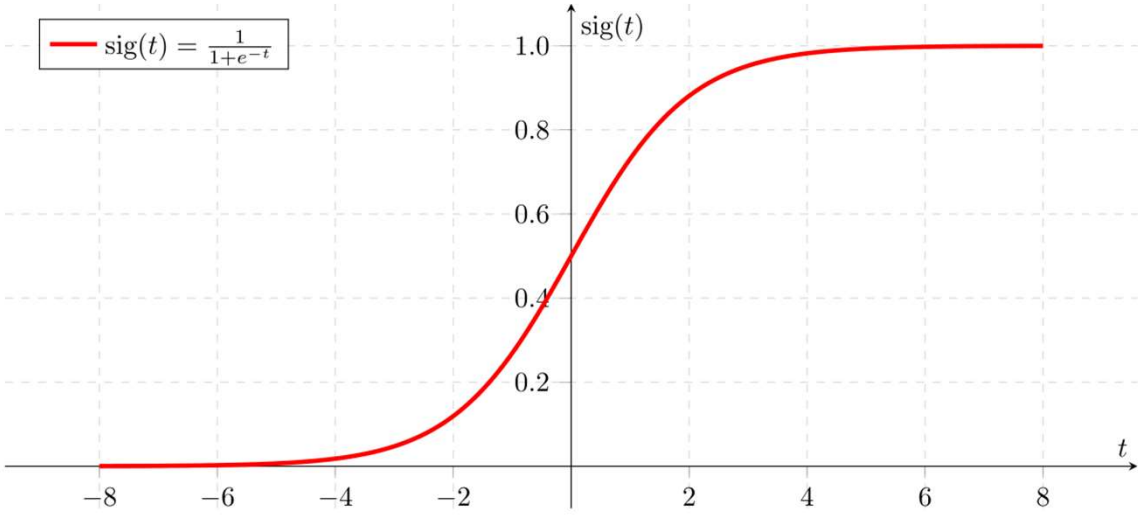
Logistic Regression

We have binary outputs 0 or 1 (a simple on/off classification)

Through parameters $\boldsymbol{\theta}$, we want to estimate the conditional probability

$$p(y = 1 \mid \boldsymbol{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \boldsymbol{x})$$

where $\sigma(\cdot)$ is a sigmoid function.

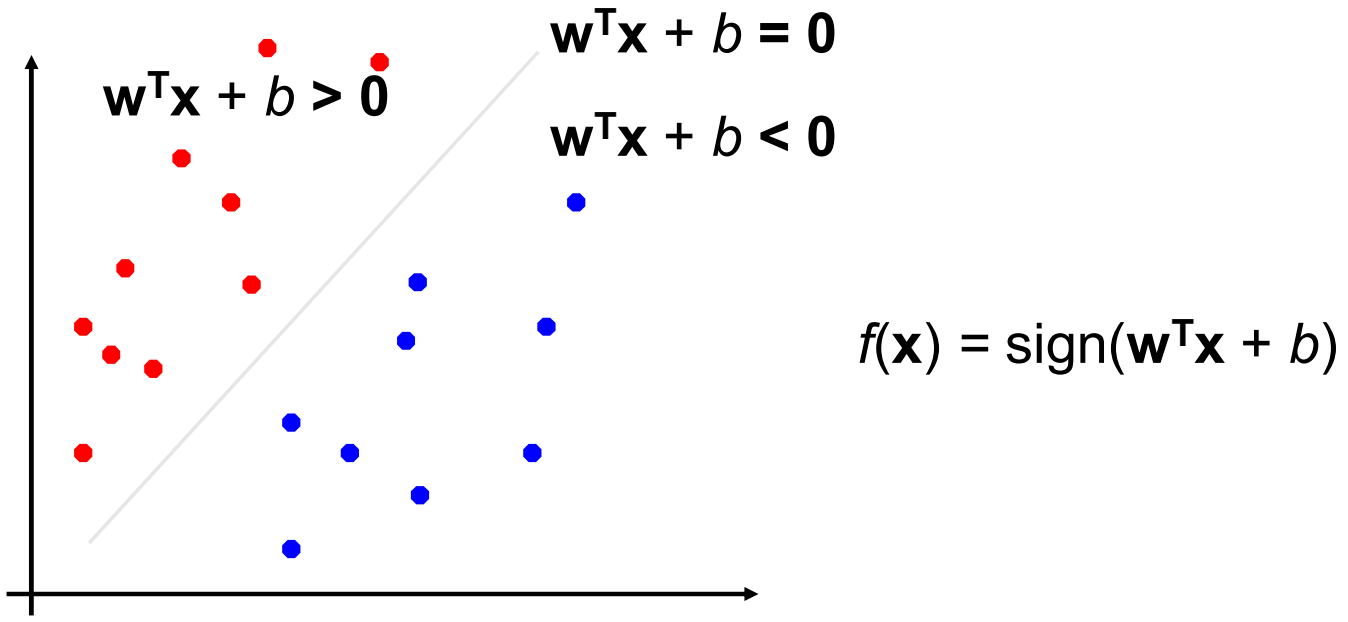


Support Vector Machines

One of the most influential supervised learning approaches

Similar to logistic regression in that it is driven by $\boldsymbol{w}^T \boldsymbol{x} + b$, but it does not provide probabilities

Suppose that we have data examples, and want to make a binary classification with a linear decision boundary

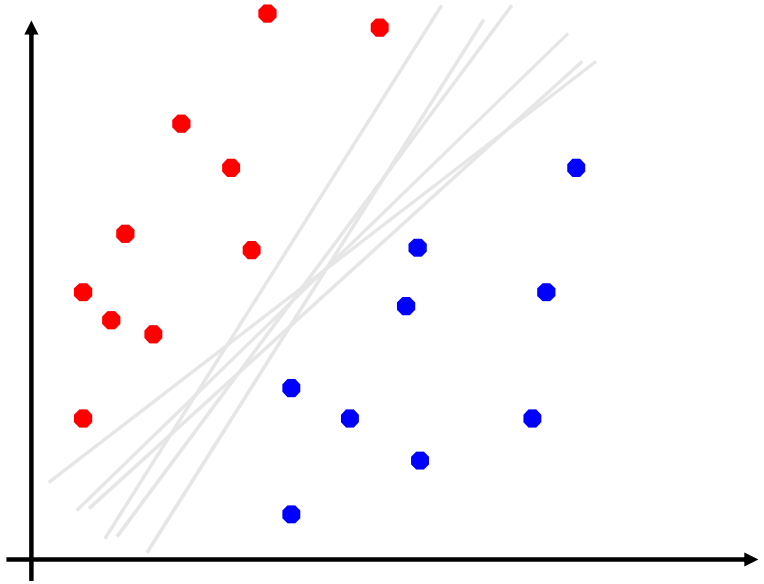


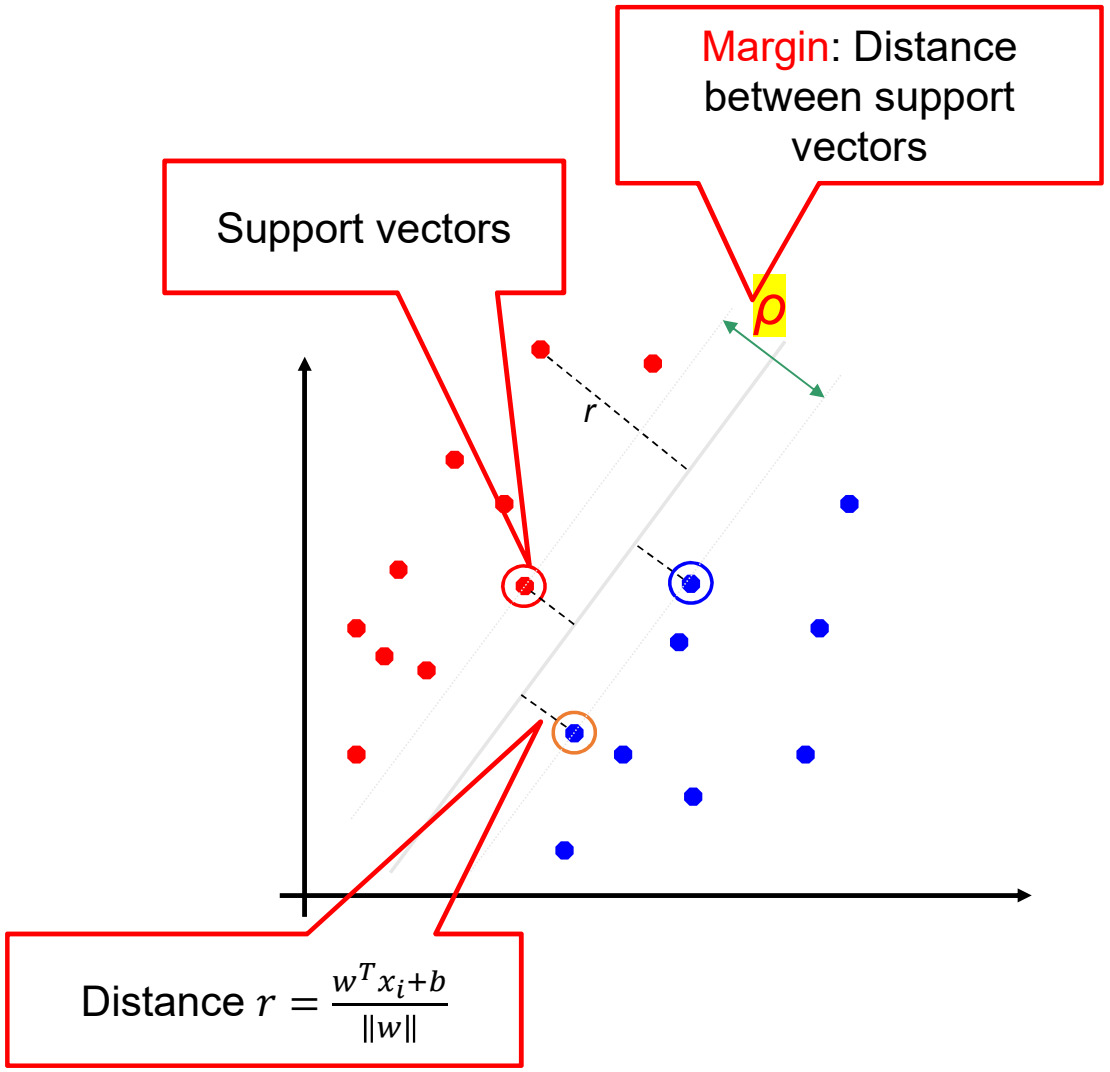
$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$$

Support Vector Machines

Which of the linear separators is optimal?

We want to maximize the margin ρ.





Support Vector Machines

Linear SVM

- Let training set $\{(\mathbf{x}_i, y_i)\}_{i=1..n}$, $\mathbf{x}_i \in \mathbf{R}^d$, $y_i \in \{-1, 1\}$ be separated by a hyperplane with margin ρ. Then for each training example $(\mathbf{x}_i, y_i)$:

$$\frac{\boldsymbol{w}^T\boldsymbol{x}_i + b}{\|\boldsymbol{w}\|} \leq -\rho/2 \quad \text{if } y_i = -1$$
$$\frac{\boldsymbol{w}^T\boldsymbol{x}_i + b}{\|\boldsymbol{w}\|} \geq \rho/2 \quad \text{if } y_i = 1$$
$$\Leftrightarrow \quad y_i \cdot \frac{\boldsymbol{w}^T\boldsymbol{x}_i + b}{\|\boldsymbol{w}\|} \geq \rho/2$$

- For every support vector $\mathbf{x}_s$ the above inequality is an equality.
- We rescale $\mathbf{w}$ and b by $\rho/2$ in the equality, i.e., $\boldsymbol{w} \leftarrow \boldsymbol{w} \cdot \left(\frac{2}{\rho}\right), b \leftarrow b \cdot \left(\frac{2}{\rho}\right)$.
- Then, we get $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$.
- Thus, we obtain that distance between each $\mathbf{x}_s$ and the hyperplane is

$$r = \frac{y_s\left(\boldsymbol{w}^T\boldsymbol{x}_s + b\right)}{\|\boldsymbol{w}\|} = \frac{1}{\|\boldsymbol{w}\|}$$

- Then the margin can be expressed through (rescaled) $\mathbf{w}$ and b as:

$$\rho = 2r = \frac{2}{\|\boldsymbol{w}\|}$$

Support Vector Machines

Linear SVM

- Algorithm:

Find $\mathbf{w}$ and b such that

$\frac{2}{\|\boldsymbol{w}\|}$ is maximized

and for all $(\mathbf{x}_i, y_i)$, i=1..n : $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Find $\mathbf{w}$ and b such that

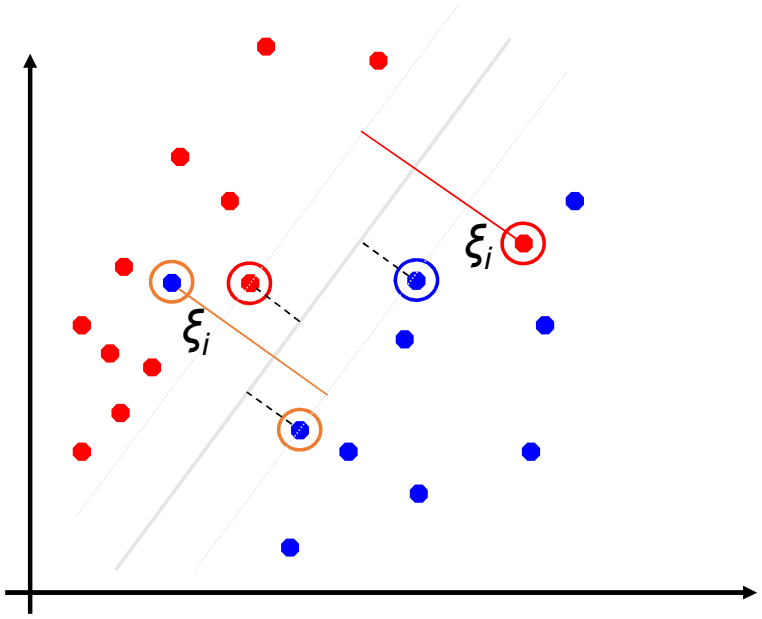
$\boldsymbol{\Phi}(\mathbf{w}) = \|\mathbf{w}\|^2 = \mathbf{w}^T\mathbf{w}$ is minimized

and for all $(\mathbf{x}_i, y_i)$, i=1..n : $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Support Vector Machines

Soft Margin Classification

- What if the training set is not linearly separable?
- Slack variables ξ_i can be added to allow misclassification of difficult or noisy examples, resulting margin called soft.



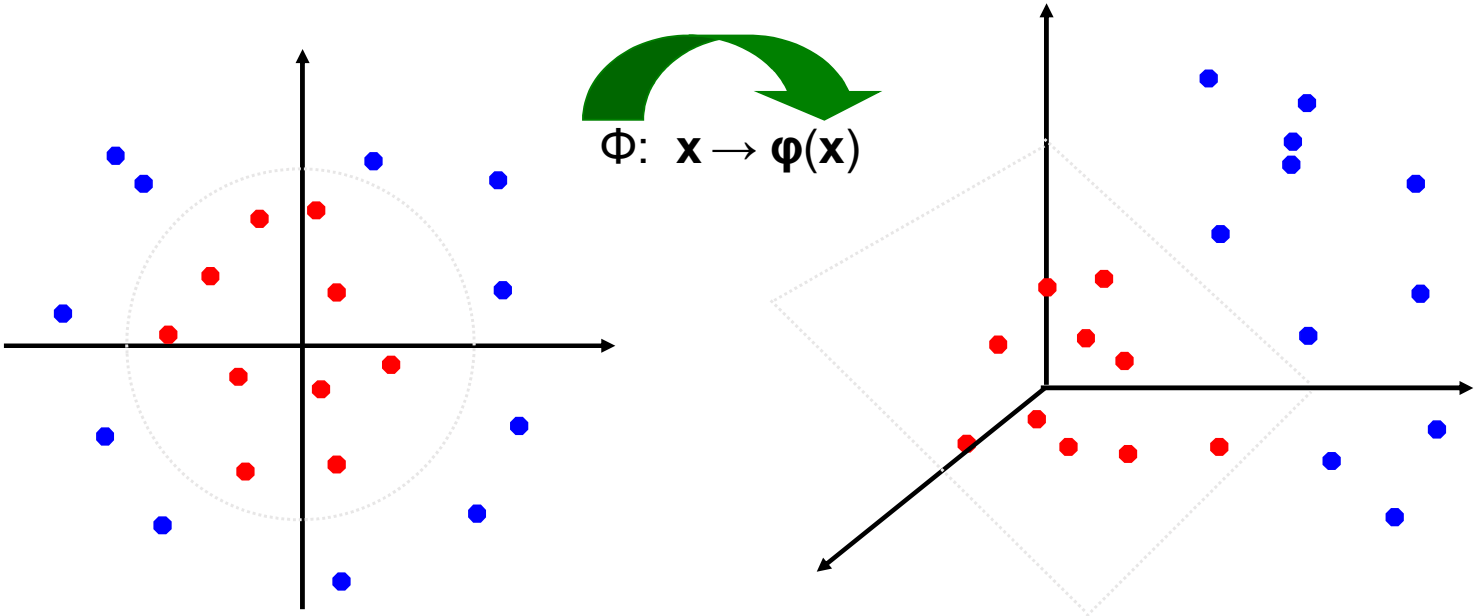
Find $\mathbf{w}$ and b such that
$\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$, $\xi_i \geq 0$

Parameter C can be viewed as a way to control overfitting: it "trades off" the relative importance of minimizing the margin and fitting the training data.

Support Vector Machines

Kernel Trick

- What if our data cannot be separable with linear boundaries?
- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



Support Vector Machines

Kernel Trick

- The linear classifier relies on inner product between vectors $K(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j$
- If every datapoint is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \varphi(\mathbf{x})$, the inner product becomes:

$$K(\mathbf{x}_i,\mathbf{x}_j)= \boldsymbol{\varphi}(\mathbf{x}_i)^{\mathrm{T}}\boldsymbol{\varphi}(\mathbf{x}_j)$$

- A kernel function is a function that is eqiuvalent to an inner product in some feature space.
- Example:

  2-dimensional vectors $\mathbf{x}=[x_1 \;\; x_2]$; let $K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j)^2$,

  Need to show that $K(\mathbf{x}_i,\mathbf{x}_j)= \boldsymbol{\varphi}(\mathbf{x}_i)^{\mathrm{T}}\boldsymbol{\varphi}(\mathbf{x}_j)$:

  $$K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j)^2 = 1+ x_{i1}^2x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2}+ x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}$$

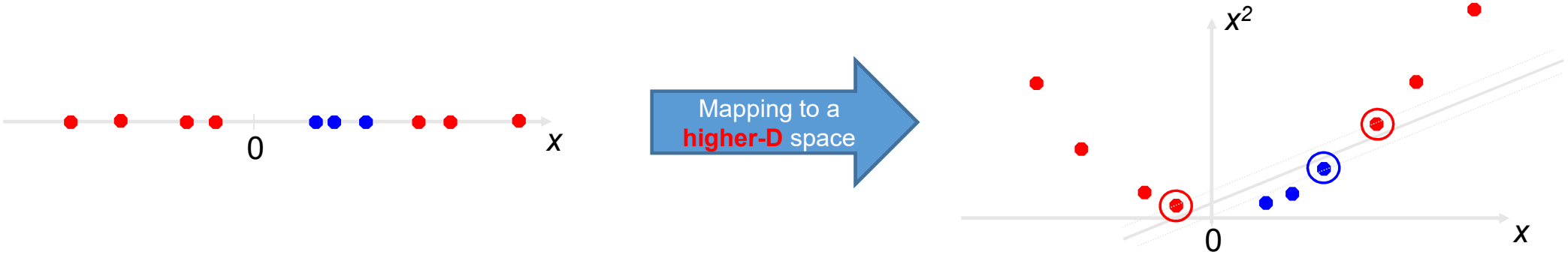
  $$= [1 \;\; x_{i1}^2 \;\; \sqrt{2}\, x_{i1}x_{i2} \;\; x_{i2}^2 \;\; \sqrt{2}x_{i1} \;\; \sqrt{2}x_{i2}]^{\mathrm{T}}\, [1 \;\; x_{j1}^2 \;\; \sqrt{2}\, x_{j1}x_{j2} \;\; x_{j2}^2 \;\; \sqrt{2}x_{j1} \;\; \sqrt{2}x_{j2}]$$

  $$= \boldsymbol{\varphi}(\mathbf{x}_i)^{\mathrm{T}}\boldsymbol{\varphi}(\mathbf{x}_j), \quad \text{where } \boldsymbol{\varphi}(\mathbf{x}) = [1 \;\; x_1^2 \;\; \sqrt{2}\, x_1x_2 \;\; x_2^2 \;\; \sqrt{2}x_1 \;\; \sqrt{2}x_2]$$

- Thus, a kernel function implicitly maps data to a high-dimensional space (without the need to compute each $\boldsymbol{\varphi}(\mathbf{x})$ explicitly).

Support Vector Machines

Kernel Trick



Support Vector Machine

Kernel Trick

- For some functions $K(\mathbf{x}_i,\mathbf{x}_j)$ checking that $K(\mathbf{x}_i,\mathbf{x}_j)= \varphi(\mathbf{x}_i)^{\mathrm{T}}\varphi(\mathbf{x}_j)$ can be cumbersome.
- Mercer's theorem:

Every semi-positive definite symmetric function is a kernel

- Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:

$K=$

$K(\mathbf{x}_1,\mathbf{x}_1)$	$K(\mathbf{x}_1,\mathbf{x}_2)$	$K(\mathbf{x}_1,\mathbf{x}_3)$	…	$K(\mathbf{x}_1,\mathbf{x}_n)$
$K(\mathbf{x}_2,\mathbf{x}_1)$	$K(\mathbf{x}_2,\mathbf{x}_2)$	$K(\mathbf{x}_2,\mathbf{x}_3)$		$K(\mathbf{x}_2,\mathbf{x}_n)$
…	…	…	…	…
$K(\mathbf{x}_n,\mathbf{x}_1)$	$K(\mathbf{x}_n,\mathbf{x}_2)$	$K(\mathbf{x}_n,\mathbf{x}_3)$	…	$K(\mathbf{x}_n,\mathbf{x}_n)$

Support Vector Machine

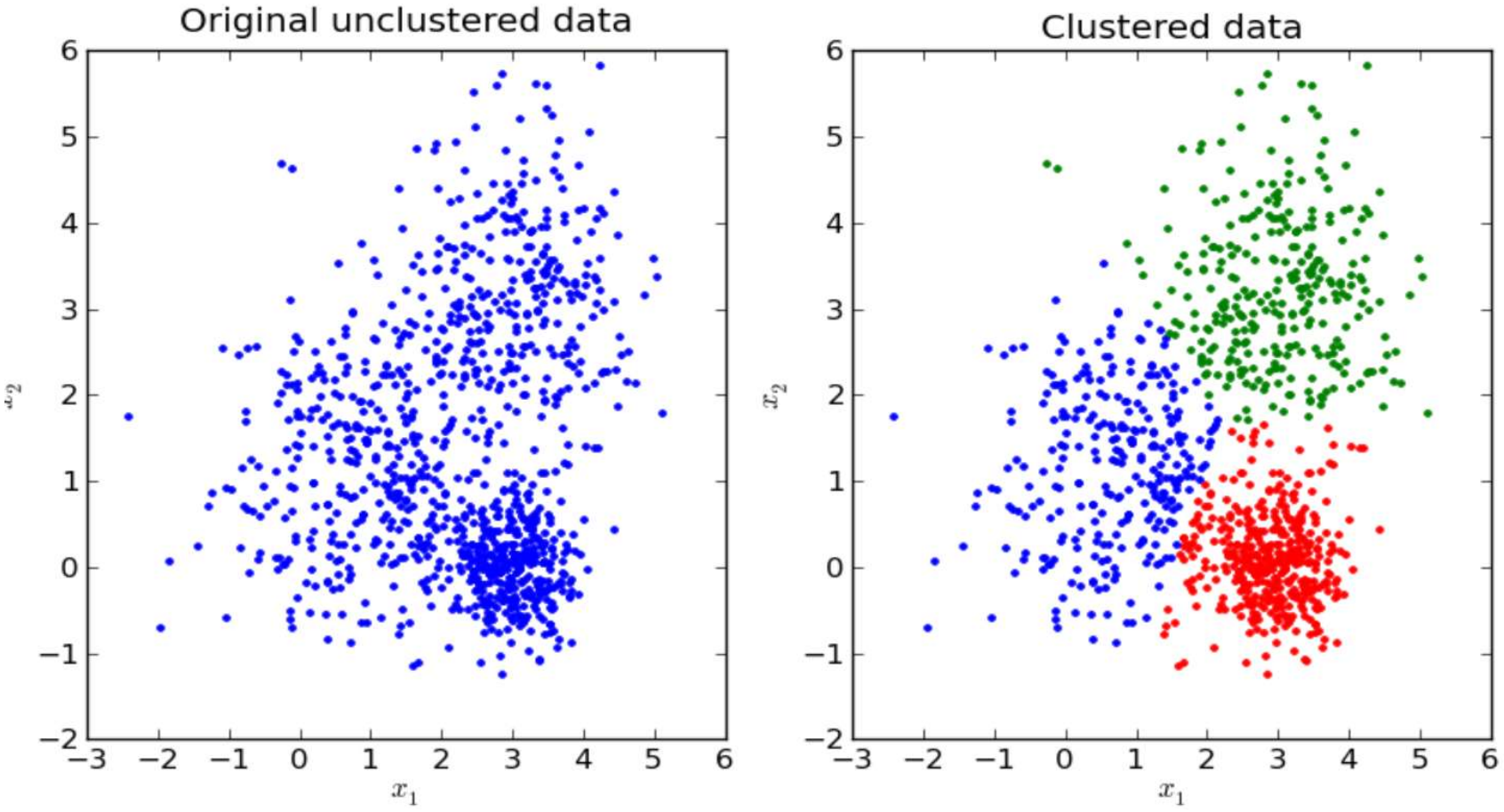
Kernel Trick

- Linear: $K(\mathbf{x}_i,\mathbf{x}_j)= \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j$
  - Mapping Φ: $\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$, where $\boldsymbol{\varphi}(\mathbf{x})$ is $\mathbf{x}$ itself

- Polynomial of power p: $K(\mathbf{x}_i,\mathbf{x}_j)= (1+ \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j)^p$
  - Mapping Φ: $\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$, where $\boldsymbol{\varphi}(\mathbf{x})$ has $\binom{d+p}{p}$ dimensions

- Gaussian (radial-basis function): $K(\mathbf{x}_i,\mathbf{x}_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$
  - Mapping Φ: $\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$, where $\boldsymbol{\varphi}(\mathbf{x})$ is infinite-dimensional

k-means Clustering



k-means Clustering

Partition a set of points $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n$ into k clusters $S_1, S_2, \cdots, S_k$ such that the following is minimized

$$\sum_{i=1}^{k} \sum_{\boldsymbol{x}_j \in S_j} \left\| \boldsymbol{x}_j - \boldsymbol{\mu}_i \right\|^2$$

Mean of the cluster S_i

Algorithm:

Initialize a set of k centers

Repeat

Assign each point to its nearest center

Recompute the set of centers

Until convergence

k-means Clustering