Machine Learning: Comprehensive Study Material

Charles University Study Guide

June 4, 2026

Contents

1 Machine Learning: Basic Notions and Linear Models for Regression

1.1 General Principles and Statistical Decision Theory

Machine learning approaches fit functions according to specific criteria, commonly aiming for minimal expected error (loss), maximal likelihood, and applying penalties for model complexity.

Let $X \in \mathbb{R}^p$ denote a real-valued random input vector consisting of features, and let $Y \in \mathbb{R}$ denote a real-valued random output variable. The joint distribution is given by $P(X, Y)$.

The theory requires a loss function $L(Y, f(X))$ for penalizing errors in predictions. The most common error measure is the squared error loss:

$$L(Y, f(X)) = (Y - f(X))^2 \tag{1}$$

Choosing a function f requires minimizing the Expected Prediction Error (EPE), which is defined as:

$$\text{EPE}(f) = \mathbb{E}_{P(X,Y)}(Y - f(X))^2 \tag{2}$$

By conditioning on X, this expectation can be rewritten as:

$$\text{EPE}(f) = \mathbb{E}_{P(X)}\mathbb{E}_{P(Y|X)}\left([Y - f(X)]^2 \mid X\right) \tag{3}$$

It suffices to minimize the EPE pointwise to find the optimal conditional expectation (the regression function):

$$f(x) = \text{argmin}_c \mathbb{E}_{Y|X}\left([Y - c]^2 \mid X = x\right) \tag{4}$$

1.2 k-Nearest-Neighbor Methods

The k-nearest-neighbor method (where $k \in \mathbb{N}_{>0}$) directly estimates the conditional expectation $\mathbb{E}(Y \mid X = x)$ using observations in the training set $\mathcal{T}$ that are closest in the input space to a target x.

The function $\hat{f}(x)$ computes the mean of the responses over a specific neighborhood:

$$\hat{f}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i \tag{5}$$

Here, $N_k(x)$ is the neighborhood containing the k points in $\mathcal{T}$ that are closest to x. Under classification tasks, a majority vote is used instead of a numerical mean.

Key characteristics of k-Nearest-Neighbor:

- The training error typically increases with increasing k.
- The effective number of parameters is defined as N/k, which is generally larger than the parameter count p in linear regression.
- The prediction complexity is naive $\mathcal{O}(Np)$.
- Under mild regularity conditions on $P(X, Y)$, as $k, N \to \infty$ such that $k/N \to 0$, the estimate converges: $\hat{f}(x) \to \mathbb{E}(Y \mid X = x)$.
- The rate of convergence decreases significantly as dimensionality increases.

1.3 Overfitting and Model Selection

Overfitting occurs when a complex model minimizes training error but simultaneously exhibits an increase in expected prediction error on independent test data. Selecting the appropriate model complexity is critical. This involves choosing the parameter k via cross-validation or introducing penalty regularization.

In overparameterized models (where the number of parameters N_p exceeds data N), adding regularization can make the model perform better, exhibiting a double descent curve beyond the interpolation threshold.

Techniques to improve upon baseline models include kernel methods, assigning different weights to dimensions, utilizing local regression fits, using linear models fit to basis expansions, and utilizing sums of non-linearly transformed linear models.

1.4 Training Data Definition and Notation

Let the features be represented by a set of random variables $X_1, \ldots, X_p$. Let Y (or G for grouping/classification) represent the numerical goal variable. The training data $\mathcal{T}$ contains N samples:

$$\mathcal{T} = \{(x_1, y_1), \ldots, (x_N, y_N)\} \tag{6}$$

We define the corresponding vectors and matrices as follows:

- $X^T = \langle X_1, \ldots, X_p \rangle$ is the vector of features.
- $x_i^T = [x_{i1}, \ldots, x_{ip}]$ is the p-dimensional column vector representing the i-th sample.
- $\mathbf{X}$ is the $N \times p$ data matrix.
- x_j is the N-vector consisting of all observations on the single variable X_j.
- $\mathbf{y} = (y_1, \ldots, y_N)^T$ is the column vector of the goal variable observations.

1.5 Linear Regression

Given an input vector $X^T = (X_1, \ldots, X_p)$, the predicted output $\hat{Y}$ is generated via the linear model parameterized by $\beta \in \mathbb{R}^{p+1}$:

$$\hat{Y} = \hat{f}_\beta(X) = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j \tag{7}$$

By including a constant variable 1 in the input vector X and absorbing the intercept bias $\hat{\beta}_0$ into the parameter vector $\hat{\beta}$, the model can be written in vector form as an inner product:

$$\hat{Y} = X^T \hat{\beta} \tag{8}$$

For an entire dataset matrix $\mathbf{X}$ and goal vector $\mathbf{y}$, the goal is to search for the optimal $\hat{\beta}$ that minimizes the Residual Sum of Squares (RSS):

$$\text{RSS}(\beta) = \sum_{i=1}^{N} (y_i - x_i^T \beta)^2 = (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta) \tag{9}$$

Differentiating the RSS with respect to β yields the normal equations:

$$\mathbf{X}^T (\mathbf{y} - \mathbf{X}\beta) = 0 \tag{10}$$

Assuming $\mathbf{X}^T\mathbf{X}$ is non-singular (which may require removing dependent features), the unique least squares solution is:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{11}$$

For a given x_i, the prediction is $\hat{y}_i = x_i^T\hat{\beta}$. The complete predicted output vector is evaluated using the Hat matrix $\mathbf{H}$:

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\beta} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{12}$$

The Hat matrix is defined as $\mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$. Its trace evaluates to the effective parameters: $\text{trace}(\mathbf{H}) = \sum \text{diag}(\mathbf{H}) = p + 1$.

1.6 Computational Complexity of Linear Regression

The complexity of standard linear regression algorithms is as follows:

- **Direct Approach**: Computing $\mathbf{X}^T\mathbf{X}$ takes $\mathcal{O}(p^2N)$, and matrix inversion of the $p \times p$ result takes $\mathcal{O}(p^3)$. Total training complexity is $\mathcal{O}(p^2N + p^3)$.
- **Cholesky Decomposition**: $\mathcal{O}(p^3 + \frac{p^2}{2}N)$.
- **QR Decomposition**: $\mathcal{O}(p^2N)$.
- **Prediction Complexity**: Calculating $\beta^T X$ for a new sample takes $\mathcal{O}(p)$.

1.7 Improving the Least Squares Estimate

Standard least squares estimates can be improved to increase prediction accuracy (decrease variance) and improve interpretability. Techniques include:

- **Best Subset Selection**
- **Forward- and Backward-Stepwise Selection**
- **Forward-Stagewise Regression**: Operates identically to Forward-Stepwise but does not change previous coefficients. It features slow convergence but may be highly useful in high dimension p (!)
- **Penalized Methods**

1.8 Data Preprocessing: Centering, Standardization, and Covariance

Data is commonly preprocessed before fitting penalized models.

To center the variables, replace each feature so it has a zero mean:

$$x_{ij} \leftarrow x_{ij} - \overline{x}_j \tag{13}$$

The sample variance of a feature X_j is defined as:

$$s_j^2 \leftarrow \frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \overline{x}_j)^2 \tag{14}$$

Standardization performs centering and divides features by their respective standard deviation:

$$x_{ij} \leftarrow \frac{x_{ij} - \overline{x}_j}{s_j} \tag{15}$$

The sample covariance $\mathbf{S}$ is a $p \times p$ symmetric matrix containing elements:

$$s_{j,k} = \frac{1}{N} \sum_{i=1}^{N} (x_{ij} - \overline{x}_j)(x_{ik} - \overline{x}_k) \tag{16}$$

The sample correlation between columns x_j and x_k is computed as:

$$\rho_{j,k} \leftarrow \text{corr}(x_j, x_k) \leftarrow \frac{s_{j,k}}{s_j s_k} = \frac{\frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \overline{x}_j)(x_{ik} - \overline{x}_k)}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \overline{x}_j)^2}\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_{ik} - \overline{x}_k)^2}} \tag{17}$$

For standardized features, the correlation simplifies to $\frac{x_j^T x_k}{N}$.

1.9 Penalized Regression Methods

Penalized methods append a complexity penalty term $\lambda \sum_{j=1}^{p} |\beta_j|^q$ to the RSS:

$$\hat{\beta} = \text{argmin}_\beta \left(\sum_{i=1}^{N} (y_i - \beta_0 - x_i^T \beta)^2 + \lambda \sum_{j=1}^{p} |\beta_j|^q \right) \tag{18}$$

Ridge Regression ($q = 2$): Assuming the input matrix $\mathbf{X}$ is centered ($N \times p$), the intercept is $\hat{\beta}_0 = \frac{1}{N}\sum_{i=1}^{N} y_i$, and the coefficients solve to:

$$\hat{\beta}^{\text{ridge}} = (\mathbf{X}^T\mathbf{X} - \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{19}$$

Lasso Regression ($q = 1$): Solved via quadratic programming or LARS modifications. Generates sparsity by setting coefficients strictly to zero.

Elastic Net Penalty: A compromise between Ridge and Lasso, utilizing both norms:

$$\lambda \sum_{j=1}^{p} \Big(\alpha|\beta_j|^2 + (1-\alpha)|\beta_j| \Big) \tag{20}$$

It selects variables like the Lasso but shrinks the coefficients of correlated predictors together like Ridge regression, providing considerable computational advantages over standard L_q penalties.

1.10 Least Angle Regression (LARS) and Pathwise Coordinate Optimization

LARS serves as a democratic version of forward stepwise regression, providing an extremely efficient algorithm for computing the entire Lasso path. LARS works iteratively.

Let $\mathcal{A}_k$ be the active set of predictors, with parameters $\beta_{\mathcal{A}_k}$. The current residuals are:

$$r_k = \mathbf{y} - \mathbf{X}_{\mathcal{A}_k}\beta_{\mathcal{A}_k} \tag{21}$$

The correlation of each predictor with the residuals is $\langle x_j, r_k \rangle$. The algorithm assumes correlations are equal for all predictors natively in the active set. The coefficients change in the direction δ_k:

$$\beta_{\mathcal{A}_k} \leftarrow \beta_{\mathcal{A}_k} + \alpha\delta_k \tag{22}$$

$$\delta_k = (\mathbf{X}_{\mathcal{A}_k}^T \mathbf{X}_{\mathcal{A}_k})^{-1}\mathbf{X}_{\mathcal{A}_k}^T r \tag{23}$$

LARS Algorithm Workflow:

1 Initialize coefficients: $\beta_1, \ldots, \beta_p \leftarrow 0$

2 Initialize residuals: $r \leftarrow y - \overline{y}$

3 Find the predictor X_j most correlated with r

4 Establish initial active set: $\mathcal{A}_1 \leftarrow \{x_j\}$

5 Move β_j from 0 towards its least-squares coefficient $\langle x_j, r \rangle$ until some competitor X_k has as much correlation with the current residual as does X_j.

6 For $k = 2, \ldots, \min(N-1, p)$ do:

7 Add the new competitor to active set: $\mathcal{A}_k \leftarrow \mathcal{A}_{k-1} \cup \{x_l\}$

8 Move the current set of coefficients $\beta_{\mathcal{A}_k}$ by their joint least squares coefficient of the current residual until another competitor catches up.

9 End For

Lasso Modification of the LARS algorithm:

8a If a non-zero coefficient hits zero during a move, drop its variable from the active set of variables and recompute the current joint least squares direction.

The computational complexity of LARS yields $\mathcal{O}(p^2N + p^3)$.

Pathwise Coordinate Optimization explicitly solves modifications by iterating over coordinates. Assuming predictors are standardized to zero mean and unit norm, and fixing a penalty parameter λ, we hold other parameters fixed at their current estimates $\tilde{\beta}_k(\lambda)$. The objective function to optimize parameter j becomes:

$$R(\tilde{\beta}(\lambda), \beta_j) = \frac{1}{2} \sum_{i=1}^{N} \left(y_i - \sum_{k \neq j} x_{ik} \tilde{\beta}_k(\lambda) - x_{ij} \beta_j \right)^2 + \lambda \sum_{k \neq j} |\tilde{\beta}_k(\lambda)| + \lambda |\beta_j| \tag{24}$$

Using the partial residual $y_i - \tilde{y}_i^{(j)} = y_i - \sum_{k \neq j} x_{ik} \tilde{\beta}_k(\lambda)$, the update function for $\tilde{\beta}_j$ leverages the soft-thresholding operator S:

$$\tilde{\beta}_j(\lambda) \leftarrow S \left(\sum_{i=1}^{N} x_{ij} (y_i - \tilde{y}_i^{(j)}), \lambda \right) \tag{25}$$

The soft-thresholding operator is defined as:

$$S(t, \lambda) = \text{sign}(t)(|t| - \lambda)_+ \tag{26}$$

1.11 Model Complexity and Degrees of Freedom

The effective degrees of freedom evaluated for each model structure:

- Linear regression: p
- Ridge regression: $df(\hat{y}) = \text{tr}(\mathbf{X}(\mathbf{X}^T\mathbf{X} - \lambda\mathbf{I})^{-1}\mathbf{X}^T)$
- LARS: After k steps, $df(\hat{y}) = k$
- Lasso: Roughly equals the number of predictors retained in the model (as some drop out).

1.12 Grouped Lasso

[Marked: Not presented this year]

Grouped Lasso applies to structural sets of predictors, such as dummy variables representing levels of a categorical predictor or biological pathways. The p predictors are divided into L groups, with p_l indicating the number of variables in group l.

A matrix $\mathbf{X}_l$ represents predictors corresponding to the l-th group with a specific coefficient vector β_l. The formulation minimizes the following convex criterion:

$$\min_{\beta \in \mathbb{R}^p} \left(\left\| y - \beta_0 \mathbf{1} - \sum_{l=1}^{L} \mathbf{X}_l \beta_l \right\|_2^2 + \lambda \sum_{l=1}^{L} \sqrt{p_l} \left\| \beta_l \right\|_2 \right) \tag{27}$$

Here, $\|\cdot\|_2$ denotes the standard Euclidean norm (not squared) and the $\sqrt{p_l}$ term correctly accounts for varying group sizes.

1.13 Linear Methods for Classification

When applying linear techniques to classification tasks (e.g., discrete classes BLUE and ORANGE), the categories are encoded to numeric forms such as BLUE $= 0$ and ORANGE $= 1$.

The linear regression estimates continuous fitted values $\hat{Y}$, which are subsequently transformed to a fitted class output $\hat{G}$ using threshold bounds:

- $\hat{G} =$ BLUE if $\hat{Y} \leq 0.5$
- $\hat{G} =$ ORANGE if $\hat{Y} > 0.5$

The boundary cleanly splitting the classes forms a hyperplane known as the decision boundary:

$$\{x : x^T \hat{\beta} = 0.5\} \tag{28}$$

2 Machine Learning: Kernel Methods, Basis Expansion and Regularization

2.1 Basis Expansion and Splines

Linear regression assumes that the target is a linear function of the inputs X. In regression, we estimate $f(X) = \mathbb{E}(Y|X)$. To generalize this, we replace the vector of inputs X with additional variables $h_m(X)$ representing transformations of the input space.

We define a linear basis expansion in X with basis functions $h_m(X) : \mathbb{R}^p \to \mathbb{R}$ for $m = 1, \ldots, M$.

$$f(x) = \sum_{m=1}^{M} \beta_m h_m(X) \tag{29}$$

The only structural change is a different matrix of features X; the subsequent fitting process remains the same. Usually, we search $f_j(X_j)$ for each dimension using a backfitting algorithm in a Generalized Additive Model (GAM), where f_j's are unspecified smooth functions, X_j are the predictors, and Y is the outcome.

$$\mathbb{E}(Y|X_1, \ldots, X_p) = \alpha + f_1(X_1) + \cdots + f_p(X_p) \tag{30}$$

2.2 Simple Derived Features

For a one-dimensional feature X, several derived basis functions can be used:

- $h_m(X) = X_m$ for $m = 1, \ldots, M$ recovers the original linear model.
- $h_m(X) = X_j^2$ or $h_m(X) = X_j X_k$ introduces polynomial terms to achieve higher-order Taylor expansions. **!** The number of variables grows exponentially in the degree of the polynomial.
- $h_m(X) = \log(X_j)$, $\sqrt{X_j}$, $\|X\|$, and other nonlinear transformations.
- $h_m(X) = I(L_m \leq X_k < U_m)$ acts as an indicator for a region of X_k, yielding a piecewise constant contribution. Non-overlapping regions are used in regression trees.
- $h_m(X) = (X_j - \xi_k)_+ = \max((X_j - \xi_k)^3, 0)$ is a piecewise-polynomial spline basis.
- Wavelet bases can also be utilized.

2.3 Piecewise Polynomials and Splines

A piecewise polynomial function $f(X)$ is obtained by dividing the domain of X into continuous intervals separated by knots, and representing f by a separate polynomial in each interval.

For a system with two knots ξ_1 and ξ_2, the basis functions are formulated as follows:

- Three base interval indicator functions: $h_1(X) = I(X < \xi_1)$, $h_2(X) = I(\xi_1 \leq X < \xi_2)$, $h_3(X) = I(\xi_2 \leq X)$.
- Additional linear functions: $h_{m+3} = h_m(X) \cdot X$ for $m = 1, 2, 3$.
- Additional cubic functions: $h_{m+6} = h_m(X) \cdot X^2$ and $h_{m+9} = h_m(X) \cdot X^3$ for $m = 1, 2, 3$.

By enforcing a continuity restriction (ensuring the value at each knot ξ_j is unique), we construct a continuous piecewise linear basis. Two parameters are spared for the two continuity conditions.

- $h_1(X) = 1$
- $h_2(X) = X$
- $h_3(X) = (X - \xi_1)_+$
- $h_4(X) = (X - \xi_2)_+$

2.4 Cubic Splines and Order-M Splines

Cubic spline! A cubic spline is a piecewise cubic fit with continuous first and second derivatives at the knots ξ_i. The basis functions with two knots ξ_1 and ξ_2 are:

- $h_1(X) = 1$
- $h_2(X) = X$
- $h_3(X) = X^2$
- $h_4(X) = X^3$
- $h_5(X) = (X - \xi_1)^3_+$
- $h_6(X) = (X - \xi_2)^3_+$

The parameter count for this cubic spline is derived as:

$$(3 \text{ regions}) \times (4 \text{ pars per region}) - (2 \text{ knots}) \times (3 \text{ constraints per knot}) = 6$$

A cubic spline is an order-4 spline. Generally, an **Order-M spline** with knots ξ_j ($j = 1, \ldots, K$) is a piecewise-polynomial of order $(M-1)$ that has continuous derivatives up to order $(M-2)$. General truncated basis functions are defined as:

- $h_j(X) = X^{j-1}$ for $j = 1, \ldots, M$
- $h_{M+l}(X) = (X - \xi_l)^{M-1}_+$ for $l = 1, \ldots, K$

Regression splines are splines with fixed knots, typically placed at percentiles of the data X. The number of knots is specified by the degrees of freedom $(df - M)$, where h_0 does not count.

2.5 B-splines

B-splines use another basis that describes the same linear feature space. They are more stable numerically and useful for large numbers of knots K. **Note:** B-splines have quite a difficult recursive formula (*not needed for the exam*).

The recursive formula starts with:

$$B_{i,1}(x) = \begin{cases} 1 & \text{if } \xi_i \leq x \leq \xi_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{31}$$

Subsequent higher-order B-splines are defined as:

$$B_{i,k+1}(x) = \omega_{i,k}(x) B_{i,k}(x) + [1 - \omega_{i+1,k}(x)] B_{i+1,k}(x) \tag{32}$$

Where the weighting function is:

$$\omega_{i,k}(x) = \begin{cases} \frac{x - \xi_i}{\xi_{i+k} - \xi_i} & \text{if } \xi_{i+k} \neq \xi_i \\ 0 & \text{otherwise} \end{cases} \tag{33}$$

2.6 Natural Cubic Spline

Because polynomial fits tend to be erratic near the boundaries, the **Natural cubic spline!** enforces that the function is linear beyond the boundary knots. The basis functions N_i for $i = 1, \ldots, K$ are:

- $N_1(X) = 1$
- $N_2(X) = X$
- $N_{k+2}(X) = d_k(X) - d_{K-1}(X)$

Where $d_k(X)$ is defined as:

$$d_k(X) = \frac{(X - \xi_k)_+^3 - (X - \xi_K)_+^3}{\xi_K - \xi_k} \tag{34}$$

2.7 Smoothing Splines

Smoothing Spline! uses the maximal number of knots N (one at each observation), but regularizes the fit by adding a penalty for model complexity based on the second derivative.

$$RSS(f, \lambda) = \sum_{i=1}^{N} (y_i - f(x_i))^2 + \lambda \int (f''(t))^2 dt \tag{35}$$

Here, λ is the smoothing parameter:

- If $\lambda = 0$, f can be any function that perfectly interpolates the data.
- If $\lambda = \infty$, the penalty forces a zero second derivative, resulting in a simple least squares line fit.

This problem has a unique finite-dimensional minimizer, which is a natural cubic spline with knots at the unique values of x_i for $i = 1, \ldots, N$. The solution takes the form:

$$f(x) = \sum_{j=1}^{N} N_j(x)\theta_j \tag{36}$$

The objective criterion reduces to matrix form:

$$RSS(\theta, \lambda) = (\mathbf{y} - \mathbf{N}\theta)^T(\mathbf{y} - \mathbf{N}\theta) + \lambda\theta^T\Omega_N\theta \tag{37}$$

Where the elements are defined as $\{\mathbf{N}\}_{ij} = N_j(x_i)$ and the penalty matrix is $\{\Omega\}_{jk} = \int N_j''(t)N_k''(t)dt$.

2.8 Structure of the Penalty Matrix Ω

Let $a = x_1 = 0$, $b = x_{101} = 1$, and knots $\xi_l = x_{l+1}$ for $l = 1, \ldots, K$ and $K = 99$. For a cubic spline $M = 4$, the basis functions are $h_j(x) = x^{j-1}$ for $j = 1, \ldots, M$, and $h_{M+l}(x) = (x - \xi_l)_+^{M-1}$ for $l = 1, \ldots, K$. The matrix $\mathbf{H}$ has entries $h_j(x_i)$.

Let $\Omega = (\omega_{i,j})_{(M+K)\times(M+K)}$ be a symmetric matrix where the upper triangular components $\omega_{i,j} = \int_a^b h_i''(t)h_j''(t)dt$ are defined analytically:

- $\omega_{i,j} = 0$ for $i < M$
- $\omega_{M,j} = \frac{1}{3}b^3 - \frac{1}{2}b^2\xi_j + \frac{1}{6}\xi_j^3$ for $j > M$

For $j \geq i > M$, where $\xi_0 = \max\{\xi_{i-M}, \xi_{j-M}\}$, the entries are:

$$\omega_{i,j} = \frac{1}{3}(b^3 - \xi_0^3) - \frac{1}{2}(b^2 - \xi_0^2)(\xi_{i-M} + \xi_{j-M}) + (b - \xi_0)\xi_{i-M}\xi_{j-M} \tag{38}$$

2.9 Smoothing Splines Solution and Degrees of Freedom

The smoothing spline solution is a generalized ridge regression:

$$\hat{\theta} = (\mathbf{N}^T\mathbf{N} + \lambda\Omega_N)^{-1}\mathbf{N}^T\mathbf{y} \tag{39}$$

The fitted smoothing spline is then $\hat{f}(x) = \sum_{j=1}^{N} N_j(x)\hat{\theta}_j$. The smoothing spline acts as a linear smoother characterized by the smoother matrix $\mathbf{S}_\lambda$:

$$\hat{\mathbf{f}} = \mathbf{N}(\mathbf{N}^T\mathbf{N} + \lambda\Omega_N)^{-1}\mathbf{N}^T\mathbf{y} = \mathbf{S}_\lambda\mathbf{y} \tag{40}$$

The effective degrees of freedom are given by the trace of the smoother matrix:

$$df_\lambda = \text{trace}(\mathbf{S}_\lambda) \tag{41}$$

As $\lambda \to 0$, $df_\lambda \to N$ and $\mathbf{S}_\lambda \to \mathbf{I}$. As $\lambda \to \infty$, $df_\lambda \to 2$ and $\mathbf{S}_\lambda \to \mathbf{H}$, where $\mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is the hat matrix for simple linear regression.

The eigendecomposition of the smoother matrix is:

$$\mathbf{S}_\lambda = \sum_{k=1}^{N} \rho_k(\lambda) u_k u_k^T \tag{42}$$

Where the eigenvalues $\rho_k(\lambda)$ respond to the smoothing parameter via:

$$\rho_k(\lambda) = \frac{1}{1 + \lambda d_k} \tag{43}$$

The optimal degrees of freedom df_λ (or penalty λ) are usually selected to minimize the expected prediction error (EPE) via cross-validation.

2.10 Multidimensional Splines

For multidimensional domains where $X \in \mathbb{R}^2$, given basis functions $h_{1k}(X_1)$ for $k = 1, \ldots, M_1$ in the first coordinate and $h_{2k}(X_2)$ for $k = 1, \ldots, M_2$ in the second coordinate, an $M_1 \times M_2$ dimensional tensor product basis is defined:

$$g_{jk}(X) = h_{1j}(X_1)h_{2k}(X_2) \tag{44}$$

This can represent a two-dimensional function $g(X)$ with coefficients θ_{jk} fitted by least squares:

$$g(X) = \sum_{j=1}^{M_1}\sum_{k=1}^{M_2} \theta_{jk} g_{jk}(X) \tag{45}$$

2.11 Multidimensional Smoothing Splines

For multidimensional smoothing splines (e.g., Thin Plate Splines), we place a knot at each example and add a complexity penalty $J[f]$. The solution formulation is:

$$f(x) = \beta_0 + \beta^T x + \sum_{j=1}^{N} \alpha_j h_j(x) \tag{46}$$

Where the basis uses a radial function $h_j(x) = \eta(\|x - x_j\|)$, specifically with $\eta(z) = z^2 \log z^2$. This method has a complexity of $\mathcal{O}(N^3)$ or $\mathcal{O}(NK^2 + K^3)$ if using K knots. The corresponding complexity penalty in $\mathbb{R}^2$ is:

$$J[f] = \iint_{\mathbb{R}^2} \left[\left(\frac{\partial^2 f(x)}{\partial x_1^2} \right) + \left(\frac{\partial^2 f(x)}{\partial x_1 \partial x_2} \right) + \left(\frac{\partial^2 f(x)}{\partial x_2^2} \right) \right] dx_1 dx_2 \tag{47}$$

2.12 Kernel Methods

Kernel methods estimate the regression function $f(x) \in \mathbb{R}$ by constructing a simple model separately at each query point x_0. The resulting $\hat{f}(X)$ is smooth in $\mathbb{R}^p$. Localization is achieved via a weighting function (or kernel) $k_\lambda(x_0, x_i)$ that assigns weight based on the distance from x_0, where λ dictates the neighborhood width. These are memory-based methods requiring little to no training because the entire training set serves as the model.

k-Nearest Neighbour Kernel: $N_k(x)$ is the set of k points nearest to x in squared distance, where all points have equal weight. The function is bumpy and discontinuous:

$$\hat{f}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i \tag{48}$$

Nadaraya-Watson Kernel-Weighted Average!

$$\hat{f}(x_0) = \frac{\sum_{i=1}^{N} k_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{N} k_\lambda(x_0, x_i)} \tag{49}$$

The **Epanechnikov quadratic kernel** takes the form $k_\lambda(x_0, x) = D\left(\frac{|x - x_0|}{\lambda}\right)$ using the function:

$$D(t) = \begin{cases} \frac{3}{4}(1 - t^2) & \text{if } |t| \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{50}$$

Other common kernels allow variable width using the general form $k_\lambda(x_0, x) = D\left(\frac{|x - x_0|}{h_\lambda(x_0)}\right)$:

- **k-NN Kernel:** $k_k(x_0, x) = \delta_{x \in \text{Neighbours}_k(x_0)}$ where $h_k(x_0) = |x_0 - x_{[k]}|$ ($x_{[k]}$ is the k-th closest point).
- **Tri-cube Kernel:** $D(t) = (1 - |t|^3)^3$ if $|t| \leq 1$, and 0 otherwise.
- **Gaussian Kernel:** $D(t) = \frac{1}{\lambda} e^{-\frac{\|x - x_0\|^2}{2\lambda}}$.

2.13 Local Linear and Polynomial Regression

Locally-weighted averages can be badly biased on the domain boundaries or when X values are not equally spaced. Fitting local straight lines (Local Linear Regression) or quadratic curves can reduce this bias.

The locally weighted regression optimization objective is:

$$\min_{\alpha(x_0), \beta(x_0)} \sum_{i=1}^{N} k_\lambda(x_0, x_i)[y_i - \alpha(x_0) - \beta(x_0) x_i]^2 \tag{51}$$

The estimate at the query point is $\hat{f}(x_0) = \hat{\alpha}(x_0) + \hat{\beta}(x_0) x_0$. In matrix notation, let $\mathbf{X}$ be an $N \times (p + 1)$ matrix mapping $x^T \to (1, x)$, and let $\mathbf{W}(x_0)$ be an $N \times N$ diagonal matrix containing the kernel weights $k_\lambda(x_0, x_i)$. The locally weighted linear function of $\mathbf{y}$ is:

$$\hat{f}(x_0) = x_0^T (\mathbf{X}^T \mathbf{W}(x_0) \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W}(x_0) \mathbf{y} \tag{52}$$

Local linear fits help dramatically at boundaries, while local quadratic fits help reduce bias caused by interior curvature. It is recommended to fix the polynomial degree based on the application rather than combining degrees.

For **Structured Local Regression in** $\mathbb{R}^p$, a positive semidefinite matrix $\mathbf{A}$ is introduced to weigh the different coordinates within the kernel:

$$k_\lambda(x_0, x) = D\left(\frac{(x - x_0)^T \mathbf{A} (x - x_0)}{h_\lambda(x_0)}\right) \tag{53}$$

2.14 Computational Consideration

Kernel smoothing and basis expansion techniques exhibit distinct time complexities:

- For kernel smoothing, the model comprises the entire training set, meaning fitting occurs entirely at evaluation/prediction time. A single observation x_0 fit costs $\mathcal{O}(N)$.
- Expanding into M basis functions costs $\mathcal{O}(M)$ per evaluation, where typically $M \sim \mathcal{O}(\log N)$.
- Basis function methods (e.g., splines) carry an initial cost of at least $\mathcal{O}(NM^2 + M^3)$. For regression splines with K knots, this is $\mathcal{O}(N(K+M)^2 + (K+M)^3)$. For B-splines using Cholesky decomposition, computing the solution takes $\mathcal{O}(N(M+1))$.
- Selecting the smoothing parameter λ offline via cross-validation generally costs $\mathcal{O}(N^2)$.
- Popular local regression implementations (like `loess`) accelerate predictions by computing the exact fit at M locations taking $\mathcal{O}(NM)$ and interpolating elsewhere in $\mathcal{O}(M)$ per evaluation.

3 Machine Learning: Linear and k-NN Methods for Classification and Their Extensions

3.1 Likelihood, Entropy, and KL Divergence

The probability of the data given the model is called the likelihood of the model θ given the data. Predicting probabilities via maximum likelihood estimate is equivalent to the maximum log-likelihood estimate.

Let G be the domain of possible classes, p be the true probability distribution, and $q = \theta$ be the estimated probability distribution. The log-likelihood function $l(\theta)$ is defined as:

$$\text{loglik}(\theta) = \mathbb{E}_{g\sim p(g)}[\log \theta_g] \hat{=} \sum_{g\in G} p(g) \log \theta_g = l(\theta) \tag{54}$$

The entropy $H(p)$ of a probability distribution p on $g \in G$ is defined as:

$$H(p) = -\mathbb{E}_{g\sim p(g)}[\log p(g)] = -\sum_{g\in G} p(g) \log p(g) \tag{55}$$

The cross-entropy $H(p, q)$ of distributions p and q on the same domain G is:

$$H(p, q) = -\mathbb{E}_{g\sim p(g)}[\log q(g)] = -\sum_{g\in G} p(g) \log q(g) \tag{56}$$

The Kullback-Leibler (KL) divergence between p and q is defined as:

$$D_{KL}(p\|q) = \sum_{g\in G} p(g) \log \frac{p(g)}{q(g)} \tag{57}$$

The relationship between cross-entropy, entropy, and KL divergence is given by:

$$H(p, q) = H(p) + D_{KL}(p\|q) \tag{58}$$

Maximizing log-likelihood is equivalent to minimizing cross-entropy or minimizing KL-divergence. KL divergence is minimal (zero) if and only if $p = q$ almost everywhere.

3.2 Logistic Regression

Logistic regression ensures that the predicted probability falls within $(0, 1)$ by transforming a linear prediction using the logistic function (sigmoid). The inverse function is called the logit.

- **Logistic function:**
$$f(x) = \frac{1}{1 + e^{-k(x-x_0)}}$$
- **Logit function:**
$$\text{logit}(p) = \log \frac{p}{1-p}$$

For K-class classification, we estimate $(p + 1) \times (K - 1)$ parameters denoted by $\theta = \{\beta_{10}, \beta_1^T, \ldots, \beta_{(K-1)0}, \beta_{K-1}^T\}$. The log-odds for classes 1 to $K - 1$ against the reference class K are modeled as linear functions:

$$\log \frac{\Pr(G = g_k \mid X = x)}{\Pr(G = g_K \mid X = x)} = \beta_{k0} + \beta_k^T x \quad \text{for } k = 1, \ldots, K-1 \tag{59}$$

The corresponding probabilities for each class k are computed as:

$$\Pr(G = g_k \mid X = x) = \frac{e^{\beta_{k0}+\beta_k^T x}}{1 + \sum_{l=1}^{K-1} e^{\beta_{l0}+\beta_l^T x}} \quad \text{for } k = 1, \ldots, K-1 \tag{60}$$

$$\Pr(G = g_K \mid X = x) = \frac{1}{1 + \sum_{l=1}^{K-1} e^{\beta_{l0}+\beta_l^T x}} \tag{61}$$

3.2.1 Fitting Two-Class Logistic Regression

In the two-class model, the class label g_i is encoded via a 0/1 response y_i, where $y_i = 1$ if $g_i = g_1$. Let $p(x;\theta) = \Pr(G = g_1 \mid x;\theta)$. The log-likelihood $l(\theta)$ to be maximized is:

$$l(\beta) = \sum_{i=1}^{N} \left(y_i \log p(x_i;\beta) + (1-y_i)\log(1-p(x_i;\beta))\right) = \sum_{i=1}^{N} \left(y_i \beta^T x_i - \log(1+e^{\beta^T x_i})\right) \tag{62}$$

To maximize, the first derivative is set to zero, yielding $p+1$ nonlinear equations:

$$\frac{\partial l(\beta)}{\partial \beta} = \sum_{i=1}^{N} x_i (y_i - p(x_i;\beta)) = 0 \tag{63}$$

3.2.2 Newton-Raphson Algorithm (IRLS)

To solve the nonlinear score equations, the Newton-Raphson algorithm is used. The Hessian matrix (second derivative) is:

$$\frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T} = -\sum_{i=1}^{N} x_i x_i^T p(x_i;\beta)(1-p(x_i;\beta)) \tag{64}$$

A single Newton-Raphson update step is:

$$\beta^{\text{new}} = \beta^{\text{old}} - \left(\frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T}\right)^{-1} \frac{\partial l(\beta)}{\partial \beta} \tag{65}$$

In matrix notation, let:

- $\mathbf{y}$: the vector of responses y_i
- $\mathbf{X}$: the $N \times (p+1)$ data matrix
- $\mathbf{p}$: the vector of fitted probabilities with i-th element $p(x_i;\beta^{\text{old}})$
- $\mathbf{W}$: a diagonal matrix with weights $w_{ii} = p(x_i;\beta^{\text{old}})(1-p(x_i;\beta^{\text{old}}))$

The gradient and Hessian become:

$$\frac{\partial l(\beta)}{\partial \beta} = \mathbf{X}^T(\mathbf{y}-\mathbf{p}) \tag{66}$$

$$\frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T} = -\mathbf{X}^T\mathbf{W}\mathbf{X} \tag{67}$$

The update rule forms the Iteratively Reweighted Least Squares (IRLS) algorithm:

$$\beta^{\text{new}} = \beta^{\text{old}} + (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{y}-\mathbf{p}) = (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}\mathbf{z} \tag{68}$$

where the adjusted response $\mathbf{z}$ is defined as:

$$\mathbf{z} = \mathbf{X}\beta^{\text{old}} + \mathbf{W}^{-1}(\mathbf{y}-\mathbf{p}) \tag{69}$$

At each iteration, $\mathbf{p}$, $\mathbf{W}$, and $\mathbf{z}$ change, effectively solving:

$$\beta^{\text{new}} \leftarrow \arg\min_{\beta} (\mathbf{z}-\mathbf{X}\beta)^T\mathbf{W}(\mathbf{z}-\mathbf{X}\beta) \tag{70}$$

Significance Testing: A Wald test uses the Z-score. A value of $|Z| > 2$ is significant at the 5% level.

3.3 Regularized Logistic Regression

3.3.1 L_1 Regularization (Lasso-like)

L_1 penalty moves coefficients towards 0 (some become exactly 0). The unpenalized intercept β_0 is excluded from the penalty. The objective is to maximize:

$$\arg\max_{\beta_0,\beta} \left(\sum_{i=1}^{N} \left[y_i(\beta_0 + \beta^T x_i) - \log(1 + e^{\beta_0+\beta^T x_i}) \right] - \lambda \sum_{j=1}^{p} |\beta_j| \right) \tag{71}$$

3.3.2 Linear Regression with Elastic Net Penalty

The Elastic Net penalty combines L_1 and L_2 terms. The optimization objective across all classes K is:

$$\max_{\{\beta_{0k},\beta_k \in \mathbb{R}^p\}_1^K} \left[\sum_{i=1}^{N} \log \Pr(g_i \mid x_i) - \lambda \left(\sum_{k=1}^{K} \sum_{j=1}^{p} \left(\alpha|\beta_{kj}| + (1-\alpha)\beta_{kj}^2 \right) \right) \right] \tag{72}$$

3.4 Local Likelihood and Generalized Additive Models

3.4.1 Local Likelihood (Local Logistic Regression)

We fit the model locally at point x_0, weighting the log-likelihood by a kernel $k_\lambda(x_0, x_i)$ and centering the estimate at x_0:

$$l(\beta_{x_0}) = \sum_{i=1}^{N} k_\lambda(x_0, x_i) \left\{ y_i \beta_{x_0}^T (x_i - x_0) - \log(1 + e^{\beta_{x_0}^T (x_i - x_0)}) \right\} \tag{73}$$

3.4.2 Generalized Additive Model (GAM)

Each feature X_j is approximated by a natural spline. The overall model is:

$$\text{logit}[\Pr(\text{class} \mid X)] = \theta_0 + h_1(X_1)^T\theta_1 + h_2(X_2)^T\theta_2 + \cdots + h_p(X_p)^T\theta_p \tag{74}$$

Here, θ_j are vectors of coefficients for their associated natural spline basis functions h_j. These basis functions evaluated over N samples form a basis matrix $\mathbf{H}$. The covariance of the estimated parameters $\hat{\theta}$ is:

$$\hat{\Sigma} = (\mathbf{H}^T\mathbf{W}\mathbf{H})^{-1} \tag{75}$$

The variance of a single variable j is $v_j(X_j) = \text{Var}[\hat{f}_j(X_j)] = h_j(X_j)^T \hat{\Sigma}_{jj} h_j(X_j)$. The error bounds are constructed as $\hat{f}_j(X_j) \pm 2\sqrt{v_j(X_j)}$.

3.5 Discriminant Analysis

Linear Discriminant Analysis (LDA) models the data using a multivariate Gaussian distribution for each class, assuming a common covariance matrix Σ.

$$\phi_k(x) = \frac{1}{\sqrt{|2\pi\Sigma|}} e^{-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)} \tag{76}$$

The model parameters are the class prior probability π_k, the class mean μ_k, and the shared covariance Σ, evaluated from N_k training instances per class G_k:

$$\hat{\pi}_k = \frac{N_k}{N} \quad , \quad \hat{\mu}_k = \frac{\sum_{x_i:G(x_i)=g_k} x_i}{N_k} \quad , \quad \hat{\Sigma} = \sum_{k=1}^{K} \sum_{x_i:G(x_i)=g_k} \frac{(x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T}{N - K} \tag{77}$$

Posterior probability given by Bayes rule:

$$\Pr(G = g_k \mid X = x) = \frac{\phi_k(x)\pi_k}{\sum_{l=1}^{K} \phi_l(x)\pi_l} \tag{78}$$

To classify a new instance x, LDA predicts the class G_k that maximizes the linear discriminant function $\delta_k(x)$:

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \tag{79}$$

3.5.1 Quadratic Discriminant Analysis (QDA)

QDA relaxes the LDA assumption by allowing a distinct covariance matrix Σ_k for each class:

$$\hat{\Sigma}_k = \sum_{x_i : G(x_i) = g_k} \frac{(x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T}{|G_k| - 1} \tag{80}$$

The QDA discriminant function to maximize is:

$$\delta_k(x) = -\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) - \frac{1}{2} \log|\Sigma_k| + \log \pi_k \tag{81}$$

3.5.2 Regularized Discriminant Analysis (RDA)

RDA provides a weighted average between the independent covariances of QDA and the common covariance of LDA to tune model complexity using parameter α:

$$\hat{\Sigma}_k(\alpha) = \alpha \hat{\Sigma}_k + (1 - \alpha)\hat{\Sigma} \tag{82}$$

3.5.3 Diagonal Linear Discriminant Analysis

For very high-dimensional data (e.g., gene expression), Σ is restricted to a diagonal matrix. The discriminant function is:

$$\delta_k(x^*) = -\sum_{j=1}^{p} \frac{(x_j^* - \bar{x}_{jk})^2}{s_j^2} + 2 \log(\pi_k) \tag{83}$$

where s_j is the pooled within-class standard deviation of the j-th feature and $\bar{x}_{jk}$ is the centroid of class k for feature j.

3.5.4 Computations for LDA

The standard computational complexity for LDA is $O(N^3)$, heavily dependent on matrix inversion. However, it can be computed using matrix decomposition:

- Compute the eigendecomposition of the common covariance: $\hat{\Sigma} = \mathbf{U}\mathbf{D}\mathbf{U}^T$.
- Sphere the data with respect to $\hat{\Sigma}$:

$$X^* \leftarrow \mathbf{D}^{-\frac{1}{2}} \mathbf{U}^T X$$

- Classify to the closest class centroid in the transformed space, adjusting for the prior probabilities π_k.

3.6 Nearest-Neighbor Methods

3.6.1 k-Nearest Neighbors (k-NN)

The k-NN algorithm identifies the neighborhood $N_k(x)$ of the k closest observations to the query point x and uses majority voting to classify:

$$\hat{g}(x) = \text{majority}_{x_i \in N_k(x)} g(x_i) \tag{84}$$

3.6.2 Curse of Dimensionality

In high dimensions, points become sparse, forcing the algorithm to rely heavily on extrapolation near the boundaries of the space. For N instances uniformly distributed in a p-dimensional unit ball, the median distance $d(p, N)$ from the origin to the nearest neighbor is:

$$d(p, N) = \left(1 - \left(\frac{1}{2}\right)^{\frac{1}{N}}\right)^{\frac{1}{p}} \tag{85}$$

This indicates that as dimension p grows, the nearest neighbor is increasingly found far from the origin, approaching the boundary.

3.6.3 Discriminant Adaptive Nearest-Neighbor Methods (DANN)

To address boundary issues in k-NN, DANN adjusts the distance metric locally at query point x_0:

$$D(x, x_0) = (x - x_0)^T \Sigma (x - x_0) \tag{86}$$

where Σ is computed using the within-class covariance $\mathbf{W}$ and between-class covariance $\mathbf{B}$ fitted in the neighborhood:

$$\Sigma = \mathbf{W}^{-\frac{1}{2}} \left[\mathbf{W}^{-\frac{1}{2}} \mathbf{B} \mathbf{W}^{-\frac{1}{2}} + \epsilon \mathbf{I}\right] \mathbf{W}^{-\frac{1}{2}} \tag{87}$$

Parameter $\epsilon = 1$ adjusts the neighborhood. Near the class boundary, the metric shrinks the neighborhood orthogonally to the boundary; in the interior, it remains circular.

3.7 Method Summary and Extensions

The slides summarize the following topics:

- Likelihood example
- Logistic regression
- *ext.* Logistic regression with L_1 penalty, elastic net penalty
- Linear and quadratic discriminant analysis
- *ext.* Regularized discriminant analysis
- *ext.* Reduced rank discriminant analysis
- *ext.* Diagonal discriminant analysis
- Nearest-neighbor methods (k-NN)
- *ext.* Local likelihood (local logistic regression)
- *ext.* Discriminant Adaptive NN methods (DANN)
- **?** Support Vector Machines (marked with a question mark indicating possible future topic or not covered)

4 Machine Learning: Model Assessment and Selection

4.1 Vanilla Machine Learning Pipelines

The procedure for Vanilla Machine Learning without Hyperparameter Tuning is defined as follows:

- Procedure `FIND_ML_MODEL(X, y)`
- `X_train0, y_train, X_test0, y_test` ← `train_test_split(X, y, train_size=2/3)`
- `preproc` ← `preprocessing method (standardization, ...)`
- `X_train` ← `preproc.fit_transform(X_train0, y_train)`
- `X_test` ← `preproc.transform(X_test0)`
- `model` ← `LinearRegression()`
- `model.fit(X_train, y_train)`
- return `model`, `test_error(model.predict(X_test), y_test)`

The procedure for Vanilla Machine Learning with Hyperparameter Tuning iterates through cross-validation folds:

- Procedure `FIND_ML_MODEL(X, y)`
- `X_train0, y_train, X_test0, y_test` ← `train_test_split(X, y, train_size=2/3)`
- `preproc` ← `preprocessing method (standardization, ...)`
- `X_train` ← `preproc.fit_transform(X_train0, y_train)`
- `X_test` ← `preproc.transform(X_test0)`
- `model` ← `Ridge()`
- `X_folds` ← `split_to_folds(X_train, y_train)`
- For each `fold` in `X_folds` do:
  - For each λ in `hyperparameter_grid` do:
    - `model(`λ`).fit(X_folds` $\setminus$ {`fold`}`)`
    - `err(`λ`, fold) = error(model.predict(fold.X), fold.y)`
- To select tuning parameter λ:
  - `err(`λ`)` $= \text{mean}_{\texttt{fold} \in \texttt{X\_folds}}$`(err(`$\lambda$`, fold))`
  - $\lambda^* = \arg\min_\lambda$ `err(`λ`)`
- `model` ← `model(`λ^*`).fit(X_train, y_train)`
- return `model`, `test_error(model.predict(X_test), y_test)`

4.2 Error Estimates

Generalization error is defined as the expected prediction error over an independent test sample:

$$Err = \mathbb{E}[L(Y, \hat{f}(X))] \tag{88}$$

where both X and Y are drawn randomly from their joint distribution (population).

Training error is computed on the data used to fit the model:

$$\overline{err} = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}(x_i)) \tag{89}$$

Test error is an evaluation on data not used to fit the model, serving as a point estimate of the generalization error:

$$err = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}(x_i)) \tag{90}$$

4.3 Loss Functions

For continuous outcomes, common loss functions for regression include:

- Square error loss:
$$L(y, \hat{y}) = (y - \hat{y})^2$$
- Absolute error loss:
$$L(y, \hat{y}) = |y - \hat{y}|$$
- Huber error loss:
$$L_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{for } |y - \hat{y}| \leq \delta \\ \delta|y - \hat{y}| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$$

For classification, a qualitative response G takes one of K values, typically classified as $\hat{G}(X) = \arg\max_k \hat{p}_k(X)$. Commonly applied classification loss functions (with binary variants encoded in $\{-1, +1\}$) include:

- 0-1 loss, misclassification:
$$L(G, \hat{G}(X)) = I(G \neq \hat{G}(X))$$
- Log-likelihood, cross-entropy, deviance:
$$L(G, \hat{p}(X)) = -2I(G = k) \log \hat{p}_k(X)$$
- The expectation for binary log-likelihood:
$$\mathbb{E}[L_{\{0,1\}}(g, \hat{p})] = -2[p \log \hat{p} + (1 - p) \log(1 - \hat{p})]$$
- Exponential:
$$e^{-yf}$$
- Support vector:
$$\max(0, 1 - yf(x))$$

4.4 Cross-Validation

Data is often split into a Train, Validation, and Test set with recommended ratios of $\frac{1}{2} : \frac{1}{4} : \frac{1}{4}$, or $\frac{2}{3} : \frac{1}{3}$ when validation is omitted. Cross-validation splits data into K equal-sized parts (e.g., $K = 5$, $K = 10$, or $K = N$ for one-leave-out). Given a partition function $\kappa : \{1, \ldots, N\} \to \{1, \ldots, K\}$ and model $\hat{f}^{-\kappa(i)}$ fitted without the k-th part, the estimate is:

$$CV = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-\kappa(i)}(x_i)) \tag{91}$$

When evaluating parameters α:

$$CV(\alpha) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-\kappa(i)}(x_i, \alpha)) \tag{92}$$

The One standard error rule selects the most parsimonious model whose error is no more than one standard error above the best model's error.

4.5 Bias-Variance Decomposition

Assuming an underlying true function $Y = f(X) + \epsilon$ with $\mathbb{E}(\epsilon) = 0$ and $\text{Var}(\epsilon) = \sigma_\epsilon^2$. The expected prediction error at point $X = x_0$ under squared-error loss expands as:

$$Err(x_0) = \mathbb{E}[(Y - \hat{f}(x_0))^2 \mid X = x_0] \tag{93}$$

$$Err(x_0) = \sigma_\epsilon^2 + [\mathbb{E}\hat{f}(x_0) - f(x_0)]^2 + \mathbb{E}[\hat{f}(x_0) - \mathbb{E}\hat{f}(x_0)]^2 \tag{94}$$

This structurally reflects Irreducible Error + Bias^2 + Variance.

For K-Nearest neighbor:

$$Err(x_0) = \sigma_\epsilon^2 + \left[f(x_0) - \frac{1}{k} \sum_{l=1}^{k} f(x_{(l)}) \right]^2 + \frac{\sigma_\epsilon^2}{k} \tag{95}$$

For a linear fit where $h(x_0)^T y = x_0^T (X^T X)^{-1} X^T y = \hat{f}_p(x_0)$:

$$Err(x_0) = \sigma_\epsilon^2 + [\mathbb{E}\hat{f}_p(x_0) - f(x_0)]^2 + \|h(x_0)\| \sigma_\epsilon^2 \tag{96}$$

Averaging the variance over N points gives $\frac{p}{N} \sigma_\epsilon^2$:

$$\frac{1}{N} \sum_{i=1}^{N} Err(x_i) = \sigma_\epsilon^2 + \frac{1}{N} \sum_{i=1}^{N} [f(x_i) - \mathbb{E}\hat{f}(x_i)]^2 + \frac{p}{N} \sigma_\epsilon^2 \tag{97}$$

Applying penalized complexity in Ridge Regression with fit $h(x_0) y = x_0^T (X^T X + \lambda I)^{-1} X^T y$, the bias decomposes further against the unpenalized optimal fit β_*:

$$\mathbb{E}_{x_0}[f(x_0) - \mathbb{E}\hat{f}_\lambda(x_0)]^2 = \mathbb{E}_{x_0}[f(x_0) - \beta_*^T x_0]^2 + \mathbb{E}_{x_0}[\beta_*^T x_0 - \mathbb{E}\hat{\beta}_\lambda^T x_0]^2 \tag{98}$$

This isolates Average Model Bias and Average Estimation Bias.

4.6 Optimism of the Training Error Rate

The in-sample error relies on fixed observations X_i while drawing new response samples Y^{new}:

$$Err_{in} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_y \mathbb{E}_{Y^{new}} L(Y_i^{new}, \hat{f}(x_i)) \tag{99}$$

Optimism compares this error with the expected training error:

$$op = Err_{in} - \mathbb{E}_y(\overline{err}) \tag{100}$$

For squared error, 0-1, and related losses, this is computed as:

$$op = \frac{2}{N}\sum_{i=1}^{N} \text{cov}(\hat{y}_i, y_i) \tag{101}$$

In an additive error linear model with d basis functions, the covariance sum is $d\sigma_\epsilon^2$:

$$Err_{in} = \mathbb{E}_y \overline{err} + 2\frac{d}{N}\sigma_\epsilon^2 = C_p = AIC_{gauss} \tag{102}$$

Derivation of the Optimism Covariance penalty:

$$\begin{aligned} N(\text{Err}_{\text{in}} - \overline{\text{err}}) &= (A_2 + B + C + D_2 + E + F_2) - (A_1 + B + C + D_1 + E + F_1) \\ &= (A_2 - A_1) + (D_2 - D_1) + (F_2 - F_1) \end{aligned} \tag{103}$$

After taking expectations, terms D_1, D_2, and F_2 equal 0 due to independence and standard definitions ($\mathbb{E}(y_i) = f(x_i)$). The term $\mathbb{E}(F_1)$ evaluates to $-2\sum_i \text{cov}(y_i, \hat{y}_i)$. Expanding the covariance for a linear smoother $\hat{y} = X(X^TX)^{-1}X^Ty$:

$$\text{cov}(\hat{y}, y) = (X(X^TX)^{-1}X^T)\text{cov}(y, y) \tag{104}$$

Summing the diagonal elements with $\text{cov}(y, y) = \sigma_\epsilon^2$:

$$\sum_{i=1}^{N} \text{cov}(\hat{y}_i, y_i) = \text{trace}(X(X^TX)^{-1}X^T)\sigma_\epsilon^2 = \text{trace}(I_d)\sigma_\epsilon^2 = d\sigma_\epsilon^2 \tag{105}$$

Similarly, for a smoothing matrix S_λ:

$$\sum_{i=1}^{N} \text{cov}(S_\lambda y_i, y_i) = \text{trace}(S_\lambda) \tag{106}$$

4.7 Information Criteria and Generalized Cross-Validation

Akaike Information Criterion (AIC) for logistic regression:

$$AIC = -\frac{2}{N}\text{loglik} + 2\frac{d}{N} \tag{107}$$

Gaussian AIC with variance σ_ϵ^2:

$$AIC(\lambda) = \overline{err(\lambda)} + 2\frac{d(\lambda)}{N}\sigma_\epsilon^2 \tag{108}$$

The effective number of parameters for a linear model is defined as $d(S) = \text{trace}(S)$.

Bayesian Information Criterion (BIC) approximates the integral via Laplace method:

$$\log P(Z \mid \mathcal{M}_m) = \text{loglik} - \frac{d_m}{2} \log N + O(1) \tag{109}$$

$$BIC_m = -2\text{loglik}_m + (\log N) \cdot d_m \tag{110}$$

BIC compares model posterior probabilities using:

$$\frac{e^{\frac{1}{2} \cdot BIC_m}}{\sum_{l=1}^{M} e^{\frac{1}{2} \cdot BIC_l}} \tag{111}$$

Generalized cross-validation (GCV) approximates linear methods $\hat{y} = Sy$:

$$\frac{1}{N} \sum_{i=1}^{N} [y_i - \hat{f}^{-i}(x_i)]^2 = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{y_i - \hat{f}(x_i)}{1 - S_{ii}} \right]^2 \tag{112}$$

The GCV formal approximation uses the trace:

$$GCV = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{y_i - \hat{f}(x_i)}{1 - \text{trace}(S)/N} \right]^2 \tag{113}$$

4.8 Bootstrap

Bootstrap samples elements with replacement. The expected probability of not selecting a sample is:

$$(1 - \frac{1}{N})^N \approx e^{-1} = 0.368 \tag{114}$$

This yields the .632 bootstrap error estimate:

$$err = 0.632 \cdot e_{test} + 0.368 \cdot e_{train} \tag{115}$$

4.9 Confusion Matrix and Metrics

Performance metrics derive from True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN):

- Accuracy (celková správnost):

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

- Error (chyba):

$$Err = \frac{FP + FN}{TP + TN + FP + FN}$$

- Precision (přesnost):

$$Prec = \frac{TP}{TP + FP}$$

- Recall, Sensitivity (úplnost, sensitivita):

$$Rec = \frac{TP}{TP + FN}$$

- Specificity (specificita):

$$Specificity = \frac{TN}{TN + FP}$$

- F measure (F míra):

$$F = \frac{2 \cdot Prec \cdot Rec}{Prec + Rec} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

- TP rate:

$$\frac{TP}{TP + FN}$$

- FP rate (1 - Specificity):

$$\frac{FP}{FP + TN}$$

Explained variance R^2 evaluates regression effectiveness:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \overline{y}_i)^2} \tag{116}$$

To account for class imbalance or explicit operational costs, a non-negative loss matrix L forces the Gini index to represent differential costs:

$$Gini(m) = \sum_{k \neq k'} L_{kk'} \hat{p}_{mk} \hat{p}_{mk'} \tag{117}$$

Models apply classification in leaves following:

$$k(m) = \arg\min_{k} \sum_{l} L_{lk} \hat{p}_{ml} \tag{118}$$

5 Machine Learning: Additive Models, Trees, and Related Methods

5.1 Overview of Methods

The document covers several methods and algorithms related to decision trees and additive models. These include:

- Generalized Additive Models (GAM), noted as an advanced topic.
- **CART!** Classification and Regression Trees.
- **!** Cost sensitive pruning utilizing α penalties.
- PRIM (Patient Rule Induction Method), included as a note.
- MARS (Multivariate Additive Regression Splines), noted as an advanced topic.

5.2 Generalized Additive Models (GAM)

Generalized additive models are automatic flexible statistical methods used to identify and characterize nonlinear regression effects. The model takes the following mathematical form.

$$\mathbb{E}(Y|X_1,\ldots,X_p) = \alpha + f_1(X_1) + \cdots + f_p(X_p) \tag{119}$$

In this formulation:

- Y represents the outcome variable.
- X_j represents the predictors.
- f_j represents unspecified smooth functions.
- The smoothers used can be a cubic smoothing spline, local polynomial regression, or a kernel smoother.
- All p functions are estimated simultaneously.

5.3 GAM for Non-Gaussian Distributions $\times$

This topic is marked with a $\times$, indicating it may be excluded or supplementary. For a two-class classification using 0-1 encoding, the conditional mean is denoted as $\mu(X) = \Pr(Y = 1|X)$. Standard logistic regression is recalled as follows.

$$\log\left(\frac{\mu(X)}{1-\mu(X)}\right) = \alpha + \beta_1 X_1 + \cdots + \beta_p X_p \tag{120}$$

The additive logistic regression model replaces the linear terms with smoothers.

$$\log\left(\frac{\mu(X)}{1-\mu(X)}\right) = \alpha + f_1(X_1) + \cdots + f_p(X_p) \tag{121}$$

More generally, the conditional mean $\mu(X)$ of a response Y relates to an additive function of the predictors via a link function g.

$$g[\mu(X)] = \alpha + f_1(X_1) + \cdots + f_p(X_p) \tag{122}$$

Examples of classical link functions include:

- $g(\mu) = \mu$: The identity link, used for linear and additive models for Gaussian response data.
- $g(\mu) = \text{logit}(\mu)$: Used for logistic regression.
- $g(\mu) = \text{probit}(\mu)$: The probit link function for binomial probabilities, which is the inverse of the Gaussian cumulative distribution function, $\text{probit}(\mu) = \Phi^{-1}(\mu)$.
- $g(\mu) = \log(\mu)$: Used for log-linear or log-additive models for Poisson count data.

5.4 Models with Feature Interactions

Categorical variables are typically treated as identifiers, utilizing 0-1 or -1,1 encoding. In logistic regression, this leads to a constant β_j fitted for the variable. The slope β_{-j} of other variables does not depend on the identifier.

To allow for different slopes or shapes of a feature Z based on a qualitative variable V, an interaction term between the two features is required.

$$g(\mu) = f(X) + g_k(Z) \tag{123}$$

Generally, a function $g_{ZW}(Z, W)$ of two or more features can be added.

$$g(\mu) = f(X) + g_{ZW}(Z, W) \tag{124}$$

Semiparametric models combine linear regression for some predictors with a general function f for others.

$$g(\mu) = X^T\beta + \alpha_k + f(Z) \tag{125}$$

Note that logit, probit, log, gamma, and negative-binomial distributions belong to an exponential family, giving them desirable properties for fitting.

5.5 Algorithm: Generalized Additive Model Fitting

The backfitting algorithm is used to fit additive models given inputs $\mathbf{X}$ and target $\mathbf{y}$.

1. Initialize $\hat{\alpha} = \frac{1}{N}\sum_{i=1}^{N} y_i$ and set $\hat{f}_j \equiv 0$ for all i, j.
2. Repeat for $j = 1, 2, \ldots, p, \ldots, 1, 2, \ldots$.
3. Apply the smoother $\mathcal{S}_j$ (such as a smoothing spline with predefined degrees of freedom) to the partial residuals:

$$\hat{f}_j \leftarrow \mathcal{S}_j \left[\left\{ y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k(x_{ik}) \right\}_{i=1}^{N} \right]$$

4. Center the function to maintain a zero mean, as the constant is absorbed by $\hat{\alpha}$:

$$\hat{f}_j \leftarrow \hat{f}_j - \frac{1}{N}\sum_{i=1}^{N} \hat{f}_j(x_{ij})$$

5. Repeat the process until the functions $\hat{f}_j$ change less than a prespecified threshold.

Re-normalization is recommended during this procedure to prevent rounding errors.

5.6 Algorithm: Additive Logistic Regression ×

This topic is marked with a ×. It fits an additive model to targets z_j with weights w_i using a weighted backfitting algorithm. The inputs are $\mathbf{X}$ and $\mathbf{y}$ in 0-1 encoding.

1. Initialize $\hat{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$, set $\hat{\alpha} = \log\left(\frac{\hat{y}}{1-\hat{y}}\right)$, and set $\hat{f}_j \equiv 0$ for all j.
2. Repeat for $j = 1, 2, \ldots, p, \ldots, 1, 2, \ldots$.
3. For $i = 1, \ldots, N$:
   - Calculate $\hat{\eta}_i \leftarrow \hat{\alpha} + \sum_k \hat{f}_k(x_{ik})$.
   - Calculate probabilities $\hat{p}_i \leftarrow \frac{1}{1+\exp(-\hat{\eta}_i)}$.
   - Update weights $w_i \leftarrow \hat{p}_i(1-\hat{p}_i)$.
4. Construct the working target variable z_i:

$$z_i \leftarrow \hat{\eta}_i + \frac{y_i - \hat{p}_i}{\hat{p}_i(1-\hat{p}_i)}$$

5. Update the smoothers using the weighted targets:

$$\hat{f}_j \leftarrow \mathcal{S}_j^{\text{weighted}}\left[\left\{z_i - \hat{\alpha} - \sum_{k\neq j}\hat{f}_k(x_{ik})\right\}_{i=1}^{N}, \{w_i\}_{i=1}^{N}\right]$$

6. Repeat until the functions change less than a prespecified threshold.

5.7 Decision Trees: Principles and Construction

A decision tree for a given goal attribute G is defined as a rooted tree where the root and inner nodes are labeled by attributes. For each possible value of the attribute, an outgoing edge originates from the node. The leaves are labeled with the predicted goal class $g \in G$, assuming other attributes have the values indicated on the path from the root. Attributes that are not present on the path from the root to the leaf are considered irrelevant.

The core idea of tree construction involves:

- Selecting an attribute to create a node, then splitting the data according to the value of that attribute.
- Constructing a subtree based on the corresponding data subset for each attribute value.
- Stopping the recursion if there is a unique value of the goal G in the data, or if no attributes remain to split.
- In the stopping case, a leaf is created and labeled with the most common class $g \in G$.

5.8 Impurity Measures: Entropy and Gini Index

Entropy represents the uncertainty or negative information of an attribute A. It is designed to be zero for pure data (only one value of the attribute) and reaches its maximum for a uniform distribution of values (providing no information). The base of the logarithm is usually e, but sometimes 2. The mathematical definition of entropy for a probability distribution is given below.

$$H([p_1, \ldots, p_k]) = -\sum_{i=1}^{k} p_i \log p_i \tag{126}$$

For a goal attribute G, the entropy over a dataset is calculated using the ratios p_g of occurrences of $G = g$.

$$H_G(\text{data}) = \sum_{g \in G} -\frac{|\text{data}_{G=g}|}{|\text{data}|} \cdot \log_2 \left(\frac{|\text{data}_{G=g}|}{|\text{data}|} \right) = \sum_{g=1}^{|G|} -p_g \cdot \log_2 p_g \tag{127}$$

An alternative to entropy is the Gini index.

$$\text{Gini} = 1 - \sum_i (p_i)^2 \tag{128}$$

5.9 Information Gain

Information Gain measures the reduction in entropy when splitting data by an attribute X_j.

$$\text{Gain}(\text{data}, X_j) = H_G(\text{data}) - \sum_{x_j \in X_j} \frac{|\text{data}_{X_j=x_j}|}{|\text{data}|} H_G(\text{data}_{X_j=x_j}) \tag{129}$$

Maximizing the Information Gain is mathematically equivalent to minimizing the weighted entropy of the subsets after the split.

$$\arg\min_{X_j} \sum_{x_j \in X_j} \frac{|\text{data}_{X_j=x_j}|}{|\text{data}|} \sum_{g \in G} -\frac{|\text{data}_{G=g \wedge X_j=x_j}|}{|\text{data}_{X_j=x_j}|} \cdot \log_2 \left(\frac{|\text{data}_{G=g \wedge X_j=x_j}|}{|\text{data}_{X_j=x_j}|} \right) \tag{130}$$

5.10 Algorithm: ID3

The ID3 algorithm builds a decision tree by iteratively maximizing Information Gain.

1. Create a root node root.
2. If there are no attributes left in the set Attributes, label the root by the most frequent class g in the data and return g.
3. If all instances in the data share the same class g, label the root g and return g.
4. Select attribute X_j from Attributes that yields the maximal $\text{Gain}(\text{data}, X_j)$.
5. Label the root as X_j.
6. For each possible value x_j of X_j:
   - Add an outgoing edge from root labeled $X_j = x_j$.
   - Define $\text{data}_{X_j=x_j}$ as the subset of the data where $X_j = x_j$.
   - If $\text{data}_{X_j=x_j}$ is empty, add a leaf labeled by the most common class g in the dataset.
   - Else, attach a subtree generated by a recursive call: $\text{ID3}(\text{data}_{X_j=x_j}, G, \text{Attributes} \setminus \{X_j\})$.
7. Return root.

5.11 Categorical and Numerical Attributes Handling

The ID3 algorithm utilizes a penalized Information Gain criterion to avoid bias towards categorical attributes with many unique values.

$$\text{Gain}^*(X_i, \text{data}) = \frac{\text{Gain}(X_i, \text{data})}{H(X_i)} \tag{131}$$

For an identifier feature containing completely unique values, this formulation resolves to:

$$\text{Gain}^*(X_i, \text{data}) = \frac{\text{Gain}(X_i, \text{data})}{\log N} \tag{132}$$

For numerical attributes, algorithms typically require binary splits. The information gain is calculated for each possible split point, and multiple splits based on the same numerical attribute are permitted. The sklearn Decision Tree Classifier (which implements CART) does not natively support categorical attributes and utilizes only binary splits. Searching for an optimal binary split across categories has exponential complexity. A recommended heuristic is to sort the categories according to their goal class probabilities, allowing a linear time search for the split.

5.12 Handling Missing Values and Weights

Decision trees handle missing data well without needing to omit samples.

- If the data is not missing at random, the missingness itself should be treated as another discrete value of the attribute.
- If the data is missing at random, the algorithm splits the instance according to the data ratio following each branch.
- Predictions on leaves are then weighted and averaged.
- When using weighted samples, a weighted information gain is utilized to select attributes.

5.13 Decision Tree Pruning Techniques

To prevent overfitting, unnecessary nodes are removed from the tree through pruning.

- **Prepruning:** Pruning executed during tree construction. This is risky because it might prematurely prune attributes combined by functions like XOR, which individually have near-zero information gain.
- **Postpruning:** The usual approach, where the full tree is built and pruned afterwards.
- **Subtree Replacement:** A selected subtree is replaced by a leaf. This increases the training error but may decrease the validation error. Subtrees are iteratively pruned if doing so does not increase validation error.
- **Subtree Raising:** An inner node is removed, and data samples are reassigned to the remaining branches. Used in C4.5, it is computationally expensive and typically evaluated only for the most frequent branch.
- **Reduced Error Pruning:** Validates splits by retaining a subset of data for pruning. For each inner node, the algorithm compares the validation error with the node as a leaf against the validation error of the pruned subtree, selecting the option yielding the lower error.

5.14 CART! Classification and Regression Trees

Regression trees predict numeric targets using binary splits. Model trees apply a linear fit within the leaves but are less popular due to increased complexity and discontinuity. Standard CART predicts the average target value in its leaves and utilizes the decrease in square error loss to select the splitting attribute.

Given regions t_m (which correspond to the leaves), the tree predicts a constant c_m. The regression tree function is expressed as follows.

$$f(x) = \sum_{m=1}^{M} c_m I(x \in t_m) \tag{133}$$

The best estimate for the constant c_m inside region t_m is the average of the targets.

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in t_m} y_i \tag{134}$$

When constructing a single regression tree, the algorithm starts with all data in one region t_0. It selects the optimal attribute j and split value s by minimizing the sum of squared errors across the two resulting sub-regions t_1 and t_2.

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in t_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in t_2(j,s)} (y_i - c_2)^2 \right] \tag{135}$$

The inner minimums $\hat{c}_1$ and $\hat{c}_2$ resolve to the local region averages. The algorithm iterates recursively until the number of samples in a leaf reaches a minimum threshold n_0.

5.15 ! Minimal Cost-Complexity Pruning

Minimal cost-complexity pruning is a criterion based strictly on the training data, penalizing the size of the tree. The sum of squared errors for any node t (if treated as a leaf) is defined as follows.

$$R(t) = \sum_{x_i \in t} \left(y_i - \frac{1}{|x \in t|} \sum_{x_i \in t} y_i \right)^2 \tag{136}$$

The overall cost-complexity of a tree T applies a penalty α for its number of leaves $|T|$.

$$R_\alpha(T) = \sum_{t \in \text{leaves}(T)} R(t) + \alpha |T| \tag{137}$$

For a single isolated node t treated as a leaf, its cost complexity is evaluated as:

$$R_\alpha(t) = R(t) + \alpha \tag{138}$$

There exists an effective α, denoted $\alpha_{\text{eff}}(t)$, where the cost of the node equals the cost $R(T_t)$ of the entire subtree rooted at t.

$$\alpha_{\text{eff}}(t) = \frac{R(t) - R(T_t)}{|T_t| - 1} \tag{139}$$

The non-terminal node that exhibits the smallest value of α_{eff} is identified as the "weakest link" and is sequentially pruned.

5.15.1 Cross-Validation for α Selection

To select the optimal penalty α, a K-fold cross-validation procedure is utilized.

1. For each fold k:
   - Train the tree T using all data except fold k.
   - Build a nested sequence of pruned subtrees $T^k \supset T_1^k \supset T_2^k \cdots \supset T_{|T|}^k$ by always joining two leaves with the minimal increase in training error.
   - Use fold k to calculate the cross-validation error $R_\alpha(T^k)$ as a function of α.
2. Select the global optimal α by minimizing the aggregated error across all folds:

$$\alpha \leftarrow \arg\min_{\alpha} \sum_{k} R_\alpha(T^k)$$

3. Rebuild the full tree on the entire training dataset and return the specific subtree that corresponds to the optimal α.

5.16 Complexity Considerations

For a dataset containing N instances and p attributes, assuming a reasonably balanced CART tree with a depth of $O(\log N)$.

- At each depth level, each instance is evaluated exactly once across p attributes. Over $\log N$ levels, the total construction time requires $O(p \cdot N^2 \cdot \log N)$.
- Subtree replacement requires $O(N)$ operations.
- Subtree raising requires $O(N(\log N)^2)$ operations.
- The total naive tree construction complexity is bounded by $O(p \cdot N^2 \cdot \log N) + O(N(\log N)^2)$.
- By pre-sorting features and utilizing clever indexing, the overall tree construction complexity is reduced to $O(p \cdot N \cdot \log N) + O(N(\log N)^2)$.

6 Machine Learning: Ensemble Methods

6.1 Learning Method Comparison

Various learning methods are compared based on their characteristics, including Neural Networks (before deep learning), Support Vector Machines (SVM - representing logistic regression with a non-linear transformation that worsens scalability), Trees, MARS, and k-Nearest Neighbors (k-NN). The fundamental characteristics for algorithmic comparison include:

- Natural handling of data of mixed types
- Handling of missing values
- Robustness to outliers in input space
- Insensitivity to monotone transformations of inputs
- Computational scalability (large N)
- Ability to deal with irrelevant inputs
- Ability to extract linear combinations of features
- Interpretability
- Predictive power

6.2 Ensemble Methods Overview

To improve the predictive power of a single decision tree, results are combined from a bag of trees. Common ensemble methods include:

- Random Forest (+ Bagging)
- Boosting (AdaBoost for classification, Gradient Boosting for regression and classification)
- Stacking
- MARS (equivalent to the earth package)

6.3 Bootstrap

The bootstrap process involves selecting elements with replacement. Given N data samples, we select N samples with replacement to form a bootstrap sample Z^{*b}. Some samples are selected more than once, while some are not selected at all. The unselected samples are used for testing.

The probability of not-selecting a specific sample is:

$$\left(1 - \frac{1}{N}\right)^N \approx e^{-1} \approx 0.368 \tag{140}$$

The selected samples are used to learn a model (usually a tree), and the unselected samples are used for the Out-Of-Bag (OOB) error computation. B training sets $Z^{*1}, \ldots, Z^{*B}$, each of size N, are drawn with replacement from the original dataset $Z = (z_1, z_2, \ldots, z_N)$. The quantity of interest $S(Z)$ is computed from each bootstrap training set, giving values $S(Z^{*1}), \ldots, S(Z^{*B})$ which are used to assess the statistical accuracy of $S(Z)$.

6.4 Random Forest for Regression or Classification

(!) Crucial: At each split, select m variables at random from p total variables.

Random Forest Procedure:

1. Given X, y training data, run for $b = 1, 2, \ldots, B$:
   (a) Draw a bootstrap sample Z^* of size N.
   (b) Grow a random forest tree T_b using the sample Z^*.
   (c) **(!)** Select m variables at random from p variables.
   (d) Pick the best variable and split-point among the selected m variables.
   (e) Split the node into two child nodes.
   (f) Repeat until the minimum node size $n_{\min}$ is reached (usually fully grown with no pruning).
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

- **Regression:**
$$\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$$
- **Classification:** Let $\hat{C}_b(x)$ be the class prediction of the b-th random-forest tree.
$$\hat{C}_{\text{rf}}^B(x) = \text{majority vote}\{\hat{C}_b(x)\}_1^B$$

6.5 Bagging (Bootstrap Aggregating)

Bagging is essentially a Random Forest where all predictors are used at each split, meaning $m = p$. It applies to both regression and classification. Given training data $Z = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, the bagging predictor is:
$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x) \tag{141}$$
Bagging averages over different trees to add smoothness in predicted values (addressing the constant prediction at tree leaves).

Bagging for Classification: For each bootstrap sample $b = 1, 2, \ldots, B$, fit a model giving prediction $\hat{f}^{*b}(x)$. To predict:

- Predict probabilities of classes and find the class with the highest predicted probability over bootstrap samples:
$$\hat{G}(x) = \arg\max_k \sum_{b=1}^{B} \hat{f}^{*b}(x)$$
- Or predict class via majority vote:
$$\hat{G}_{\text{bag}}(x) = \text{majority vote}\{\hat{G}^{*b}(x)\}_{b=1}^B$$

6.6 Behind Random Forest and Variance Reduction

The variance of the random forest estimate is defined as:

$$\text{Var}(\hat{f}_{\text{rf}}^B(x)) = \mathbb{E}(\hat{f}(x) - \mathbb{E}\hat{f}(x))^2 \tag{142}$$

- For i.i.d. data variables with independent features, each with variance σ^2, the variance is $\frac{1}{B}\sigma^2$.
- For identically distributed data, each with variance σ^2 with a positive pairwise correlation ρ, the variance is:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

Bagging addresses the second term of the variance formula. Random forests address the first term by de-correlating the trees. Before each split, random forest selects $m \leq p$ variables as candidates for splitting. Typical values are $m = \sqrt{p}$ for regression (sometimes as low as 1), and $m = p/3$ for classification.

6.7 Out of Bag Error (OOB)

Definition (Out of bag error): For each observation $z_i = (x_i, y_i)$, construct a random forest predictor by averaging only those trees corresponding to bootstrap samples in which z_i did not appear. An OOB error estimate is almost identical to that obtained by N-fold cross-validation. Unlike other nonlinear estimators, random forests can be fit in one continuous sequence.

6.8 Feature Importance: Mean Decrease in Impurity

The Variable Importance of a predictor X_l in a single tree T is:

$$\mathcal{I}_l^2(T) = \sum_{t=1}^{J} \hat{i}_t^2 \cdot I(v(t) = l) \tag{143}$$

Where $\hat{i}_t^2$ is the Gini or RSS improvement of the predictor in the internal node t, and $v(t)$ is the split variable at node t.

- Gini impurity before and after the split: $\hat{p}_k(t)(1 - \hat{p}_k(t))$.
- Weighted by the probability of reaching the node t.

For a set of M trees, the variable importance is averaged over all trees:

$$\mathcal{I}_l^2 = \frac{1}{M}\sum_{m=1}^{M} \mathcal{I}_l^2(T_m) \tag{144}$$

6.9 Feature Importance based on Feature Permutation

Randomization voids the effect of a variable to empirically measure its importance. **Procedure:**

1. For $b = 1, 2, \ldots, B$:
   (a) Draw a bootstrap sample Z^* of size N.
   (b) Grow a random forest tree T_b.
   (c) Calculate accuracy on OOB samples.
   (d) For $j = 1, 2, \ldots, p$:
      i. Permute the values for the j-th variable randomly in the OOB samples.
      ii. Calculate the decrease in the accuracy.
2. Output the average accuracy gain for each $j = 1, 2, \ldots, p$.

6.10 Proximity Plot

Procedure:

1. For $b = 1, 2, \ldots, B$:
   (a) Draw a bootstrap sample Z^* of size N.
   (b) Grow a random forest tree T_b.
   (c) Calculate prediction accuracy on OOB samples.
   (d) For any pair of OOB samples sharing the same leaf: increase their proximity score by one.

Samples in the 'star center' of proximity plots are close to the decision boundary, while distinct samples come from pure regions.

6.11 Overfitting in Random Forests

The random forest cannot overfit the limit distribution:

$$\hat{f}_{\text{rf}}(x) = \mathbb{E}_\Theta T(x;\Theta) = \lim_{B\to\infty} \hat{f}_{\text{rf}}^B(x) \tag{145}$$

However, the limit distribution itself (the average of fully grown trees) may overfit the training data. A small number of relevant variables combined with many irrelevant ones hurts the random forest approach, though it remains robust with a higher number of relevant variables.

6.12 Boosting

(!) Use a weak classifier as a decision stump (a decision tree with depth = 1).
AdaBoost.M1 Classifier:

1. Initialize the observation weights $w_i \leftarrow \frac{1}{N}$.
2. For $m = 1, 2, \ldots, M$:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute the error:
   $$err_m \leftarrow \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}$$
   (c) Compute the classifier weight:
   $$\alpha_m \leftarrow \log \frac{(1 - err_m)}{err_m}$$
   (d) Update and normalize the observation weights:
   $$w_i \leftarrow w_i \cdot e^{I(y_i \neq G_m(x_i))\cdot \alpha_m}$$
3. Output the final classifier (using encoding $Y \in \{-1, 1\}$):
   $$G(x) = \text{sign}\left[\sum_{m=1}^M \alpha_m G_m(x)\right]$$

The weighted sample error is measured as $\overline{err} = \frac{1}{N}\sum_{i=1}^N I(y_i \neq G(x_i))$.

6.13 Additive Model

Encoding the binary goal by $Y \in \{-1, +1\}$, boosting fits an additive model:

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m) \tag{146}$$

Where β_m are the expansion coefficients and $b(x; \gamma_m) \in \mathbb{R}$ are simple functions of the multivariate argument x characterized by parameters γ. Forward stagewise additive modeling sequentially adds one new basis function without adjusting the parameters and coefficients of previously fitted functions. For squared-error loss, $L(y, f(x)) = (y - f(x))^2$:

$$L(y_i, f_{m-1}(x) + \beta_m b(x_i; \gamma_m)) = (y_i - f_{m-1}(x) - \beta_m b(x; \gamma_m))^2 = (r_{im} - \beta_m b(x_i; \gamma_m))^2 \tag{147}$$

6.14 Exponential Loss and AdaBoost

Using $Y \in \{-1, 1\}$ and the exponential loss:

$$L(y, f(x)) = e^{-yf(x)} \tag{148}$$

To minimize the loss at iteration m:

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} e^{-y_i(f_{m-1}(x_i) + \beta G(x_i))} = \arg\min_{\beta, G} \sum_{i=1}^{N} w_i^{(m)} e^{-y_i \beta G(x_i)} \tag{149}$$

Where the weight $w_i^{(m)} = e^{-y_i f_{m-1}(x_i)}$ depends on $f_{m-1}(x_i)$ and changes with each iteration m, but does not depend on β or $G(x)$.

Solving for G_m and β:

$$(\beta_m, G_m) = \arg\min_{\beta, G} \left[(e^{\beta} - e^{-\beta}) \cdot \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i)) + e^{-\beta} \cdot \sum_{i=1}^{N} w_i^{(m)} \right] \tag{150}$$

For any $\beta > 0$, the optimal $G_m(x; \gamma)$ is:

$$G_m = \arg\min_{\gamma} \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i; \gamma)) \tag{151}$$

The optimal β_m is:

$$\beta_m = \frac{1}{2} \log \frac{1 - err_m}{err_m} \tag{152}$$

The function approximation is updated as $f_m(x) = f_{m-1}(x) + \beta_m G_m(x)$, which updates the weights for the next iteration:

$$w_i^{m+1} = w_i^m \cdot e^{-\beta_m y_i G_m(x_i)} \tag{153}$$

Using the identity $-y_i G_m(x_i) = 2 \cdot I(y_i \neq G_m(x_i)) - 1$, the weights can be equivalently expressed as:

$$w_i^{m+1} = w_i^m \cdot e^{2\beta_m I(y_i \neq G_m(x_i))} \cdot e^{-\beta_m} \tag{154}$$

6.15 Why Exponential Loss? [× Skipped in Lecture]

The population minimizer for exponential loss is:

$$f^*(x) = \arg\min_{f(x)} \mathbb{E}_{Y|x}(e^{-Yf(x)}) = \frac{1}{2} \log \frac{\Pr(Y = 1|x)}{\Pr(Y = -1|x)} \tag{155}$$

Which yields the probability:

$$\Pr(Y = 1|x) = \frac{1}{1 + e^{-2f^*(x)}} \tag{156}$$

This same function minimizes deviance (cross-entropy/binomial negative log-likelihood).

6.16 Forward Stagewise Additive Modeling

Procedure:

1. Initialize $f_0 \leftarrow 0$.
2. For $m = 1, 2, \ldots, M$:
   (a) Compute:
   $$(\beta_m, \gamma_m) \leftarrow \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$$
   (b) Set:
   $$f_m(x) \leftarrow f_{m-1}(x) + \beta_m b(x; \gamma_m)$$

6.17 Gradient Tree Boosting Algorithm

Procedure:

1. Initialize the model with a constant value:
   $$f_0(x) \leftarrow \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$$
2. For $m = 1, 2, \ldots, M$:
   (a) For $i = 1, 2, \ldots, N$, compute the pseudo-residuals. For square error loss, this evaluates to:
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)} = y_i - f_{m-1}(x_i)$$
   (b) Fit a regression tree to the target r_{im}, yielding terminal regions $\{R_{jm}\}_{j=1}^{J_m}$.
   (c) For $j = 1, 2, \ldots, J_m$, compute the optimal terminal node predictions:
   $$\gamma_{jm} = \arg\min_{\gamma} \sum_{i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$
   (d) Update the model:
   $$f_m(x) \leftarrow f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$$
3. Output $\hat{f}(x) = f_M(x)$.

6.18 Regularization: Shrinkage and Subsampling

- **Shrinkage:** Slows down the learning rate to avoid overfitting by adding a shrinkage parameter $0 < \nu < 1$ into the model update step:
$$f_m(x) \leftarrow f_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm}) \tag{157}$$
- **Subsampling:** In each step, select a fraction without replacement (e.g., $\eta = \frac{1}{2}$) of the data samples to train the subsequent base learner.

6.19 Stacking

Stacking learns a simple aggregate model (such as a linear regression) over a set of distinct base models to form a final ensemble. Assume base predictions $\hat{f}_1(x), \hat{f}_2(x), \ldots, \hat{f}_M(x)$. To avoid giving unfairly high weight to highly complex models, cross-validated predictions $\hat{f}_m^{-i}(x)$ (trained without the i-th example) are used to fit the stacking weights $w = (w_1, \ldots, w_M)$:

$$\hat{w}^{\text{st}} = \arg\min_w \sum_{i=1}^{N} \left[y_i - \sum_{m=1}^{M} w_m \hat{f}_m^{-i}(x) \right]^2 \tag{158}$$

Optimal stability is achieved by restricting the weights to be nonnegative and to sum to 1. The final stacked prediction is:

$$\hat{f}^{\text{st}}(x) = \sum_{m=1}^{M} w_m^{\text{st}} \hat{f}_m(x) \tag{159}$$

7 Machine Learning: Bayesian Learning and EM Algorithm

7.1 Fundamental Concepts

- **Maximum a posteriori probability hypothesis (MAP):** The most probable hypothesis given the data.
- **Maximum likelihood hypothesis (ML):** The hypothesis that maximizes the likelihood of the data.
- **Bayesian optimal prediction (Bayes Rate):** The weighted average of predictions of all hypotheses.
- **Complexity Penalty:** Represents the prior distribution or preference on parameters, favoring simpler models.

7.2 Maximum Aposteriori Probability Hypothesis (MAP)

For a hypothesis h_i and data observation $B = c$, Bayes' formula gives:

$$P(h_i|B = c) = \frac{P(B = c|h_i) \cdot P(h_i)}{\sum_j P(B = c|h_j) \cdot P(h_j)} = \frac{P(B = c|h_i) \cdot P(h_i)}{P(B = c)} \tag{160}$$

The MAP hypothesis maximizes the posterior probability:

$$h_{\text{MAP}} = \arg\max_i P(h_i|B = c) = \arg\max_i P(B = c|h_i) \cdot P(h_i) \tag{161}$$

7.3 Bayesian Optimal Prediction

Bayesian optimal prediction is the weighted average of predictions of all hypotheses:

$$P(N = c|\text{data}) = \sum_j P(N = c|h_j, \text{data}) \cdot P(h_j|\text{data}) \tag{162}$$

Assuming independent and identically distributed (i.i.d.) data where hypotheses fully describe data behavior:

$$P(N = c|\text{data}) = \sum_j P(N = c|h_j) \cdot P(h_j|\text{data}) \tag{163}$$

The error of the Bayesian optimal prediction is called the **Bayes error rate**, analogous to irreducible error from the bias-variance decomposition.

7.4 Maximum Likelihood Estimate (ML)

When prior probabilities of hypotheses are unknown or assumed equal, MAP reduces to the ML estimate:

$$h_{\text{ML}} = \arg\max_i P(\text{data}|h_i) \tag{164}$$

7.4.1 ML Estimate for a Continuous Parameter

Given a parameter $\theta \in [0, 1]$, the probability of observing c successes (e.g., cherry candies) and l failures (e.g., lime candies) is:

$$P(\text{data}|h_\theta) = \theta^c \cdot (1 - \theta)^l \tag{165}$$

Taking the logarithm yields the log-likelihood:

$$l(h_\theta; \text{data}) = c \cdot \log_2 \theta + l \cdot \log_2(1 - \theta) \tag{166}$$

Setting the derivative to zero finds the maximum:

$$\frac{\partial l(h_\theta; \text{data})}{\partial \theta} = \frac{c}{\theta} - \frac{l}{1-\theta} = 0 \tag{167}$$

$$\theta = \frac{c}{c+l} \tag{168}$$

7.4.2 ML Estimate of Gaussian Distribution Parameters

Assuming a feature X has a Gaussian distribution with unknown parameters μ and σ, the hypothesis is:

$$h_{\mu,\sigma} = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{169}$$

Given observations $x_1, \ldots, x_N$, the log-likelihood is:

$$LL = \sum_{j=1}^{N} \log\left(\frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x_j-\mu)^2}{2\sigma^2}}\right) = N \cdot \left(\log \frac{1}{\sqrt{2\pi}\sigma}\right) - \sum_{j=1}^{N} \frac{(x_j-\mu)^2}{2\sigma^2} \tag{170}$$

7.4.3 Linear Gaussian Distribution

Given a random variable X and a target variable Y with linear Gaussian distribution:

$$\mu = b \cdot x + b_0 \tag{171}$$

$$\Pr(Y|X=x) = \mathcal{N}(b \cdot x + b_0; \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(y-(b\cdot x+b_0))^2}{2\sigma^2}} \tag{172}$$

For observations $\{\langle x_1, y_1\rangle, \ldots, \langle x_N, y_N\rangle\}$, the ML estimates for b and b_0 minimize the sum of squared errors:

$$\arg\min_{b,b_0} \sum_{i=1}^{N} (y_i - (b \cdot x_i + b_0))^2 \tag{173}$$

7.5 Naive Bayes Model and Bayes Classifier

The Naive Bayes Classifier assumes independent features given the class variable. The most probable class for a new observation of features is predicted as:

$$\arg\max_{c_i} P(x_j|c_i) \cdot P(c_i) \tag{174}$$

The **Bayes factor** for updating class ratio sequentially after observing x_p:

$$\frac{P(c_i|x_1, \ldots, x_p)}{P(c_k|x_1, \ldots, x_p)} = \frac{P(c_i)}{P(c_k)} \cdot \frac{P(x_1|c_i)}{P(x_1|c_k)} \cdots \frac{P(x_p|c_i)}{P(x_p|c_k)} \tag{175}$$

7.6 Parameter Estimate as a Probability Distribution

- For binary features, the **Beta Function** is used, where $a-1$ is the number of positive examples and $b-1$ is the number of negative examples:

$$\text{beta}[a,b](\theta) = \alpha\theta^{a-1}(1-\theta)^{b-1}$$

- For categorical features, **Dirichlet priors** and multinomial distributions are used.
- For Gaussian distributions, μ has a Gaussian prior and $1/\sigma$ has a Gamma prior.

7.7 Bayesian Methods and Smoothing

Specifying a sampling model $P(Z|\theta)$ and a prior distribution $P(\theta)$, the posterior distribution is computed as:

$$P(\theta|Z) = \frac{P(Z|\theta) \cdot P(\theta)}{\int P(Z|\theta) \cdot P(\theta) d\theta} \tag{176}$$

Providing the Bayesian optimal predictive distribution:

$$P(z^{\text{new}}|Z) = \int P(z^{\text{new}}|\theta) \cdot P(\theta|Z) d\theta \tag{177}$$

7.7.1 Bayesian Smoothing Example

For training data $Z = \{z_1, \ldots, z_N\}$ with $z_i = (x_i, y_i)$, fit a cubic spline using B-spline basis functions $h_j(x)$ for $j = 1, \ldots, 7$:

$$\mu(x) = \sum_{j=1}^{7} \beta_j h_j(x) \tag{178}$$

Let H be the $N \times 7$ matrix $h_j(x_i)$. The Residual Sum of Squares (RSS) estimate is:

$$\hat{\beta} = (H^T H)^{-1} H^T y \tag{179}$$

Assuming fixed variance σ^2 and specifying a prior on $\beta \sim \mathcal{N}(0, \tau\Sigma)$:

$$\mathbb{E}(\beta|Z) = \left(H^T H + \frac{\sigma^2}{\tau} \Sigma^{-1} \right)^{-1} H^T y \tag{180}$$

$$\mathbb{E}(\mu(x)|Z) = h(x)^T \left(H^T H + \frac{\sigma^2}{\tau} \Sigma^{-1} \right)^{-1} H^T y \tag{181}$$

7.8 MAP and Penalized Methods

The complexity penalty (e.g., Lasso, Ridge) acts as a prior distribution $P(h)$, giving higher probability to simpler models:

$$h_{\text{MAP}} = \arg\min_h [-\log_2 P(\text{data}|h) - \log_2 P(h)] \tag{182}$$

This is equivalent to minimizing RSS + complexity penalty for Gaussian models, or maximizing log lik − complexity penalty for categorical models.

7.9 Expectation Maximization Algorithm (EM Algorithm)

The EM algorithm estimates a maximum likelihood model for data with missing values. Let Z be observed data, Z^m be missing data, and $T = (Z, Z^m)$ be the complete data. $l(\theta; Z)$ is the log-likelihood of the model, and $l_0(\theta; T)$ is the log-likelihood of the complete data.

7.9.1 EM as a Maximization-Maximization Procedure

Consider the function F defined for parameters θ' and latent distribution $\hat{P}$:

$$F(\theta', \hat{P}) = \mathbb{E}_{\hat{P}}[l_0(\theta'; (Z, Z^m))] - \mathbb{E}_{\hat{P}}[\log \hat{P}(Z^m)] \tag{183}$$

Taking the expectation with respect to $T|Z$ governed by parameter θ gives:

$$l(\theta'; Z) = \mathbb{E}[l_0(\theta'; T)|\theta, Z] - \mathbb{E}[l_1(\theta'; Z^m|Z)|\theta, Z] \equiv Q(\theta', \theta) - R(\theta', \theta) \tag{184}$$

7.9.2 The EM Algorithm Workflow

1. Start with an initial guess $\hat{\theta}^{(0)}$ (usually close to uniform distribution).
2. **Expectation Step (E step):** At the j-th step, compute Q as a function of the dummy argument θ':
$$Q(\theta', \hat{\theta}^{(j)}) = \mathbb{E}(l_0(\theta'; T)|Z, \hat{\theta}^{(j)})$$
3. **Maximization Step (M step):** Determine the new estimate $\hat{\theta}^{(j+1)}$ as the maximizer of Q:
$$\hat{\theta}^{(j+1)} = \arg\max_{\theta'} Q(\theta', \hat{\theta}^{(j)})$$
4. Repeat E and M steps until convergence.

Note: Generalized EM (GEM) algorithms only require finding $\hat{\theta}^{(j+1)}$ such that $Q(\hat{\theta}^{(j+1)}, \hat{\theta}^{(j)}) > Q(\hat{\theta}^{(j)}, \hat{\theta}^{(j)})$.

7.10 EM Learning of Mixture of K Gaussians !

Estimating a Gaussian Mixture Model (GMM) with latent class variable C (representing the cluster). Model parameters are $\pi_1, \ldots, \pi_K$, $\mu_1, \ldots, \mu_K$, $\Sigma_1, \ldots, \Sigma_K$ such that $\sum_{k=1}^{K} \pi_k = 1$.

- **E step (Expectation):** Compute weights of unobserved fill-ins for variable C:
$$p_{ik} = P(C_i = k|x_i) = \frac{\pi_k \phi_{\theta_k}(x_i)}{\sum_{l=1}^{K} \pi_l \phi_{\theta_l}(x_i)}$$
$$p_k = \sum_{i=1}^{N} p_{ik}$$
- **M step (Maximization):** Update mean, variance, and cluster prior:
$$\mu_k \leftarrow \sum_{i=1}^{N} \frac{p_{ik}}{p_k} x_i$$
$$\Sigma_k \leftarrow \sum_{i=1}^{N} \frac{p_{ik}}{p_k} (x_i - \mu_k)(x_i - \mu_k)^T$$
$$\pi_k \leftarrow \frac{p_k}{\sum_{l=1}^{K} p_l}$$

(Note: Bayesian Network example for EM is marked as "can be omitted")

7.11 Hierarchical Mixture of Experts

A hierarchical extension of naive Bayes modeled as a decision tree with soft, probabilistic splits.

- **Gating Network:** Represents non-terminal nodes controlling split probabilities. For children of the root:
$$g_j(x, \gamma_j) = \frac{e^{\gamma_j^T x}}{\sum_{k=1}^{K} e^{\gamma_k^T x}}$$
$$g_{l|j}(x, \gamma_{jl}) = \frac{e^{\gamma_{jl}^T x}}{\sum_{k=1}^{K} e^{\gamma_{jk}^T x}}$$
- **Expert Network:** Represents terminal nodes. For Gaussian linear regression models: $\theta_{jl} = (\beta_{jl}, \sigma_{jl}^2)$, $Y = \beta_{jl}^T x + \epsilon$. For linear logistic regression classification:
$$\Pr(Y = 1|x, \theta_{jl}) = \frac{1}{1 + e^{-\theta_{jl}^T x}}$$

7.12 Handling Missing Data

- **Missing Completely at Random (MCAR):** Probability that a value is missing is independent of both observed and unobserved values.
- **Missing at Random (MAR):** Probability that a value is missing depends only on the observed values.
- **Non-ignorable:** Neither MAR nor MCAR.

7.13 Patient Rule Induction Method (PRIM) / Bump Hunting

A rule induction method iteratively searching for regions with high target variable values.

1. Consider the whole space and all data. Set $\alpha = 0.05$ or 0.10.
2. Find feature X_j and its upper or lower boundary such that peeling away $\alpha \cdot 100\%$ observations yields the maximal mean of the remaining data.
3. Repeat peeling until fewer than 10 observations are left.
4. Enlarge the region in any direction that increases the mean value.
5. Select the number of regions by cross-validation. Denote the best region B_1.
6. Create a rule describing B_1 and remove all its data from the dataset.
7. Repeat the entire process to create B_2, continuing until the final condition is met.

7.14 Multivariate Adaptive Regression Splines (MARS)

A generalization of linear regression and CART to address the high variance and stepwise boundaries of decision trees.

- Uses reflected pairs of basis functions: $(x-t)_+$ and $(t-x)_+$, where $+$ denotes the non-negative part. The set of functions across all variables and data points is:

$$\mathcal{C} = \{(X_j - t)_+, (t - X_j)_+\}_{t \in \{x_{1,j},\ldots,x_{N,j}\}, j=1,\ldots,p}$$

- The model is an additive expansion:

$$f(X) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(X)$$

- **Basis Selection:** Start with $h_0 = 1$ in model $\mathcal{M}$. Consider the product of any member $h_l \in \mathcal{M}$ with any pair from $\mathcal{C}$. Iteratively select the term minimizing the training error RSS.
- **Model Pruning:** Remove functions to prevent overfitting by minimizing generalized cross-validation (GCV) for different numbers of parameters λ:

$$GCV(\lambda) = \frac{\sum_{i=1}^{N}(y_i - \hat{f}_\lambda(x_i))^2}{(1 - M(\lambda)/N)^2}$$

  where $M(\lambda)$ is the effective number of parameters: $M(\lambda) = r + 3K$ (r is the number of functions h_m, and K is the number of knots).

8 Machine Learning: Clustering

8.1 Unsupervised Learning Concepts

Unsupervised learning focuses on finding relations in data where there is no goal class (neither Y nor G).

- **Clustering / Segmentation**: Are the data organized in natural clusters? Techniques include k-means, hierarchical clustering, Expectation-Maximization (EM) algorithm for clustering, Dirichlet Process Mixture Models, and Spectral Clustering.
- **Association Rules**: Are there frequent combinations or implication relations? (e.g., Market Basket Analysis).
- **Principal Component Analysis (PCA)**: Linear algebra method finding k linear combinations of features minimizing reconstruction error (the first k principal components).
- **Independent Component Analysis (ICA)**: Finding independent components.
- **Self Organizing Maps (SOM)**.

8.2 K-Means Algorithm [!]

The K-Means algorithm partitions unlabeled training data into K clusters by assigning the same cluster label to nearby points.

Procedure: K-MEANS(X, K) Where X is the data and K is the number of clusters:

1. Select randomly K centers of clusters μ_k (either random data points or random points in the feature space).
2. **Repeat** until no change in assignment:
3. For each data record x_i, compute the assignment:

$$C(x_i) \leftarrow \arg\min_{k \in \{1,\dots,K\}} d(x_i, \mu_k)$$

4. For each cluster k, find new centers:

$$\mu_k = \sum_{x_i : C(x_i)=k} \frac{x_i}{|C(k)|}$$

Properties & Complexity:

- The t iterations of the K-means algorithm take $O(tKpN)$ time.
- Finding the global optimum is NP-hard.
- The result depends on initial values and may get stuck in a local minimum.
- It may not be robust to data sampling (e.g., bootstrap method may yield different cluster centers).
- Sensitive to outliers, as each record must belong to a cluster.

8.3 Distance Measures

The choice of distance measure $d(x_i, \mu_k)$ is key and application-dependent. Scaling of data is generally recommended.

- **Euclidean Distance**:
$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{p}(x_{ir} - x_{jr})^2}$$

- **Hamming (Manhattan) Distance**:
$$d(x_i, x_j) = \sum_{r=1}^{p}|x_{ir} - x_{jr}|$$

- **Overlap (Categorical Variables)**:
$$d(x_i, x_j) = \sum_{r=1}^{p} I(x_{ir} \neq x_{jr})$$

- **Cosine Similarity**:
$$s(x_i, x_j) = \frac{\sum_{r=1}^{p}(x_{ir} \cdot x_{jr})}{\sqrt{\sum_{r=1}^{p}(x_{jr} \cdot x_{jr}) \cdot \sum_{r=1}^{p}(x_{ir} \cdot x_{ir})}}$$

- **Cosine Distance**:
$$d(x_i, x_j) = 1 - s(x_i, x_j)$$

- **Correlation Proximity**:
$$\rho_{X,Y} = \text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

Weighting Attributes: Weights w_j for equally important attributes j are defined as:

$$w_j = \frac{1}{\hat{d}_j} \tag{185}$$

Where the empirical distance variance is:

$$\hat{d}_j = \frac{1}{N^2}\sum_{i_1=1}^{N}\sum_{i_2=1}^{N} d_j(x_{i_1}, x_{i_2}) = \frac{1}{N^2}\sum_{i_1=1}^{N}\sum_{i_2=1}^{N}(x_{i_1}[j] - x_{i_2}[j])^2 \tag{186}$$

Total distance is calculated as a weighted sum of attribute distances. Distances may also be specified directly by a symmetric matrix, 0 at the diagonal, fulfilling the triangle inequality:

$$d(x_i, x_l) \leq d(x_i, x_r) + d(x_r, x_l) \tag{187}$$

8.4 Cluster Evaluation Metrics

Within Cluster Variation:

$$W(C) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k} d(x_i, x_{i'}) \tag{188}$$

Total Cluster Variation: The sum of between-cluster variation $B(C)$ and within-cluster variation $W(C)$.

$$T(C) = \frac{1}{2}\sum_{i,i'=1}^{N} d(x_i, x_{i'}) = W(C) + B(C) \tag{189}$$

$$T(C) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\left(\sum_{C(i')=k} d(x_i, x_{i'})\right) + \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\left(\sum_{C(i')\neq k} d(x_i, x_{i'})\right) \tag{190}$$

8.5 GAP Function for Number of Clusters Selection [x]

The GAP statistic evaluates the difference between the actual within-cluster variation and the expected variation under a uniform distribution W_k^u.

$$G(K) = \log(W_k^u) - \log(W(k)) \tag{191}$$

The optimal number of clusters K^* is selected via:

$$K^* = \arg\min\{k \mid G(k) \geq G(k+1) - s'_{k+1}\} \tag{192}$$

Where $s'_k = s_k\sqrt{1+\frac{1}{20}}$ and s_k is the standard deviation of $\log(W_k)$.

8.6 Silhouette Analysis

For each data sample X_i, calculate:

- Average distance to other points in the same cluster C_i:

$$a(i) = \frac{1}{|C_i| - 1}\sum_{j\in C_i, i\neq j} d(i,j)$$

- Minimum average distance to points in other clusters C_k:

$$b(i) = \min_{k\neq i}\frac{1}{|C_k|}\sum_{j\in C_k} d(i,j)$$

Definition (Silhouette):

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \quad \text{if } |C_i| > 1 \tag{193}$$

If $|C_i| = 1$, then $s(i) = 0$. The value bounds are $-1 \leq s(i) \leq 1$.

Definition (Silhouette Score):

$$\text{Score} = \frac{1}{N}\sum_{i=1}^{N} s(i) \tag{194}$$

8.7 K-Medoids Algorithm

Uses actual data points as centroids and requires only a distance metric (e.g., number of differences in binary attributes).

Procedure: K-MEDOIDS(X, K)

1. Select randomly K data samples to be centroids of clusters.
2. **Repeat** until no change in assignment:
3. For each data record, assign to the closest cluster.
4. For each cluster k, find new centroids $i_k^* \in C_k$:

$$i_k^* \leftarrow \arg\min_{i:C(i)=k} \sum_{C(i')=k} d(x_i, x_{i'})$$

Complexity: The t iterations of K-medoids take $O(tkpN^2)$, requiring quadratic time to find a centroid compared to linear time in K-means.

8.8 Multidimensional Scaling (MDS)

Definition: For given data $x_1, \ldots, x_N$ with a distance matrix d, we search for projections $(z_1, \ldots, z_N) \in \mathbb{R}^p$ minimizing the stress function S_D, optimized via gradient descent.

$$S_D(z_1, \ldots, z_N) = \left[\sum_{i \neq l} (d[x_i, x_l] - \|z_i - z_l\|)^2 \right]^{\frac{1}{2}} \tag{195}$$

8.9 Hierarchical Clustering (Bottom-Up)

Start with each data sample in its own cluster. Iteratively join the two nearest clusters. Represented visually as a Dendrogram.

Linkage Measures for Join:

- **Single Linkage**: Distance between closest points.
- **Complete Linkage**: Distance between maximally distant points.
- **Average Linkage**:

$$d_{GA}(C_A, C_B) = \frac{1}{|C_A| \cdot |C_B|} \sum_{x_i \in C_A, x_j \in C_B} d(x_i, x_j)$$

- **Ward Distance**: Minimizes the sum of squared differences within all clusters.

$$Ward(C_A, C_B) = \sum_{i \in C_A \cup C_B} d(x_i, \mu_{A \cup B})^2 - \sum_{i \in C_A} d(x_i, \mu_A)^2 - \sum_{i \in C_B} d(x_i, \mu_B)^2$$

$$Ward(C_A, C_B) = \frac{|C_A| \cdot |C_B|}{|C_A| + |C_B|} \cdot d(\mu_A, \mu_B)^2$$

Where μ_A, μ_B, and $\mu_{A \cup B}$ are cluster centers.

8.10 Gaussian Mixture Model (GMM) [!]

Assume data originates from K Gaussian distributions.

- Prior probability: π_k
- Mean: μ_k
- Covariance matrix: Σ_k

Gaussian density function:

$$\phi_{\mu_k,\Sigma_k}(x) = \frac{1}{\sqrt{(2\pi)^p|\Sigma_k|}} \exp\left(-\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)\right) \tag{196}$$

8.11 EM Learning of Mixture of K Gaussians [!]

Optimizing model parameters π_k, μ_k, Σ_k subject to $\sum_{k=1}^K \pi_k = 1$.

Expectation Step (E-step): Calculate weights of unobserved fill-ins of variable C:

$$p_{ik} = \Pr(C = k \mid x_i) = \frac{\pi_k \phi_{\theta_k}(x_i)}{\sum_{l=1}^K \pi_l \phi_{\theta_l}(x_i)} \tag{197}$$

Let $p_k = \sum_{i=1}^N p_{ik}$.

Maximization Step (M-step): Update mean, variance, and prior:

$$\mu_k \leftarrow \sum_i \frac{p_{ik}}{p_k} x_i \tag{198}$$

$$\Sigma_k \leftarrow \sum_i \frac{p_{ik}}{p_k}(x_i-\mu_k)(x_i-\mu_k)^T \tag{199}$$

$$\pi_k \leftarrow \frac{p_k}{\sum_{l=1}^K p_l} \tag{200}$$

8.12 Dirichlet Process Mixture Modeling (DPMM) [x]

A nonparametric Bayesian model allowing an infinite number of clusters, modeled via a Stick-breaking process.

Likelihood of the Finite Mixture Model:

$$p(y_i \mid \mu_1,\ldots,\mu_K,\Sigma_1,\ldots,\Sigma_K,\pi_1,\ldots,\pi_K) = \sum_{k=1}^K \pi_k \mathcal{N}(x_i;\mu_k,\Sigma_k) \tag{201}$$

Dirichlet Process Prior: Given concentration parameter α, the prior probability of assigning to a new cluster $K+1$ is $\frac{\alpha}{N-1+\alpha}$. The probability of assigning to an existing cluster k is:

$$p(c_i \mid c_{-i}, \alpha) = \frac{n_{-i,k}}{N-1+\alpha} \tag{202}$$

Where $n_{-i,k}$ is the number of samples in cluster k excluding sample i.

Posterior Updates (Conjugate Priors): Assuming known irreducible noise σ_x^2 and a prior $\mathcal{N}(\mu_0,\Sigma_0)$ with concentration τ.

$$\mu_k = \frac{\sum_{i\in k} x_i\tau_k + \mu_0\tau_0}{n_k\tau_k + \tau_0} \tag{203}$$

$$\sigma_k^2 = \frac{1}{n_k\tau_k+\tau_0} + \sigma_x^2 \tag{204}$$

Probability of sample i belonging to cluster k:

$$p(c_i \mid c_{-i}, \mu_k, \tau_k, \alpha) = \frac{n_{-i,k}}{N-1+\alpha} \mathcal{N}\left(x_i; \frac{\sum_{i \in k} x_i \tau_k + \mu_0 \tau_0}{n_k \tau_k + \tau_0}, \frac{1}{n_k \tau_k + \tau_0} + \sigma_x^2\right) \tag{205}$$

Algorithm 1: DPMM Algorithm (MCMC Re-assignment): Input: $\alpha, \mu_0, \sigma_0^2, \sigma_y^2$. Let $\tau_0 = 1/\sigma_0^2$ and $\tau_y = 1/\sigma_y^2$.

1. Given state of Markov chain, sample new $\{\mu_k^{(t)}, \tau_k^{(t)}\}$ and $c^{(t)}$.
2. For $t \leftarrow 1$ to maxiter:
3. For $i \leftarrow 1$ to n:
4. Remove y_i from cluster c_i.
5. If cluster becomes empty, remove it and rearrange indices $1, \ldots, K$.
6. Draw $c_i \mid c_{-i}, y$:
7. For $k \leftarrow 1$ to $K+1$:
8. If $k \leq K$, compute $p(c_i = k \mid c_{-i}, y_i) \propto \frac{n_{-i,k}}{n-1+\alpha} \mathcal{N}\left(y_i; \frac{\bar{y}_k n_k \tau_k + \mu_0 \tau_0}{n_k \tau_k + \tau_0}, \frac{1}{n_k \tau_k + \tau_0} + \sigma_y^2\right)$
9. If $k = K+1$, compute $p(c_i = K+1 \mid c_{-i}, y_i) \propto \frac{\alpha}{n-1+\alpha} \mathcal{N}(y_i; \mu_0, \sigma_0^2 + \sigma_y^2)$
10. Assign c_i based on probabilities. Update cluster statistics.
11. Set $c^{(t)} = c_i$.
12. Return c.

8.13 Kernel Density Estimation (KDE)

Unsupervised procedure smoothing density estimate within neighborhood $\mathcal{N}(x_0)$ utilizing a Parzen kernel estimate with lengthscale λ:

$$\hat{f}_X(x_0) = \frac{1}{N\lambda} \sum_{i=1}^{N} K_\lambda(x_0, x_i) \tag{206}$$

A popular choice for K_λ is the Gaussian kernel density ϕ_λ.

8.14 Kernel Density Classification

Estimate KDE for each target class $k \in \{1, \ldots, K\}$, estimate class priors π_k, and apply Bayes' theorem:

$$\hat{\Pr}(G = k \mid X = x_0) = \frac{\pi_k \hat{f}_k(x_0)}{\sum_{j=1}^{K} \pi_j \hat{f}_j(x_0)} \tag{207}$$

8.15 Radial Basis Functions and Kernels for Regression

Kernels are not restricted to observation points. Fit prototype parameters $\xi_j \in X$ and scales $\lambda_j \in \mathbb{R}$ for M kernels.

$$f(x) = \sum_{j=1}^{M} K_{\lambda_j}(\xi_j, x) \beta_j \tag{208}$$

8.16 Mean Shift Clustering

Procedure: MEAN SHIFT CLUSTERING(X, kernel $K(\cdot)$, bandwidth λ)

1. $\mathcal{C} \leftarrow \emptyset$
2. For each data record x:
3. **Repeat** shift mean to weighted average until convergence:

$$m(x) \leftarrow \frac{\sum_{i=1}^{N} K(x_i - x) x_i}{\sum_{i=1}^{N} K(x_i - x)}$$

4. Add the new $m(x)$ to $\mathcal{C}$
5. Return pruned $\mathcal{C}$

Kernels:

- Flat kernel (ball)
- Gaussian kernel:

$$K(x_i - x) = \exp\left(-\frac{\|x_i - x\|^2}{\lambda^2}\right)$$

9 Machine Learning: Rule Learning and Association Rules

9.1 Rule Learning Overview

- **Supervised learning**: Includes rules derived from decision trees (or the Sequential covering algorithm) and PRIM (Bump hunting).
- **Unsupervised learning**: Includes association rules, version space search for rules, and Inductive Logic Programming (ILP, MIL).

9.2 Association Rules and Market Basket Analysis

- Designed for very large datasets. For example, with $p \approx 10^4$ and $N \approx 10^8$ in a unit ball, the distance to the nearest neighbor is approximately 0.9981.
- The algorithm searches for frequent itemsets, which represent high-density areas.
- It tests on a feature X_j being either equal to a specific value or having no restriction at all. The value 1 is considered more important than 0.
- Selects combinations of items with a higher number of occurrences (support) than a predefined threshold t.
- Categorical variables may be coded by dummy variables in advance (if there are not too many). For instance, an OneHotEncoder is applied for each class g, creating a new variable $X_g = [X == g]$ without dropping any values.

9.3 Apriori Algorithm

The objective is to find frequent itemsets.

1. $i \leftarrow 1$
2. Generate a list of candidates of length i
3. **while** Candidate set is not empty **do**
   (a) **for each** data sample **do**
      i. **for each** candidate **do**
         A. **if** all items of the candidate appear in the data sample **then** increase the candidate counter by 1
   (b) $i \leftarrow i + 1$
   (c) Discard candidates with support less than t
   (d) Generate a list of candidates of length i
   (e) Join any two candidates from the previous step having $i - 2$ elements in common. (More pruning is possible.)

9.4 Properties of the Apriori Algorithm

- Applicable for very large data (with a high threshold t).
- **Key idea**: Only a few of the 2^K combinations have a high support $> t$. Furthermore, any subset of a high-support combination also has high support.
- The number of passes through the data is equal to the size of the longest supported combination. The data do not need to be in memory simultaneously.
- As an alternative, the FP-growth algorithm needs only two passes through the data.

9.5 Association Rules [!]

From each supported itemset $\mathcal{K}$ found by the Apriori algorithm, we create a list of association rules, which are implications of the form $A \Rightarrow B$, where:

- A and B are disjoint and $A \cup B = \mathcal{K}$.
- A is called the **antecedent**.
- B is called the **consequent**.
- The support of the rule $T(A \Rightarrow B)$ is defined as the normalized support of the itemset $\mathcal{K}$, which is the normalized support of the conjunction $A \wedge B$:

$$T(\mathcal{K}) = \frac{|\text{data}_{\mathcal{K}}|}{|\text{data}|}$$

$$T(A \Rightarrow B) = \frac{|\text{data}_{A \cup B}|}{|\text{data}|}$$

9.6 Rule Confidence and Lift

There are several important measures for evaluating a rule $A \Rightarrow B$:

- **Confidence** (predictability, přesnost): An estimate of $\Pr(B \mid A)$.

$$C(A \Rightarrow B) = \frac{T(A \Rightarrow B)}{T(A)}$$

- **Support**: $T(B)$ serves as an estimate of $\Pr(B)$.
- **Lift**: The ratio of confidence to expected precision. It acts as an estimate of $\frac{\Pr(A \wedge B)}{\Pr(A) \cdot \Pr(B)}$.

$$L(A \Rightarrow B) = \frac{C(A \Rightarrow B)}{T(B)}$$

- **Leverage**: The difference of the supports.

$$\text{Leverage}(A \Rightarrow B) = T(A \Rightarrow B) - T(A) \cdot T(B)$$

- **Conviction**: Evaluated as the following ratio:

$$\text{Conviction}(A \Rightarrow B) = \frac{1 - T(B)}{1 - C(A \Rightarrow B)}$$

9.7 The Goal of Apriori Algorithm [!]

- Apriori finds all rules with high support.
- Frequently, it finds many rules. We usually select a lower threshold c on confidence; that is, we select rules satisfying $T(A \Rightarrow B) > t$ and $C(A \Rightarrow B) > c$.
- The conversion of itemsets to rules is usually relatively fast compared to the initial search for frequent itemsets.

9.8 Frequent Pattern-Tree (FP-Tree) Data Structure

- The number of passes through the data for Apriori equals the length of the longest frequent itemset.
- With an internal data structure, we can reduce this to exactly 2 passes.
- The process involves building an internal structure called an **FP-tree** and calling **FP-growth** to generate frequent itemsets.
- Each construction of a conditional tree needs 2 passes through the parent tree. An optimized version requiring only 1 pass exists (requires an additional array data structure).
- **FP-max**: Algorithm to find maximal itemsets (where none of the immediate supersets are frequent).
- **FP-close**: Algorithm to find closed itemsets (where none of the immediate supersets have the same support).
- **Principle**: If X and Y are two itemsets, the count of itemsets $X \cup Y$ in the database is exactly that of Y in the database restricted to those transactions containing X.

9.9 FP-Tree Procedure

1. Calculate counts of items (singletons).
2. Create a table header ordered by decreasing item count.
3. **for each** data sample **do**
   (a) order items according to the header.
   (b) insert the branch into the tree.
   (c) increase all counters on the inserted branch.
4. **return** the tree.

9.10 FP-Growth* Procedure

Procedure for generating patterns given a conditional FP-tree, T.

1. **if** T only contains a single path P **then**
   (a) **for each** subpath Y of P **do**
      i. output pattern $Y \cup T$.base with count = smallest count of nodes in Y.
2. **else**
   (a) **for each** i in T.header **do**
      i. $Y \leftarrow T$.base $\cup \{i\}$ with i.count.
      ii. **if** T.array is not NULL **then** construct a new header table for Y's FP-tree from T.array.
      iii. **else** construct a new header table for Y's FP-tree from T.
      iv. construct Y's conditional FP-tree T_Y and its array A_Y.
      v. **if** $T_Y \neq \emptyset$ **then** call FPGROWTH*(T_Y).

9.11 Unsupervised Learning as Supervised Learning

- Add an additional attribute Y_G.
- Set $Y_G = 1$ for all available real data.
- Randomly generate a dataset of a similar size using a uniform distribution, and set $Y_G = 0$ for these artificial data points.
- The task then becomes a supervised classification problem to separate $Y_G = 1$ and $Y_G = 0$.

9.12 Generalize Association Rules

- The goal is to search for high lift where the probability of the conjunction is significantly greater than statistically expected.
- A hypothesis is specified by column indexes J and subsets of values s_j corresponding to features X_j.
- We aim to satisfy:

$$\hat{\Pr}\left(\bigcap_{j\in J}(X_j \in s_j)\right) = \frac{1}{N}\sum_{i=1}^{N} I\left(\bigcap_{j\in J}(x_{i,j} \in s_j)\right) \gg \prod_{j\in J}\hat{\Pr}(X_j \in s_j) \tag{209}$$

- On this data, CART (decision tree algorithm) or PRIM ('bump hunting') may be used.

9.13 Version Space Search

- A hypothesis is a conjunction of attribute tests that imply a specific target label (e.g., Target = yes).
- A symbol like `?` in a hypothesis indicates it is satisfied by any value.
- Hypotheses with combinations that cannot be satisfied are considered equivalent.
- The hypothesis space is partially ordered by subsumption.
- **More General / More Specific**: The hypothesis h_g is more general than h_s (denoted $h_g \geq h_s$) if and only if any sample that satisfies h_s also satisfies h_g. In this case, h_s is considered more specific.
- The most general hypothesis is satisfied by all data.
- The most specific hypothesis (e.g., using $\emptyset$ or 0) is not satisfied by any data.

9.14 Find-S Algorithm

The search targets a hypothesis satisfied by all positive examples and no negative examples.

1. $h \leftarrow (0, 0, 0, 0)$ (Initialize with the most specific hypothesis).
2. **for each** positive data sample X_i **do**
   (a) **for each** attribute condition $X_j = x_{i,j}$ in h **do**
      i. **if** X_i does not satisfy $X_j = x_{i,j}$ **then** replace the condition by the closest, more general condition satisfied by X_i.
3. **return** h.

9.15 Version Space Boundaries

- **Version Space**: The version space for the hypothesis space H and the data X is the subset of H that is perfectly consistent with X:

$$VS(H, X) = \{h \in H \mid \text{Consistent}(h, X)\}$$

- **General Boundary (G)**: The set of the most general hypotheses from H that are consistent with X:

$$G(H, X) = \{g \in H \mid \text{Consistent}(g, X) \wedge \nexists g_1 \in H[g_1 > g \wedge \text{Consistent}(g_1, X)]\}$$

- **Specific Boundary (S)**: The set of the most specific hypotheses from H that are consistent with X:

$$S(H, X) = \{s \in H \mid \text{Consistent}(s, X) \wedge \nexists s_1 \in H[s > s_1 \wedge \text{Consistent}(s_1, X)]\}$$

9.16 Candidate-Elimination Algorithm

1. $G \leftarrow \{\langle ?, ?, ?, ? \rangle\}$ and $S \leftarrow \{\langle \emptyset, \emptyset, \emptyset, \emptyset \rangle\}$
2. **for each** data sample X_i **do**
   (a) **if** X_i is positive **then**
      i. remove from G all hypotheses h inconsistent with X_i.
      ii. **for each** $s \in S$ inconsistent with X_i **do**
         A. add to S all minimal generalizations h such that Consistent$(h, X_i) \wedge (\exists g \in G)(g \geq h)$.
         B. remove from S any hypothesis s that is not most specific; meaning remove $\{s \mid (\exists s_1 \in S)(s > s_1)\}$.
   (b) **else** (X_i is a negative example)
      i. remove from S all hypotheses h inconsistent with X_i.
      ii. **for each** $g \in G$ inconsistent with X_i **do**
         A. add to G all minimal specifications h such that Consistent$(h, X_i) \wedge (\exists s \in S)(h \geq s)$.
         B. remove from G any hypothesis g that is not most general; meaning remove $\{g \mid (\exists g_1 \in G)(g_1 > g)\}$.
3. **return** G, S.

10 Machine Learning: Inductive Logic Programming (ILP)

10.1 Predicate Logic Definitions

- **CNF, DNF**: The conjunctive and disjunctive normal forms.
- **Clause**: A disjunction of literals. For example:

$$father(X, Y) \vee \neg parent(X, Y) \vee \neg male(X)$$

- **Horn Clauses**: Clauses with at most one positive literal, typically written as a rule.
- **Definite Clause**: A Horn clause with exactly one positive literal. Example:

$$father(X, Y) :- male(X), parent(X, Y).$$

- **Fact**: A clause with no negative literals. Example: $male(adam)$.
- **Goal Clause**: A clause with no positive literals. Example: $false :- father(X, bob)$.
- **Ground term / Ground clause**: A term or a clause without variables.
- **Data Representation**: Data is provided in the form of a set of clauses B, E^+, and E^-, where the background knowledge B is a set of Horn clauses, and the positive and negative examples E^+, E^- are sets of ground literals (facts).

10.2 Substitution and Subsumption

Clauses are implicitly generally quantified. They should not have a variable with the same name.

Definitions:

- **Substitution**: Given a substitution $\theta = \{v_i/t_i\}$ and formula F, $F\theta$ is formed by replacing every variable v_i in F by t_j.
- **Unification**: Substitution unifies atom A and atom B in the case $A\theta = B\theta$.
- **Subsumption (Atom)**: Atom A subsumes atom B ($A \geq B$) iff there exists a substitution θ such that $A\theta = B$.
- **Subsumption (Clause)**: Clause C subsumes clause D ($C \geq D$) iff there exists a substitution θ such that $C\theta \subseteq D$.

10.3 Generalisation

Definitions:

- Clause C is more general than clause D, iff $C \models D$.
- Clause C is more general than clause D with respect to B, iff $B, C \models D$, where B is the background knowledge. This is denoted as $C \geq D$ with respect to B.

10.4 ILP General Logical Setting

The background knowledge B and the hypothesis H should entail E. The properties of the hypothesis are defined as follows:

- **Necessity**: $B \not\models E^+$ (we need H).
- **Sufficiency / Completeness**: $B \wedge H \models E^+$ (H explains positive examples).
- **Weak consistency**: $B \wedge H \not\models$ false (H does not contradict B).
- **(Strong) consistency**: $B \wedge H \wedge E^- \not\models$ false (neither negative examples are contradicted).

ILP Task: Given background knowledge B (logic program) and examples E^+, E^- (sets of ground unit clauses), find a logic program H such that it is necessary, sufficient, and consistent.
(!) Important: Often, we assume noisy data and accept some errors, but we try to minimize them.

10.5 Hypothesis Space and Mode Declarations

To specify or restrict the hypothesis space, mode declarations are used.

Definition (Mode Declarations): Mode declarations denote which literals may appear in the head or body of a rule. A mode declaration is of the form:

$$mode(recall, pred(m_1, m_2, ..., m_a))$$

where:

- *recall* is the maximum number of occurrences of the predicate.
- m_i are the argument types which may be assigned as input (+), output (-), or constant (#).

10.6 Non-monotonic Reasoning

In Prolog, there is negation as a failure.

- **Normal logic program**: Normal logic programs may include negated literals in the body of a clause. Example:
$$h :- b_1, ..., b_n, \text{not } b_{n+1}, ..., \text{not } b_m$$

10.7 Aleph ILP System (based on Progol)

Input:

- A set of mode declarations M.
- Background knowledge B in the form of a normal program (allows negation, with the semantics negation as a failure).
- Positive E^+ and negative E^- examples as a set of ground facts.

Output: A normal program hypothesis H such that:

- H is consistent with M.
- $\forall e \in E^+, H \cup B \models e$ (H is complete).
- $\forall e \in E^-, H \cup B \not\models e$ (H is consistent).

Aleph Algorithm Steps:

1. Select a positive example to generalize.
2. Construct the most specific clause consistent with M that entails the example (the bottom clause).
3. Search for the 'best' clause more general than the bottom clause.
4. Add the clause to the hypothesis and remove all examples covered.
5. If a positive example is left, return to step 1.

Bottom Clause Construction: Let H be a clausal hypothesis and C be a clause. The bottom clause $\perp(C)$ is the most specific clause such that:

$$H \cup \perp(C) \models C \tag{210}$$

The purpose is to bound the search in step 3 of the algorithm. **(!) Important:** Without mode declarations, the bottom clause may have infinite cardinality.

Clause Search and Evaluation:

- Aleph performs a bounded-breadth-first search to enumerate the shorter clauses before the longer ones. The search is bounded by parameters such as max clause size and max proof depth.
- **Evaluation Function**: Aleph's default evaluation function is coverage, defined as $P - N$, where P is the number of positive examples covered by the clause, and N is the number of negative examples covered by the clause (meaning it accepts some noise).
- **Refinement**: Aleph starts from the most general hypothesis and tries to specialize the clause by adding literals to the body (selected from the bottom clause) or by instantiating variables. Each specialization is called a refinement.

10.8 Metagol

Metagol is a form of ILP based on a Prolog meta-interpreter.

Input:

- A set of metarules M.
- Background knowledge B in the form of a normal program.
- Positive E^+ and negative E^- examples as a set of facts (atoms).

Output: A definite program hypothesis H that:

- H is consistent with M.
- $\forall e \in E^+, H \cup B \models e$ (H is complete).
- $\forall e \in E^-, H \cup B \not\models e$ (H is consistent).
- $\forall h \in H, \exists m \in M$ such that $h = m\theta$, where θ is a substitution that grounds all the existentially quantified variables in m.

Metagol Algorithm Steps:

1. Select a positive example to generalize.

- If none exists, test the hypothesis on the negative examples.
- If the hypothesis does not entail any negative example, stop and return the hypothesis.
- Otherwise, backtrack to a choice point at step 2 and continue.

2. Try to prove the atom by:
   - Using given Background Knowledge (BK) or an already induced clause.
   - Unifying the atom with the head of a metarule.
   - Binding the variables in the metarule to symbols in the predicate and constant signatures.
   - Saving the substitution.
   - Trying to prove the body of the metarule by treating the body atoms as examples and applying step 2 to them.

Key Features:

- **Recursion**: Metagol can learn recursive programs (e.g., reachability in a graph, tail recursive metarules).
- **Iterative Deepening**: Metagol uses iterative deepening to search for hypotheses. At depth $d = 1$, at most one metasub is allowed. At iteration d, it introduces $d - 1$ new predicate symbols and is allowed to use d clauses.

10.9 ASPAL Algorithm

ASPAL uses Answer Set Programming (ASP). An ASP program can have one, many, or no models (answer sets). Computation in ASP is the process of finding models. We may specify the range of the number of clauses from a set being true (e.g., $0\{sunny., weekday., happy(A) : -lego\_builder(A)\}3$) and an evaluation function to optimize (like minimizing the size of the hypothesis).

ASPAL Workflow:

1. Generate all possible rules consistent with the given mode declarations.
2. Assign each rule a unique identifier and add a guessable atom in each rule (e.g., $rule(r1)$).
3. Use an ASP solver to find a minimal subset of the rules by formulating the problem as an ASP optimization problem.

11 Machine Learning: Undirected Graphical Models

11.1 Hidden Conditional Random Fields (HCRF)

Hidden Conditional Random Fields (HCRF) can be used for tasks such as Gesture Recognition. An HCRF utilizes different window sizes w and learns hidden states for each frame to base the overall classification on these estimated states.

The potentials formula for the HCRF model is given by:

$$\Psi(y, s, x; \theta, \omega) = \sum_{j=1} \varphi(x, j, \omega) \cdot \theta_s[s_j] + \sum_{j=1} \theta_y[y, s_j] + \sum_{(j,k)\in E} \theta_e[y, s_j, s_k] \tag{211}$$

Where:

- y is the gesture type (estimated per frame).
- s represents the hidden states.
- x represents the input data (features).
- θ are the model parameters $(\theta_s, \theta_y, \theta_e)$.
- ω represents the window size.
- E is the set of edges in the model.

11.2 Undirected (Pairwise, Continuous) Graphical Models

A generative model represents the full probability distribution $P(X)$. Missing edges in the graph represent the conditional independence of the variables. **[Note: Other restricted models not covered this year]**.

For a Sparse Conditional Gaussian Graphical Model analyzing dataset variables (e.g., $p_Y = 54$ continuous expressions and $p_X = 188$ discrete markers), the conditional Gaussian distribution is defined as:

$$Y^{p_Y} | X^{p_X} \sim \mathcal{N}(\mathbf{M}^{p_Y \times p_X} X^{p_X}, \Sigma^{p_Y \times p_Y}) \tag{212}$$

Where $\mathbf{M}$ is the coefficient matrix and Σ is the covariance matrix.

11.3 Gaussian Graphical Models (Undirected Graphs)

An undirected Gaussian graphical model is represented by an undirected graph $\mathcal{G} = (X, E)$, where $X = \{X_1, \ldots, X_p\}$ represents the set of variables and E is the set of undirected edges.

The Multivariate Gaussian Distribution on variables X is defined as:

$$\phi(x) = \frac{1}{\sqrt{|2\pi\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \tag{213}$$

Where:

- μ is the mean vector.
- Σ is the covariance matrix.
- $|2\pi\Sigma| = (2\pi)^p |\Sigma|$ is the determinant.

If Σ is not invertible, its columns are linearly dependent (rank $l < p$). In this case, new coordinates z with l dimensions can be found such that $x = \mathbf{A}z + \nu$, assuming Σ has a full rank in the new coordinates.

11.4 Concentration Matrix and Partial Correlation

When a random vector x follows a Gaussian distribution $\mathcal{N}_p(\mu, \Sigma)$, the graph $\mathcal{G}$ represents the model where $K = \Sigma^{-1}$ is a positive definite matrix.

Definition (Concentration Matrix): The concentration (or precision) matrix is $K = \Sigma^{-1}$. **Lemma:** For $u \neq v$, $k_{uv} = 0$ if and only if X_u and X_v are conditionally independent given all other variables.

Definition (Partial Correlation Matrix): The partial correlation matrix is defined from K by:

$$\rho_{uv|V\setminus\{uv\}} = \frac{-k_{uv}}{\sqrt{k_{uu}k_{vv}}} \tag{214}$$

Lemma: Partial correlations are invariant under a change of scale and origin. If $X_j^* = a_j X_j + b_j$, then $a_v a_u k_{uv}^* = k_{uv}$ and $\rho_{uv|V\setminus\{uv\}}^* = \rho_{uv|V\setminus\{uv\}}$.

11.5 Marginalization and Conditioning

Marginalization: The marginal of a Gaussian distribution is calculated by removing appropriate dimensions from the mean and covariance matrix. For a subset of variables $\{x_3, x_5, x_7\}$, the marginal distribution ϕ_{x_3,x_5,x_7} uses $\mu_{3,5,7}$ and $\Sigma_{3,5,7}$ directly extracted from μ and Σ.

Conditioning: To find the conditional distribution $\phi(A|B = b)$ where subset A has q elements and subset B has $p - q$ elements, partition the matrices as:

$$x = \begin{bmatrix} x_A \\ x_B \end{bmatrix}, \quad \mu = \begin{bmatrix} \mu_A \\ \mu_B \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \Sigma_{AA} & \Sigma_{AB} \\ \Sigma_{BA} & \Sigma_{BB} \end{bmatrix} \tag{215}$$

The parameters of the conditional Gaussian distribution $\mathcal{N}(\mu_{A|B=b}, \Sigma_{A|B=b})$ are:

$$\mu_{A|B=b} = \mu_A + \Sigma_{AB}\Sigma_{BB}^{-1}(b - \mu_B) \tag{216}$$

$$\Sigma_{A|B=b} = \Sigma_{AA} - \Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{BA} \tag{217}$$

Note that the conditional covariance matrix $\Sigma_{A|B=b}$ depends on the fact that B was observed, but does not depend on the specific observation values b.

11.6 Partition Matrix Inverse Properties & Regression Coefficients

Because $K = \Sigma^{-1}$:

$$\begin{pmatrix} K_{AA} & K_{AB} \\ K_{BA} & K_{BB} \end{pmatrix} \begin{pmatrix} \Sigma_{AA} & \Sigma_{AB} \\ \Sigma_{BA} & \Sigma_{BB} \end{pmatrix} = \begin{pmatrix} \mathbf{I}_{AA} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{BB} \end{pmatrix} \tag{218}$$

This yields critical identities:

$$\Sigma_{AB}\Sigma_{BB}^{-1} = -K_{AA}^{-1}K_{AB} \tag{219}$$

$$K_{AA}^{-1} = \Sigma_{AA} - \Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{BA} \tag{220}$$

For a linear Gaussian Conditional Probability Distribution (CPD) where x_1 is a linear function of the other variables $x_{2\ldots p}$ with noise $\epsilon_1 \sim \mathcal{N}(0, \sigma_1^2)$:

$$x_{1|2\ldots p} = \beta_1 + \beta_{12}x_2 + \beta_{13}x_3 + \cdots + \beta_{1p}x_p + \epsilon_1 \tag{221}$$

The variance and regression coefficients can be expressed via the concentration matrix elements:

$$\sigma_1^2 = \frac{1}{k_{11}} \tag{222}$$

$$(\beta_{12}, \ldots, \beta_{1p}) = -\frac{(k_{12}, \ldots, k_{1p})}{k_{11}} \tag{223}$$

11.7 Parameter Learning for Gaussian Graphical Models

Given data $x_1, \ldots, x_N \sim \mathcal{N}_p(\mu, \Sigma)$, let $S = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T$ be the empirical covariance matrix. The penalized log-likelihood for parameter matrix $\Theta = \Sigma^{-1}$ is proportional to:

$$\log|\Theta| - \text{tr}(\Theta S) \tag{224}$$

By replacing equality conditions (for missing edges) with a Lagrange multiplier matrix Γ (where $\Gamma_{jk} \neq 0$ only for missing edges), the objective to maximize is:

$$l_C(\Theta) = \log|\Theta| - \text{tr}(\Theta S) - \sum_{(j,k)\notin E} \gamma_{jk}\theta_{jk} \tag{225}$$

The gradient set to zero yields:

$$\Theta^{-1} - S - \Gamma = \mathbf{0} \tag{226}$$

11.8 Algorithm: Graphical Regression

This algorithm estimates undirected graphical model parameters by iterating over rows/columns.

1. **Initialize:** $W \leftarrow S$ (where S is the sample covariance).
2. **Repeat until convergence:**
3. For $j = 1, 2, \ldots, p$ do:
4. Partition W into the j-th row/column and the rest (W_{11}).
5. Solve $W_{11}^{*}\beta^{*} - s_{12}^{*} = \mathbf{0}$ for the reduced system.
6. Pad $\hat{\beta}^{*}$ with zeros to form $\hat{\beta}$.
7. Update $w_{12} \leftarrow W_{11}\hat{\beta}$.
8. **Final Update (after convergence):**
9. For $j = 1, 2, \ldots, p$ do:
10. $\hat{\theta}_{22} \leftarrow \frac{1}{w_{22} - w_{12}^T\hat{\beta}}$
11. $\hat{\theta}_{12} \leftarrow -\hat{\beta} \cdot \hat{\theta}_{22}$

11.9 Structure Learning and the Graphical Lasso

To learn the structure, an L_1 penalty (Lasso) is added to the log-likelihood to enforce sparsity, ignoring the diagonal elements:

$$\log|\Theta| - \text{tr}(\Theta S) - \lambda\|\Theta\|_1 \tag{227}$$

The gradient equation utilizing sub-gradients becomes:

$$\Theta^{-1} - S - \lambda\,\text{Sign}(\Theta) = \mathbf{0} \tag{228}$$

Where $\text{Sign}(\theta_{jk}) = \text{sign}(\theta_{jk})$ for $\theta_{jk} \neq 0$, and $\text{Sign}(\theta_{jk}) \in [-1, 1]$ for $\theta_{jk} = 0$.

11.10 Algorithm: Graphical Lasso

1. **Initialize:** $W \leftarrow S + \lambda \mathbf{I}$.
2. **Repeat until convergence:**
3. For $j = 1, 2, \ldots, p$ do:
4. Partition W into the j-th row/column and the rest (W_{11}).
5. Solve $W_{11}\beta - s_{12} + \lambda\,\text{Sign}(\beta) = \mathbf{0}$ using cyclical coordinate-descent.
6. Update $w_{12} \leftarrow W_{11}\hat{\beta}$.
7. **Final Update:** (Same logic as Graphical Regression for $\hat{\theta}_{22}$ and $\hat{\theta}_{12}$).

Subroutine: Coordinate Descent for solving $V\hat{\beta} - s_{12} + \lambda\,\text{Sign}(\hat{\beta}) = \mathbf{0}$ where $V \leftarrow W_{11}$:

1. Repeat $j = 1, 2, \ldots, p-1$ until convergence:
2. $\hat{\beta}_j \leftarrow S\left(s_{12j} - \sum_{k \neq j} V_{kj}\hat{\beta}_k, \lambda\right)/V_{jj}$

Where the soft-thresholding operator is $S(x,t) = \text{sign}(x)(|x| - t)_+$.

11.11 Model Quality and Selection

Definition (Deviance and Likelihood Ratio Test):

- **Saturated model:** Full model with all edges (maximal log-likelihood).
- **Independent model:** No edges (minimal likelihood).
- **Deviance (D):**

$$D = \text{dev} = 2 \cdot (\hat{l}_{\text{sat}} - \hat{l}) = N \log \frac{|S^{-1}|}{|\hat{K}|} = -N \log |S\hat{K}|$$

- **iDeviance (iD):**

$$iD = \text{idev} = 2 \cdot (\hat{l} - \hat{l}_{\text{ind}}) = N\left(\log|\hat{K}| + \sum_{i=1}^{p} \log s_{ii}\right)$$

- **Likelihood Ratio Test (lrt):** For nested models $\mathcal{M}_1 \subseteq \mathcal{M}_0$:

$$\text{lrt} = 2 \cdot (\hat{l}_0 - \hat{l}_1) = N \log \frac{|\hat{K}_0|}{|\hat{K}_1|}$$

11.12 Markov Properties for Undirected Graphs

[**! Important: Zeros are dangerous**] - The properties below are only strictly equivalent for strictly positive measures. Counterexamples exist for measures with zero probability regions.

Definition (Markov Properties): Let $\mathcal{G}$ be an undirected graph over V.

- **Global Markov (GM):** For any subgraphs $\mathcal{A}, \mathcal{B}, \mathcal{C} \subseteq V$, if $\mathcal{C}$ separates $\mathcal{A}$ and $\mathcal{B}$, then:

$$\mathcal{A} \perp_{\mathcal{G}} \mathcal{B}|\mathcal{C} \implies \mathcal{A} \perp_P \mathcal{B}|\mathcal{C} \text{ in } P$$

- **Local Markov (LM):** Every node is conditionally independent of all other nodes given its neighbors:

$$(\forall A \in V) : A \perp_P V \setminus (A \cup N_A)|N_A$$

- **Pairwise Markov (PM):** Any two non-adjacent nodes are conditionally independent given all other nodes:

$$(\forall A, B \in V, A \neq B) : A \perp_P B|V \setminus \{A, B\}$$

11.13 Markov Random Fields (MRF)

A probability density function f over a Markov graph $\mathcal{G}$ with maximal cliques $\{C_1, \ldots, C_k\}$ can be represented via positive clique potential functions $\psi_i(x_{C_i})$:

$$f(x) = \frac{1}{Z} \prod_{i=1}^{k} \psi_i(x_{C_i}) \tag{229}$$

The partition function Z normalizes the distribution:

$$Z = \int_X \exp\left(\sum_{i=1}^{k} \log g_i(x_{C_i}) \right) \tag{230}$$

MRF for Image Denoising: Given a noisy image v, recover an image u that minimizes the pairwise energy function on a 4-connected grid P:

$$E(u) = \sum_{(m,n)\in P} (u_{m,n} - v_{m,n})^2 + \lambda \sum_{(m,n)\in P} \Big[(u_{n+1,m} - u_{n,m})^2 + (u_{n,m+1} - u_{n,m})^2 \Big] \tag{231}$$

Iterative solution updates for $u_{n,m}$ using neighbor sum $s_{m,n} = u_{n-1,m} + u_{n+1,m} + u_{n,m-1} + u_{n,m+1}$:

$$u_{n,m}^{(t+1)} = \begin{cases} \frac{1}{1+4\lambda} \left(v_{n,m} + \lambda s_{n,m}^{(t)} \right) & \text{for } (n,m) \in v \\ \frac{1}{4} s_{n,m}^{(t)} & \text{for missing } v \end{cases} \tag{232}$$

11.14 Ising Model and Restricted Boltzmann Machines

Definition (Ising Model): A model over binary variables $X_i \in \mathcal{X}$ with interactions represented by parameter matrix Θ:

$$p(X, \Theta) = \exp\left\{ \sum_{(j,k)\in E} \theta_{jk} X_j X_k - \Phi(\Theta) \right\} \tag{233}$$

Where $\Phi(\Theta)$ is the log-partition function:

$$\Phi(\Theta) = \log \sum_{x\in\mathcal{X}} \left[\exp\left\{ \sum_{(j,k)\in E} \theta_{jk} x_j x_k \right\} \right] \tag{234}$$

Restricted Boltzmann Machine (RBM): Contains a visible layer v and a hidden layer h, with no edges inside any single layer. The energy function is:

$$E(v, h) = -\sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} h_i v_j - \sum_{j=1}^{m} b_j v_j - \sum_{i=1}^{n} c_i h_i \tag{235}$$

Algorithm: Contrastive Divergence (for RBM parameter update):

1. Repeat until convergence:
2. For each training sample $v^{(0)}$ in a batch S:
3. Sample H given $v^{(0)}$.
4. Sample $v^{(\text{last})}$ given H.
5. Sample H given $v^{(\text{last})}$.

6. Update weights and biases:

7. $\Delta w_{ij} \leftarrow \Delta w_{ij} + p(H_i = 1|v^{(0)}) \cdot v_j^{(0)} - p(H_i = 1|v^{(\text{last})}) \cdot v_j^{(\text{last})}$

8. $\Delta b_j \leftarrow \Delta b_j + v_j^{(0)} - v_j^{(\text{last})}$

9. $\Delta c_i \leftarrow \Delta c_i + p(H_i = 1|v^{(0)}) - p(H_i = 1|v^{(\text{last})})$

10. Apply accumulated $\Delta w_{ij}, \Delta b_j, \Delta c_i$ to adjust the parameters.

11.15 Mixed Interaction Models

For models containing both discrete variables i and continuous variables y, the conditional Gaussian density is formulated as:

$$f(i, y) = p(i)(2\pi)^{-\frac{q}{2}}|\Sigma|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(y - \mu(i))^T \Sigma^{-1}(y - \mu(i))\right) \tag{236}$$

In exponential (canonical) form, this is parameterized by canonical parameters $g(i), h(i), K$:

$$f(i, y) = \exp\left\{g(i) + h(i)^T y - \frac{1}{2} y^T K y\right\} \tag{237}$$

12 Machine Learning: Gaussian Processes and Bayesian Optimization

12.1 Bayesian Optimization

Bayesian Optimization is utilized when solving the following objective:

$$x^* = \arg\min_x f(x) \tag{238}$$

The optimization is applied under the following conditions:

- $f(x)$ is a black box function.
- f is expensive to evaluate.
- The evaluations may be noisy.

! If any of the above conditions is not true, a better algorithm exists.

The goal is to search for the optimal point x to observe. We aim to minimize y and search for the maximal probability of improvement. The "chance to improve" is mathematically expressed by the Expected Improvement (EI).

Bayesian Optimization Algorithm:

1. Evaluate y on the initial set X. Let $y = y(X)$ and calculate conditional means and covariances.
2. Repeat forever:
   - Find the new point x^{new} that maximizes Expected Improvement:

     $$x^{\text{new}} = \arg\max_x \text{EI}(x)$$

   - Add x^{new} into X.
   - Evaluate $y = y(x^{\text{new}})$ and add the new y to the observations.
   - Re-estimate the Gaussian process (update the parameters of the covariance or kernel function).

12.2 Gaussian Processes

Definition (Gaussian Process): A Gaussian process is a set of random variables where any finite subset follows a multivariate Gaussian distribution.

We define the mean $m(x)$ and the symmetric positive semidefinite covariance function $k(x, x')$:

$$m(x) = \mathbb{E}[f(x)] \tag{239}$$

$$k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))] \tag{240}$$

A Gaussian process is denoted as:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')) \tag{241}$$

We generally assume $m(x) = 0$ as it simplifies the formulas. Because it defines an infinite (continuous) number of Gaussian variables, any variable x follows $\mathcal{N}(\mu = f(x), \Sigma_{x|\text{observed}})$. Since we have only a finite number of observations, we can marginalize unobserved variables out to continuously predict at any x.

12.3 Brownian Motion (Wiener Process)

Definition (Brownian motion 1):

- $B_0 = 0$ almost surely (for sure).
- B_t has stationary and independent increments.
- The increments follow a Gaussian distribution:

$$B_s - B_t \sim \mathcal{N}(0, s-t)$$

Lemma: It follows that $K(0,0) = \min(0,0) = 0$.
Definition (Brownian motion 2): A Gaussian process with $m = 0$ and the following covariance:

$$k(x, x') = \min(x, x') \tag{242}$$

The covariance is positive semidefinite:

$$\min(t, s) = \int_0^\infty f_t(x) f_s(x) dx \tag{243}$$

where $f_t(x)f_s(x) = 1$ if and only if $x \in [0, t]$ and $x \in [0, s]$. The process has variance 0 at $t = 0$, and $m(0) = 0$. The covariance is linear in both arguments. For $s \geq t$:

$$\text{cov}(B_s - B_t, B_s - B_t) = \text{cov}(B_s, B_s) - \text{cov}(B_t, B_s) - \text{cov}(B_s, B_t) + \text{cov}(B_t, B_t) = s - 2t + t = s - t \tag{244}$$

[Note: Independence derivation skipped, from Gaussian vectors]. For increments where $s \geq t \geq b \geq a$:

$$\text{cov}(B_b - B_a, B_s - B_t) = \text{cov}(B_b, B_s) - \text{cov}(B_a, B_s) - \text{cov}(B_b, B_t) + \text{cov}(B_a, B_t) = b - a - b + a = 0 \tag{245}$$

12.4 Prediction and Predictive Distribution

The covariance on y is defined by the covariance on the inputs X. The covariance also defines the distribution on functions f, such that $f_* \sim \mathcal{N}(0, K(X_*, X_*))$.

For noise-free observations $y = f(x)$:

$$\text{cov}(y_p, y_q) = k(x_p, x_q) \tag{246}$$

For noisy observations $y = f(x) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$:

$$\text{cov}(y_p, y_q) = k(x_p, x_q) + \sigma_n^2 \delta_{pq} \tag{247}$$

$$\text{cov}(\mathbf{y}) = K(X, X) + \sigma_n^2 I \tag{248}$$

When observing $\mathbf{y}$ to predict f_*, the joint distribution is:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X, X) + \sigma_n^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right) \tag{249}$$

The predictive distribution is:

$$\mathbf{f}_* | X, \mathbf{y}, X_* \sim \mathcal{N}(\bar{\mathbf{f}}_*, \text{cov}(\mathbf{f}_*)) \tag{250}$$

where the conditional mean is defined as:

$$\bar{\mathbf{f}}_* \triangleq \mathbb{E}[\mathbf{f}_* | X, \mathbf{y}, X_*] = K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} \mathbf{y} \tag{251}$$

and the conditional covariance is:

$$\text{cov}(\mathbf{f}_*) = K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} K(X, X_*) \tag{252}$$

The predictive distribution is a linear function of the observations $\mathbf{y}$. By defining $\boldsymbol{\alpha} \leftarrow (K + \sigma_n^2 I)^{-1}\mathbf{y}$, we predict the function as:

$$\bar{f}(x_*) = \sum_{i=1}^{N} \alpha_i k(x_i, x_*) \tag{253}$$

12.5 Kernel Functions

[!] Definition (First Set of Kernel Functions!):

- **Radial Basis Function (RBF) covariance function:** Defined with the length scale parameter l:

$$\text{cov}(f(x_p), f(x_q)) = \text{RBF}(x_p, x_q) = \exp\left(-\frac{1}{2}\frac{|x_p - x_q|^2}{l^2}\right)$$

- **Constant covariance function:** Defined with a constant parameter:

$$\text{cov}(f(x_p), f(x_q)) = \text{Constant}(x_p, x_q) = \text{constant}$$

- **Squared exponential (SE) covariance function:** With hyperparameters l^2 (lengthscale) and σ_f^2 (signal variance). It can be defined as a product kernel of the Constant and RBF kernels:

$$k(x_p, x_q) = \sigma_f^2 \exp\left(-\frac{1}{2}\frac{|x_p - x_q|^2}{l^2}\right) = \text{Constant}(x_p, x_q) \times \text{RBF}(x_p, x_q)$$

Definition (Further Kernel Functions):

- **ExpSineSquared kernel function:** Defined with parameters length scale l and periodicity $p > 0$ (where d is distance). Useful for periodic functions:

$$\text{cov}(f(x_q), f(x_r)) = \exp\left(-\frac{2\sin^2(\pi d(x_q, x_r)/p)}{l^2}\right)$$

- **Dot product kernel function:** Defined with inhomogeneity parameter σ_0. Useful to capture trends, often combined with an exponential kernel:

$$\text{cov}(f(x_p), f(x_q)) = \sigma_0 + x_p \cdot x_q$$

- **Rational Quadratic kernel function:** With hyperparameters l^2 (lengthscale) and mixture α. Represents the mixture of many RBF kernel length scales:

$$k(x_p, x_q) = \left(1 + \frac{d(x_p, x_q)^2}{2\alpha l^2}\right)^{-\alpha}$$

Definition (Matérn Kernel): Most kernel functions have many derivatives, but the Matérn kernel $\nu = 1.5$ has only the first derivative, allowing it to model less smooth functions. As $\nu \to \infty$, it approaches an RBF kernel. It is defined with parameters $\nu = k + \frac{1}{2}$ and length scale l:

$$k(x_p, x_q) = \frac{1}{2^{\nu-1}\Gamma(\nu)} \left(\frac{\sqrt{2\nu}}{l} d(x_p, x_q) \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}}{l} d(x_p, x_q) \right) \tag{254}$$

[Note: No need to memorize the following formulas] The Modified Bessel functions are defined as:

$$I_\alpha(x) = \sum_{m=0}^{\infty} \frac{1}{m!\Gamma(m + \alpha + 1)} \left(\frac{x}{2} \right)^{2m+\alpha} \tag{255}$$

$$K_\alpha(x) = \frac{\pi}{2} \frac{I_{-\alpha}(x) - I_\alpha(x)}{\sin(\alpha\pi)} \tag{256}$$

12.6 Marginal Likelihood

The parameters may be automatically tuned by gradient maximization of the marginal likelihood.

Lemma: The marginal log-likelihood is derived as follows. For noise-free in-sample prediction where $f \sim \mathcal{N}(0, \Sigma_*)$ and $\Sigma_* = K(X, X)$:

$$\log \Pr(\mathbf{y}|X) = \log \Pr(\mathbf{f}|X) = -\frac{1}{2}\mathbf{f}^\top \Sigma_*^{-1}\mathbf{f} - \frac{1}{2}\log|\Sigma_*| - \frac{N}{2}\log 2\pi \tag{257}$$

For noisy observations where $\mathbf{y}|\mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma_n^2 I)$ and $\mathbf{y} \sim \mathcal{N}(0, \Sigma_* + \sigma_n^2 I)$:

$$\log \Pr(\mathbf{y}|X) = -\frac{1}{2}\mathbf{y}^\top (\Sigma_* + \sigma_n^2 I)^{-1}\mathbf{y} - \frac{1}{2}\log|\Sigma_* + \sigma_n^2 I| - \frac{N}{2}\log 2\pi \tag{258}$$

The noise level may also be tuned by adding a `WhiteKernel`. The `WhiteKernel` outputs the noise level if and only if addressing the same variable (x_p, x_p); otherwise, it is 0.

12.7 Acquisition Functions in Bayesian Optimization

To minimize y with training data $X, \mathbf{y}$, the Expected Improvement (EI) formulation represents the maximum probability of improvement:

$$\mathrm{EI}(x) = \mathbb{E}[(\min(Y(X)) - Y(x))^+ | Y(X) = \mathbf{y}] = \mathbb{E}[(\min(\mathbf{y}) - Y(x))^+ | Y(X) = \mathbf{y}] \tag{259}$$

This can be solved analytically using the cumulative distribution function (Φ) and probability density function (ϕ) of a Gaussian distribution:

$$\mathrm{EI}(x) = (\min(\mathbf{y}) - \mu(x))\Phi\left(\frac{\min(\mathbf{y}) - \mu(x)}{\sigma(x)} \right) + \sigma(x)\phi\left(\frac{\min(\mathbf{y}) - \mu(x)}{\sigma(x)} \right) \tag{260}$$

For maximizing a target y (where $\xi > 0$ ignores small improvements):

$$\mathrm{EI}(x) = (\mu(x) - \max(\mathbf{y}) - \xi)\Phi\left(\frac{\mu(x) - \max(\mathbf{y}) - \xi}{\sigma(x)} \right) + \sigma(x)\phi\left(\frac{\mu(x) - \max(\mathbf{y}) - \xi}{\sigma(x)} \right) \tag{261}$$

Parallelization: The Constant Liar If direct formulas are unavailable, the following expression is solved by Markov Chain simulation:

$$\mathrm{EI}(x) = \mathbb{E}[(\min(Y(X)) - \min(Y(x^{(n+1)}), \ldots, Y(x^{(n+k)})))^+ | Y(X) = \mathbf{y}] \tag{262}$$

The algorithm estimates the observations y by an estimate (min, max, or mean) and runs the evaluation in parallel. The covariance is correctly estimated, while the mean must be corrected later.

Definition (Other Acquisition Functions):

- **Probability of Improvement:**

$$\mathrm{PI}(x_*) = \Pr(f(x_*) < \min(\mathbf{y})) = \Phi\left(\frac{\min(\mathbf{y}) - \mu(x_*)}{\sigma(x_*)}\right)$$

- **Lower Confidence Bound:**

$$\mathrm{LCB}(x) = \mu(x) - \kappa \cdot \sigma(x)$$

12.8 Gaussian Processes for Classification

Gaussian processes for classification are more complex and rely on an approximation. A latent function f is estimated as before and linked to the $(0, 1)$ interval using the sigmoid function. The log-marginal-likelihood does not have a closed analytical form anymore and can be approximated by a Hessian matrix. The algorithm works in $\mathcal{O}(N^3)$ time complexity.

12.9 POMDP Applications: Aircraft Collision Avoidance

Algorithms evaluate state and dynamics uncertainty to optimize maneuvers (turns, horizontal/vertical plane accelerations).

State variables (2D/3D):

- $r_x, r_y, (r_z)$: relative range states / intruder range components
- $v_{ox}, v_{oy}, (v_{oz})$: velocities for the ownship
- $v_{ix}, v_{iy}, (v_{iz})$: velocities for the intruder
- $d_x, d_y, (d_z)$: absolute displacement from the desired trajectory

The desired trajectory is normalized to unit velocity in the x-axis and zero velocity in the y-axis.

MDP Transitions: The prediction horizon is very short (updates every 0.1 to 1 seconds). Simple update equations are sufficient:

- $\dot{r}_x = v_{ix} - v_{ox}$
- $\dot{r}_y = v_{iy} - v_{oy}$
- $\dot{v}_{ox} = a_x + N_{ox}$
- $\dot{v}_{oy} = a_y + N_{oy}$
- $\dot{v}_{ix} = N_{ix}$
- $\dot{v}_{iy} = N_{iy}$
- $d_x = v_{tx} - v_{ox}$
- $d_y = v_{ty} - v_{oy}$

Variables a_x and a_y denote ownship acceleration. N_* is noise applied to the ownship or intruder in the x and y axes (e.g., N_o with $\mu = 0, 0.30\mathrm{s}^{-2}$ and N_i with $\mu = 0, 0.45\mathrm{s}^{-2}$).

Bellman Update: Transitioning from state s with action a to state s':

$$Q[s, a] \leftarrow R(s, a) + \gamma \sum_{s'} T(s'|s, a) \max_{a'} Q[s', a'] \tag{263}$$

Reward Function: The minimum reward is $R_{\min}$ (applied for collisions or physically impossible states). The function prefers no acceleration, long distances to the intruder, and short distances to the desired trajectory.

$$R(s,a) = \max\left[R_{\min}, -(k_{ax}|a_x| + k_{ay}|a_y|) - k_s\frac{1}{k_{rx}r_x^2 + k_{ry}r_y^2} - K_T(k_{dx}d_x^2 + k_{dy}d_y^2)\right] \quad (264)$$

Parameters K_s, K_T, and $R_{\min}$ are learned weights. The k weights are set to 1.

Reward Tuning with Bayesian Optimization: We tune the parameters $R_P = (K_T, K_s, R_{\min})$. The objective function models $F(R_P)$ where β weights the two goals:

$$F(R_P) = \beta \times (r_{5\%\text{CPA}})^{-1} + (1-\beta) \times \mu_{\text{dev}} \quad (265)$$

Here, CPA is the closest point of approach, and μ_{dev} is the mean deviation distance from the desired trajectory. A Gaussian process models $F(R_P)$, determining the point with the largest expected improvement $E[I(F(R_P))]$ over the current minimum. This set of R_P is passed to QMDP.

Beliefs and Policy Calculation: State uncertainty is incorporated only when actions are selected. A set of potential states (beliefs b) is calculated from observations at each step.

$$\pi(b) = \max_a\left[\sum_k Q(s^{(k)}, a)b^{(k)}\right] \quad (266)$$

The value $Q(s^{(k)}, a)b^{(k)}$ is approximated from QMDP solutions using rectangular interpolation (between 2^n nearest neighbors) or simplex interpolation (between $n+1$ nearest neighbors). Value Iteration utilizes $\gamma \leftarrow 0.99$.

13 Machine Learning: Support Vector Machines

13.1 Linear Models for Classification

- The feature space X is separated by a hyperplane (or a set of hyperplanes for more than two classes).
- **Linear Discriminant Analysis (LDA)** is optimal if assumptions are met, specifically $N(\mu_c, \sigma_c^2)$.
- **Logistic Regression** combines a logistic function and a linear model.
- **Perceptron** (a neural network with one neuron) finds a separating hyperplane if the data is linearly separable. The exact position of the hyperplane depends on the initial parameters.

13.2 Optimal Separating Hyperplane (Separable Case)

We define the Optimal Separating Hyperplane as a separating hyperplane with maximal free space M without any data point falling inside this margin around the hyperplane.

The hyperplane is formally defined as:

$$\beta_0 + \beta^T x = 0 \tag{267}$$

The margin is derived as $M = \frac{1}{\|\beta\|}$. The optimization problem seeks to maximize this margin:

$$\max_{\beta,\beta_0,\|\beta\|=1} M \tag{268}$$

subject to the condition $y_i(x_i^T\beta + \beta_0) \geq M$ for all $i = 1, \ldots, N$, where $y_i \in \{-1, 1\}$ encodes the binary target class.

By redefining the norm constraint, the condition $\|\beta\| = 1$ can be integrated into the inequality:

$$\frac{1}{\|\beta\|} y_i(x_i^T\beta + \beta_0) \geq M \tag{269}$$

Since any positively scaled multiple of β and β_0 satisfying these inequalities will also work, we can conveniently set $\|\beta\| = \frac{1}{M}$. This transforms the problem into a convex optimization problem:

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 \tag{270}$$

subject to $y_i(x_i^T\beta + \beta_0) \geq 1$ for all $i = 1, \ldots, N$.

To find the saddle point with respect to β and β_0, we construct the Lagrange function:

$$L_P = \frac{1}{2}\|\beta\|^2 - \sum_{i=1}^{N} \alpha_i[y_i(x_i^T\beta + \beta_0) - 1] \tag{271}$$

Setting the derivatives to zero yields:

- $\beta = \sum_{i=1}^{N} \alpha_i y_i x_i$
- $0 = \sum_{i=1}^{N} \alpha_i y_i$

Substituting these back into L_P gives the Wolfe dual formulation:

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k x_i^T x_k \tag{272}$$

which is maximized in the positive orthant subject to $\alpha_i \geq 0$.

The solution must satisfy the Karush-Kuhn-Tucker (KKT) conditions:

$$\alpha_i[y_i(x_i^T\beta + \beta_0) - 1] = 0 \tag{273}$$

For any $\alpha_i > 0$, the corresponding data point x_i lies exactly on the boundary, meaning $y_i(x_i^T\beta + \beta_0) - 1 = 0$. These specific points are called **support vectors**. For all points strictly outside the boundary, $\alpha_i = 0$.

New observations are classified using the function:

$$\hat{G}(x) = \text{sign}(x^T\beta + \beta_0) \tag{274}$$

where $\beta = \sum_{i=1}^{N} \alpha_i y_i x_i$ and the intercept is calculated using any support vector ($\alpha_s > 0$) as $\beta_0 = y_s - x_s^T\beta$.

13.3 Optimal Separating Hyperplane (Non-separable Case)

For non-separable data, the algorithm must allow for incorrectly classified instances by introducing slack variables $\xi = (\xi_1, \ldots, \xi_N)$:

- ξ_i represents the proportional distance of x_i from the margin boundary if it falls on the wrong side.
- $\xi_i = 0$ for points on the correct side of the margin.

The total amount of error is bounded by requiring $\sum_{i=1}^{N} \xi_i \leq K$. The objective becomes:

$$\max_{\beta, \beta_0, \|\beta\|=1} M \tag{275}$$

subject to $y_i(x_i^T\beta + \beta_0) \geq M(1 - \xi_i)$, where $\xi_i \geq 0$ and $\sum_{i=1}^{N} \xi_i \leq K$.

Equivalently, using a tuning parameter γ instead of the strict constant limit K, we minimize:

$$\min_{\beta, \beta_0} \frac{1}{2}\|\beta\|^2 + \gamma \sum_{i=1}^{N} \xi_i \tag{276}$$

subject to $\xi_i \geq 0$ and $y_i(x_i^T\beta + \beta_0) \geq (1 - \xi_i)$.

- $\gamma = \infty$ reverts the formulation to the strictly separable case.
- Large γ: Results in fewer support vectors (and a complex boundary when using kernels).
- Small γ: Results in a robust model with many support vectors (yielding a smoother boundary). The value of γ is typically selected via cross-validation.

The primal Lagrangian with multipliers α_i and μ_i is defined as:

$$L_P = \frac{1}{2}\|\beta\|^2 + \gamma \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] - \sum_{i=1}^{N} \mu_i \xi_i \tag{277}$$

Setting the derivatives with respect to β, β_0, ξ_i to zero produces:

- $\beta = \sum_{i=1}^{N} \alpha_i y_i x_i$
- $0 = \sum_{i=1}^{N} \alpha_i y_i$
- $\alpha_i = \gamma - \mu_i$

Substituting these back provides the dual formulation, which aims to maximize L_D:

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k x_i^T x_k \tag{278}$$

subject to $0 \leq \alpha_i \leq \gamma$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$.

The solution adheres to the KKT conditions for the non-separable model:

- $\alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] = 0$
- $\mu_i \xi_i = 0$
- $y_i(x_i^T\beta + \beta_0) - (1 - \xi_i) \geq 0$

The optimal boundary points ($\hat{\alpha}_i > 0$) define the model:

- Points exactly on the margin boundary: $\hat{\xi}_i = 0$, leading to $0 < \hat{\alpha}_i < \gamma$.
- Points on the wrong side of the margin: $\hat{\xi}_i > 0$, forcing $\hat{\alpha}_i = \gamma$.

Any point with $\hat{\xi}_i = 0$ can be used to calculate the intercept $\hat{\beta}_0$. Usually, an average over all such eligible support vectors is utilized.

13.4 Kernel Functions

Support Vector Machines extend to non-linear boundaries by replacing the standard inner product $\langle x_i, x \rangle$ with a kernel function $K(x_i, x)$ that acts as a dot product in an expanded, higher-dimensional feature space.

The generalized classification function is:

$$\hat{f}(x) = \sum_{i=1}^{N} \hat{\alpha}_i y_i K(x_i, x) + \hat{\beta}_0 \tag{279}$$

Standard kernel functions include:

- **Polynomial of degree d:** $K(x, x') = (1 + \langle x, x' \rangle)^d$
- **Radial Basis Function (RBF):** $K(x, x') = \exp\left(\frac{-\|x - x'\|^2}{l}\right)$
- **Neural Network (Sigmoid):** $K(x, x') = \tanh(\kappa_1 \langle x, x' \rangle + \kappa_2)$

Example of Feature Mapping (Degree-2 Polynomial for 2D Input):

$$K(x, x') = (1 + \langle x, x' \rangle)^2 = 1 + 2x_1x_1' + 2x_2x_2' + (x_1x_1')^2 + (x_2x_2')^2 + 2x_1x_1'x_2x_2' \tag{280}$$

This explicitly maps into a space of $M = 6$ features: $h_1(x) = 1$, $h_2(x) = \sqrt{2}x_1$, $h_3(x) = \sqrt{2}x_2$, $h_4(x) = x_1^2$, $h_5(x) = x_2^2$, $h_6(x) = \sqrt{2}x_1x_2$. The evaluation never explicitly computes $h(x)$, but merely evaluates the scalar product $\langle h(x), h(x_i) \rangle$ through the kernel.

String Kernels and Protein Classification: For sequence data, considering all possible sequences $a \in \mathcal{A}_m$ of length m:

- We define a feature map $\Phi_m(x) = \{\phi_a(x)\}_{a \in \mathcal{A}_m}$
- $\phi_a(x)$ tracks the number of occurrences of substring a in x.
- The kernel function is the inner product of the feature maps:

$$K_m(x_1, x_2) = \langle \Phi_m(x_1), \Phi_m(x_2) \rangle \tag{281}$$

13.5 SVM as a Penalization Method

An SVM can be viewed similarly to smoothing splines by framing it as a regularized loss minimization problem. For a fitted function $f(x) = h(x)^T\beta + \beta_0$, using the **Hinge Loss**:

$$L(y, f) = [1 - yf]_+ \tag{282}$$

The formulation corresponds to solving:

$$\min_{\beta,\beta_0} \sum_{i=1}^{N} [1 - y_i f(x_i)]_+ + \lambda\|\beta\|^2 \tag{283}$$

which is structurally analogous to a smoothing spline penalty $\min_{\alpha,\alpha_0} \sum_{i=1}^{N}[1 - y_i f(x_i)]_+ + \lambda\alpha^T K\alpha$, where $J(f) = \alpha^T K\alpha$ serves as the smoothing penalty.

Comparison of Loss Functions $L[y, f(x)]$ and their Minimizing Functions:

- **Binomial Deviance:** $\log[1 + e^{-yf(x)}]$ minimized by $f(x) = \log \frac{\Pr(Y=+1|x)}{\Pr(Y=-1|x)}$
- **SVM Hinge Loss:** $[1 - yf(x)]_+$ minimized by $f(x) = \text{sign}\left[\Pr(Y = +1 \mid x) - \frac{1}{2}\right]$
- **Squared Error:** $[1 - yf(x)]^2$ minimized by $f(x) = 2\Pr(Y = +1 \mid x) - 1$
- **"Huberised" Square Hinge Loss:**

$$\begin{cases} -4yf(x) & \text{if } yf(x) < -1 \\ [1 - yf(x)]_+^2 & \text{otherwise} \end{cases}$$

  minimized by $f(x) = 2\Pr(Y = +1 \mid x) - 1$

13.6 SVM Complexity and Model Selection

The computational complexity of the SVM solver is roughly $\mathcal{O}(m^3 + mN + mpN)$, where m denotes the number of support vectors, N is the dataset size, and p is the dimensionality.

When evaluated against added pure noise features, SVM classifiers may severely overfit compared to additive spline techniques (such as MARS and BRUTO) which inherently possess the ability to ignore noisy dimensions. Optimal performance hinges significantly on the precise tuning of cost C (or inversely λ) and the kernel parameters (such as γ for the RBF lengthscale).

13.7 Support Vector Regression (SVR)

In a regression setting, we aim to fit a continuous function $f(x) = x^T\beta + \beta_0$. The model replaces the standard squared error with the ϵ-insensitive error function:

$$V_\epsilon(r) = \begin{cases} 0 & \text{if } |r| < \epsilon \\ |r| - \epsilon & \text{otherwise} \end{cases} \tag{284}$$

The optimization target to minimize is:

$$\text{SVR}(\beta, \beta_0) = \sum_{i=1}^{N} V_\epsilon(y_i - f(x_i)) + \frac{\lambda}{2}\|\beta\|^2 \tag{285}$$

The regression solution relies on a set of dual non-negative parameters $\hat{\alpha}_i, \hat{\alpha}_i^* \geq 0$, leading to:

$$\hat{\beta} = \sum_{i=1}^{N} (\hat{\alpha}_i^* - \hat{\alpha}_i) x_i \tag{286}$$

$$\hat{f}(x) = \sum_{i=1}^{N} (\hat{\alpha}_i^* - \hat{\alpha}_i)\langle x, x_i \rangle + \beta_0 \tag{287}$$

This is obtained by solving the corresponding quadratic programming problem:

$$\min_{\alpha_i, \alpha_i^*} \epsilon \sum_{i=1}^{N} (\alpha_i^* + \alpha_i) - \sum_{i=1}^{N} y_i(\alpha_i^* - \alpha_i) + \frac{1}{2} \sum_{i,i'=1}^{N} (\alpha_i^* - \alpha_i)(\alpha_{i'}^* - \alpha_{i'})\langle x_i, x_{i'} \rangle \tag{288}$$

subject to the constraints:

- $0 \leq \alpha_i, \alpha_i^* \leq 1/\lambda$
- $\sum_{i=1}^{N} (\alpha_i^* - \alpha_i) = 0$
- $\alpha_i \alpha_i^* = 0$

The support vectors for the regression task are purely the observations maintaining a non-zero coefficient sequence, that is, where $(\hat{\alpha}_i^* - \hat{\alpha}_i) \neq 0$. If the response variable y is scaled beforehand, the default value for the epsilon parameter ϵ is often sufficient, while the penalty term λ is usually optimized through cross-validation.

14 Machine Learning: Further ML Methods

14.1 Overview of Further ML Methods

- Linear Projections
- Principal Component Analysis (PCA): finding 'the most spread variance' directions.
- Sparse PCA.
- Partial Least Squares [**Not mentioned here**].
- Archetypal analysis: focuses on extremes instead of 'centers' from clustering; data is modeled as a linear combination of archetypes.
- Nonnegative Matrix Factorization (NMF): acts as a 'linear r-dimensional autoencoder'.
- Factor analysis: a view on 'independent factors' observed via a linear combination mixture with a gaussian noise.
- Independent Component Analysis (ICA): splits the signal according to non-gaussian features (maximum divergence from gaussian).
- Procrustes transformation curve fitting.
- Principal curves and surfaces: uses predefined coordinate functions $f_j(\lambda)$ and a curve parameter λ.
- Kernel PCA.
- Spectral Clustering.

14.2 Principal Component Analysis (PCA), Curves, and Surfaces

The principal components of a set of data in $\mathbb{R}^p$ provide a sequence of best linear approximations to that data of all ranks $q \leq p$.

- $p \times q$ represents the dimensionality.
- μ is a location vector in $\mathbb{R}^p$.
- V_q is a matrix with q orthogonal unit vectors as columns.
- λ is a q-vector of parameters.

An affine hyperplane of rank q is represented by:

$$f(\lambda) = \mu + V_q \lambda \tag{289}$$

We minimize the reconstruction error by least squares:

$$\min_{\mu, \{\lambda_i\}} \sum_{i=1}^{N} \|x_i - \mu - V_q \lambda_i\|^2 \tag{290}$$

We can partially optimize this with respect to μ and λ_i:

$$\hat{\mu} = \overline{x} \tag{291}$$

$$\hat{\lambda}_i = V_q^T (x_i - \overline{x}) \tag{292}$$

This leaves us to find the orthogonal matrix V_q:

$$\min_{V_q} \sum_{i=1}^{N} \|(x_i - \overline{x}) - V_q V_q^T (x_i - \overline{x})\|^2 \tag{293}$$

To simplify formulas, we typically center the data such that $\overline{x} = 0$.

For a handwritten digit '3' example, given $x \in \mathbb{R}^{256}$, the projection on the first two components is:

$$\hat{f}(\lambda) = \overline{x} + \lambda_1 v_1 + \lambda_2 v_2 \tag{294}$$

14.3 Sparse Principal Components

We often interpret PCA by examining loadings (direction vectors v_j). This interpretation is easier if the loadings are sparse.

Definition (Sparse PCA for a single component): The sparse principal component technique solves for a single component by minimizing:

$$\min_{\theta, v} \sum_{i=1}^{N} \|x_i - \theta v^T x_i\|_2^2 + \lambda \|v\|_2^2 + \lambda_1 \|v\|_1 \tag{295}$$

subject to $\|\theta\|_2 = 1$.

- If both $\lambda = \lambda_1 = 0$ and $N > p$, then $v = \theta$, which is the largest principal component direction.
- When $p \gg N$, the solution may not be unique unless $\lambda > 0$.
- For $\lambda > 0$ and $\lambda_1 = 0$, the solution is proportional to the largest principal component direction.

Sparse PCA for multiple components: Minimizes over Θ and V ($p \times K$ matrices):

$$\min_{\Theta, V} \sum_{i=1}^{N} \|x_i - \Theta V^T x_i\|_2^2 + \lambda \sum_{k=1}^{K} \|v_k\|_2^2 + \sum_{k=1}^{K} \lambda_{1k} \|v_k\|_1 \tag{296}$$

subject to $\Theta^T \Theta = I_K$.

14.4 Archetypal Analysis

Archetypal Analysis approximates data points by a linear combination of prototypes (archetypes) that are themselves linear combinations of data points. Each data point is approximated by a convex combination of prototypes, forcing the prototypes to lie on the convex hull of the data cloud. It functions as a "linear autoencoder" of dimension r.

Definition (Archetypal Analysis):

- A non-negative $N \times p$ data matrix X is modeled as $X \sim WH$.
- $H = BX$ is an $r \times p$ matrix of r archetypes (rows of H).
- B is an $r \times N$ matrix where $b_{ki} \geq 0$ and $\sum_{i=1}^{N} b_{ki} = 1$ for each k.
- W is an $N \times r$ matrix where $w_{ik} \geq 0$ and $\sum_{k=1}^{r} w_{ik} = 1$ for each i.

We minimize over W and B:

$$J(W, B) = \|X - WH\|^2 = \|X - WBX\|^2 \tag{297}$$

This is minimized in an alternating fashion, with each separate minimization involving a convex optimization that converges to a local minimum.

14.5 Non-negative Matrix Factorization (NMF)

Definition (Non-negative Matrix Factorization):

- A centered $N \times p$ data matrix X is modeled as $X \sim WH$.
- W is an $N \times r$ matrix and H is an $r \times p$ matrix.
- $r \leq \max(N, p)$.
- Assumption: $X_{ij}, W_{ik}, h_{kj} \geq 0$.

NMF assumes X_{ij} has a Poisson distribution with mean $(WH)_{ij}$. We maximize the log-likelihood over W and H:

$$L(W, H) = \sum_{i=1}^{N} \sum_{j=1}^{p} [x_{ij} \log(WH)_{ij} - (WH)_{ij}] \tag{298}$$

Algorithm: NMF Given X as centered data:

1. Repeat until convergence:

$$w_{ik} \leftarrow w_{ik} \frac{\sum_{j=1}^{p} h_{kj} x_{ij} / (WH)_{ij}}{\sum_{j=1}^{p} h_{kj}}$$

$$h_{kj} \leftarrow h_{kj} \frac{\sum_{i=1}^{N} w_{ik} x_{ij} / (WH)_{ij}}{\sum_{i=1}^{N} b_{ik}}$$

2. Return W, H.

The sklearn `decomposition.NMF` implementation has the objective function:

$$\begin{aligned} 0.5\|X - WH\|_{\text{Fro}}^2 + \alpha_W l1_{\text{ratio}} p \|\text{vec}(W)\|_1 + 0.5\alpha_W (1 - l1_{\text{ratio}}) p \|W\|_{\text{Fro}}^2 \\ + \alpha_H l1_{\text{ratio}} N \|\text{vec}(H)\|_1 + 0.5\alpha_H (1 - l1_{\text{ratio}}) N \|H\|_{\text{Fro}}^2 \end{aligned} \tag{299}$$

Where the L_1 and Frobenius norms are defined as:

$$\|\text{vec}(W)\|_1 = \sum_{i,j} |W_{i,j}| \tag{300}$$

$$\|W\|_{\text{Fro}}^2 = \sum_{i,j} W_{i,j}^2 \tag{301}$$

14.6 Latent Variables and Factor Analysis

Take the singular value decomposition (SVD) where columns of X have zero mean, D is a diagonal matrix, and U is orthogonal:

$$X = UDV^T \tag{302}$$

X has a latent variable decomposition:

$$X = SA^T \tag{303}$$

Where:

- $S = \sqrt{N}U$
- $A^T = \frac{1}{\sqrt{N}} DV^T$

Each column of X is a linear combination of the columns of S. The columns of S have zero mean, are uncorrelated, and have unit variance, meaning $\text{Cov}(S) = I$.

For any orthogonal $p \times p$ matrix R:

$$X = AS = AR^T RS = A^* S^* \tag{304}$$

Factor Analysis Model: With rank $q < p$, a factor analysis model has the form:

$$X = AS + \epsilon \tag{305}$$

Expanding this yields:

$$X_j = a_{j1} S_1 + a_{j2} S_2 + \cdots + a_{jq} S_q + \epsilon_j \tag{306}$$

Where:

- S is a vector of $q < p$ underlying latent variables or factors.
- A is a $p \times q$ matrix of factor loadings.
- ϵ_j are uncorrelated zero-mean disturbances.

The parameters reside in the covariance matrix:

$$\Sigma = AA^T + D_\epsilon \tag{307}$$

Where $D_\epsilon = \text{diag}[\text{Var}(\epsilon_1), \text{Var}(\epsilon_2), \ldots, \text{Var}(\epsilon_p)]$.

14.7 Independent Component Analysis (ICA)

ICA assumes multivariate data forms multiple indirect measurements from underlying sources:

$$X_j = a_{j1} S_1 + a_{j2} S_2 + \cdots + a_{jp} S_p \tag{308}$$

Unlike factor analysis, S are assumed to be statistically independent (all orders of interactions) rather than just uncorrelated (second order interactions).

We assume X has been whitened to have $\text{Cov}(X) = I$. The simplest method is multiplying by $W = \Sigma^{-1/2}$, typically achieved via the SVD to $D^{-1/2} V^T$. Because $\text{Var}(S) = I$, A is orthogonal. ICA searches for an orthogonal S such that $S = A^T X$ are independent (non-Gaussian) components.

Entropy and Mutual Information: Entropy is defined as:

$$H(Y) = -\int g(y) \log g(y) dy \tag{309}$$

Mutual information is defined as:

$$I(Y) = \sum_{j=1}^{p} H(Y_j) - H(Y) = \sum_{j=1}^{p} H(Y_j) - H(X) - \log|\det(A)| \tag{310}$$

Since $\text{Cov}(X) = I$, $Y = A^T X$, and A is orthogonal, the $\log|\det(A)|$ term drops out:

$$I(Y) = \sum_{j=1}^{p} H(Y_j) - H(X) \tag{311}$$

We search for A to minimize $I(Y) = I(A^T X)$. This orthogonal transformation leads to the most independence between its components by minimizing the sum of the entropies, which amounts to maximizing their departures from Gaussianity.

Negentropy: For each Y_j, let Z_j be a Gaussian random variable with the same variance as Y_j. Negentropy $J(Y_j)$ measures the departure of Y_j from Gaussianity:

$$J(Y_j) = H(Z_j) - H(Y_j) \tag{312}$$

This can be approximated by:

$$J(Y_j) \approx [\mathbb{E}G(Y_j) - \mathbb{E}G(Z_j)]^2 \tag{313}$$

where $G(u) = \frac{1}{a}\log\cosh(au)$ for $1 \leq a \leq 2$.

Algorithm: FastICA Given centered and whitened data X with $\text{Cov}(X) = I$ (typically achieved via SVD: $X \leftarrow \sqrt{D}U$ from $X = UDV^T$), $q \leq p$ components, and parameter $a \in [1, 2]$:

1. Initialize $w_1, \ldots, w_q$ as random N-dimensional weight vectors.
2. For $l = 1, \ldots, q$ do:
3. Repeat until convergence:

$$w_l^+ \leftarrow \sum_{i=1}^{N} x \tanh(a w_l^T x) - \left(\sum_{i=1}^{N} \frac{1}{\cosh^2(w_l^T x)}\right) w_l$$

$$w_l \leftarrow w_l^+ - \sum_{j=1}^{l-1} w_l^T w_j w_j \quad \# \text{ orthogonalize to previous components}$$

$$w_l \leftarrow \frac{w_l}{\sqrt{w_l^T w_l}} \quad \# \text{ normalize}$$

14.8 Procrustes Transformations

Assume shapes X_1 and X_2 are $N \times 2$ matrices with column means $\overline{x}_1$ and $\overline{x}_2$, and they are centered to $\tilde{X}_1$ and $\tilde{X}_2$.

Let R be an orthonormal $p \times p$ matrix. We compute the SVD:

$$\tilde{X}_1^T \tilde{X}_2 = UDV^T \tag{314}$$

And calculate the optimal rotation and a p-vector of location coordinates μ:

$$\hat{R} \leftarrow UV^T \tag{315}$$

$$\mu \leftarrow \overline{x}_2 - \hat{R}\overline{x}_1 \tag{316}$$

We minimize the Procrustes distance:

$$\min_{\mu, R} \|X_2 - (X_1 R + \mathbf{1}\mu^T)\|_F^2 \tag{317}$$

Where the squared Frobenius matrix norm is defined as:

$$\|X\|_F^2 = \text{trace}(X^T X) = \sum_{i=1}^{N}\sum_{j=1}^{p} |x_{ij}|^2 \tag{318}$$

[! Important] From now on, we assume the data are centered.

Definition (Procrustes Average): The Procrustes average of a collection of L shapes is M that minimizes:

$$\min_{\{R_l\}_1^L, M} \sum_{l=1}^{L} \|X_l R_l - M\|_F^2 \tag{319}$$

Algorithm: Procrustes Average Given $N \times p$ shapes $\{X_l\}_{l=1}^L$:

1. $M \leftarrow X_1$ (Initialize the average)
2. Repeat until convergence:
3. With M fixed, solve L Procrustes rotations $\hat{R}_l$:

$$X_l' \leftarrow X_l \hat{R}_l$$

4. Update the average:

$$M \leftarrow \frac{1}{L} \sum_{l=1}^{L} X_l'$$

14.9 Principal Curves and Surfaces

To find a principal curve $f(\lambda)$ of a distribution, we define its coordinate functions $f(\lambda) = [f_1(\lambda), f_2(\lambda), \ldots, f_p(\lambda)]$ and let $X^T = (X_1, \ldots, X_p)$.

Algorithm: Principal Curve Given coordinate functions $f(\lambda)$ and data X:

1. Repeat until convergence:

$$\hat{f}_j(\lambda) \leftarrow \mathbb{E}[X_j \mid \lambda(X) = \lambda] \quad \text{for } j = 1, \ldots, p$$

$$\hat{\lambda}_f(x) \leftarrow \arg\min_{\lambda'} \|x - \hat{f}(\lambda')\|^2$$

A scatterplot smoother is used to estimate the conditional expectations by smoothing each X_j as a function of the arc-length $\lambda(\hat{X})$.

14.10 Kernel Principal Components

We select a kernel function, such as the radial kernel given scale parameter c:

$$K(x_i, x_{i'}) = e^{-\frac{\|x_i - x_{i'}\|^2}{c}} \tag{320}$$

We set $M = \mathbf{1}\mathbf{1}^T/N$ and calculate the double-centered version of K:

$$\tilde{K} = (I - M)K(I - M) = UD^2U^T \tag{321}$$

The principal component variables are $Z = UD$. The elements of the m-th component z_m (m-th column of Z) can be written (up to centering) as:

$$z_{im} = \sum_{j=1}^{N} \alpha_{jm} K(x_i, x_j) \tag{322}$$

where:

$$\alpha_{jm} = \frac{u_{jm}}{d_m} \tag{323}$$

14.11 Spectral Clustering

Spectral clustering puts close points into the same cluster by forming a weighted adjacency graph for data samples.

Algorithm: Spectral Clustering Given X as N points in $\mathbb{R}^p$, scale $c > 0$, and number of clusters $k > 0$:

1. Calculate the similarity matrix:

$$s_{ii'} \leftarrow \exp(-d(i, i')^2/c)$$

2. Initialize W, G as zero matrices of dimension $N \times N$.

3. For i, i' symmetric nearest neighbors do:

$$W_{ii'} \leftarrow s_{ii'} \quad \# \text{ connect them}$$

4. For $i \in X$ do:

$$g_{ii} \leftarrow \sum_{i'} W_{ii'} \quad \# \text{ compute the degree of vertex } i$$

5. Compute the unnormalized graph Laplacian:

$$L \leftarrow G - W$$

(Alternatively, the normalized graph Laplacian $\tilde{L} \leftarrow I - G^{-1}W$)

6. Find m eigenvectors $Z_{N \times m}$ with the smallest eigenvalues of L.

7. Return the rows of $Z_{N \times m}$ clustered by standard k-means.

For any vector f:

$$f^T L f = \sum_{i=1}^{N} g_{ii} f_i^2 - \sum_{i=1}^{N} \sum_{i'=1}^{N} f_i f_{i'} W_{ii'} = \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} W_{ii'} (f_i - f_{i'})^2 \tag{324}$$

For any graph, $\mathbf{1}^T L \mathbf{1} = 0$. For a graph with m connected components reordered so that L is block diagonal, L will have m eigenvectors with an eigenvalue of exactly zero. In practice, zero eigenvalues are approximated by small eigenvalues.