DSA5105 Principles of Machine Learning

AY2024/25 Sem1 By Zhao Peiduo

Supervised Learning

Lecture 1

Supervised Learning Supervised learning is the most common type of machine learning problem, where the goal is to make predictions and learn a function that maps an input to an output based on example input-output pairs.

Problem Setup Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, where x_i are the inputs and y_i are the corresponding outputs or labels, the goal is to learn the mapping $f : X \to Y$ such that $f(x_i) \approx y_i$ for all i.

Hypothesis Space The **oracle** $f*$ is unknown to us except through the dataset. The function f is chosen from a hypothesis space $\mathcal{H}$, which is a set of candidate functions. For example, in linear regression, the hypothesis space might be $\mathcal{H} = \{f : f(x) = w_0 + w_1 x \mid w_0, w_1 \in \mathbb{R}\}$.

Three Paradigms of Supervised Learning

- **Approximation:** Analyze the breadth and depth of the hypothesis space $\mathcal{H}$ to determine if it contains, or closely approximates, the optimal function. (How large is our hypothesis space?)
- **Optimization:** Design and implement efficient algorithms to address the empirical risk minimization problem and find or closely approximate the best function within $\mathcal{H}$. (How can we find or get close to an approximation $\hat{f}$ of f^*?)
- **Generalization:** Evaluate whether the optimized model can effectively generalize to new, unseen data, focusing on the interplay between data size and model complexity. This is done through minimizing the population risk: $R_{\text{pop}}(f) = \mathbb{E}_{(x,y)\sim P}[L(y, f(x))]$.(Can the $\hat{f}$ found generalized to unseen examples?)

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:$L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$

Ordinary Least Squares Formula The formula for Ordinary Least Squares in a simple linear regression context is given by: $\beta = (X^T X)^{-1} X^T y$. In 1D context, the empericial risk minization problem can be defined as $\min_{w_0,w_1} \frac{1}{2N}\sum_{i=1}^N (w_0 + w_1 x_i - y_i)^2$. By setting the partial derivatives to zero, the ordinary least sqaures formula 1D is given by: $\hat{w}_0 = \bar{y} - \hat{w}_1 \bar{x} \quad \hat{w}_1 = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2} \quad \bar{x} = \frac{1}{N}\sum_i x_i \quad \bar{y} = \frac{1}{N}\sum_i y_i$

Huber Loss The Huber loss, used for robust regression, is defined as: $L_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| \le \delta \\ \delta|y - \hat{y}| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$

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: $\hat{w}(u) = \Phi^\dagger y + \left(I - \Phi^\dagger\Phi\right)u \quad u \in \mathbb{R}^M$

Overfitting: Overfitting occurs when the hypothesis space $\mathcal{H}$ is too large, allowing the model to fit the noise in the training data. This results in poor generalization to new data.

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:** $\lambda\sum_{j=1}^p |w_j|$

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

Cross-Entropy Loss Commonly used in classification tasks, the cross-entropy loss is: $L(y, p) = -\sum_i y_i \log(p_i)$

Lecture 2

Another view of feature maps One can also view feature maps as implicitly defining some sort of **similarity measure**.Consider two vectors u and v. Then, $u^T v$ measures how similar they are. Feature maps define a **similarity** between two samples x, x' by computing the dot product in **feature space**.

Reformulation of Ridge Regression We rewrite the regularized least squares solution in another way:

$\hat{w} = \left(\Phi^\top\Phi + \lambda I_M\right)^{-1}\Phi^\top y = \Phi^\top\left(\Phi\Phi^\top + \lambda I_N\right)^{-1} y$

Proof:

$\left(\Phi^\top\Phi + \lambda I_M\right)\Phi^\top = \Phi^\top\left(\Phi\Phi^\top + \lambda I_N\right)^{-1}$

$= \Phi^\top\left(\Phi\Phi^\top + \lambda I_N\right)^{-1} y$

$= \Phi^\top\Phi^\top y\left(\Phi^\top + \lambda I_N\right)^{-1}$

$\lambda I_N^{-1} y = \left(\Phi^\top\Phi + \lambda I_M\right)^{-1}\Phi^\top y$

Reformulation of Ridge Regression

$\hat{f}(x) = \sum_{i=1}^N \alpha_i \varphi(x_i)^\top \varphi(x)$

$\alpha = (G + \lambda I_N)^{-1} y \quad$ where $\quad G_{ij} = \varphi(x_i)^\top \varphi(x_j)$ 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.

Kernel Construction To build a kernel, we can write an expression in the form of a dot product of the same function with variable x and x', but this method does not always work.

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:

- **Symmetry:** $K(x, x') = K(x', x)$
- **Positive Semi-Definiteness:** $\sum_{i,j} \alpha_i \alpha_j K(x_i, x_j) \ge 0$ for any α_i

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:

$k(x, x') = ck_1(x, x') \qquad k(x, x') = f(x)k_1(x, x')f(x')$

$k(x, x') = q(k_1(x, x')) \qquad k(x, x') = \exp(k_1(x, x'))$

$k(x, x') = k_1(x, x') + k_2(x, x') \qquad k(x, x') = k_1(x, x')k_2(x, x')$

$k(x, x') = k_3(\varphi(x), \varphi(x')) \qquad k(x, x') = x^\top A x'$

$k(x, x') = k_a(x_a, x'_a) + k_b(x_b, x'_b) \qquad k(x, x') = k_a(x_a, x'_a)k_b(x_b, x'_b)$

Lecture 3

Support Vector Machines (SVM)

Support Vector Machines (SVMs) are designed for binary classification. They find a hyperplane that separates two classes with the maximum margin, defined as the minimum distance between the separating hyperplane and the data points.

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 $\mathcal{L}(z, \mu) = F(z) + \mu^T G(z)$. 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 $\hat{\mu}_j G_j(\hat{z}_j) = 0$

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

KKT conditions for SVM

1. From Stationarity $\hat{w} = \sum_{i=1}^N \hat{\mu}_i y_i x_i, \quad 0 = \sum_{i=1}^N \hat{\mu}_i y_i$

2. From Dual Feasibility $\hat{\mu}_i \ge 0 \quad$ for $\quad i = 1, \ldots, N$

3. From Complementary Slackness$\hat{\mu}_i = 0 \quad$ or $\quad y_i(\hat{w}^T x_i + \hat{b}) = 1$

4. The multipliers $\hat{\mu}$ can be found by the dual problem $\max_{\mu\in\mathbb{R}^N} \sum_{i=1}^N \mu_i - \frac{1}{2}\sum_{i,j} \mu_i\mu_j y_i y_j (x_i^T x_j)$

Dual formulation of SVM

$\hat{\mu} = \arg\max_{\mu\in\mathbb{R}^N} \sum_{i=1}^N \mu_i - \frac{1}{2}\sum_{i,j} \mu_i\mu_j y_i y_j (x_i^T x_j)$

Subject to: $\hat{\mu} \ge 0 \quad$ and $\quad \sum_{i=1}^N \hat{\mu}_i y_i = 0$

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

Complementary slackness: $\hat{\mu}_i = 0 \quad$ or $\quad 1 = y_i(\hat{w}^T x_i + \hat{b})$

Kernel Support Vector Machines

$\hat{\mu} = \arg\max_{\mu\in\mathbb{R}^N} \sum_{i=1}^N \mu_i - \frac{1}{2}\sum_{i,j=1}^N \mu_i\mu_j y_i y_j k(x_i, x_j)$

Subject to: $\hat{\mu} \ge 0 \quad$ and $\quad \sum_{i=1}^N \hat{\mu}_i y_i = 0$

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

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

Kernel Ridge Regression (KRR)
Kernel ridge regression is an extension of ridge regression that allows for computation in high-dimensional feature spaces using a kernel function $K(x, x')$. The goal is to minimize the regularized loss:

$$\min_w \left\{ \frac{1}{2} \|y - Xw\|^2 + \lambda \|w\|^2 \right\}$$

The solution is given by:

$$f(x) = \sum_{i=1}^{N} \alpha_i K(x_i, x)$$

where $\alpha = (K + \lambda I)^{-1} y$ and $K(x, x')$ is a symmetric positive definite kernel. This approach generalizes linear models to non-linear relationships without explicit feature transformations, using the kernel trick.
Model Ensembling
Bagging reduces the variance by aggregating predictions from multiple models trained on different random subsamples of the data. The final prediction for regression is:

$$f(x) = \frac{1}{m} \sum_{i=1}^{m} f_i(x)$$

For classification, the majority vote is taken. This method helps stabilize high-variance models like decision trees.
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:

$$f(x) = \sum_{t=1}^{T} \alpha_t f_t(x)$$

where α_t are coefficients based on each learner's performance. Boosting helps reduce bias.
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.

Bias-Variance Tradeoff
The bias-variance tradeoff is critical for understanding model performance:

$$\text{MSE} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Bias measures the error introduced by assuming a simple model. Increasing complexity reduces bias but increases **variance**, which is the model's sensitivity to the training set.
Regularization: Ridge Regression
Ridge regression adds an L_2 penalty to linear regression to prevent overfitting:

$$\min_w \left\{ \frac{1}{2} \|y - Xw\|^2 + \lambda \|w\|^2 \right\}$$

The solution is:

$$w = (X^\top X + \lambda I)^{-1} X^\top y$$

where λ controls the tradeoff between bias and variance. Larger λ values increase bias but decrease variance, helping avoid overfitting.
Neural Networks
Neural networks consist of multiple layers where each layer applies a weighted linear transformation followed by a non-linear activation. The general form is:

$$f(x) = v^\top \sigma(Wx + b)$$

Training neural networks involves minimizing the loss function using gradient descent:

$$w^{(t+1)} = w^{(t)} - \eta \frac{\partial L}{\partial w}$$

Backpropagation efficiently computes gradients using the chain rule, allowing neural networks to adjust weights and minimize error.