(1.1): Feature Scaling Scale the data to a fixed range $[0,1]$. Min-Max Scaling:

$$x_{minmax} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{1.1}$$

(1.2): The Task, T Tasks are usually described in terms of how the machine learning should process an example: $x \in \mathbb{R}^n$ where each entry x_i is a feature.

Classification: Learn $f : \mathbb{R}^n \to \{1, \ldots, k\}$

Regression: Learn $f : \mathbb{R}^n \to \mathbb{R}$

Transcription: Unstructured representation to discrete textual form. Example: Optical character recognition, speech recognition.

Machine translation: Sequence to sequence. Example: Translate English to French.

Synthesis and sampling: Generate examples that are similar to those in the training data. Example: Generate textures for video games, speech synthesis.

(1.2.1): Supervised Learning y is real-valued → regression. y is categorical → classification.

(1.3): Cross-Validation

(1.3.1): k-fold cross-validation Divide data into k folds. Train on $k-1$ folds, use the kth to measure error. Repeat k times; use average error to measure generalization accuracy. Statistically valid and gives good accuracy estimates.

(1.3.2): Leave-one-out cross validation (LOOCV) k-fold cross validation with $k = N$, where N is the number of data points. Quite accurate, but also expensive, since it requires building N models.

(1.4): AIC/BIC Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) – compares different models to choose one that best fits the data. The goal of both AIC and BIC is to balance the goodness-of-fit of the model with its complexity, in order to avoid overfitting or underfitting. Both AIC and BIC penalize models with large number of parameters relative to the size of the data, but BIC penalizes more severely.

$$\begin{aligned} \min AIC &= \min\{2m - 2\log(L)\} \\ \min BIC &= \min\{m\log(n) - 2\log(L)\} \end{aligned} \tag{1.2}$$

where m is the number of model parameters, n is the number of data points, and L is the maximum likelihood of the model.

(2.1): Perceptron Algorithm Set time $t = 1$, start with vector $\mathbf{w}_1 = \vec{0}$. Given example $\mathbf{x}$, predict positive iff (if and only if) $\mathbf{w}_1 \cdot \mathbf{x} \geq \mathbf{0}$. On a mistake, update as follows: false positive, then update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathbf{x}$; false negative, then update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{x}$.

(2.2): Kernel Closure Properties Easily create new kernels using basic ones. If $K_1(\cdot,\cdot)$ and $K_2(\cdot,\cdot)$ are kernels and $c_1 \geq 0$, $c_2 \geq 0$, then – $K = c_1K_1 + c_2K_2$ is a kernel, $K = K_1K_2$ is a kernel.

(2.3): Kernel Benefits Offers great modularity. No need to change the underlying learning algorithm to accommodate a particular choice of kernel function. Can substitute a different algorithm while maintaining the same kernel.

(2.4): Conclusion Strengths: Fast. Training is relatively easy. It scales relatively well high dimensional data. Tradeoff between classifier complexity and error can be controlled explicitly. Non-traditional data like strings and trees can be used as input to SVM, instead of feature vectors.
Weaknesses: Need to choose a "good" kernel function.
0 – 1 loss: $L(h(\mathbf{x}), y) = 1, h(\mathbf{x}) \neq y$ otherwise $L(h(\mathbf{x}), y) = 0$
L_2 Loss: $L(h(\mathbf{x}), y) = (h(\mathbf{x}) - y)^2$
Hinge Loss: $L(h(\mathbf{x}), y) = \max\{0, 1 - yh(\mathbf{x})\}$
Exponential Loss: $L(h(\mathbf{x}), y) = e^{-yh(\mathbf{x})}$

$$\begin{aligned} P(E) &= \sum_{rows\ matching\ E} P(row) \\ P(E_1 \mid E_2) &= \frac{E_1 \wedge E_2}{P(E_2)} = \frac{\sum_{\text{rows matching } E_1 \text{ and } E_2} P(row)}{\sum_{\text{rows matching } E_2} P(row)} \end{aligned} \tag{3.1}$$

(3.1): Probability Distribution

(3.1.1): Bernoulli Distribution Random Variable X takes values $\{0,1\}$ (flipping a coin) such that

$$P(X = 1) = p = 1 - P(X = 0)$$

(3.1.2): Binomial Distribution Random Variable X takes values $\{1, 2, \ldots, n\}$ representing the number of successes $X = 1$ in n Bernoulli trials (flipping a coin n times).

$$P(X = k) = f(n, p, k) = C_n^k p^k (1-p)^{n-k}$$

(3.1.3): Categorical Distribution Random Variable X takes on values in $\{1, 2, \ldots, k\}$ such that (rolling a die)

$$P(X = i) = p_i \text{ and } \sum_1^k p_i = 1$$

(3.1.4): Multinomial Distribution Let the random variables $X_i (i = 1, 2, \ldots, k)$ indicates the number of times outcome i was observed over the n trials (rolling a die n times). The vector $X = (X_1, X_2, \ldots, X_k)$ follows a multinomial distribution (n, p) where $p = (p_1, p_2, \ldots, p_k)$ and $\sum_1^k = 1$

$$f(x_1, x_2, \ldots, x_k, n, p) = P(X_1 = x_1, X_2 = x_2, \ldots, X_k = x_k)$$

$$= \frac{n!}{x_1! \times x_2! \times \cdots \times x_k!} \times p_1^{x1} \times p_2^{x2} \times \cdots \times p_k^{xk} \text{ where } \sum_{i=1}^k x_i = n$$

(3.2): Bayes' Rule

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \tag{3.2}$$

(3.3): Maximum APosteriori Estimate

$$\begin{aligned} P(h|D) &= \frac{P(D|h) \times P(h)}{P(D)} \\ h_{MAP} &= \arg\max_{h \in H} P(h|D) = \arg\max_{h \in H} P(D|h) \times P(h) \end{aligned} \tag{3.3}$$

(3.4): Maximum Likelihood Estimate Here we just look for the hypothesis that best explains the data

(3.5): Na ive Bayes Classifier

$$\begin{aligned} Y_{NB} &= \arg\max_{y_j \in Y} P(y_j) \prod_{i=1}^n P(x_i|y_j) \\ P(x_i|y) &= \frac{\text{number of } x_i \text{ labeled as } y}{\text{total number of label } y} = \frac{n_i}{n} = \frac{n_i + \alpha}{n + \alpha d} \end{aligned} \tag{3.4}$$

where $\alpha > 0$ is a smoothing parameter, d is the dimension of the input.

(3.6): Continuous Features Assume $P(x_i|y)$ has a Gaussian (normal) distribution. It is a continuous distribution with probability density function:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.5}$$

μ is the mean of the distribution. σ^2 is the variance of the distribution. x is a continuous variance $(-\infty \leq x \leq \infty)$

(3.7): Training Bayesian Classifier During training, typically log-space is used.

$$y_{NB} = \arg\max_y \left[\log P(y) \prod_{i=1}^n P(x_i|y)\right] = \arg\max_y \left[\log P(y) + \sum_{i=1}^n \log P(x_i|y)\right]$$

(3.8): Evaluating Classifiers Gold Label is the correct output class label of an input. Confusion Matrix is a table for visualizing how a classifier performs with respect to the gold labels, using two dimensions (system output and gold labels), and each cell labeling a set of possible outcomes. TP and TN are correctly classified outputs belonging to the positive and negative class, respectively. FP and FN are incorrectly classified outputs.

$$\textbf{Precision} = \frac{\text{true positives}}{\text{true positives + false positives}} \tag{3.6}$$

$$\textbf{Recall} = \frac{\text{true positives}}{\text{true positives + false negatives}} \tag{3.7}$$

$$\textbf{F} - \textbf{measure} = F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{3.8}$$

(3.9): ROC Curve The ROC curve is the plot of the true positive rate (recall) (TPR) (3.7) against the false positive rate (FPR).

$$\textbf{FPR} = \frac{\text{false positives}}{\text{false positives + true negatives}} \tag{3.9}$$

(3.10): Naïve Bayes: Two Classes Take logarithm; we predict $y = 1$ iff

$$\log \frac{P(y_j = 1)}{P(y_j = 0)} + \sum_i \log \frac{1 - p_i}{1 - q_i} + \sum_i \left(\log \frac{p_i}{1 - p_i} - \log \frac{q_i}{1 - q_i}\right) x_i > 0$$

(4.1): Generative Classifiers If we are distinguishing cat from dog images using a Generative Classifier, we build a model of what is in a cat image. It knows about whiskers, ears, eyes and assigns a probability to any image to determine how cat-like is that image? Similarly, build a model of what is in a dog image. Now given a new image, run both models and see which one fits better.

(4.2): Discriminative Classifiers If we are distinguishing cat from dog images using a Discriminative Classifier. Just try to distinguish dogs from cats. Oh look, dogs have collars. Ignore everything else.

(4.3): Generative vs Discriminative Classifiers Generative Classifiers (Naïve Bayes) – Assume some functional form for conditional independence. Estimate parameters of $P(D|h)$, $P(h)$ directly from training data. Use Bayes' rule to calculate $P(h|D)$. Why not learn $P(h|D)$ or the decision boundary directly? Discriminative Classifiers (Logistic Regression) – Assume some functional form for $P(h|D)$ or for the decision boundary. Estimate parameters of $P(h|D)$ directly from training data.

(4.4): Learning a Logistic Regression Classifier Given n input-output pairs – 1) A feature representation of the input. For each input observation x_i, a vector of features $[x_1, x_2, \ldots, x_d]$. 2) A classification function that computes y, the estimated class, via $P(y|x)$, using the sigmoid of softmax functions. 3) An objective function for learning, like cross-entropy loss. 4) An algorithm for optimizing the objective function, like stochastic gradient ascent/descent.

(4.5): Logistic Regression Logistic Regression assumes the following function form for $P(y|x)$:

$$P(y = 1|x) = \frac{1}{1 + e^{-(\sum_i w_i x_i + b)}}$$

$$\frac{P(y = 1|x)}{P(y = 0|x)} = e^{(\sum_i w_i x_i + b)} > 1 \Rightarrow \sum_i w_i x_i + b > 0$$

Logistic Regression is a linear classifier. Turning a probability into a classifier using the logistic function:

$$y_{LR} \begin{cases} 1 & \text{if } P(y = 1|x) \geq 0.5 \quad \leftarrow w_i x_i + b \geq 0 \\ 0 & \text{otherwise} \quad \leftarrow w_i x_i + b < 0 \end{cases}$$

(4.6): Training Logistic Regression We'll focus on binary classification. We parameterize (w_i, b) as θ:

$$P(y_i = 0|x_i, \theta) = \frac{1}{e^{\sum_i w_i x_i + b} + 1}$$

$$P(y_i = 1|x_i, \theta) = \frac{e^{\sum_i w_i x_i + b}}{e^{\sum_i w_i x_i + b} + 1}$$

$$P(y_i|x_i, \theta) = \frac{e^{y_i \sum_i w_i x_i + b}}{e^{\sum_i w_i x_i + b} + 1}$$

How do we learn parameters θ?

(4.7): Cross-Entropy Loss We want to know how far is the classifier output $\hat{y}$ from the true output y. Let's call this difference $L(\hat{y}, y)$. Since there are only 2 discrete outcomes (0 or 1), we can express the probability $P(y|x)$ from our classifiers as:

$$P(y|x) = \hat{y}^y \cdot (1 - \hat{y})^{1-y}$$

Goal: maximize the probability of the correct label $P(y|x)$. Maximize:

$$\begin{aligned} P(y|x) &= \hat{y}^y \cdot (1 - \hat{y})^{1-y} \\ \log(P(y|x)) &= \log\left(\hat{y}^y \cdot (1 - \hat{y})^{1-y}\right) \\ &= y\log(\hat{y}) + (1 - y)\log(1 - \hat{y}) \end{aligned}$$

We want to minimize the cross-entropy loss:

$$\min_\theta L_{CE}(\hat{y}, y) = \log\left(1 + e^{\sum_i w_i x_i + b}\right) + y\left(\sum_i w_i x_i + b\right)$$

(4.8): Minimizing Cross-Entropy Loss $\min_\theta L_{CE}(\hat{y}, y)$ Convex/Concave functions have a global minimum/maxima. Gradient Ascent/Descent is used for finding the maximum/minimum of a concave/convex function.

(4.9): Gradients Gradient Ascent/Descent: Find the gradient of the function at the current point and move in the same/opposite direction.

(4.10): Gradient Descent for Logistic Regression Let us represent $\hat{y} = f(x, \theta)$. Gradient:

$$\nabla_\theta L(f(x, \theta), y) = \left[\frac{\partial L(f(x, \theta), y)}{\partial b}, \frac{\partial L(f(x, \theta), y)}{\partial w_1}, \frac{\partial L(f(x, \theta), y)}{\partial w_2}, \ldots, \frac{\partial L(f(x, \theta), y)}{\partial w_d}\right] \tag{4.1}$$

Update Rule:

$$\begin{aligned} \Delta\theta &= \eta \cdot \nabla_\theta L(f(x, \theta), y) \\ \theta_{t+1} &= \theta_t - \eta \cdot \frac{\partial}{\partial(w, b)} L(f(x, \theta), y) \end{aligned} \tag{4.2}$$

Gradient descent algorithm will iterate until $\Delta\theta < \epsilon$.

$$L_{CE}(f(x, \theta), y) = \log\left(1 + e^{\sum_i w_i x_i + b}\right) - y\left(\sum_i w_i x_i + b\right)$$

$$\theta_{t+1} = \theta_t - \eta \cdot x_i \left[\hat{P}(y = 1|x, \theta_t) - y\right]$$

(4.11): Learning Rate (η) Large $\eta \Rightarrow$ Fast convergence but larger residual error. Also, possible oscillations. Small $\eta \Rightarrow$ Slow convergence but small residual error.

(4.12): Batch Training Stochastic gradient descent is called stochastic because it chooses a single random example at a time, moving the weights to improve performance on that single example. This results in very choppy movements, so it's common to compute the gradient over batches of training instances rather than a single instance. Training data: $\{x_i, y_i\}_{i=i...n}$ where $x_i = (x_{i1}, x_{i2}, \dots, x_{id})$, n is the total instances in a batch and d is the dimension of an instance.

$$\theta_{t+1} = \theta_t - \frac{\eta}{n} \times \sum_{i=1}^{n} xij \left[\frac{1}{1+e^{-\theta^T \mathbf{x}}} - y_i \right] \tag{4.3}$$

(4.13): Regularization Regularization is used to avoid overfitting. The weights for features will attempt to perfectly fit details of the training set, modeling even noisy data that just accidentally correlate with the class. The problem is called overfitting. A good model should generalize well from the training data to the unseen test set, but a model that overfits will have poor generalization. To avoid overfitting, a new regularization term $R(\theta)$ is added to the loss function.

$$\min_{\theta} L_{reg}(\hat{y}, y) = -\frac{1}{n} \sum_{i=1}^{n} [y \log(\hat{y}) + (1-y)\log(1-\hat{y})] + \lambda R(\theta) \tag{4.4}$$

(4.14): L1 Regularization Uses the L1 norm (Manhattan distance) of the weights.

$$R(\theta) = ||\theta||_1 = \sum_{j=0}^{d} |\theta_j| \tag{4.5}$$

(4.15): L2 Regularization L2 Regularization is also called Ridge Regularization. Uses the square of the L2 (Euclidean) norm of the weights.

$$\min_{\theta} R(\theta) = ||\theta||_2^2 = \sum_{j=0}^{d} \theta_j^2 \tag{4.6}$$

(4.16): Training Phase

$$\theta_{t+1} = \theta_t - \frac{\eta}{n} \times \sum_{i=1}^{n} x_{ij} \left[\frac{1}{1+e^{-\theta^T \mathbf{x}}} - y_i \right]$$

1) Calculate the factor $-\theta^T \mathbf{x}$ for every example in the dataset. 2) Calculate the factor $\sum_{i=1}^{n} x_{ij} \left[\frac{1}{1+e^{-\theta^T \mathbf{x}}} - y_i \right]$ for every example in the dataset, for every θ 3) Compute every θ

(4.17): Multinomial Logistic Regression The loss function for multinomial logistic regression generalizes the loss function for binary logistic regression from 2 to K classes. The true label y is a vector with K elements, each corresponding to a class, with $y_c = 1$ if the correct class is c, with all other elements of y being 0. The classifier will produce an estimate vector with K elements $\hat{y}$, each element $\hat{y}_k$ of which represents the estimated probability $P(y_k = 1|x)$.

$$\textsc{Softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp z_j} \quad 1 \le i \le K \tag{4.7}$$

$$L_{CE}(\hat{y}, y) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k) \tag{4.8}$$

$$\begin{aligned} L_{CE}(\hat{y}, y) &= -\log(\hat{y}_c) \\ &= -\log\left(\frac{e^{\sum_{i=1}^{d} w_c x_i + b_c}}{\sum_{j=1}^{K} e^{\sum_{i=1}^{d} w_j x_i + b_j}} \right) \end{aligned} \tag{4.9}$$

$$\frac{\partial L_{CE}}{\partial(w_k, b_k)} = x_i \left[\frac{e^{\sum_{i=1}^{d} w_k x_i + b_k}}{\sum_{j=1}^{K} e^{\sum_{i=1}^{d} w_j x_i + b_j}} - y_k \right] \tag{4.10}$$

(5.1): Simplest Linear Regression The regression model is:

$$y = w_1 x + w_0$$

Two parameters: the slope of the line w_1, the y-intercept w_0.

(5.2): Linear Regression Function Model

$$f(x) = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_d x_d = w_0 + \sum_{j=1}^{d} w_j x_j = \mathbf{w}^T \mathbf{x}$$

$w_0, w_1, \dots, w_d$ are the parameters (weights) and $\mathbf{x} = [1, x_1, x_2, \dots, x_d]$

(5.3): Error Error function measures how much our predictions deviate from the desired answers. Mean-squared error (MSE):

$$\begin{aligned} J_n &= \frac{1}{2n} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i))^2 \\ &= \frac{1}{2n} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \end{aligned} \tag{5.1}$$

Learning: We want to find the weights minimizing the error. In (5.1), $y_i - \mathbf{w}^T \mathbf{x}_i$ is the residual and $\sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$ is the residual sum of squares (RSS).

(5.4): Optimization For the optimal set of parameters, derivatives of the error with respect to each parameter must be 0.

$$\frac{\partial}{\partial w_j} J_n(\mathbf{w}) = -\frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 x_{i0} - w_1 x_{i1} - \dots - w_d x_{id}) x_{ij} = 0$$

Vector of derivatives:

$$\nabla_w (J_n(\mathbf{w})) = -\frac{1}{n} \sum_{i=1}^{n} \left(y_i - \mathbf{w}^T \mathbf{x}_i \right) \mathbf{x}_i = \vec{0}$$

By rearranging the terms, we get a system of linear equations with $d+1$ unknowns.

$$w_0 \sum_{i=1}^{n} x_{i0} \cdot x_{ij} + \dots + w_1 \sum_{i=1}^{n} x_{i1} \cdot x_{ij} + \dots + w_d \sum_{i=1}^{n} x_{id} \cdot x_{ij} + \sum_{i=1}^{n} y_i \cdot x_{ij} = \sum_{i=1}^{n} y_i \cdot x_{ij}$$

Can also be solved through matrix inversion if the matrix is not singular.

$$\mathbf{Aw} = \mathbf{b} \Rightarrow \mathbf{w} = \mathbf{A}^{-1}\mathbf{b}$$

(5.5): Linear Regression as a System of Linear Equations The linear regression model is akin to a system of linear equations. Assuming n training examples with $d+1$ features each –

$$\begin{aligned} 1^{\text{st}} \text{ training example:} \quad & y_1 = w_0 + x_{11} w_1 + x_{12} w_2 + \dots + x_{1d} 2_d \\ 2^{\text{nd}} \text{ training example:} \quad & y_2 = w_0 + x_{21} w_1 + x_{22} w_2 + \dots + x_{2d} 2_d \\ \vdots \\ n^{\text{th}} \text{ training example:} \quad & y_n = w_0 + x_{n1} w_1 + x_{n2} w_2 + \dots + x_{nd} 2_n \end{aligned}$$

(5.6): Solving Linear Regression

(5.6.1): Using Matrices $J_n(\mathbf{w})$ can be rewritten in terms of data matrices X and vectors:

$$J_n(\mathbf{w}) = \frac{1}{2} (\mathbf{y} - \mathbf{Xw})^T (\mathbf{y} - \mathbf{Xw})$$

$$\nabla J_n(\mathbf{w}) = -\mathbf{X}^T (\mathbf{y} - \mathbf{Xw})$$

Set derivatives to 0 and solve to obtain $\mathbf{w}$.

$$\begin{aligned} J_n(\mathbf{w}) &= 0 \\ -\mathbf{X}^T (\mathbf{y} - \mathbf{Xw}) &= 0 \\ -\mathbf{X}^T \mathbf{y} + \mathbf{X}^T \mathbf{Xw} &= 0 \\ \mathbf{X}^T \mathbf{Xw} &= \mathbf{X}^T \mathbf{y} \\ \mathbf{w} &= \left(\mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{y} \end{aligned}$$

(5.6.2): Using Gradient Descent Linear regression problem comes down to the problem of solving a set of linear equations:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot \nabla_{\mathbf{w}} J_n(\mathbf{w})$$

$$\nabla J_n(\mathbf{w}) = -\mathbf{X}^T (\mathbf{y} - \mathbf{Xw})$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot \mathbf{X}^T (\mathbf{Xw} - \mathbf{y})$$

(5.7): Online Linear Regression The error function defined for the whole dataset for the linear regression is:

$$J_n = \frac{1}{2n} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i))^2$$

Online Gradient Descent: use the most recent sample at each iteration. Instead of MSE for all data points, it uses MSE for an individual sample.

$$\begin{aligned} J_{online} &= Error_i(\mathbf{w}) \\ &= \frac{1}{2} (y_i - f(\mathbf{x}_i))^2 \end{aligned}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot \nabla_{\mathbf{w}} Error_i(\mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot (f(\mathbf{x}_i) - y_i) \cdot \mathbf{x}_i$$

(5.8): Input Normalization Makes the data very roughly on the same scale. Can make a huge difference in online learning.

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot (f(\mathbf{x}_i) - y_i) \cdot \mathbf{x}_i$$

For inputs with a large magnitude, the change in the weight is huge. Solution: Make all inputs vary in the same range.

$$\begin{aligned} \bar{x}_j &= \frac{1}{n} \sum_{i=1}^{n} x_{ij} \\ \sigma_j^2 &= \frac{1}{n-1} \sum_{i=1}^{n} (x_{ij} - \bar{x}_j)^2 \end{aligned} \tag{5.2}$$

New output:

$$\hat{x_{ij}} = \frac{x_{ij} - \bar{x}_j}{\sigma_j}$$

(5.9): L1/L2 Regularization Using L1/L2 Regularization, we can rewrite our loss function as:

$$L_{lasso} = \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - f(\mathbf{w}^T \mathbf{x}_i) \right)^2 + \lambda ||\mathbf{w}||_1$$

$$L_{ridge} = \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - f(\mathbf{w}^T \mathbf{x}_i) \right)^2 + \lambda ||\mathbf{w}||_2^2$$

(5.10): Other Ways to Control Overfitting

Early-stopping : stopping training when a monitored metric has stopped improving.

Bagging : learning multiple models in parallel and applying majority voting to choose final predictor.

Dropout : in each iteration, don't update some of the weights.

Injecting noise in the inputs.

(5.11): Bias-Variance Tradeoff Bias captures the inherent error present in the model. The bias error originates from erroneous assumption(s) in the learning algorithms. Bias is the contrast between the mean prediction of our model and the correct prediction. Variance captures how much the model changes if it is trained on a different training set. Variance is the variation or spread of model prediction values across different data samples. Underfitting happens when a model is unable to capture the underlying pattern of the data. Such models usually have high bias and low variance. It usually happens when there is much fewer amount of data to build an accurate model or when a linear model is used to learn non-linear data. Overfitting happens when our model captures the noise along with the underlying pattern in data. It usually happens when the model is trained a lot over a noisy dataset. These models have low bias and high variance.
Bias:

$$(y - \bar{y}) \tag{5.3}$$

Variance:

$$\frac{1}{k-1} \sum_{j=1}^{k-1} (\hat{y}_j - \bar{y})^2 \tag{5.4}$$

Total Error:

$$TE = Bias^2 + Variance = (y - \bar{y})^2 + \frac{1}{k-1} \sum_{j=1}^{k-1} (\hat{y}_j - \bar{y})^2 \tag{5.5}$$

$$\text{Expected Loss} = \text{Total Error} = \text{Bias}^2 + \text{Variance}$$

(5.12): Fitting the Data R^2 is a metric to determine how well does the learned model fit the data, because simply having a low MSE does not guarantee that the model is not overfitting. R^2 captures the fraction of the total variance explained by the model. Let $\hat{y}_i$ be a predicted value, and $\bar{y}$ be the sample mean.

$$\begin{aligned} R^2 &= 1 - \frac{\text{Residual Variance}}{\text{Total Variance}} \\ &= 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y}_i)^2} \end{aligned} \tag{5.6}$$

(5.13): Alternative Loss Functions

Square loss Very commonly used for regression. Leads to an easy-to-solve optimization problem.

$$(y_n - f(\mathbf{x}_n))^2 \tag{5.7}$$

Absolute loss Grows more slowly than squared loss. Better suited when data has some outliers (inputs on which model makes large errors).

$$|y_n - f(\mathbf{x}_n)| \tag{5.8}$$

Huber loss Squared loss for small errors (up to δ); absolute loss for larger errors. Good for data with outliers.

ϵ-insensitive loss (Vapnik loss) Zero loss for small errors (say up to ϵ); absolute loss for larger errors.

$$|y_n - f(\mathbf{x}_n)| - \epsilon \tag{5.9}$$