DSA5105 Principles of Machine Learning

AY2024/25 Sem1 By Zhao Peiduo

Supervised Learning

Lecture 1

Empirical Risk Minimization (ERM) The learning process aims to find a function $f \in \mathcal{H}$ that minimizes the empirical risk, $R_{\text{emp}}(f) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f(x_i))$, where $y_i = f^*(x_i)$.

Loss Function To quantify how well a function f fits the data, we use a loss function $L(y, \hat{y})$, where y is the true output, and $\hat{y} = f(x)$ is the predicted output. Common loss functions include the mean squared error (MSE) for regression tasks:$L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$, cross-entropy loss for classification task: $L(y, p) = -\sum_i y_i \log(p_i)$, and huber loss for robust regression: $L_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y-\hat{y})^2 & \text{if } |y-\hat{y}| \le \delta \\ \delta|y-\hat{y}| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$

Softmax Function For a multi-class classification problem with K classes, the softmax function is defined as:$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{K};\exp(z_j)}$

General Ordinary Least Squares Formula Consider $x \in \mathbb{R}^d$ and the new hypothesis space $\mathcal{H}_M = \left\{ f : f(x) = \sum_{j=0}^{M-1} w_j \varphi_j(x) \right\}$ Each $\varphi_j : \mathbb{R}^d \to \mathbb{R}$ is called a **basis function** or **feature map**.

We can rewrite the ERM $\min_{w \in \mathbb{R}^M} \frac{1}{2N}\sum_{i=1}^{N}\left(\sum_{j=0}^{M-1} w_j\varphi_j(x_i) - y_i\right)^2$ into $\min_{w\in\mathbb{R}^M} \frac{1}{2N}\|\Phi w - y\|^2$. Solving by setting $\nabla R_{\text{emp}}(\hat{w}) = 0$, we have $\hat{w} = (\Phi^\top\Phi)^{-1}\Phi^\top y$, given invertible $\Phi^T\Phi$.

For cases where $\Phi^T\Phi$ is not invertible, the formula using the Moore-Penrose pseudoinverse is: $\hat{w}(u) = \Phi^\dagger y + \left(I - \Phi^\dagger\Phi\right)u \quad u \in \mathbb{R}^M$

Regularization: To prevent overfitting, regularization techniques add a penalty to the loss function: $\min_{w\in\mathbb{R}^M} \frac{1}{2N}\|\Phi w - y\|^2 + \lambda C(w) \frac{1}{2N}\|\Phi w - y\|^2 + \lambda C(w)$. Minimizing the ERM we get $\hat{w} = \left(\Phi^\top\Phi + \lambda I_M\right)^{-1}\Phi^\top y$ which is always invertible for positive λ.

Common regularization terms:**L_2 (Ridge) regularization:** $\lambda\sum_{j=1}^{p} w_j^2$, **$L_1$ (Lasso) regularization:** $\lambda\sum_{j=1}^{p}|w_j|$

Lecture 2

Reformulation of Ridge Regression We rewrite the regularized least squares solution in another way:
$\hat{w} = \left(\Phi^\top\Phi + \lambda I_M\right)^{-1}\Phi^\top y = \Phi^\top\left(\Phi\Phi^\top + \lambda I_N\right)^{-1} y$

Reformulation of Ridge Regression
$\hat{f}(x) = \sum_{i=1}^{N}\alpha_i\varphi(x_i)^\top\varphi(x)$ $\alpha = (G + \lambda I_N)^{-1}y$ where $G_{ij} = \varphi(x_i)^\top\varphi(x_j)$ is the gram matrix

Kernel Ridge Regression Essentially, the reformulation computes the similarity score between x and x', which can be replaced by a kernel function $K(x_i, x_j)$, allowing computation in high-dimensional feature spaces without explicit feature transformations. The solution to kernel ridge regression is: $f(x) = \sum_{i=1}^{N}\alpha_i K(x_i, x)$ where α_i are coefficients determined based on the training data and the kernel.

Mercer's Theorem and SPD Kernels Suppose k is a SPD kernel. Then, there exists a feature space $\mathcal{H}$ and a feature map $\varphi : \mathbb{R}^d \to \mathcal{H}$ such that $k(x, x') = \varphi(x)^\top\varphi(x')$

SPD kernels properties: $K(x, x') = K(x', x)$ (**Symmetry**) For any n and $\{x_1, \ldots, x_n\}$, the Gram matrix $G_{ij} = k(x_i, x_j)$ is positive semi-definite. (Recall: a matrix G is positive semi-definite if $c^TGc \ge 0$ for any vector c) (**Positive Semi-definiteness**)

Examples of SPD Kernels: **Linear Kernel:** $K(x, x') = x^\top x'$ **Polynomial Kernel:** $K(x, x') = (1 + x^\top x')^d$

Gaussian RBF Kernel: $K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$

Constructing kernels Given valid kernels $k_1(x, x')$ and $k_2(x, x')$, the following new kernels will also be valid:
$k(x, x') = ck_1(x, x')$ $\quad k(x, x') = f(x)k_1(x, x')f(x')$ $\quad k(x, x') = q(k_1(x, x'))$ $\quad k(x, x') = \exp(k_1(x, x'))$

$k(x, x') = k_1(x, x') + k_2(x, x')$ $\quad k(x, x') = k_1(x, x')k_2(x, x')$ $\quad k(x, x') = k_3(\varphi(x), \varphi(x'))$

$k(x, x') = x^\top Ax'$ $\quad k(x, x') = k_a(x_a, x'_a) + k_b(x_b, x'_b)$ $\quad k(x, x') = k_a(x_a, x'_a)k_b(x_b, x'_b)$

Lecture 3

SVM Max Margin Formulation $\max_{\mathbf{w}, b}\frac{1}{\|\mathbf{w}\|}\min_{i=1,\ldots,N}|\mathbf{w}^T\mathbf{x}_i + b|$ subject to $y_i(\mathbf{w}^T\mathbf{x}_i + b) > 0 \quad \forall i$

Optimization Problem $\min_{w,b}\frac{1}{2}\|w\|^2$ subject to: $y_i(w^\top x_i + b) \ge 1 \quad \forall i$. Introducing Lagrange multipliers $\alpha_i \ge 0$, the Lagrangian is: $\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N}\alpha_i[y_i(w^\top x_i + b) - 1]$

Karush-Kuhn-Tucker (KKT) Conditions: Define the Lagrangian $\mathcal{L}(z, \mu) = F(z) + \mu^TG(z)$. Then, under technical conditions, for each locally optimal $\hat{z}$, there exists Lagrange multipliers $\hat{\mu} \in \mathbb{R}^m$ such that:

stationarity $\nabla_z\mathcal{L}(\hat{z}, \hat{\mu}) = 0$ **Primal Feasibility** $G(\hat{z}) \le 0$
Dual Feasibility $\hat{\mu} \ge 0$ **Complementary Slackness** $\hat{\mu}_jG_j(\hat{z}_j) = 0$

Dual Problem $\max_{\mu\ge 0}\tilde{F}(\mu)$ where $\tilde{F}(\mu) = \min_z \mathcal{L}(z, \mu)\mathcal{L}(z, \mu) = F(z) + \mu^TG(z)$

KKT conditions for SVM [noitemsep, topsep=0pt]

1. **From Stationarity** $\hat{w} = \sum_{i=1}^{N}\hat{\mu}_iy_ix_i, \quad 0 = \sum_{i=1}^{N}\hat{\mu}_iy_i$
2. **From Dual Feasibility** $\hat{\mu}_i \ge 0 \quad \text{for} \quad i = 1, \ldots, N$
3. **From Complementary Slackness**$\hat{\mu}_i = 0 \quad \text{or} \quad y_i(\hat{w}^Tx_i + \hat{b}) = 1$
4. **The multipliers** $\hat{\mu}$ can be found by the dual problem $\max_{\mu\in\mathbb{R}^N}\sum_{i=1}^{N}\mu_i - \frac{1}{2}\sum_{i,j}\mu_i\mu_jy_iy_j(x_i^Tx_j)$

Dual formulation of SVM
$\hat{\mu} = \arg\max_{\mu\in\mathbb{R}^N}\sum_{i=1}^{N}\mu_i - \frac{1}{2}\sum_{i,j}\mu_i\mu_jy_iy_j(x_i^Tx_j)$ Subject to: $\hat{\mu} \ge 0$ and $\sum_{i=1}^{N}\hat{\mu}_iy_i = 0$

Decision function: $\hat{f}(x) = \text{sgn}\left(\sum_{i=1}^{N}\hat{\mu}_iy_ix_i^Tx + \hat{b}\right)$ Complementary slackness: $\hat{\mu}_i = 0$ or $1 = y_i(\hat{w}^Tx_i + \hat{b})$

Kernel SVMs

$\hat{\mu} = \arg\max_{\mu\in\mathbb{R}^N}\sum_{i=1}^{N}\mu_i - \frac{1}{2}\sum_{i,j=1}^{N}\mu_i\mu_jy_iy_jk(x_i, x_j)$ Subject to: $\hat{\mu} \ge 0$ and $\sum_{i=1}^{N}\hat{\mu}_iy_i = 0$

Decision function: $\hat{f}(x) = \text{sgn}\left(\sum_{i=1}^{N}\hat{\mu}_iy_ik(x_i, x) + \hat{b}\right)$

Only support vectors satisfying $1 = y_i(\hat{w}^T\varphi_i(x) + \hat{b})$ matter for predictions. This is a **sparse kernel method**.

Lecture 4

Classification and Regression Trees Suppose that the input space is $\mathcal{X}$. A **partition** of $\mathcal{X}$ is a collection of subsets $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_J$ such that $\mathcal{R}_i \cap \mathcal{R}_j = \emptyset$ for $i \ne j$ and $\bigcup_{j=1}^{J}\mathcal{R}_j = \mathcal{X}$

The general decision tree hypothesis space is: $\mathcal{H} = \left\{ f : f(x) = \sum_{j=1}^{J} a_j\mathbb{I}_{x\in\mathcal{R}_j}, \{\mathcal{R}_j\} \text{ is a partition of } \mathcal{X}, a_j \in \mathcal{Y} \right\}$

where $\mathbb{I}_{x\in\mathcal{R}_j} = \begin{cases} 1, & \text{if } x \in \mathcal{R}_j \\ 0, & \text{otherwise} \end{cases}$

A decision tree model $f(x) = \sum_{j=1}^{J}a_j\mathbb{I}_{x\in\mathcal{R}_j}$ depends on both a_j and $\mathcal{R}_j$. For regression we take the **average** label values $a_j = y_j = \frac{\sum_i y_i\mathbb{I}_{x\in\mathcal{R}_j}}{\sum_i\mathbb{I}_{x\in\mathcal{R}_j}}$; For classification we take the **mode** label values $a_j = \text{mode}\{y_i : x_i \in \mathcal{R}_j\}$

Loss function for Decision Trees Classification Entropy: $-\sum_{k=1}^{K}\sum_{j=1}^{J}p_{jk}\log p_{jk}$

Gini Impurity: $\sum_{k=1}^{K}\sum_{j=1}^{J}p_{jk}(1 - p_{jk})$ where p_{jk} is the proportion of samples in $\mathcal{R}_j$ belonging to class k.

Model Ensembling Bagging reduces the variance by aggregating predictions from multiple models trained on different random subsamples of the data:**Regression** $\bar{f}(x) = \frac{1}{m}\sum_{j=1}^{m}f_j(x)$ **Classification** $\bar{f}(x) = \text{Mode}\{f_j(x) : j = 1, \ldots, m\}$

Boosting works by training weak learners sequentially, where each learner focuses on correcting the mistakes of the previous ones. The combined model is a weighted sum of the weak learners: $f(x) = \sum_{t=1}^{T}\alpha_tf_t(x)$ where α_t are coefficients based on each learner's performance. Boosting helps reduce bias.

Key Ideas of AdaBoost 1. Initialize with uniform weight across all training samples
2. Train a classifier/regressor f_1
3. Identify the samples that f_1 got wrong (classification) or has large errors (regression)
4. Weight these samples more heavily and train f_2 on this reweighted dataset
5. Repeat steps 3-5

AdaBoost Implementation

- **Data:** $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$
- Initialize $w_i^{(1)} = \frac{1}{N}$ for all $i = 1, \ldots, N$;
- For $j = 1, \ldots, m$ do
    1. Obtain f_j from: $f_j = \arg\min_{f\in\mathcal{H}}\sum_{i=1}^{N}w_i^{(j)}\mathbb{I}_{y_i\ne f(x_i)}$
    2. Compute combination coefficients: $\delta_j = \frac{\sum_{i=1}^{N}w_i^{(j)}\mathbb{I}_{y_i\ne f_j(x_i)}}{\sum_{i=1}^{N}w_i^{(j)}}$ $\alpha_j = \log\left(\frac{1-\delta_j}{\delta_j}\right)$
    3. Update weights:$w_i^{(j+1)} = w_i^{(j)}\exp\left(\alpha_j\mathbb{I}_{y_i\ne f_j(x_i)}\right)$
- Return: $\bar{f}(x) = \text{Sign}\left(\sum_{j=1}^{m}\alpha_jf_j(x)\right)$

Cross-Validation Cross-validation is essential for tuning model hyperparameters. In k-fold cross-validation, the data is split into k subsets. The model is trained on $k - 1$ subsets and validated on the remaining one. This process is repeated k-times, and the average performance is evaluated. Cross-validation ensures model generalizability.

Lecture 5

Activation Functions
Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}, [0, 1]$ **Tanh:** $\sigma(z) = \tanh(z), [-1, 1]$ **ReLU** $\sigma(z) = \max(0, z), [0, \inf]$, leaky ReLU: δzif $z < 0$ instead of 0

Gradient Descent $w^{(t+1)} = w^{(t)} - \eta\frac{\partial L}{\partial w}$ When η is too small, updates are **slow**; When η is too large, the updates may become **unstable**.

Stochastic Gradient Descent (SGD): $\theta_{k+1} = \theta_k - \eta\frac{1}{B}\sum_{i\in I_B}\nabla\Phi_i(\theta_k)$, this diminishes the probability of stucking at local minima as the main drawback for GD.

Convex function is a function satisfying: $\Phi(\lambda\theta + (1-\lambda)\theta') \le \lambda\Phi(\theta) + (1-\lambda)\Phi(\theta')$ for all $\theta, \theta' \in \mathbb{R}^p$ and $\lambda \in [0, 1]$
Deep Neural Networks (DNNs) Deep neural networks are an extension of shallow networks. The idea is to stack n hidden layers together and forward pass the x sequentially.

Back-propagation Algorithm

- Initialize $x_0 = x \in \mathbb{R}^d$.
- For $t = 0, 1, \ldots, T$: $x_{t+1} = g_t(x_t, W_t) = \sigma(W_tx_t)$
- Set $p_{T+1} = \nabla_xL(x_{T+1}, y)$.
- For $t = T, T-1, \ldots, 1$:
    - $\nabla_{W_t}\Phi = p_{t+1}^T\nabla_Wg_t(x_t, W_t)$
    - $p_t = [\nabla_xg_t(x_t, W_t)]^Tp_{t+1}$
- Return $\{\nabla_{W_t}\Phi : t = 0, \ldots, T\}$.

Neural Network Architecture Example

The architecture of the network consists of: $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad x_1 \in \mathbb{R}, \quad x_2 \in \mathbb{R}$ **Input layer**: $x_0 = x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$

1st hidden layer with 2 neurons. **2nd hidden layer with 3 neurons**. **Output**: Scalar.

1st Hidden Layer $x_1 = W_0x_0 + b_0 \quad W_0 \in \mathbb{R}^{2\times 2}, \quad b_0 \in \mathbb{R}^{2\times 1} \quad x_1 = \begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix}, \quad x_1 \in \mathbb{R}^{2\times 1} \quad x_1 = \text{ReLU}(x_1)$

2nd Hidden Layer $x_2 = W_1x_1 + b_1 \quad W_1 \in \mathbb{R}^{3\times 2}, \quad b_1 \in \mathbb{R}^{3\times 1} \quad x_2 = \text{ReLU}(x_2), \quad x_2 \in \mathbb{R}^{3\times 1}$

Back-propagation Example Computation
The model is defined as: $y(x) = v\delta(w_1\delta(w_0x)), \quad x, w_0, w_1, v \in \mathbb{R}$, where δ is the identity function, i.e., $\delta(z) = z$. The loss function is: $L = (y(x) - y)^2$ We want to compute the gradients: $\frac{\partial L}{\partial v}, \quad \frac{\partial L}{\partial w_1}, \quad \frac{\partial L}{\partial w_0}$

Forward pass: $x_1 = w_0 x \; x_2 = v w_1 x_1 \; L = (x_2 - y)^2$

Backward pass:

$$p_2 = \frac{\partial L}{\partial x_2} = 2(x_2 - y) \quad \frac{\partial L}{\partial v} = \frac{\partial L}{\partial x_2} \cdot \frac{\partial x_2}{\partial v} = 2(x_2 - y) w_1 x_1 \qquad \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_1} = 2(x_2 - y) v x_1$$

$$p_1 = \frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial x_2} \cdot \frac{\partial x_2}{\partial x_1} = 2(x_2 - y) v w_1 \qquad \frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial x_1} \cdot \frac{\partial x_1}{\partial w_0} = 2(x_2 - y) v w_1 x$$

Unsupervised Learning

Lecture 6

PCA Algorithm Implementation

- **Data**: $\mathcal{D} = \{x_i\}_{i=1}^N, x_i \in \mathbb{R}^d$ for all i.
- **Hyper-parameters**: m (reduction dimension).
- Compute sample covariance matrix $S = \frac{1}{N} \sum_{i=1}^N x_i x_i^T$.
- Compute the first m eigenvectors $\{u_1, \dots, u_m\}$ and eigenvalues $\{\lambda_1, \dots, \lambda_m\}$.
- Form $d \times m$ matrix U_m whose j^{th} column is u_j.
- Compute $Z_m = XU_m$.
- Return Principal component scores Z_m, Eigenvalues, and eigenvectors λ_j, u_j for $j = 1, \dots, m$.

PCA in Feature Space

- **Data**: $\mathcal{D} = \{x_i\}_{i=1}^N, x_i \in \mathbb{R}^d$ for all i
- **Hyper-parameters**: m (reduction dimension), ϕ (feature maps);
- Compute design matrix $\Phi_{ij} = \phi_j(x_i)$;
- Center design matrix $\Phi_{ij} \leftarrow \Phi_{ij} - \frac{1}{N} \sum_{i=1}^N \Phi_{ij}$;
- Compute sample covariance matrix $S_\phi = \frac{1}{N} \Phi^\top \Phi$;
- Compute the first m eigenvectors $\{u_1, \dots, u_m\}$ and eigenvalues $\{\lambda_1, \dots, \lambda_m\}$ of S_ϕ;
- Form $d \times m$ matrix U_m whose j^{th} column is u_j;
- Compute $Z_m = \Phi U_m$;
- **Return**: Principal component score Z_m, Eigenvalues and eigenvectors λ_j, u_j for $j = 1, \dots, m$.

PCA whitening transform Principal component scores are given by $Z = XU$ where X is the original features and U is the matrix of eigenvectors. The transformation $X' = XU\Lambda^{-\frac{1}{2}}$, where Λ is the matrix of eigenvalues makes $\text{cov}(X') = I$.

Useful derivatives
$\frac{\partial}{\partial x}\left(x^T A x\right) = 2Ax \quad \frac{\partial}{\partial x}\left(A^T x\right) = A$