Machine Learning in Finance and Insurance

Ramon Kaspar, ETH Zürich (Fall 2024)

1 Basic Notions of Statistical Learn.

Expected Loss (Expected Risk)

$$\mathbb{E}[\ell(f(X),Y)] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(f(x),y)\rho(dx,dy).$$

For a measurable *loss function* $\ell : \mathcal{Y}\times\mathcal{Y} \to \mathbb{R}^+$

Goal: Find f that minimizes $\mathbb{E}[\ell(f(X),Y)]$.

Empirical Loss (Empirical Risk)

With training data $(X_i,Y_i)_{i=1}^m$:

$$\mathbb{E}[\ell(f(X),Y)] \approx \frac{1}{m}\sum_{i=1}^m \ell(f(X_i),Y_i).$$

$\Rightarrow$ Empirical loss minimizer depends on hypothesis class $\mathcal{H}$ and loss function ℓ. For square loss and constant functions, the minimizer is $\bar{Y} = \frac{1}{m}\sum_{i=1}^m Y_i$. There's a *Bias-Variance Tradeoff*; the best balance depends on the unknown distribution of Y.

PRED. PROB.: Find meas. func. $f : \mathcal{X} \to \mathcal{Y}$, s.t. $f(X) \approx Y$.

HYPOTHESIS CLASS: A *hypothesis class* is a family $\mathcal{H}$ of measurable functions $f : \mathcal{X} \to \mathcal{Y}$, e.g. all affine functions.

MODEL ESTIMATION: Find a numerical solution $\hat{f}_m$ to:

$$\min_{f\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^m \ell(f(X_i),Y_i)$$

- **Deterministic algorithm:** $\hat{f}_m(\cdot) = \hat{\varphi}\left(\cdot,(X_i,Y_i)_{i=1}^m\right)$
- **Stochastic algorithm** (e.g., random initialization): $\hat{f}_m(\cdot) = \hat{\varphi}\left(\cdot,(X_i,Y_i)_{i=1}^m,V\right)$, where V indep. of (X,Y) & random.

Define the average model $\hat{f}_m^{\text{avg}}(x) := \mathbb{E}[\hat{f}_m(x)]$.

RELATIONSHIPS BETWEEN X AND Y:

- *Deterministic dependence:* $Y = g(X)$
- *Homoscedastic additive noise:* $Y = g(X)+\epsilon$, for ϵ independent of X with $\mathbb{E}[\epsilon] = 0$
- *Heteroscedastic additive noise:* $Y = g(X)+h(X)\epsilon$, for ϵ independent of X with $\mathbb{E}[\epsilon] = 0$
- *Non-additive noise:* $Y = g(X,\epsilon)$, for ϵ independent of X
- *Model-free:* the form of the underlying model is not known, only iid observations $(X_i,Y_i)_{i=1}^m$ are available

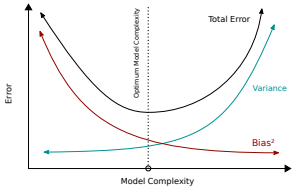
REGRESSION FUNCTION: $\bar{f}(x)$ minimizes $w \mapsto \int_{\mathcal{Y}} \ell(w,y)\rho_{Y|X}(dy|x)$ and is called the *regression function*. $\mathbb{E}[\ell(\bar{f}(X),Y)]$ is the *irreducible error*.
Square Loss $\bar{f}(X) = \mathbb{E}[Y|X]$, $\mathbb{E}[\ell(\bar{f}(X),Y)] = \text{Var}(Y) - \text{Var}(\mathbb{E}[Y|X])$
Pinball Loss $\ell_\alpha(u) = \left(\alpha - 1_{\{u<0\}}\right)u$ for $u \in \mathbb{R}$, $\bar{f}(X) = q_\alpha$ and $\mathbb{E}[\ell(\bar{f}(X),Y)] = \mathbb{E}\left[\ell_\alpha(Y - q_\alpha(X))\right]$

Bias-Variance Decomposition of Square Loss

$$E\left[(\hat{f}_m(X)-Y)^2\right] = \underbrace{\mathbb{E}\left[\left(\hat{f}_m^{\text{avg}}(X)-\bar{f}(X)\right)^2\right]}_{\text{Bias}^2} +$$

$$\underbrace{E\left[\left(\hat{f}_m(X)-\hat{f}_m^{\text{avg}}(X)\right)^2\right]}_{\text{Variance}} + \underbrace{E\left[\left(\bar{f}(X)-Y\right)^2\right]}_{\text{Irreducible Error}}.$$

Bias: Error from erroneous assumptions (underfitting).
Variance: Error from sensitivity to data fluctuations (overfitting).



Trade-off: Increase model complexity $\Rightarrow$ lower bias, higher var

R^2 (goodness of in-sample fit)

Let $(X_i,Y_i)_{i=1}^m$ be the training data, and $(X_i,Y_i)_{i=m+1}^{m+n}$ the test data. Let $\hat{f}_m : \mathcal{X} \to \mathcal{Y}$ be a prediction function trained on training data:

$$R^2 = 1 - \frac{SSR}{SST}$$

$\text{SSR} = \sum_{i=1}^m \left(Y_i - \hat{f}_m(X_i)\right)^2$, $\text{SST} = \sum_{i=1}^m \left(Y_i - \overline{Y}_{\text{train}}\right)^2$ for $\overline{Y}_{\text{train}} = \frac{1}{m}\sum_{i=1}^m Y_i$. *The higher the better.*

Similarly, define out-of-sample R^2_{os} using test data; it measures **prediction power**.

Approximation Error

$$\inf_{f\in\mathcal{H}} \mathbb{E}\,\ell(f(X),Y) - \mathbb{E}\,\ell(\bar{f}(X),Y) \geq 0$$

Approximation error occurs if $\mathcal{H}$ is not flexible enough to approximate $\bar{f}$ well. Is decreasing if $\mathcal{H}$ is increasing. Is 0 if $\bar{f} \in \mathcal{H}$. If the empirical loss min. problem $\min_{f\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^m \ell(f(X_i),Y_i)$ has a solution, we call it $\hat{f}_m^{\mathcal{H}} \in \mathcal{H}$.

Sampling Error

$$\mathbb{E}\left[\ell\left(\hat{f}_m^{\mathcal{H}}(X),Y\right)\middle|\hat{f}_m^{\mathcal{H}}\right] - \inf_{f\in\mathcal{H}} \mathbb{E}[\ell(f(X),Y)] =$$

$$\int_{\mathcal{X}\times\mathcal{Y}} \ell\left(\hat{f}_m^{\mathcal{H}}(x),y\right)\rho(dx,dy) - \inf_{f\in\mathcal{H}} \int_{\mathcal{X}\times\mathcal{Y}} \ell(f(x),y)\rho(dx,dy) \geq 0$$

Results from minimizing the empirical loss instead of the expected loss. Tends to decrease if the sample size m is increasing.

Direct Optimization Error

Let $\hat{f}_m(\cdot) = \hat{\varphi}(\cdot,(X_i,Y_i)_{i=1}^m,V)$:

$$\frac{1}{m}\sum_{i=1}^m \ell\left(\hat{f}_m(X_i),Y_i\right) - \inf_{f\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^m \ell\left(f(X_i),Y_i\right) \geq 0$$

For classical methods (e.g., lin. reg.), assume $\hat{f}_m \approx \hat{f}_m^{\mathcal{H}}$. For complex ML methods (boosted trees, NNs), typically $\hat{f}_m \not\approx \hat{f}_m^{\mathcal{H}}$.

Generalization Error

$$\mathbb{E}\left[\ell(\hat{f}_m(X),Y) \mid \hat{f}_m\right] = \mathbb{E}[\ell(\bar{f}(X),Y)] \quad \textit{(irreducible error)}$$
$$+ \inf_{f\in\mathcal{H}} \mathbb{E}[\ell(f(X),Y)] - \mathbb{E}[\ell(\bar{f}(X),Y)] \quad \textit{(approximation error)}$$
$$+ \mathbb{E}\left[\ell\left(\hat{f}_m^{\mathcal{H}}(X),Y\right) \mid \hat{f}_m^{\mathcal{H}}\right] - \inf_{f\in\mathcal{H}} \mathbb{E}[\ell(f(X),Y)] \quad \textit{(sampling error)}$$
$$+ \mathbb{E}\left[\ell\left(\hat{f}_m(X),Y\right) \mid \hat{f}_m\right] - \mathbb{E}\left[\ell\left(\hat{f}_m^{\mathcal{H}}(X),Y\right) \mid \hat{f}_m^{\mathcal{H}}\right] \textit{(ind. opt. error)}$$

For fixed $m \in \mathbb{N}$ and increasing $\mathcal{H}$: approx. error decreases, sampling error increases (variance of $\hat{f}_m^{\mathcal{H}}$ grows). There's a tradeoff between them (like bias-variance). For small data, $\mathcal{H}$ should be simple; for large data, it can be complex.

Training Error and Test Error

$$E_{\text{tr}} = \frac{1}{m}\sum_{i=1}^m \ell\left(\hat{f}_m(X_i),Y_i\right), \quad E_{\text{te}} = \frac{1}{m}\sum_{i=m+1}^{m+n} \ell\left(\hat{f}_m(X_i),Y_i\right)$$

If $E_{\text{tr}} \ll E_{\text{te}}$, the data most likely was overfitted.
Sample Variance: $\sigma^2_{\text{te}} = \frac{1}{n-1}\sum_{i=m+1}^{m+n}\left(\ell\left(\hat{f}_m(X_i),Y_i\right) - E_{\text{te}}\right)^2$

2 Linear Regression

Linear regression models the relationship between predictors and response: $y = \beta_0 + \beta_1 X_1 + \cdots + \beta_d X_d + \epsilon, \quad \epsilon \sim N(0, \sigma^2)$
In matrix form:

$$y = A\beta + \epsilon$$

where A is the design matrix, β the coefficient vector.

Normal Equation

If $A^T A$ regular (or equivalently columns of A lin. indep.):

$$\hat{\beta} = \min_{b \in \mathbb{R}^{d+1}} \|Ab - y\|_2^2 = (A^\top A)^{-1} A^\top y$$

$\Rightarrow \hat{\beta} = (A^T A)^{-1} A^T (A\beta + \epsilon) \sim \mathcal{N}_{d+1}\left(\beta, \sigma^2 (A^T A)^{-1}\right)$
If columns of A are linearly dependent, $A^\top A$ is singular, and the normal equation has infinitely many solutions. The **pseudoinverse** solution $\hat{\beta} = (A^\top A)^\dagger A^\top y$ minimizes $\|b\|_2$. Here, $(A^\top A)^\dagger = V \Lambda^\dagger V^\top$, with $\Lambda^\dagger$ diagonal entries $1_{\{\lambda_j > 0\}} \lambda_j^{-1}$.

SINGULAR VALUE DECOMPOSITION (SVD)

Matrix $A \in \mathbb{R}^{m \times l}$, rank $r \leq \min(m, l)$, can be decomposed as:

$$A = U \Sigma V^\top$$

- $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{l \times l}$ orthogonal, $\Sigma^\dagger \in \mathbb{R}^{l \times m}$ diagonal
- $\lambda_1 \geq ... \geq \lambda_r$ are the positive eigenvalues of $A^\top A$ and $\Sigma = \text{diag}(\sqrt{\lambda_1}, ..., \sqrt{\lambda_r}, 0, ...) \in \mathbb{R}^{m \times l}$
- **Pseudoinverse** $A^\dagger = V \Sigma^\dagger U^\top$; for any $y \in \mathbb{R}^m, \hat{\beta} = A^\dagger y \in \mathbb{R}^n$ minimizes $b \mapsto \|Ab - y\|_2$ with minimal $\|\cdot\|_2$-norm.
- **Regularization:** Truncate small σ_i (set $\sigma_i = 0$ if $\sigma_i < c$); then $\hat{\beta}_c = A_c^\dagger y \sim \mathcal{N}_l\left(Q_k \beta, \sigma^2 V \Lambda_c^{-1} V^T\right)$ balances bias and variance: increasing c increases bias and decreases variance.

Ridge Regression

$$\hat{\beta}_\lambda = \min_{b \in \mathbb{R}^{d+1}} \left(\|Ab - y\|_2^2 + \lambda \|b\|_2^2\right) = (A^\top A + \lambda I_l)^{-1} A^\top Y$$

$\hat{\beta}_\lambda = \sum_{i=1}^r \frac{\sigma_i}{\sigma_i^2 + \lambda} v_i u_i^T y \sim \mathcal{N}_l\left(\sum_{i=1}^r \frac{\lambda_i}{\lambda_i + \lambda} v_i v_i^T \beta, \sigma^2 \sum_{i=1}^r \frac{\lambda_i}{(\lambda_i + \lambda)^2} v_i v_i^T\right)$
Increasing λ increases bias and decreases variance.

LASSO Regression

$$\hat{\beta}_\lambda = \min_{b \in \mathbb{R}^{d+1}} \left(\|Ab - y\|_2^2 + \lambda \|\beta\|_1\right)$$

$\Rightarrow$ No closed-form solution, LASSO-Regr. encourages **sparsity**.

CROSS-VALIDATION

Technique to estimate model's predictive performance and tune hyperparameters.

Divide data into K folds:

- Train on $K - 1$ folds.
- Validate on the remaining fold.
- Repeat K times; average validation error.

STANDARDIZED LINEAR REGRESSION

$$\tilde{x}_i = \frac{x_i - \bar{x}}{s_x}, \quad \tilde{y}_i = \frac{y_i - \bar{y}}{s_y}$$

with mean $\bar{x}_j$ and $s_j^2 = \frac{1}{m} \sum_{i=1}^m (x_{ij} - \bar{x}_j)^2$. The prediction $\hat{y}$ for a new datapoint $x = (x_1, ..., x_d)$ is $\hat{y} = s_y \sum_{j=1}^d \frac{x_j - \bar{x}_j}{s_j} \hat{\beta}_j + \bar{y}$

3 Gradient Descent

A function $h : \mathbb{R}^d \to \mathbb{R}$ is said to be ***convex*** if $h(\lambda x + (1-\lambda) y) \leq \lambda h(x) + (1 - \lambda) h(y)$, for all $x, y \in \mathbb{R}^d$ and $\lambda \in (0, 1)$.
Setup: Given a convex function $h : \mathbb{R}^d \to \mathbb{R}$ with minimizer x^*, start at x_0 (might be random) and update with gradient steps:

$$x_{k+1} = x_k - \eta_k \nabla h(x_k)$$

If $\|\nabla h(x_k)\|_2 \leq L$ for all k, then:

$$\min_{0 \leq k \leq K} h(x_k) - h(x^*) \leq \frac{\|x_0 - x^*\|_2^2 + L^2 \sum_{k=0}^K \eta_k^2}{2 \sum_{k=0}^K \eta_k}$$

If $\sum_{k=0}^\infty \eta_k^2 < \infty$ and $\sum_{k=0}^\infty \eta_k = \infty$, then $h(x_k) \to h(x^*)$.
For accuracy ε, one needs $K = \left\lceil \left(\frac{L \|x_0 - x^*\|_2}{\varepsilon} \right)^2 \right\rceil$ gradient steps (which does not depend on the dimension!).

Stochastic Gradient Descent (SGD)

Consider $H : \Omega \times \mathbb{R}^d \to \mathbb{R}$, convex in θ, measurable in ω, with $\mathbb{E}[|H(\theta)|] < \infty$ for all $\theta \in \mathbb{R}^d$. Let H_i be independent copies of H, and $I \in \mathbb{N}$. Define the gradient estimator:

$$g_k(\theta) = \frac{1}{I} \sum_{i=kI+1}^{(k+1)I} \nabla H_i(\theta) \approx \nabla h(\theta), \quad k \geq 0.$$

Start with $\theta_0 \in \mathbb{R}^d$ (random) and perform updates:

$$\theta_{k+1} = \theta_k - \eta_k \, g_k(\theta_k), \quad k \geq 0.$$

For $I = 1$: SGD; for $I > 1$: SGD with mini-batches.

Let $\widetilde{H}_i$, $i = 1, \ldots, v$, be independent copies of H, independent of H_i (*validation set*). Monitor the *empirical loss* $\frac{1}{v} \sum_{i=1}^v \widetilde{H}_i(\theta_k)$ for θ_k, $k = 0, 1, \ldots$. If the validation loss stops decreasing, decrease the learning rate η_k; if it increases, stop the SGD.

4 Logistic Regression

Logistic Regression (Binary Classification)

Let $Y \mid X \sim \text{Ber}(p(X))$, where $X = (X_1, \ldots, X_d), p(X) = \psi\left(\beta_0 + \beta_1 X_1 + \cdots + \beta_d X_d\right)$ and $\psi(x) = \frac{e^x}{e^x + 1} = \frac{1}{1 + e^{-x}}$.
The empirical loss function, derived from conditional negative log-likelihood, corresponds to the *cross-entropy loss*:

$$\hat{b} = \min_{b \in \mathbb{R}^{d+1}} \sum_{i=1}^m \left\{ -y_i \log(\psi(x_i^T b)) - (1 - y_i) \log(1 - \psi(x_i^T b)) \right\}$$

$$= \min_{b \in \mathbb{R}^{d+1}} \sum_{i=1}^m \left\{ \log(1 + e^{x_i^T b}) - y_i x_i^T b \underbrace{+\lambda \|b\|_2}_{\text{Regularization}} \right\}$$

$\Rightarrow$ *Convex* min. problem in $b \in \mathbb{R}^{d+1}$, can be solved with (S)GD.

Having obtained $\hat{\beta} = (\hat{\beta}_0, \ldots, \hat{\beta}_d)$ from the empirical loss minimization, we predict $Y = 1$ for a new data point $X = (x_1, \ldots, x_d)$ by computing: $\hat{p}(x) = \psi\left(\hat{\beta}_0 + \sum_{j=1}^d x_j \hat{\beta}_j\right)$. Using a decision threshold $c \in (0, 1)$, we predict: $\hat{y}(x) = \begin{cases} 1, & \text{if } \hat{p}(x) \geq c, \\ 0, & \text{if } \hat{p}(x) < c. \end{cases}$

PERFORMANCE METRICS (DIAGNOSTICS)

$$\text{TPR (Recall)} = \frac{TP}{P} \qquad \text{FPR} = \frac{FP}{N} \qquad \text{FDR} = \frac{FP}{TP + FP}$$

$$\text{Accuracy} = \frac{TP + TN}{P + N} \qquad \text{Precision (PPV)} = \frac{TP}{TP + FP}$$

$$\text{F1 Score} = 2 \frac{\text{TPR} \times \text{PPV}}{\text{TPR} + \text{PPV}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}} = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}$$

ROC: plots TPR vs. FPR for $c \in [0, 1]$. Random guessing produces the diagonal with $AUC = 1/2$. **AUROC** (Area under ROC): the larger the better (1 indicates a perfect classifier).

CREDIT ANALYTICS

Let P be a good borrower and N a bad borrower.
Try to *minimize* FDR $= FP/(FP + TP)$ (FP's result in losses),
Try to *maximize* TPR $= TP/P$ (increases business volume).
$\Rightarrow$ Try to obtain a flat FDR/TPR-curve, i.e., a small area under the plotted FDR/TPR-curve is desirable.

5 Support Vector Machine (SVM)

Derivation (Hard-margin SVM): Let $(x_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}$ for $i = 1, \dots, m$, with linear classification, i.e., there exist $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ such that $\text{sign}(\langle w, x_i \rangle + b) = y_i$ for all i. Each classifier (w, b) defines the decision hyperplane: $H(w, b) = \{x \in \mathbb{R}^d : \langle w, x \rangle + b = 0\}$. The distance from x_i to $H(w, b)$ is: $d_i = \frac{|\langle w, x_i \rangle + b|}{\|w\|}$, and the margin is the minimal distance: $\gamma = \min_i d_i$. We aim to maximize the margin: $\max_{w,b} \gamma$ subject to: $y_i(\langle w, x_i \rangle + b) \geq \gamma \|w\| \quad \forall i$. By scaling w and b such that $\gamma \|w\| = 1$, the constraints simplify to: $y_i(\langle w, x_i \rangle + b) \geq 1, \forall i$. Maximizing γ is equivalent to minimizing $\|w\|$, yielding:

Hard-margin SVM

$$\min_{w,b} \|w\|^2 \quad \text{s.t. } y_i(\langle w, x_i \rangle + b) \geq 1 \quad \forall i \in \{1, \dots, m\}.$$

Problem: Linear separation may be infeasible or lead to overfitting. Solutions using *RKHS learning*:

- Lift x_i into a *Hilbert space* H_0 via a feature map $\Phi : \mathbb{R}^d \to H_0$. The classifier becomes $w \in H_0$.
- Introduce *slack variables* $\xi_i \geq 0$ to allow constraint violations with a penalty in the objective.

Soft-margin SVM (Kernelized)

$$\begin{aligned}
\hat{w}_{SVM} &= \min_{w,b,\xi} \|w\|_2^2 + C \sum_{i=1}^{m} \xi_i \quad \text{s.t. } y_i \left(\langle w, \Phi(x_i) \rangle + b\right) \geq 1 - \xi_i \\
&= \min_{(w,b) \in H_0 \times \mathbb{R}} \|w\|_2^2 + \lambda \sum_{i=1}^{m} \underbrace{\max\left(0, 1 - y_i(\langle w, \Phi(x_i) \rangle + b)\right)}_{\text{Hinge loss}} \\
&= \min_{f \in H} \frac{1}{m} \sum_{i=1}^{m} \ell_{\text{Hinge}}(y_i, f(x_i)) + \lambda \|f\|_H^2
\end{aligned}$$

where $H = \{f : \mathbb{R}^d \to \mathbb{R} : f = \langle w, \Phi(\cdot) \rangle\}$ is a Hilbert Space

Support Vector Regression (SVR): Find $(w, b) \in H_0 \times \mathbb{R}$ to approximate $y \in \mathbb{R}$. Goal: Fit data within an ϵ-tube around $f(x) = \langle w, \Phi(x) \rangle + b$. The optimization problem becomes: $\min_{(w,b,\xi)} \|w\|^2 + C \sum_{i=1}^m \xi_i$ s.t. $|y_i - \langle w, \Phi(x_i) \rangle - b| \leq \epsilon + \xi_i$ with $\xi_i \geq 0$. Or similarly: $\min_{f \in H} \lambda \|f\|_H^2 + \frac{1}{m} \sum_{i=1}^m \ell_\epsilon(y_i, f(x_i) + b)$ where ϵ-insensitive loss $\ell_\epsilon(y, y') = \max(0, |y - y'| - \epsilon)$.

6 Kernels & Hilbert Spaces

A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a **kernel** on $\mathcal{X}$ if there exists a Hilbert space H and a map $\Phi : \mathcal{X} \to H$ such that for all $x, x' \in \mathcal{X}$ we have $k(x, x') = \langle \Phi(x), \Phi(x') \rangle_H$.

A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a **kernel** if and only if it is **symmetric** and **pos. semidefinite** ($\sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j k(x_j, x_i) \geq 0$).

- Linear kernel: $k(x, x') = \langle x, x' \rangle$
- Polynomial kernel: $p \in \mathbb{N}$ and $c \in \mathbb{R}_+$, $k(x, x') = (\langle x, x' \rangle + c)^p$
- Gaussian kernel: for $\gamma > 0$, $k(x, x') = \exp\left(-\frac{\|x - x'\|_2^2}{\gamma^2}\right)$
- Compactly supported radial basis kernel: for $p > d/2$, $k(x, x') = \max\{1 - \|x - x'\|_2, 0\}^p$
- Radial basis function kernels: $k(x, x') = \phi(\|x - x'\|_2)$ for $\phi : \mathbb{R}_+ \to \mathbb{R}$. The factor γ, s.t. $k_\gamma(x, x') = \phi(\|x - x'\|_2/\gamma)$ is the *bandwidth* (the higher, the smoother the function).

Decomposition Rules: $k(\mathbf{x}, \mathbf{y}) = k_1(\mathbf{x}, \mathbf{y}) + k_2(\mathbf{x}, \mathbf{y})$, $k(\mathbf{x}, \mathbf{y}) = k_1(\mathbf{x}, \mathbf{y}) k_2(\mathbf{x}, \mathbf{y})$, $k(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}) f(\mathbf{y})$, $k(\mathbf{x}, \mathbf{y}) = ck(\mathbf{x}, \mathbf{y})$ for $c > 0$, $k(\mathbf{x}, \mathbf{y}) = k(\phi(\mathbf{x}), \phi(\mathbf{y}))$.

REPRODUCING KERNEL HILBERT SPACES

We call H a **reproducing kernel Hilbert space (RKHS)**, if $\delta_x : H \to \mathbb{R}$ given by $\delta_x(f) := f(x)$ is continuous for all $x \in \mathcal{X}$. A kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a **reproducing kernel of** H, if $k(\cdot, x) \in H$ and $f(x) = \langle f, k(\cdot, x) \rangle$ for all $x \in \mathcal{X}, f \in H$. If such a kernel k exists, we call $\Phi : \mathcal{X} \to H$ given by $\Phi(x) := k(\cdot, x)$ the canonical feature map.

Theorem: Every RKHS has a unique reproducing kernel, and conversely, every positive definite kernel k corresponds to a unique RKHS H.

Mercer's Theorem

If k is a continuous, symmetric, positive definite kernel on a compact set $\mathcal{X}$, then: $k(x, x') = \sum_{i=1}^\infty \lambda_i \phi_i(x) \phi_i(x')$, where $\lambda_i \geq 0$ are eigenvalues and $\{\phi_i\}$ are orthonormal in $L^2(\mathcal{X})$. Furthermore, the RKHS H associated with k is: $H = \left\{ f = \sum_{i=1}^\infty a_i \phi_i : \sum_{i=1}^\infty \frac{a_i^2}{\lambda_i} < \infty \right\}$, with inner product $\langle f, g \rangle_H = \sum_{i=1}^\infty \frac{a_i b_i}{\lambda_i}$, for $f = \sum a_i \phi_i$, $g = \sum b_i \phi_i$.

Representer Theorem

Any minimizer $f \in H$ of the regularized empirical risk $\min_{f \in H} \lambda \|f\|_H^2 + \sum_{i=1}^m \ell(y_i, f(x_i))$ admits a representation of the form $f^*(x) = \sum_{i=1}^m \alpha_i k(x_i, x)$ for some coefficients $\alpha_i \in \mathbb{R}$.

Implications: Mercer's Theorem allows kernels to implicitly map data into high-dimensional spaces. Representer Theorem ensures solutions are finite sums over training data using kernels (*kernel trick*).

Corollary: Let $\ell(y, y') = (y - y')^2$. Assume $x_1, \dots, x_m$ are distinct and k is strictly positive definite. Then, the parameters $a = (\alpha_1, \alpha_2, \dots, \alpha_m)$ of the optimizer $f_m^* = \sum_{i=1}^m \alpha_i k(x_i, \cdot)$ from *Representer Theorem* are given by

$$a = (\lambda m I_m + K_m)^{-1} b,$$

where $I_m \in \mathbb{R}^{m \times m}$ is the identity matrix, $K_m \in \mathbb{R}^{m \times m}$ is given by $K_m[i, j] = k(x_i, x_j)$, and $b = (y_1, \dots, y_m)$.

NUMERICAL APPROACHES

Regression: $\hat{w} = \mathbf{\Phi}^\top \hat{\alpha}$ (`sklearn.kernel_ridge.KernelRidge`).
Binary Classification: Use SMO Algorithm. For *Hinge loss* use `sklearn.svm.SVC`, for ϵ-sens. loss use `sklearn.svm.SVR`.

Analytic solution not feasible for large datasets. Solutions:

- **Feature selection**: select a subset $I \subset \{1, ..., m\}, |I| = k < m$ and build an estimator of the form $\hat{f}(x) = \sum_{i \in I} \alpha_i k(x_i, x)$. E.g. Nyström (`sklearn.kernel_approximation.Nystroem`)
- **Preconditioning:** Approximate A^{-1} via $BB^\top \approx A^{-1}$ to solve $Ax = b$ efficiently (e.g., FALKON).

FUNCTION APPROXIMATION WITH RKHS

Kernel k is **universal**, if for every continuous function $f : X \to R$ and $\epsilon > 0$, there exists $h \in H$ such that: $\|h - f\|_{\text{inf}} \leq \epsilon$.
Examples: Exponential kernel: $k(x, x') = \exp(\gamma \langle x, x' \rangle)$ for $\gamma > 0$, Gaussian kernel: $k(x, x') = \exp\left(-\gamma \|x - x'\|_2^2\right)$ for $\gamma > 0$, Binomial kernel: $k(x, x') = (1 - \langle x, x' \rangle)^{-\alpha}$ for $\alpha > 0$, $\mathcal{X} \subseteq \{x \in \mathbb{R}^d : \|x\|_2 < 1\}$.
Kernel Rate: For estimators $\hat{f}_m$ in RKHS with smoothness s, the estimation error decreases at rate $m^{-\frac{s}{2s+d}}$, i.e.,
$\lim_{m \to \infty} \mathbb{P}\left(\|\hat{f}_m^* - \bar{f}\|_2 \geq C m^{-\frac{s}{2s+d}}\right) = 0$

Minimax Rate: This rate $m^{-\frac{s}{2s+d}}$ is the optimal rate achievable by any estimator over functions with smoothness s, i.e.,
$\lim_{m \to \infty} \inf_{\hat{T}_m} \sup_{\theta \in \Theta} \mathbb{P}\left(\|\hat{T}_m - \bar{f}_\theta\|_2 \geq c m^{-\frac{s}{2s+d}}\right) = 1$.
$\Rightarrow$ Kernel methods achieve optimal convergence rates for estimating $\bar{f}$, making them effective for high-dimensional nonparametric regression. However, neural networks can be seen as kernel methods where the feature map is learned from data and thus better than any a-priori fixed kernel.

7 Neural Networks

Definition Feedforward Neural Network (FNN)

A **FNN** is a function $F_\theta : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ defined as:

$$F_\theta = F^{(L)} \circ \rho \odot F^{(L-1)} \circ \cdots \circ \rho \odot F^{(1)}$$

- each $F^{(k)} : \mathbb{R}^{N_{k-1}} \to \mathbb{R}^{N_k}$ an affine function: $F^{(k)}(x) = W^{(k)} \cdot x + b^{(k)}$, where $W^{(k)} \in \mathbb{R}^{N_k \times N_{k-1}}$ are weights and $b^{(k)} \in \mathbb{R}^{N_k}$ biases,
- N_k the number of neurons in the k-th layer and $(N_0, \ldots, N_L) \in \mathbb{N}^{L+1}$ is the network's architecture,
- $\theta = ((W^{(k)}, b^{(k)}), k = 1, \ldots, L)$ are network parameters,
- parameter space is $\mathbb{R}^{P(N_0,\ldots,N_L)} \ni \theta$, with $P := P(N_0, \ldots, N_L) = \sum_{k=1}^{L} N_k N_{k-1} + N_k$,
- $\rho : \mathbb{R} \to \mathbb{R}$ is the non-linear activation function applied to vectors element-wise.

- $k = 0$ is the *input layer*, $k = L$ is the *output layer*, $k \in \{1, \ldots, L-1\}$ are the *hidden layers*. $L+1$ is the *number of layers* and L is the *depth*,
- $\|N\|_\infty = \max_{0 \le k \le L} N_k$ is the *width* of the network.

Algorithm 1 Forward propagation

Require: Params $\theta = \left((W^{(k)}, b^{(k)}), k = 1, \ldots, L\right)$; datapoint (x, y)
Ensure: Loss value $\ell(F_\theta(x), y)$
1: $a^{(0)} := x$
2: **for** $k = 1, \ldots, L-1$ **do**
3: $\quad \tilde{a}^{(k)} := W^{(k)} a^{(k-1)} + b^{(k)}$
4: $\quad a^{(k)} := \rho(\tilde{a}^{(k)})$
5: **end for**
6: $\hat{y} := W^{(L)} a^{(L-1)} + b^{(L)}$
7: **return** $\ell(\hat{y}, y)$

Mini-batch Stochastic Gradient Descent (SGD)

$$\theta_0 \sim \mathbb{P}_{\text{initialization}}, \quad \theta_{t+1} = \theta_t - \eta_t \nabla_\theta \mathcal{L}(\theta, D_t)\big|_{\theta=\theta_t}$$

where D_t is a mini-batch of size B. For $B = 1$, this reduces to standard SGD.

- Non-convex loss ⇒ SGD result depends on initialization.
- Identical initial weights (e.g., $\theta_0 = c$) lead to identical gradients ⇒ Initialize weights differently.

***Xavier Initialization*:** (often used in practice)

$$W_{j,l}^{(k)} \sim \mathcal{N}\left(0, \frac{1}{N_k}\right), \quad b^{(k)} = 0,$$

where N_k is the width of the k-th layer.

Algorithm 2 Back-propagation $\mathcal{O}(L)$ time & memory

Require: Params $\theta = \left((W^{(k)}, b^{(k)}), k = 1, \ldots, L\right)$; datapoint (x, y)
Ensure: Gradient $\nabla_\theta \ell(F_\theta(x), y)$
1: Compute $\ell(F_\theta(x), y)$ using forward propagation
2: $\texttt{grad} \leftarrow \nabla_{\hat{y}} \ell(\hat{y}, y)$
3: $\nabla_{W^{(L)}} \ell(\hat{y}, y) = \texttt{grad} \cdot {a^{(L-1)}}^T$
4: $\nabla_{b^{(L)}} \ell(\hat{y}, y) = \texttt{grad}$
5: **for** $k = L-1, \ldots, 1$ **do**
6: $\quad \texttt{grad} \leftarrow \nabla_{\tilde{a}^{(k)}} \ell(\hat{y}, y) = \texttt{grad}^T \cdot W^{(k+1)} \cdot \operatorname{diag}\left(\rho'(\tilde{a}^{(k)})\right)$
7: $\quad \nabla_{W^{(k)}} \ell(\hat{y}, y) = \texttt{grad} \cdot {a^{(k-1)}}^T$
8: $\quad \nabla_{b^{(k)}} \ell(\hat{y}, y) = \texttt{grad}$
9: **end for**
10: **return** $\left((\nabla_{W^{(k)}} \ell(\hat{y}, y), \nabla_{b^{(k)}} \ell(\hat{y}, y)) \text{ for } k = 1, \ldots, L\right)$

BATCH NORMALIZATION

Problem: Parameter updates cause *internal covariate shift*, slowing training.
Solution: Normalize activations in each layer using batch mean and variance:

$$a_j^{(k)}(x_i) \leftarrow \frac{a_j^{(k)}(x_i) - \mu_j^{(k)}}{\sqrt{\sigma_j^{(k)2} + \epsilon}},$$

$$\mu_j^{(k)} = \frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} a_j^{(k)}(x_i), \quad \sigma_j^{(k)2} = \frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} \left(a_j^{(k)}(x_i) - \mu_j^{(k)}\right)^2$$

$\epsilon \approx 10^{-5}$ prevents division by zero, Gradients are backpropagated through normalization.

REGULARIZATION:

Loss functions are typically regularized, i.e. minimize objective:

$$\tilde{\mathcal{L}}(\theta, D) = \mathcal{L}(\theta, D) + \lambda \mathcal{R}(\theta),$$

Ridge: $\mathcal{R}(\theta) = \|\theta\|_2^2$ (shrinks θ, prevents overfitting)
Lasso: $\mathcal{R}(\theta) = \|\theta\|_1$ (leads to sparse θ)
Notes: Only weights W are regularized (not biases), and different λ can be used per layer.
Training Techniques as Regularizers:

- **Early Stopping:** Stop training if no improvement on validation set for p steps (patience). Limits parameter space (similar to L^2-regularization).
- **Dropout:** Set each neuron to zero with probability p before each gradient step; encourages sparse, redundant represent..
- **Other Techniques:** Bagging, dataset augmentation, weight robustness, parameter sharing.

NEURAL TANGENT KERNEL (NTK)

Generalization Puzzle: Overparametrized NNs generalize well despite classical bias-variance trade-off.
Goal: Study the dynamics of a NN $F_\theta : \mathbb{R}^{N_0} \to \mathbb{R}$ during GD.

For a NN $F_\theta : \mathbb{R}^{N_0} \to \mathbb{R}$, the **NTK** is defined as:

$$K(x, x'; \theta_t) = D_{\theta_t} F_\theta(x) \cdot D_\theta F_{\theta_t}(x')^\top \in \mathbb{R}.$$

Loss: $\mathcal{L}(\theta_t, D) = \frac{1}{m} \sum_{i=1}^{m} \ell(F_{\theta_t}(x_i), y_i)$.
Gradient flow: $\frac{d\theta_t}{dt} = -D_\theta \mathcal{L}(\theta_t, D)^\top$.
Using chain rule: $\frac{d}{dt}\mathcal{L}(\theta_t, D) = -\frac{1}{m^2} \sum_{i,j=1}^{m} D_{\hat{y}} \ell(F_{\theta_t}(x_i), y_i) \cdot K(x_i, x_j; \theta_t) \cdot D_{\hat{y}} \ell(F_{\theta_t}(x_j), y_j)^\top = -\left\|D_{\hat{y}} \ell(F_{\theta_t}(\cdot), y)\right\|^2_{K(\cdot,\cdot,\theta_t), D}$

CONVERGENCE RESULTS [Jacot et. al, 2018]

Theorem (NTK at Initialization): For shallow NN $(N_0, N_1, 1)$ with $\theta_0 \sim \mathcal{N}(0, I)$, as $N_1 \to \infty$: $F_{\theta_0} \to \text{GP}(0, C)$ where $C(x, x') = \mathbb{E}_{f \sim \text{GP}(0,\Sigma)}[\rho(f(x))\rho(f(x'))] + \beta^2$ with $\Sigma(x, x') = \frac{x^\top x'}{N_0} + \beta^2$
Theorem (NTK During Training): For any $T > 0$ with bounded $\int_0^T \frac{1}{m} \sum_i \|D_{\hat{y}} \ell(F_{\theta_t}(x_i), y_i)\|_2^2 dt$: $\sup_{t \in [0,T]} \|K(x, x'; \theta_t) - \tilde{K}(x, x')\| \xrightarrow{P} 0$ as width $\to \infty$.

MINIMUM-NORM SOLUTION

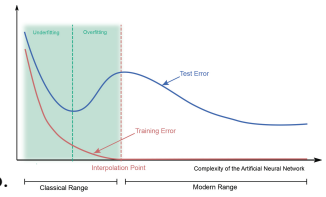
For squared loss $\ell(\hat{y}, y) = (\hat{y} - y)^2$, as $t \to \infty$: $F_{\theta_\infty}(x) = F_{\theta_0}(x) + \sum_{i=1}^{m} \sum_{j=1}^{m} \tilde{K}^{(L)}(x, x_i) \bar{K}_{i,j}^{-1} (y_j - F_{\theta_0}(x_j))$ where $\bar{K}_{i,j} := \tilde{K}^{(L)}(x_i, x_j)$ (limiting NTK evaluated on dataset), First term is initial network output and Second term is correction towards minimum-norm interpolator in RKHS.
Key insight: Solution converges to minimum-norm interpolator in NTK RKHS: $\arg\min_{f \in \mathcal{H}, f(x_i) = y_i} \|f\|_{\mathcal{H}}$.

GENERALIZATION IN OVERPARAMETRIZED REGIME

Linear Regression Setting: Model: $Y = X^\top \beta + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2)$, where $X = (X_1, ..., X_p) \in \mathbb{R}^p$ with covariance Σ. $\to \hat{\beta} = (A^\top A)^\dagger A^\top Y$.
Bias-Variance Decomposition (conditioned on A):

$$\mathbb{E}[(\hat{f}_D(X) - f^*(X))^2 | A] = \underbrace{\beta^\top \Pi \Sigma \Pi \beta}_{B_A = \text{bias}^2} + \underbrace{\frac{\sigma^2}{m} \text{Tr}(\hat{\Sigma}^\dagger \Sigma)}_{V_A = \text{variance}}$$

where $\hat{\Sigma} = \frac{1}{m} A^\top A$, $\Pi = I_p - \hat{\Sigma}^\dagger \hat{\Sigma}$
Theorem (Asymptotic Variance): For $p, m \to \infty$ with $p/m \to \gamma$: $V_A \to \sigma^2 \frac{1 - \max 1-\gamma, 0}{|1-\gamma|}$. *[Hastie et al., 2022]*
Double Descent Phenomenon: γ quantifies model complexity ($\gamma \gg 1 \Rightarrow p \gg m \Rightarrow$ Complex model). Variance increase for $\gamma \in (0, 1)$ (classical bias-variance trade-off), but decreases for $\gamma > 1$ (overparametrized).
⇒ Variance decreases as soon as model complexity passes the interpolation threshold. The test error can decrease even after the train error has reached zero.



8 Convolutional Neural Networks

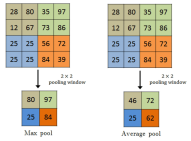
Three components: *convolution-*, *detector-* and *pooling layer*.
Convolutional layer: Let $I \in \mathbb{R}^{n_1 \times n_2}$, $K \in \mathbb{R}^{m_1 \times m_2}$, $m_1 \leq n_1$, $m_2 \leq n_2$, then their **cross-correlation** $(I \circledast K)$ is:

$$(I \circledast K)_{i,j} := \sum_{m=1}^{m_1} \sum_{n=1}^{m_2} I_{m+i-1,n+j-1} K_{m,n}.$$

- **Kernels:** Square $m \times m$ matrices (m hyperparameter); convolution layer uses $M > 1$ kernels in parallel to produce multiple feature maps.
- **Padding (p):** 0's added around input to control output size.
- **Stride (s):** steps kernel moves over input matrix.

Output size formula: $\left(\frac{n_1+2p-m}{s}+1, \frac{n_2+2p-m}{s}+1, M\right)$

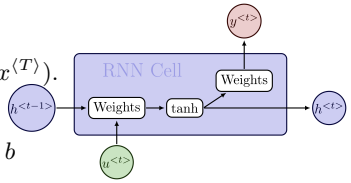
Detector layer: Applies non-linear activation ρ componentwise ($I_{i,j,k} \mapsto \rho(I_{i,j,k})$); output shape same as input; no trainable parameters.
Pooling Layer: Fixed filter (e.g., $m = 2$, $s = 2$, $p = 0$); operates per channel; no trainable parameters.



Convolution ≡ Linear transformation via sparse, parameter-shared weights (doubly block-circulant matrix).
- **Parameter sharing:** Same kernel used at every position ⇒ shared learning, translation equivariance.
- **Sparsity:** Small kernels reduce parameters ⇒ computational efficiency; layers extract local (edges) to global (digits) features.

9 Recurrent Neural Networks

Exploit data priors of sequential data, $(x^{\langle 1 \rangle}, x^{\langle 2 \rangle}, \ldots, x^{\langle T \rangle})$.



$$\begin{cases} a^{\langle t \rangle} = W \cdot h^{\langle t-1 \rangle} + U \cdot x^{\langle t \rangle} + b \\ h^{\langle t \rangle} = \rho(a^{\langle t \rangle}) \\ y^{\langle t \rangle} = V \cdot h^{\langle t \rangle} + c \end{cases}$$

Hidden states of RNN form a dynamical system with stationary transition functions: $h^{\langle t \rangle} = f_\theta\left(h^{\langle t-1 \rangle}, x^{\langle t \rangle}\right)$. The parameter $\theta = (W, U, b)$ are used at each time step t (**parameter sharing**). Hidden state $h^{\langle t \rangle}$ summarizes statistics of past sequence.

Back-Propagation Through Time (BPTT): Consider Loss $\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}^{\langle t \rangle} = \sum_{t=1}^{T} \ell(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle}), \quad t = 1, \ldots, T$

Compute recursively the gradients of $\mathcal{L}$: $D_{\hat{y}^{\langle t \rangle}} \mathcal{L} = D_{\hat{y}^{\langle t \rangle}} \ell(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle})$ and $D_{h^{\langle t \rangle}} \mathcal{L} = D_{\hat{y}^{\langle t \rangle}} \mathcal{L} \cdot V + D_{h^{\langle t+1 \rangle}} \mathcal{L} \cdot \operatorname{diag}\left(\rho'\left(a^{\langle t+1 \rangle}\right)\right) \cdot W$
Vanishing/Exploding Gradient Problem: Gradients increase/decrease exponentially over time steps:

$$D_\theta \mathcal{L}^{\langle t \rangle} = \sum_{k=1}^{t} \left(\prod_{i=k+1}^{t} D_{h^{\langle i \rangle}} h^{\langle i+1 \rangle} \right) D_{h^{\langle k \rangle}} \mathcal{L}^{\langle t \rangle} \cdot D_\theta h^{\langle k \rangle}$$

If $\|D_{h^{\langle i \rangle}} h^{\langle i+1 \rangle}\| > 1$, gradients *explode*; if < 1, gradients *vanish*.
Solution to exploding gradient problem: *gradient clipping*; at each gradient step, if the gradient is larger than a threshold K, 'clip' it to K:

$$D_\theta \mathcal{L} \leftarrow \begin{cases} D_\theta \mathcal{L} & \text{if } \|D_\theta \mathcal{L}\| < K \\ K \frac{D_\theta \mathcal{L}}{\|D_\theta \mathcal{L}\|} & \text{if } \|D_\theta \mathcal{L}\| \geq K \end{cases}$$

Solutions to vanishing gradient:
(1) GRU: Two gates control information flow (by learning optimal gate parameters, model learns how to accumulate/forget past info dynamically):

$$\begin{cases} \Gamma_r = \rho(W_r h^{\langle t-1 \rangle} + U_r x^{\langle t \rangle} + b_r) & \text{(reset)} \\ \Gamma_u = \rho(W_u h^{\langle t-1 \rangle} + U_u x^{\langle t \rangle} + b_u) & \text{(update)} \\ h^{\langle t \rangle} = (1 - \Gamma_u) \odot h^{\langle t-1 \rangle} + \Gamma_u \odot \tanh(W[\Gamma_r \odot h^{\langle t-1 \rangle}, x^{\langle t \rangle}]) \end{cases}$$

(2) LSTM: Memory cell $c^{\langle t \rangle}$ separates information flow:

$$\begin{cases} \Gamma_f, \Gamma_i, \Gamma_o = \text{forget/input/output gates} \\ c^{\langle t \rangle} = \Gamma_f \odot c^{\langle t-1 \rangle} + \Gamma_i \odot \tilde{c}^{\langle t \rangle} & \text{(memory)} \\ h^{\langle t \rangle} = \Gamma_o \odot \tanh(c^{\langle t \rangle}) & \text{(output)} \end{cases}$$

10 Classification & Regression Trees

Key Idea: Transform input x into output y via sequence of simple binary decisions. **Advantages:**

- Easy to interpret (medicine, insurance)
- Powerful with ensemble methods
- Can approximate any continuous function

Binary Tree Definition

Triplet $(\mathcal{T}, \mathcal{P}, \mathcal{V})$ where:
- $\mathcal{T} \subseteq \bigcup_{l \in \mathbb{N}_0} \{0,1\}^l$: nodes ($t^{\text{flip}}, t^{\text{cut}} \in \mathcal{T}$ if $t \in \mathcal{T} \setminus \{()\}$)
- $\mathcal{P} = \{P_t \subseteq \mathcal{X}\}$: partitions ($P_t \cup P_{t^{\text{flip}}} = P_{t^{\text{cut}}}$)
- $\mathcal{V} = \{y_t \in \mathcal{Y}\}$: values

Tree function: $f(x) = \sum_{t \in \mathcal{T}} y_t \mathbb{1}_{P_t}(x)$

Algorithm 3 Basic structure of tree growing algorithm

1: Initialize $\mathcal{T} = \{()\}$ and $P_{()} = \mathcal{X}$.
2: **while** Stopping criterion (*) is not reached **do**
3: Choose a node $t \in \mathcal{T}$ ($t \in \{0,1\}^l$), a variable $i \in \{1, \ldots, d\}$, and a set $C_i \subseteq \mathcal{X}_i$ according to criterion (**).
4: Set $\mathcal{T} \leftarrow \mathcal{T} \cup \{t0, t1\}$, where $t0 = (t_1, t_2, \ldots, t_l, 0)$ and $t1 = (t_1, t_2, \ldots, t_l, 1)$.
5: Set $P_{t0} = P_t \cap \{x \in \mathcal{X} : x_i \in C_i\}$, $P_{t1} = P_t \cap \{x \in \mathcal{X} : x_i \notin C_i\}$.
6: **end while**
7: Compute the values $\mathcal{V} = \{y_t : t \in \mathcal{T}\}$; hereby, for all $t \in \mathcal{T}$, y_t is calculated solely using the training data $\{y^i : i \in \{1, \ldots, m\}, x^i \in P_t\}$ according to criterion (***).
8: **return** $(\mathcal{T}, \mathcal{P}, \mathcal{V})$.

(***) HOW TO ASSIGN VALUES TO LEAVES?

Node Statistics: $p(t) = \frac{|\{i : x^i \in P_t\}|}{m}$, $p(y|t) = \frac{|\{i : x^i \in P_t, y^i = y\}|}{|\{i : x^i \in P_t\}|}$
Here, $p(t)$ is the estimated probability of an input x belonging to the set P_t, and $p(y|t)$ the likelihood of y being the output given that an input x belongs to the set P_t.
Regression ($\mathcal{Y} \in \mathbb{R}$): $y_t = \frac{1}{|\{i : x^i \in P_t\}|} \sum_{i : x^i \in P_t} y^i$ (emp. mean)
Classification ($\mathcal{Y} = \{1, \ldots, K\}$): $y_t = \arg\max_{c \in \mathcal{Y}} |\{i : x^i \in P_t, y^i = c\}|$ (majority vote)
General: $y_t \in \arg\min_{y \in \mathcal{Y}} \sum_{i=1}^{m} L(y, y^i) \mathbb{1}_{x^i \in P_t}$ (empirical mean for $L(y, y') = (y - y')^2$, majority vote for $L(y, y') = \mathbb{1}_{y \neq y'}$.

(**) HOW TO GROW A TREE/SPLIT NODES?

Tree Impurity: For tree structure $(\mathcal{T}, \mathcal{P})$: $I(t) \in [0, \infty)$ is impurity of node $t \in \mathcal{T}$, $\bar{I}(\mathcal{T}) = \sum_{t \in \mathcal{T}} p(t) I(t)$ is impurity of $\mathcal{T}$. *Lower impurity is better.* ⇒ The impurity determines the loss function which the overall tree function wants to minimize.
Optimal Split: $(t, i, C_i) \in \arg\min_{(t', i', C_{i'})} \bar{I}(\mathcal{T}^{(t', i', C_{i'})})$
Classification: $I(t) = \phi(p(1|t), \ldots, p(K|t)) \Rightarrow \bar{I}(T) \equiv$ cross-entropy

$$\begin{cases} \text{Gini: } \phi(p) = \frac{1}{2} \sum_{i=1}^{K} p_i (1 - p_i) \\ \text{Entropy: } \phi(p) = -\sum_{i=1}^{K} p_i \log(p_i) \end{cases}$$

Regression: $I(t) = \operatorname{Var}(\{y^i : x^i \in P_t\}) \Rightarrow \bar{I}(T) \equiv$ square loss

(*) STOPPING CRITERION/REGULARIZATION

Problem: Without stopping, tree splits until zero loss (overfitting). *Solution:* Add penalty term α for complexity, i.e., Penalizes number of leaves, locally defined for each node.
$I_\alpha(t) = I(t) + \frac{\alpha}{p(t)}$ $\qquad$ $\bar{I}_\alpha(\mathcal{T}) = \bar{I}(\mathcal{T}) + \alpha|\mathcal{T}| = \sum_{t \in \mathcal{T}} p(t) I_\alpha(t)$
Split Criterion: Split node $t \in \mathcal{T}$ if it decreases $\bar{I}_\alpha$, i.e., if:

$$\min_{(i, C_i)} \bar{I}_\alpha\left(\mathcal{T}^{(t,i,C_i)}\right) < \bar{I}_\alpha(\mathcal{T}) \Leftrightarrow \min_{i, C_i} (p(t0) I(t0) + p(t1) I(t1)) < p(t) I(t) - \alpha$$

Choice of α: Grow large tree (small α), then prune (increase α). Pruning exploits: α only affects *number* of splits, not their *structure*. Cross-validation efficient through pruning sequence.

11 Bagging & Random Forests

Key Idea: Combine multiple predictors to reduce variance.
Regression (Average): $\hat{f}(x) = \frac{1}{K}\sum_{k=1}^{K}\hat{f}^{(k)}(x)$
Classification (Maj. Voting): $\hat{f}(x) \in \arg\max_{y\in\mathcal{Y}}\sum_{k=1}^{K} 1_{\hat{f}^{(k)}(x)=y}$
Key Assumption: Estimators $\hat{f}^{(1)}, ..., \hat{f}^{(k)}$ not strongly positively correlated!

BOOTSTRAPPING

Resampling technique which generates additional artificial training data sets.

Algorithm 4 Bootstrap algorithm

1: Set $K \in \mathbb{N}$
2: **for** $k = 1, \dots, K$ **do**
3: **Non-Parametric:** Draw samples $\tilde{Z}_1^{(k)}, \dots, \tilde{Z}_m^{(k)}$ i.i.d. with repl.
4: **Parametric:** Fit distribution $\lambda(\varepsilon)$ to data, sample from it
5: $\hat{\theta}^{(k)} := g(\tilde{Z}_1^{(k)}, \dots, \tilde{Z}_m^{(k)})$
6: **end for**
7: Return $(\hat{\theta}^{(1)}, \dots, \hat{\theta}^{(K)})$ *(bootstrap-distr.* $\kappa^{(m,K)} = \frac{1}{K}\sum_{k=1}^{K}\delta_{\hat{\theta}^{(k)}}$*)*

BAGGING (**B**ootstrapping with **Agg**regat**ing**)

Aggregating is a technique to combine various models/estimators into a single (more powerful) estimator.
Subbagging: Use heuristic $\tilde{m} = \lceil m/2 \rceil$.

Algorithm 5 Bagging Algorithm (non-parametric)

1: **for** $k = 1, \dots, K$ **do**
2: Draw bootstrap sample $\tilde{Z}_1^{(k)}, \dots, \tilde{Z}_m^{(k)}$ with replacement
3: Take $\hat{f}^{(k)} := \arg\min_{f\in\mathcal{F}} \frac{1}{m}\sum_{i=1}^{m}\ell(\tilde{Y}_i^{(k)}, \tilde{X}_i^{(k)}) + \lambda R(f)$
4: **end for**
5: **return** Aggregated $\hat{f}^{agg}$ (Averaging, majority vote, or similar)

RANDOM FORESTS

Idea: Modify Bagging for more variability in $\hat{f}^{(k)}$, i.e., Randomize tree generation.

Algorithm 6 Random Forest Algorithm (non-parametric)

1: **for** $k = 1, \dots, K$ **do**
2: Draw bootstrap sample $\tilde{Z}_1^{(k)}, \dots, \tilde{Z}_m^{(k)}$ with replacement
3: Take $\hat{f}^{(k)}$ via randomized tree growing:
4: Sample feature subset $U \subset \{1, \dots, d\}$
5: Choose best split among features in U
6: **end for**
7: **return** Aggregated $\hat{f}^{agg}$ (Averaging, majority vote)

Benefits: Internal randomness reduces overfitting, increases model diversity and generalization.

12 Gradient boosted trees

Key Problem: Given binary classification data $D = \{(x_i, y_i) \in \mathcal{X} \times \{-1, +1\}\}_{i=1}^{m}$ and weak learners (slightly better than random), how to build a strong classifier?
Main Idea: Sequential training of weak learners on reweighted data, focusing on previously misclassified points. Then aggregate them into a strong classifier using error-weighted outputs.

ADABOOST ALGORITHM

Algorithm 7 AdaBoost

1: Initialize *data weights* $w_i^{(1)} = 1$ for $i = 1, \dots, m$
2: **for** $k = 1, \dots, K$ **do**
3: Fit $C_k = \arg\min_C \sum_{i=1}^{m} w_i^{(k)} \mathbb{1}_{\{C(x_i)\neq y_i\}}$
4: Compute weighted error rate $\text{err}_k := \frac{\sum_{i=1}^{m} w_i^{(k)} \mathbb{1}_{\{C_k(x_i)\neq y_i\}}}{\sum_{i=1}^{m} w_i^{(k)}}$
5: Compute classifier's weight $\alpha_k := \log(\frac{1-\text{err}_k}{\text{err}_k})$
6: Update data weights $w_i^{(k+1)} := w_i^{(k)} \exp(\alpha_k \mathbb{1}_{\{C_k(x_i)\neq y_i\}})$
7: **end for**
8: **return** $C(x) = \text{sign}(\sum_{k=1}^{K} \alpha_k C_k(x))$

Properties: Always assume $\text{err}_k < \frac{1}{2}$ (else flip predictions). Weight updates increase misclassified points:

$$w_i^{(k+1)} = \begin{cases} w_i^{(k)} & \text{if } C_k(x_i) = y_i \\ w_i^{(k)}(\frac{1-\text{err}_k}{\text{err}_k}) & \text{if } C_k(x_i) \neq y_i \end{cases}$$

$\Rightarrow$ Next weak classifier focuses more on misclassified samples.

STAGE-WISE ADAPTIVE MODELING

Additive Expansion: Build model sequentially: $f(x) = \sum_{k=1}^{K} \beta_k g_{\theta_k}(x)$ where g_{θ_k} are weak learners with parameters θ_k
AdaBoost as Special Case: Uses exponential loss: $\ell(f(x), y) = \exp(-yf(x))$. Optimal prediction: $f^*(x) = \frac{1}{2}\log(\frac{P(Y=1|X)}{P(Y=-1|X)})$. Weak learners: $g_{\theta_k}(x) = C_k(x)$ with weights $\beta_k = \alpha_k$.

GRADIENT BOOSTED TREES

Idea: Stage-wise adapt. modelling using trees as weak learners.
Model: $f_K(x) = \sum_{k=1}^{K} g_k(x)$ where $g_k = (T_k, P_k, V_k)$ small trees

Algorithm 8 Gradient Tree Boosting Algorithm

1: Initialize weights $f_0(x) = 0$.
2: **for** $k = 1, \dots, K$ **do**
3: Compute the *pseudo-residuals*: $r_{k,i} = -\left[\frac{\partial \ell(\hat{y}, y_i)}{\partial \hat{y}}\right]_{\hat{y}=f_{k-1}(x_i)}$
4: Fit a square loss regression tree $g_k = (T_k, P_k, V_k)$ to $(r_{k,i})_{i=1}^{m}$.
5: Set $f_k(x) = f_{k-1}(x) + g_k(x)$.
6: **end for**
7: **Return** f_K.

Pseudo-residual for Square loss: $r_{k,i} = y_i - f_{k-1}(x_i)$.
Regularization: Restrict Max. number of leaves J (typically 4-8), Learning rate η in update $f_k = f_{k-1} + \eta g_k$, Subsampling fraction α of data.

13 Dim. Reduction & Autoencoders

Goal: Represent high-dimensional data in low-dimensional space (*latent space*) by preserving as much information as possible.

PRINCIPAL COMPONENT ANALYSIS (PCA)

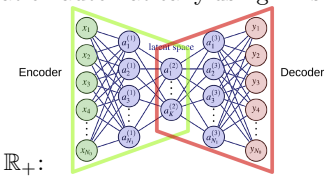
Main idea: Project data onto low-dimensional subspace while preserving max. amount of variance.
Let A be the standardized data (centered and unit variance)
Optimization Problem: Find orthonormal basis $\mathbf{v}_1, \dots, \mathbf{v}_d$ that maximize variance: $\mathbf{v}_1 = \arg\max_{\|\mathbf{w}\|_2=1} \|A\mathbf{w}\|_2^2$
Solution: Eigenvectors of A^TA (covariance matrix) corresponding to top eigenvalues $\sigma_1^2 \geq \sigma_2^2 \geq \dots$.
Dimensionality Reduction: Represent data using k top principal components, i.e., $\mathbf{x}_i \mapsto (\mathbf{x}_i^T\mathbf{v}_1, \dots, \mathbf{x}_i^T\mathbf{v}_k)$.
SVD Solution: $A = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$. Where $\mathbf{v}_i$: principal components (eigenvectors of A^TA), σ_i: singular values.
$\Rightarrow$ Low-dim representation: $A_k = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T = U_k\Sigma_k V_k^T$.
Explained Variance: A_k preserves $\frac{\sum_{i=1}^{k}\sigma_i^2}{\sum_{i=1}^{d}\sigma_i^2}$ of total variance.

KERNEL PCA

Key Idea: Extend PCA to non-linear transformations.
For kernel $k(x, x') = \langle\Phi(x), \Phi(x')\rangle_H$: (1) Compute $K_{ij} = k(x_i, x_j)$, (2) Center: $\tilde{K} = K - U_mK - KU_m + U_mKU_m$, (3) Solve $\tilde{K}\alpha = \lambda\alpha$, (4) Project: $\langle\mathbf{v}_k, \Phi(x)\rangle = \sum_{i=1}^{m} \alpha_i^{(k)} k(x_i, x)$.

AUTO-ENCODERS

Key Idea: Learn latent representation automatically using NNs.



- Encoder $F_\theta : \mathbb{R}^{N_0} \to \mathbb{R}^K$
- Decoder $G_{\tilde{\theta}} : \mathbb{R}^K \to \mathbb{R}^{N_0}$
- **Goal:** Minimize *dissimilia-rity* function $\ell : \mathbb{R}^{N_0} \times \mathbb{R}^{N_0} \mapsto \mathbb{R}_+$:

$$\arg\min_{(\theta,\tilde{\theta})} \frac{1}{m}\sum_{i=1}^{m} \ell(G_{\tilde{\theta}}(F_\theta(x_i)), x_i)$$

- **Sparse Auto-encoders (SAEs):** Penalty on latent activations. Encourage sparsity: $\mathcal{R}(F_\theta(x)) = \lambda|F_\theta(x)|_1$.
- **Denoising Auto-encoders (DAEs):** Train on corrupted inputs $\tilde{x} \sim C(\cdot|x)$. Minimize $\ell(x, G_{\tilde{\theta}}(F_\theta(\tilde{x})))$.
- **Contractive Auto-encoders (CAEs):** Penalty on input gradient: $\lambda|\nabla_x F_\theta(x)|_F^2$. Encourage robust latent representations (i.e., representations that locally do not change much).

Example: Categorical Feature Embedding
Compress high-dimensional one-hot encoded categorical features (common for insurance datasets) using autoencoders into low-dimensional embeddings; then use these latent representations in the model.

14 GNNs & Transformers

Key Idea: Combine flexibility of NNs with relational structure learning. GNNs leverage *inductive biases* to learn from graph-structured data, capturing relationships between entities.

GRAPH NEURAL NETWORKS

Graph Data: $G = (V, E)$ with Node data: $x_v \in \mathbb{R}^{d_V}$ for $v \in V$; Edge data: $x_e \in \mathbb{R}^{d_E}$ for $e \in E$; and Global data: $u \in \mathbb{R}^{d_U}$.
Aggregation functions: used to aggregate information across all edges connected to a given node (with varying #edges).

Graph Net Layer

Transforms graphical input data into graphical output data of the same shape.

1. **Edge transform:** $x'_e = \phi^E(x_e, x_v, x_w, u)$ for $e = (v, w)$.
   Edge $\to$ node agg.: $a_w = \rho^{E\to V}(\{x'_e : e = (v, w)\})$.
   Edge $\to$ global agg.: $a_E = \rho^{E\to U}(\{x'_e : e \in E\})$
2. **Node transform:** $x'_v = \phi^V(a_v, x_v, u)$
   Node $\to$ global agg.: $a_V = \rho^{V\to U}(\{x'_v : v \in V\})$
3. **Global transform:** $u' = \phi^U(a_E, a_V, u)$

$\Rightarrow$ Reason for efficiency of GNNs is that function ϕ^V (respectively its parameters) is *shared among all nodes*, and similarly ϕ^E is *shared among all edges.*

SELF-ATTENTION LAYER

Feedforward NNs $f_K, f_Q, f_V : \mathbb{R}^d \to \mathbb{R}^d$ (***K**eys, **Q**ueries, **V**alues*)

- **Edge transformation:** $\phi^E(x_{(v,w)}, x_v, x_w) = \big(A_{(v,w)}, \nu_{(v,w)}\big) := \Big(\frac{\langle f_Q(x_w), f_K(x_v)\rangle}{\sqrt{d}}, f_V(x_v)\Big) \in \mathbb{R}^{d+1}$.
- **Softmax aggr. ($\rho^{E\mapsto V}$):** $a_w = \sum_{e\in E_w} \frac{\exp(A_e)}{\sum_{e'\in E_w}\exp(A_{e'})}\nu_e \in \mathbb{R}^d$.
- **Node transformation:** $x'_v = \phi^V(a_v, x_v) = f_O(a_v) \in \mathbb{R}^d$.

TRANSFORMERS

Key Idea: Stack self-attention layers for seq. data (esp. text).

- Input tokens $w_1, ..., w_n \to$ embeddings in $\mathbb{R}^d$
- Learning objective: predict probability distribution over next token w_{n+1} given $w_1, ..., w_n$ (conditional prob. forecasting).
- Graph: Full $(V \times V)$ or Causal $\{(i, j) : i < j\}$ (i.e., masked attention, so token at position j can only pay attention tottokens at positions $i <$j).
- Layer: Multi-head attention + Norm + Residuals
- Loss: $\ell(p, y) = \sum_{i=1}^K y_i \log(p_i)$

Training: (1) Pretrain: Large-scale unsupervised (e.g., next token); **(2) Finetune:** Task-specific via full / frozen / PEFT / LoRA ($W \to W + AB, A \in \mathbb{R}^{d\times r}, B \in \mathbb{R}^{r\times k}$) / Soft-prompting.