

# Semi-supervised and transfer Learning

Makoto Yamada

myamada@i.kyoto-u.ac.jp

Kyoto University

# Review: Supervised Learning

Problem formulation of supervised learning.

- Input vector:  $\mathbf{x} = (x_1, x_2, \dots, x_d)^\top \in \mathbb{R}^d$
- Output:  $y \in \mathbb{R}$
- $(\mathbf{x}_i, y_i) \stackrel{\text{i.i.d.}}{\sim} p(\mathbf{x}, y)$
- Labeled data:  $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$
- Model:  $f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}$ . (Linear model)

Risk:  $R(\mathbf{w}) = \iint \text{loss}(y, f(\mathbf{x}; \mathbf{w})) p(\mathbf{x}, y) d\mathbf{x} dy$

Empirical Risk:  $R_{emp}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n \text{loss}(y_i, f(\mathbf{x}_i; \mathbf{w}))$

Empirical Risk Minimization (ERM):  $\widehat{\mathbf{w}} = \operatorname{argmin}_{\mathbf{w}} R_{emp}(\mathbf{w})$

# Semi-Supervised Learning

Problem formulation of semi-supervised learning.

- $(\mathbf{x}_i, y_i) \stackrel{\text{i.i.d.}}{\sim} p(\mathbf{x}, y)$
- $\mathbf{x}_i \stackrel{\text{i.i.d.}}{\sim} p(\mathbf{x})$
- Labeled data:  $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$
- Unlabeled data:  $\{\mathbf{x}_{n+1}, \mathbf{x}_{n+2}, \dots, \mathbf{x}_{n+m}\}$
- Usually  $n \ll m$  and  $n$  is small

Semi-supervised learning:

- We have both labeled and unlabeled samples.
- Semi-supervised learning uses both labeled and unlabeled samples.
- The unlabeled samples follow the **same distribution** of the marginal distribution of  $p(\mathbf{x}, y)$

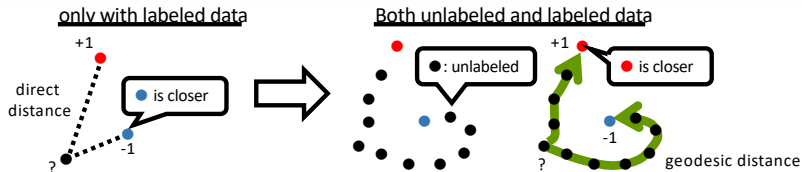
# Role of unlabeled data

## Data generation process

- Input  $\mathbf{x}$  is generated by a distribution with  $p(\mathbf{x})$ .
- Output  $y$  for  $\mathbf{x}$  is generated by conditional distribution with probability density  $p(y|\mathbf{x})$ .

## Unlabeled data can be used for capturing $p(\mathbf{x})$

- input data distribution, input space metric, or better representation.



# Semi-supervised learning frameworks

- Weighted maximum likelihood estimation
- Graph-based learning
- Self-training
- Clustering
- Generative models

# Weighted maximum likelihood

The original goal of MLE is to maximize:

$$\begin{aligned}\mathbb{E}_{p(\mathbf{x},y)}[\log p(y|\mathbf{x})] &= \iint \log P(y|\mathbf{x}; \mathbf{w}) \underbrace{p(y|\mathbf{x})p(\mathbf{x})}_{p(\mathbf{x},y)} d\mathbf{x}dy, \\ &\approx \frac{1}{n} \sum_{i=1}^n \log(P(y_i|\mathbf{x}_i; \mathbf{w}))\end{aligned}$$

where  $P(y|\mathbf{x}; \mathbf{w})$  is a model. Each training instance is equally weighted.

Note, MLE is equivalent to maximize the negative log-likelihood function:

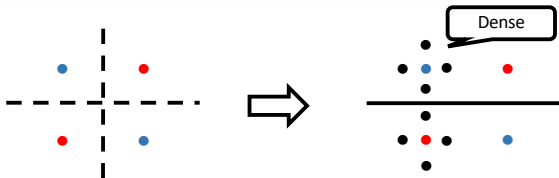
$$L(\mathbf{w}) = \log \left( \prod_{i=1}^n P(y_i|\mathbf{x}_i; \mathbf{w}) \right) \propto \frac{1}{n} \sum_{i=1}^n \log(P(y_i|\mathbf{x}_i; \mathbf{w}))$$

# Weighted maximum likelihood

Weighted maximum likelihood:

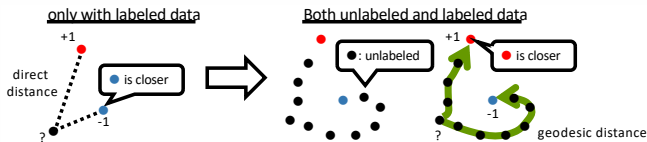
$$\max_{\mathbf{w}} \sum_{i=1}^n p(\mathbf{x}_i) \log(P(y_i|\mathbf{x}_i; \mathbf{w}))$$

- Each training data instance is weighted by  $p(\mathbf{x}_i)$ .
- $p(\mathbf{x})$  is estimated by using unlabeled data.
- Denser areas are largely weighted
- Training a classifier focusing on the dense areas



# Graph-based method

- Basic idea: construct a graph capturing the intrinsic shape of input space, and make prediction on the graph.
- Assumption: Data lie on a manifold in the feature space
- The graph represent adjacency relationships among data
- K-nearest neighbor graph (e.g.,  $A_{ij} = \{0, 1\}$ )
- Edge-weighted graph with e.g.,  $A_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2)$





# Label propagation

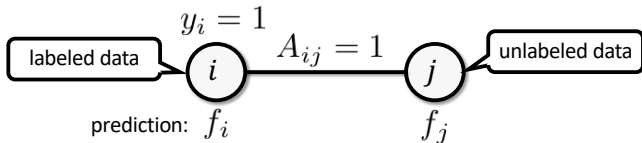
Basic idea: Adjacent instances tend to have the same label.

Transductive setting (we have test instances)

$$\min_{\mathbf{f} \in \mathbb{R}^{n+m}} \sum_{i=1}^n (f_i - y_i)^2 + \lambda \sum_{i=1}^{n+m} \sum_{j=1}^{n+m} A_{ij} (f_i - f_j)^2,$$

where  $\lambda > 0$  is the regularization parameter.

- 1st term: (squared) loss function to fit to labeled data.
- 2nd term: regularization function to make adjacent nodes to have similar predictions.



# Transfer Learning

Supervised Learning:

- Training  $\{(\mathbf{x}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^n \stackrel{\text{i.i.d.}}{\sim} p_{\text{tr}}(\mathbf{x}, y)$
- Test  $(\mathbf{x}^{\text{te}}, y^{\text{te}}) \stackrel{\text{i.i.d.}}{\sim} p_{\text{te}}(\mathbf{x}, y)$  (Not observed during training)
- $p_{\text{tr}} = p_{\text{te}}$  (Training and test distributions are same)

Semi-supervised Learning:

- Training  $\{(\mathbf{x}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^n \stackrel{\text{i.i.d.}}{\sim} p_{\text{tr}}(\mathbf{x}, y),$   
 $\{\mathbf{x}_i^{\text{tr}}\}_{i=n+1}^{n+m} \stackrel{\text{i.i.d.}}{\sim} p_{\text{tr}}(\mathbf{x}).$
- Test  $(\mathbf{x}^{\text{te}}, y^{\text{te}}) \stackrel{\text{i.i.d.}}{\sim} p_{\text{te}}(\mathbf{x}, y)$  (Not observed during training)
- $p_{\text{tr}} = p_{\text{te}}$  (Training and test distributions are same)

If  $p_{\text{tr}} \neq p_{\text{te}}$ , supervised method and semi-supervised method do not perform well. A solution: [Transfer Learning!](#)

# Problem formulations of Transfer Learning

## Unsupervised transfer learning

- $\{(\mathbf{x}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{n_{\text{tr}}} \stackrel{\text{i.i.d.}}{\sim} p_{\text{tr}}(\mathbf{x}, y),$
- $\{\mathbf{x}_j^{\text{te}}\}_{j=1}^{n_{\text{te}}} \stackrel{\text{i.i.d.}}{\sim} p_{\text{te}}(\mathbf{x}), n_{\text{tr}} \ll n_{\text{te}}$

## Supervised transfer learning

- $\{(\mathbf{x}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{n_{\text{tr}}} \stackrel{\text{i.i.d.}}{\sim} p_{\text{tr}}(\mathbf{x}, y)$
- $\{(\mathbf{x}_j^{\text{te}}, y_j^{\text{te}})\}_{j=1}^{n_{\text{te}}} \stackrel{\text{i.i.d.}}{\sim} p_{\text{te}}(\mathbf{x}, y), n_{\text{te}} \ll n_{\text{tr}}$

## Semi-supervised transfer learning

- $\{(\mathbf{x}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{n_{\text{tr}}} \stackrel{\text{i.i.d.}}{\sim} p_{\text{tr}}(\mathbf{x}, y)$
- $\{(\mathbf{x}_j^{\text{te}}, y_j^{\text{te}})\}_{j=1}^{n_{\text{te}}} \stackrel{\text{i.i.d.}}{\sim} p_{\text{te}}(\mathbf{x}, y), n_{\text{te}} \ll n_{\text{tr}}$
- $\{\mathbf{x}_j^{\text{te}}\}_{j=n_{\text{te}}+1}^{n_{\text{te}}+n'_{\text{te}}} \stackrel{\text{i.i.d.}}{\sim} p_{\text{te}}(\mathbf{x}), n_{\text{tr}} \ll n_{\text{te}}$

# Unsupervised Transfer Learning

We assume

- It does not need to have `test` label
- Need some assumption

Standard approaches

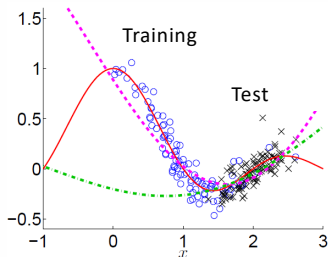
- Importance weighted method (e.g., Covariate shift adaptation)
- Subspace based method.

# Covariate Shift Adaptation [1]

Problem setup:

- $\{(\mathbf{x}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{n_{\text{tr}}} \stackrel{\text{i.i.d.}}{\sim} p_{\text{tr}}(\mathbf{x}, y),$
- $\{\mathbf{x}_j^{\text{te}}\}_{j=1}^{n_{\text{te}}} \stackrel{\text{i.i.d.}}{\sim} p_{\text{te}}(\mathbf{x}), n_{\text{tr}} \ll n_{\text{te}}$

Key idea: Learning a function so that **error in test data is minimized** under the assumption  $p_{\text{tr}}(y|\mathbf{x}) = p_{\text{te}}(y|\mathbf{x})$



# Covariate Shift Adaptation

The risk for  $p_{\text{te}}(\mathbf{x}, y)$  can be written as

$$\begin{aligned} J(\mathbf{w}) &= \iint L(y, f(\mathbf{x})) p_{\text{te}}(\mathbf{x}, y) d\mathbf{x} dy \\ &= \iint L(y, f(\mathbf{x})) \frac{p_{\text{te}}(\mathbf{x}, y)}{p_{\text{tr}}(\mathbf{x}, y)} p_{\text{tr}}(\mathbf{x}, y) d\mathbf{x} dy \\ &= \iint L(y, f(\mathbf{x})) \frac{p_{\text{te}}(y|\mathbf{x}) p_{\text{te}}(\mathbf{x})}{p_{\text{tr}}(y|\mathbf{x}) p_{\text{tr}}(\mathbf{x})} p_{\text{tr}}(y, \mathbf{x}) d\mathbf{x} dy \\ &= \iint L(y, f(\mathbf{x})) \frac{p_{\text{te}}(\mathbf{x})}{p_{\text{tr}}(\mathbf{x})} p_{\text{tr}}(y, \mathbf{x}) d\mathbf{x} dy \\ &\approx \frac{1}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} L(y_i^{\text{tr}}, f(\mathbf{x}_i^{\text{tr}})) \frac{p_{\text{te}}(\mathbf{x}_i^{\text{tr}})}{p_{\text{tr}}(\mathbf{x}_i^{\text{tr}})} \end{aligned}$$

Actually, it is a **weighted maximum likelihood** problem. Note  $\frac{p_{\text{te}}(\mathbf{x}_i^{\text{tr}})}{p_{\text{tr}}(\mathbf{x}_i^{\text{tr}})}$  is a ratio of probability densities (density-ratio).

# Covariate Shift Adaptation

Exponentially-flattened Importance weighted empirical risk minimization (IW-ERM) [1]:

$$\min_{f \in \mathcal{F}} \frac{1}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} L(y_i^{\text{tr}}, f(\mathbf{x}_i^{\text{tr}})) \left( \frac{p_{\text{te}}(\mathbf{x}_i^{\text{tr}})}{p_{\text{tr}}(\mathbf{x}_i^{\text{tr}})} \right)^{\tau}$$

where  $0 \leq \tau \leq 1$  is a tuning parameter for stabilizing the covariate shift adaptation.

- $\tau = 0 \rightarrow$  ERM
- $0 < \tau < 1 \rightarrow$  Intermediate
- $\tau = 1$  IW-ERM

Setting  $\tau$  to  $0 < \tau < 1$  is practically useful.

# Covariate Shift Adaptation

Relative Importance weighted empirical risk minimization (RIW-ERM) [2]:

$$\min_{f \in \mathcal{F}} \frac{1}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} L(y_i^{\text{tr}}, f(\mathbf{x}_i^{\text{tr}})) \frac{p_{\text{te}}(\mathbf{x}_i^{\text{tr}})}{(1 - \alpha)p_{\text{te}}(\mathbf{x}_i^{\text{tr}}) + \alpha p_{\text{tr}}(\mathbf{x}_i^{\text{tr}})}$$

where  $0 \leq \alpha \leq 1$  is a tuning parameter for stabilizing the covariate shift adaptation.

- $\alpha = 0 \rightarrow$  ERM
- $0 < \alpha < 1 \rightarrow$  Intermediate
- $\alpha = 1 \rightarrow$  IW-ERM

$$r_{\alpha}(\mathbf{x}) = \frac{p_{\text{te}}(\mathbf{x})}{(1 - \alpha)p_{\text{tr}}(\mathbf{x}) + \alpha p_{\text{tr}}(\mathbf{x})} < \frac{1}{1 - \alpha}$$

The density ratio is bounded above by  $1/(1 - \alpha)$ .



# Importance Weighted Least Squares

The importance weighted least squares problem can be written as

$$\min_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} r(\mathbf{x}_i^{\text{tr}}) \|y_i^{\text{tr}} - \mathbf{w}^\top \mathbf{x}_i^{\text{tr}}\|_2^2,$$

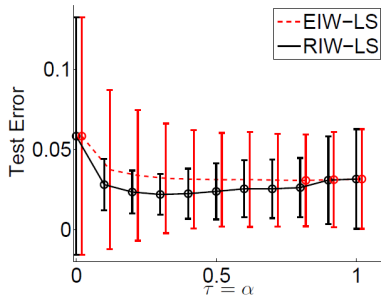
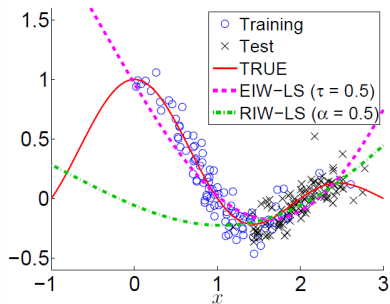
where  $r(\mathbf{x})$  is a weight function (e.g., density-ratio).

Take the derivative w.r.t.  $\mathbf{w}$  and equating it to zero.

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = -\frac{2}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} r(\mathbf{x}_i^{\text{tr}}) (y_i^{\text{tr}} - \mathbf{w}^\top \mathbf{x}_i^{\text{tr}}) \mathbf{x}_i^{\text{tr}} = 0$$
$$\widehat{\mathbf{w}} = \left( \sum_{i=1}^{n_{\text{tr}}} r(\mathbf{x}_i^{\text{tr}}) \mathbf{x}_i^{\text{tr}} \mathbf{x}_i^{\text{tr}\top} \right)^{-1} \sum_{i=1}^{n_{\text{tr}}} r(\mathbf{x}_i^{\text{tr}}) y_i^{\text{tr}} \mathbf{x}_i^{\text{tr}}$$

# Synthetic Example

Comparison of EIW-LS and RIW-LS:



# Supervised Transfer Learning

Problem formulation:

- $\{(\mathbf{x}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{n_{\text{tr}}} \stackrel{\text{i.i.d.}}{\sim} p_{\text{tr}}(\mathbf{x}, y)$
- $\{(\mathbf{x}_j^{\text{te}}, y_j^{\text{te}})\}_{j=1}^{n_{\text{te}}} \stackrel{\text{i.i.d.}}{\sim} p_{\text{te}}(\mathbf{x}, y), n_{\text{te}} \ll n_{\text{tr}}$

We assume to have a large number of training samples and a small number of paired target labeled samples.

- Frustratingly easy domain adaptation [3].
- Multi-task Learning
- Fine-tuning (Deep Learning)

# Importance Weight

Naive approach: Pooling training and test samples

$$\begin{aligned} J(\mathbf{w}) &= \iint \text{loss}(y, f(\mathbf{x}; \mathbf{w})) p_{\text{te}}(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y} \\ &= \alpha \iint \text{loss}(y, f(\mathbf{x}; \mathbf{w})) p_{\text{tr}}(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y} \\ &\quad + (1 - \alpha) \iint \text{loss}(y, f(\mathbf{x}; \mathbf{w})) p_{\text{te}}(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y} \\ &\simeq \frac{\alpha}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} \text{loss}(y_i^{\text{tr}}, f(\mathbf{x}_i^{\text{tr}}; \mathbf{w})) + \frac{(1 - \alpha)}{n_{\text{te}}} \sum_{j=1}^{n_{\text{te}}} \text{loss}(y_j^{\text{te}}, f(\mathbf{x}_j^{\text{te}}; \mathbf{w})). \end{aligned}$$

where  $0 \leq \alpha \leq 1$  is a tuning parameter to control trade off between source and target errors.

# Multi-task Learning

Problem formulation:

- Task 1:  $\{(\mathbf{x}_i^{(1)}, y_i^{(1)})\}_{i=1}^{n_1} \stackrel{\text{i.i.d.}}{\sim} p_1(\mathbf{x}, y)$
- ...
- Task  $M$ :  $\{(\mathbf{x}_j^{(M)}, y_j^{(M)})\}_{j=1}^{n_M} \stackrel{\text{i.i.d.}}{\sim} p_M(\mathbf{x}, y)$
- Linear Models:  $f_1(\mathbf{x}^{(1)}) = \mathbf{w}_1^\top \mathbf{x}^{(1)}, f_2(\mathbf{x}^{(2)}) = \mathbf{w}_2^\top \mathbf{x}^{(2)}, \dots, f_M(\mathbf{x}^{(M)}) = \mathbf{w}_M^\top \mathbf{x}^{(M)}$

$$\min_{\mathbf{w}_1, \dots, \mathbf{w}_M} \sum_{m=1}^M \frac{1}{n_m} \sum_{i=1}^{n_m} \text{loss}(y_i^{(m)}, f_m(\mathbf{x}^{(m)})) + \lambda R(\mathbf{w}_1, \dots, \mathbf{w}_M).$$

where  $R(\mathbf{w}_1, \dots, \mathbf{w}_M)$  is a regularizer.

- $\lambda = 0$  : Independently optimize  $\mathbf{w}$ s
- $\lambda > 0$  : We share some information among models.

# Multi-task Learning

Multi-task learning optimization (Graph-Laplacian).

$$\min_{\mathbf{w}_1, \dots, \mathbf{w}_M} \sum_{m=1}^M \frac{1}{n_m} \sum_{i=1}^{n_m} \text{loss}(y_i^{(m)}, f_m(\mathbf{x}_i^{(m)})) + \lambda \sum_{m=1}^M \sum_{m'=1}^M r_{m,m'} \|\mathbf{w}_m - \mathbf{w}_{m'}\|_2^2.$$

where  $r_{m,m'} \geq 0$  is a model parameter (similarity between models). If  $r_{m,m'} > 0$ , we make  $\mathbf{w}_m$  and  $\mathbf{w}_{m'}$  close.

# Multi-task Learning [4]

Other approach: Explicitly including shared parameter. We decompose  $\mathbf{w}_m = \mathbf{w}_0 + \mathbf{v}_m$

That is

- $f_1(\mathbf{x}^{(1)}) = (\mathbf{w}_0 + \mathbf{v}_1)^\top \mathbf{x}^{(1)},$
- $f_2(\mathbf{x}^{(2)}) = (\mathbf{w}_0 + \mathbf{v}_2)^\top \mathbf{x}^{(2)},$
- ...
- $f_M(\mathbf{x}^{(M)}) = (\mathbf{w}_0 + \mathbf{v}_M)^\top \mathbf{x}^{(M)}$

where  $\mathbf{w}_0$  is a common factor for all models.

For squared-loss, we can write the problem as

$$\min_{\mathbf{w}_1, \dots, \mathbf{w}_M} \frac{1}{2} \sum_{m=1}^M \frac{1}{n_m} \sum_{i=1}^{n_m} \left( y_i^{(m)} - (\mathbf{w}_0 + \mathbf{v}_m)^\top \mathbf{x}_i^{(m)} \right)^2 + \lambda (\|\mathbf{w}_0\|_2^2 + \sum_{m=1}^M \|\mathbf{v}_m\|_2^2)$$

# Supervised Transfer Learning: Frustratingly easy domain adaptation

A *frustratingly easy* feature augmentation approach:

$$\mathbf{z}^{\text{tr}} = (\mathbf{x}^{\text{tr}\top} \quad \mathbf{x}^{\text{tr}\top} \quad \mathbf{0}_d^\top)^\top, \quad \mathbf{z}^{\text{te}} = (\mathbf{x}^{\text{te}\top} \quad \mathbf{0}_d^\top \quad \mathbf{x}^{\text{te}\top})^\top,$$

The inner product of  $\mathbf{z}$  in the same domain is give as

$$\mathbf{z}^{\text{tr}\top} \mathbf{z}^{\text{tr}} = 2\mathbf{x}^{\text{tr}\top} \mathbf{x}^{\text{tr}}, \quad \mathbf{z}^{\text{te}\top} \mathbf{z}^{\text{te}} = 2\mathbf{x}^{\text{te}\top} \mathbf{x}^{\text{te}},$$

while we have

$$\mathbf{z}^{\text{tr}\top} \mathbf{z}^{\text{te}} = \mathbf{x}^{\text{tr}\top} \mathbf{x}^{\text{te}}, .$$

Then, we train a supervised learning method with the transformed vectors  $\mathbf{z}$ . **Super easy!!!!**



# Multi-task Learning

Supervised transfer learning can be regarded as a **two-task** learning problem. First task is for training and second task is for test.

Let us denote the transformed vectors as

$$\begin{aligned}z^{\text{tr}} &= (\mathbf{x}^{\text{tr}\top} \quad \mathbf{x}^{\text{tr}\top} \quad \mathbf{0}_d^\top)^\top \in \mathbb{R}^{3d}, \\z^{\text{te}} &= (\mathbf{x}^{\text{te}\top} \quad \mathbf{0}_d^\top \quad \mathbf{x}^{\text{te}\top})^\top \in \mathbb{R}^{3d},\end{aligned}$$

where  $\mathbf{0}_d \in \mathbb{R}^d$  is the vector whose elements are all zero.

And, we consider a linear regression problem: The model parameter of the linear model can be written as

$$\mathbf{w} = (\mathbf{w}_0^\top \quad \mathbf{v}_1^\top \quad \mathbf{v}_2^\top)^\top \in \mathbb{R}^{3d}$$

# Multi-task Learning

$$\begin{aligned} J(\mathbf{w}) &= \frac{1}{2n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} \|y_i^{\text{tr}} - \mathbf{z}_i^{\text{tr}\top} \mathbf{w}\|_2^2 + \frac{1}{2n_{\text{te}}} \sum_{i=1}^{n_{\text{te}}} \|y_i^{\text{te}} - \mathbf{z}_i^{\text{te}\top} \mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 \\ &= \frac{1}{2} \sum_{m=1}^M \frac{1}{n_m} \sum_{i=1}^{n_m} \left( y_i^{(m)} - (\mathbf{w}_0 + \mathbf{v}_m)^\top \mathbf{x}_i^{(m)} \right)^2 + \lambda (\|\mathbf{w}_0\|_2^2 + \sum_{m=1}^M \|\mathbf{v}_m\|_2^2), \end{aligned}$$

where we use

$$\begin{aligned} \mathbf{w}^\top \mathbf{z}^{\text{tr}} &= (\mathbf{w}_0 + \mathbf{v}_1)^\top \mathbf{x}^{\text{tr}}, & \mathbf{w}^\top \mathbf{z}^{\text{te}} &= (\mathbf{w}_0 + \mathbf{v}_2)^\top \mathbf{x}^{\text{te}} \\ \mathbf{x}^{\text{tr}} &= \mathbf{x}^{(1)}, & \mathbf{x}^{\text{te}} &= \mathbf{x}^{(2)}, \\ \|\mathbf{w}\|_2^2 &= \|\mathbf{w}_0\|_2^2 + \sum_{m=1}^2 \|\mathbf{v}_m\|_2^2. \end{aligned}$$

Frustratingly easy domain adaptation is a multi-task learning.

# Summary

- **Semi-supervised learning.** Use unlabeled samples and assume the data distribution of unlabeled data is same as training.
- Weighted Maximum Likelihood, Graph-based method.
- **Transfer Learning.** Use samples from test data. **Training and test distributions are different.**
- Covariate shift adaptation, frustratingly easy domain adaptation.

- [1] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.
- [2] Makoto Yamada, Taiji Suzuki, Takafumi Kanamori, Hirotaka Hachiya, and Masashi Sugiyama. Relative density-ratio estimation for robust distribution comparison. In *Advances in neural information processing systems*, pages 594–602, 2011.
- [3] Hal Daumé III. Frustratingly easy domain adaptation. *ACL 2007*, page 256, 2007.
- [4] Theodoros Evgeniou and Massimiliano Pontil. Regularized multi-task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109–117, 2004.