Note and Formula of DDA4210

softstar

January 27, 2026

1 Lec1: introduction

There are no formulas in Lecture 1. Good luck with the rest!

2 Lec2: Advanced Ensemble Learning

2.1 Gradient Boosting:

Boosting is an ensemble technique that combines multiple weak learners to create a strong learner. The main idea is to train models sequentially, with each model focusing on the errors made by the previous ones. (用人话说，就是一波接一波地训练模型，每一波都专注于纠正前一波的错误，从而逐步提升整体的预测能力。)

- **Final Model:**
$$H_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$$
  - The weighted sum of all weak learners(加权和).
  - $H_T(\mathbf{x})$ is the final model after T rounds.
  - $h_t(\mathbf{x})$ is the t-th weak learner.
  - α_t is the weight of the t-th weak learner.
- **Loss Function:**
$$\mathcal{L}(H) := \frac{1}{n}\sum_{i=1}^{n} \ell(H(\mathbf{x}_i), y_i)$$
  - **Each Step:** We want to add a function h to minimize the loss as fast as possible. Using first-order Taylor expansion(一阶泰勒展开):
$$\mathcal{L}(H + \alpha h) \approx \mathcal{L}(H) + \alpha\langle \nabla \mathcal{L}(H), h\rangle$$
  - **Find h:** Minimize $\langle \nabla \mathcal{L}(H), h\rangle$, i.e.:
$$h = \arg\min_{h \in \mathbb{H}} \sum_{i=1}^{n} \frac{\partial \mathcal{L}}{\partial [H(\mathbf{x}_i)]} h(\mathbf{x}_i)$$
    * Here $\frac{\partial \mathcal{L}}{\partial [H(\mathbf{x}_i)]}$ is the gradient of the loss function w.r.t. the current model's output on the i-th sample.

* We train h to fit these **negative gradients(负梯度)**.

- **GBR with squared error loss:** For regression tasks with squared error loss:

$$\ell(H(\mathbf{x}_i), y_i) = \sum_{i=1}^{n} (y_i - H(\mathbf{x}_i))^2$$

  - Solve $h_{t+1} = \arg\min_{h\in\mathbb{H}} \sum_{i=1}^{n} q_i h(\mathbf{x}_i)$, where $q_i = \frac{\partial \mathcal{L}}{\partial [H(\mathbf{x}_i)]}$
  - Let $\sum_{i=1}^{n} h^2(\mathbf{x}_i) = \text{constant}$ (we can always normalize the predictions(对结果归一化处理)) and replace q_i with $-2r_i$. We have

$$\begin{aligned}
h_{t+1} &= \arg\min_{h\in\mathbb{H}} \sum_{i=1}^{n} q_i h(\mathbf{x}_i) \\
&= \arg\min_{h\in\mathbb{H}} -2\sum_{i=1}^{n} r_i h(\mathbf{x}_i) \\
&= \arg\min_{h\in\mathbb{H}} \sum_{i=1}^{n} \left(r_i^2 - 2r_i h(\mathbf{x}_i) + (h(\mathbf{x}_i))^2\right) \\
&= \arg\min_{h\in\mathbb{H}} \sum_{i=1}^{n} (h(\mathbf{x}_i) - r_i)^2
\end{aligned} \tag{1}$$

  - We train h_{t+1} to predict r_i, which are from the old model H_t.
  - The gradient is:

$$\frac{\partial \mathcal{L}}{\partial [H(\mathbf{x}_i)]} = -2(y_i - H(\mathbf{x}_i))$$

  - So we fit h to the residuals:

$$r_i = y_i - H(\mathbf{x}_i)$$

  - Update the model:

$$H_t(\mathbf{x}) = H_{t-1}(\mathbf{x}) + \alpha_t h_t(\mathbf{x})$$

- **GBR with Absolute loss (更鲁棒):** For regression tasks with absolute loss:
  - Square loss is easy to deal with mathematically but not robust to outliers.
  - Absolute loss (more robust to outliers):

$$\ell(y, \hat{y}) = |y - \hat{y}|$$

  - The gradient is $q_i = \frac{\partial \mathcal{L}}{\partial H(\mathbf{x}_i)} = -\text{sign}(y_i - H(\mathbf{x}_i))$.
  - Then fit h on $-q_i$, $i = 1, 2, \ldots, n$. (no longer the residuals, different from using the squared loss)

- **GBR with Huber loss(更更鲁棒):**
  - Huber loss (more robust to outliers):

$$\ell(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & |y - \hat{y}| \le \delta \\ \delta(|y - \hat{y}| - \delta/2) & |y - \hat{y}| > \delta \end{cases}$$

– The gradient is:

$$\frac{\partial \mathcal{L}}{\partial H(\mathbf{x}_i)} = \begin{cases} -(y_i - H(\mathbf{x}_i)) & |y_i - H(\mathbf{x}_i)| \le \delta \\ -\delta \operatorname{sign}(y_i - H(\mathbf{x}_i)) & |y_i - H(\mathbf{x}_i)| > \delta \end{cases}$$

- **GBM for classification:**
  – Predict probability of K classes:

$$p_k(\mathbf{x}) = \frac{\exp(h^{(k)}(\mathbf{x}))}{\sum_{c=1}^{K} \exp(h^{(c)}(\mathbf{x}))} \triangleq \hat{y}^{(k)}, \quad k = 1, 2, \ldots, K$$

  – Loss: $\mathcal{L}(H) = \sum_{i=1}^{n} \ell(\mathbf{y}_i, \hat{\mathbf{y}}_i)$ (e.g. cross-entropy or KL divergence)
  – Initialize $H^{(1)}, H^{(2)}, \ldots, H^{(K)}$, iterate until converge or reach max T:
    1. Calculate negative gradients for every class:

$$-g_k(\mathbf{x}_i) = -\frac{\partial \mathcal{L}}{\partial [H^{(k)}(\mathbf{x}_i)]}, \quad i = 1, \ldots, n, \; k = 1, \ldots, K$$

    2. Fit $h^{(k)}$ to $-g_k(\mathbf{x}_i)$ (负梯度), $k = 1, 2, \ldots, K$.
    3. Update: $H^{(k)} \leftarrow H^{(k)} + \alpha h^{(k)}$, $k = 1, 2, \ldots, K$.

2.2 AdaBoost

A special case of gradient boosting with exponential loss.

- Exponential loss (learns α adaptively):

$$\mathcal{L}(H) = \sum_{i=1}^{n} e^{-y_i H(\mathbf{x}_i)}$$

- Gradient: $q_i = \frac{\partial \mathcal{L}}{\partial H(\mathbf{x}_i)} = -y_i e^{-y_i H(\mathbf{x}_i)}$
- Let $w_i = \frac{1}{Z} e^{-y_i H(\mathbf{x}_i)}$, where $Z = \sum_{i=1}^{n} e^{-y_i H(\mathbf{x}_i)}$ (constant w.r.t h), so $\sum_{i=1}^{n} w_i = 1$.
  w_i is the relative contribution of $(\mathbf{x}_i, y_i)$ to the overall loss.
- Binary classification: $y \in \{-1, +1\}$, $h(\mathbf{x}) \in \{-1, +1\}$.
- Derivation:

$$\begin{aligned} h &= \arg\min_{h \in \mathbb{H}} \sum_{i=1}^{n} q_i h(\mathbf{x}_i) \\ &= \arg\min_{h \in \mathbb{H}} -\sum_{i=1}^{n} w_i y_i h(\mathbf{x}_i) \\ &= \arg\min_{h \in \mathbb{H}} \sum_{i: h(\mathbf{x}_i) \neq y_i} w_i - \sum_{i: h(\mathbf{x}_i) = y_i} w_i \\ &= \arg\min_{h \in \mathbb{H}} \sum_{i: h(\mathbf{x}_i) \neq y_i} w_i \end{aligned}$$

  * Last equality holds because $\sum_{i=1}^{n} w_i = 1$.

- Result:

$$h = \arg\min_{h \in \mathbb{H}} \sum_{i:h(\mathbf{x}_i) \neq y_i} w_i$$

$$\epsilon := \sum_{i:h(\mathbf{x}_i) \neq y_i} w_i$$

  is the weighted classification error(加权错误率).

  Note: misclassified points by H get larger weights(分类错误的点会得到更大的权重).

- Given h, find α via:

$$\alpha = \arg\min_{\alpha} \mathcal{L}(H + \alpha h) = \arg\min_{\alpha} \sum_{i=1}^{n} e^{-y_i(H(\mathbf{x}_i) + \alpha h(\mathbf{x}_i))}$$

- Differentiate w.r.t α and set to zero:

$$\sum_{i=1}^{n} y_i h(\mathbf{x}_i) e^{-(y_i H(\mathbf{x}_i) + \alpha y_i h(\mathbf{x}_i))} = 0$$

  It follows that:

$$\sum_{i:h(\mathbf{x}_i)y_i=1} e^{-(y_i H(\mathbf{x}_i) + \alpha y_i h(\mathbf{x}_i))} - \sum_{i:h(\mathbf{x}_i)y_i=-1} e^{-(y_i H(\mathbf{x}_i) + \alpha y_i h(\mathbf{x}_i))} = 0$$

$$\sum_{i:h(\mathbf{x}_i)y_i=1} w_i e^{-\alpha} - \sum_{i:h(\mathbf{x}_i)y_i=-1} w_i e^{\alpha} = 0$$

  We have $(1-\epsilon)e^{-\alpha} - \epsilon e^{\alpha} = 0$, $e^{2\alpha} = \frac{1-\epsilon}{\epsilon}$, and get:

$$\boxed{\boldsymbol{\alpha = \frac{1}{2} \ln \frac{1-\epsilon}{\epsilon}}}$$

2.3 Mixture of Experts(MoE)

A machine learning technique where multiple expert learners (e.g. neural networks) are used to divide a problem space into homogeneous regions (distinct subtasks). (用人话 (AI) 说就是，把一个复杂的问题拆分成多个子任务，每个子任务由一个专家模型来处理，从而提升整体的学习效果。)
(骗你的人话也没看懂。。)

2.3.1 The First Attempt

- Error function:

$$E = \left\| y - \sum_{j=1}^{k} g_j O_j \right\|^2$$

- y: target vector; O_j: output of expert j; g_j: proportional contribution of expert j.

- *This error function does not ensure localisation of experts.(人话：专家没有明确的分工).

2.3.2 The Second Attempt

- Error function:

$$E = \sum_{j=1}^{k} g_j \|y - O_j\|^2$$

- The system tends to devote a single expert to each training case.
- *This may not work well in practice.(人话：实际效果可能不佳)

2.3.3 The Third Attempt [Jacobs et al. 1991]

- Error function (mixture of Gaussians):

$$E_{ME} = -\log \sum_{j=1}^{k} g_j \exp\left(-\frac{1}{2}(y - O_j)^T \Sigma^{-1} (y - O_j)\right)$$

- Assume $\Sigma = I$ (Σ: Covariance matrix, 协方差矩阵), derivative w.r.t the i-th expert:

$$\frac{\partial E_{ME}}{\partial O_i} = -\left[\frac{g_i \exp\left(-\frac{1}{2}(y - O_i)^T (y - O_i)\right)}{\sum_j g_j \exp\left(-\frac{1}{2}(y - O_j)^T (y - O_j)\right)}\right](y - O_i)$$

- Compare with derivative of second attempt:

$$\frac{\partial E}{\partial O_i} = -2g_i(y - O_i)$$

- The former considers how well expert i performs relative to others, adapting the best-fitting expert faster.

  E.g., $g_1 = 0.8$, $g_2 = 0.2$, then $\frac{0.8\times0.9}{0.8\times0.9+0.2\times0.1} \approx 0.97 > 0.8$.
  (人话：表现好的专家会得到更快的提升)

(这 nm 都是什么玩意?)

2.4 Stacking(堆叠法)

Multiple base learners' outputs $(\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_N)$ are combined by a meta-learner (stacker) to produce the final prediction $\hat{Y}$. (用人话说就是，把多个模型的预测结果再拿去训练一个新的模型，从而提升整体的预测效果。)

- Multi-level stacking: stacking can be repeated.
- Popular stackers: linear models (fast), gradient boosting (accurate).
- Base models should be diverse, expert at different data parts.
- Trade-off: accurate but slow to predict.

(为什么一讲能有这么多神秘小知识点和神秘公式？)

3 Lec3: Advanced Applications

3.1 Recommendation Systems

3.1.1 Collaborative Filtering(协同过滤)

Core idea: Use behavioral data from many users (e.g., ratings, clicks) to predict what the current user might like. (人话：大数据推荐你喜欢的东西)

- **User-Item Interaction:**
  - Explicit Feedback(显式反馈): ratings, purchases.
  - Implicit Feedback(隐式反馈): clicks, browsing time.
- **User-Item Rating Matrix:** Rows = users, columns = items, values = ratings. Typically large and sparse.
- **Distance/Similarity Measurement(相似度度量):**
  - Euclidean distance: $\mathrm{sim}(user_i, user_j) = \frac{1}{1+\|\mathbf{x}_i - \mathbf{x}_j\|_2}$
  - Cosine similarity: $\mathrm{sim}(user_i, user_j) = \frac{\mathbf{x}_i \cdot \mathbf{x}_j}{\|\mathbf{x}_i\|\|\mathbf{x}_j\|}$
  - Pearson correlation: $\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y}$
- **Nearest-Neighbor Collaborative Filtering(最近邻协同过滤):**

  Predict utility of item i based on similar users who rated that item.
  - $\mathcal{N}$: neighborhood set (most similar users to u who rated item i)(人话：给你推荐东西的那些“邻居”用户)
  - $w_{uv} \in [0,1]$: similarity weight between users u and v(人话：你和邻居用户的相似度)
  - Prediction:
  $$\hat{x}_{ui} = \bar{x}_u + \sum_{v \in \mathcal{N}} \left((x_{vi} - \bar{x}_v) \times \frac{w_{uv}}{\sum_{v' \in \mathcal{N}} w_{uv'}} \right)$$
  - $\bar{x}_u = \frac{1}{|I_u|} \sum_{i \in I_u} x_{ui}$ (average rating of user u, where I_u is the set of items rated by u)
  - $\bar{x}_v = \frac{1}{|I_v|} \sum_{i \in I_v} x_{vi}$ (average rating of user v, where I_v is the set of items rated by v)
- **Matrix Factorization Collaborative Filtering(矩阵分解协同过滤):**
  - Notations:
    * $R = [r_{ui}] \in \mathbb{R}^{m \times n}$: incomplete user-item rating matrix
    * Ω: set of observed entries (known ratings)
    * $P = [p_1, \dots, p_u, \dots, p_m] \in \mathbb{R}^{f \times m}, \quad Q = [q_1, \dots, q_i, \dots, q_n] \in \mathbb{R}^{f \times n}$
  - Basic SVD ($R \approx P^T Q$):
  $$\min_{P,Q} \sum_{(u,i) \in \Omega} \left\{ (r_{ui} - p_u^T q_i)^2 + \lambda (\|p_u\|^2 + \|q_i\|^2) \right\}$$
  - SVD with bias ($b_{ui} = \mu + b_u + b_i$):
  $$\min_{P,Q,B} \sum_{(u,i) \in \Omega} \left\{ (r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda (\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2) \right\}$$

- μ: global mean; b_u: user bias; b_i: item bias;
- $\lambda(\cdot)$: regularization term (prevents overfitting).
- Optimization: GD/SGD or Alternating Least Squares(交替最小二乘法).

- **Pros & Cons of Collaborative Filtering:**
  - Pros: No domain knowledge needed; captures diverse user preferences(人话：不需要领域知识，能捕捉多样的用户偏好).
  - Cons: Cold-start problem (new users/items); data sparsity(人话：新来的用户和物品数据较少，导致推荐效果差).

3.1.2 Content-Based Methods

- **Content analysis:** item $\to$ feature vector v (e.g. TF-IDF, image features).
- **Profile learning:** user $\to$ feature vector z (e.g. age, sex, education).
- **Filtering module:** train classification/regression model to predict user's utility for an item.
- **Recommendation for user:**
  - n: # of items;
  - $z_u \in \mathbb{R}^d$:
  - user u's feature vector;
  - $h : \mathbb{R}^d \to \mathbb{R}^n$ (e.g. neural network);
  - h_i: i-th output of h.
  - ℓ: loss function (e.g. squared loss).

$$\min_h \sum_{(u,i)\in\Omega} \ell(r_{ui}, h_i(z_u))$$

- **Recommendation for item:**
  - m: # of users;
  - $v_i \in \mathbb{R}^{d'}$: item
  - i's feature vector;
  - $g : \mathbb{R}^{d'} \to \mathbb{R}^m$ (e.g. neural network);
  - g_u: u-th output of g.
  - ℓ: loss function (e.g. squared loss).

$$\min_g \sum_{(u,i)\in\Omega} \ell(r_{ui}, g_u(v_i))$$

- **Pros & Cons of Content-Based Methods:**
  - Pros: User-independent; explainable; handles new items/users well.
  - Cons: Needs domain knowledge; narrow recommendations (similar items).

3.1.3 Hybrid Methods(看起来是不重要的知识点)

Most modern systems are hybrid recommenders.

- Combine separate recommenders (CF + CB): ensemble techniques (linear weighting, stacking, etc.)
- Add content-based aspects to CF: e.g. matrix factorization with side information.

3.1.4 Evaluation Metrics for RS

3.1.4.1 Prediction Metrics(评分预测指标)

- **Mean Absolute Error (MAE):**

$$\text{MAE} = \frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} |r_{ui} - \hat{r}_{ui}|$$

- **Root Mean Squared Error (RMSE):**

$$\text{RMSE} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} (r_{ui} - \hat{r}_{ui})^2}$$

  Here $\mathcal{T}$ denotes the set of user–item pairs used for evaluation (e.g., the test set).(人话：测试集中所有用户和物品的组合)

3.1.4.2 Ranking-based Metrics(基于排序的指标)

- **Precision@K:** fraction of top-K recommended items that are relevant(人话：前 K 个推荐中有多少是相关的).

$$\text{Prec}(R)_k = \frac{|\{r \in R : \ r \leq k\}|}{k}$$

- **Recall@K:** fraction of relevant items covered in top-K(人话：前 K 个推荐覆盖了多少相关的物品).

$$\text{Recall}(R)_k = \frac{|\{r \in R : \ r \leq k\}|}{|R|}$$

  - (Precision = TP/(TP+FP); Recall = TP/(TP+FN)).
  - r = rank position of a recommended item;
  - k = cut-off (top-k);
  - R = set of relevant items for the user.

- **Average Precision (AP@N):** average of precision values at ranks of relevant items.(前 N 个推荐中相关物品的精确率)

$$\text{AP@}N = \frac{1}{m} \sum_{k=1}^{N} P(k) \cdot \text{rel}(k)$$

  where $P(k)$ is precision@k, m number of relevant items, and rel(k) is indicator if item at rank k is relevant.

- **Mean Average Precision (MAP):** mean of AP over Q users:

$$\text{MAP} = \frac{1}{Q} \sum_{q=1}^{Q} \text{AP}(q)$$

- **Normalized Discounted Cumulative Gain (NDCG):** evaluates ranked relevance with position discounting.

$$\mathrm{NDCG}_p = \frac{\mathrm{DCG}_p}{\mathrm{IDCG}_p}, \quad \mathrm{DCG}_p = \sum_{i=1}^{p} \frac{2^{\mathrm{rel}_i} - 1}{\log_2(i+1)}$$

  where rel_i is relevance of item at rank i, and IDCG_p is the ideal DCG (sorted by relevance). Range: $[0, 1]$. (NDCG 越接近 1 越好)

  - **Example:** 5 recommended items with relevances $[3, 2, 1, 0, 2]$ (in rank order).

$$\begin{aligned}
\mathrm{DCG}_5 &= \frac{2^3 - 1}{\log_2(1+1)} + \frac{2^2 - 1}{\log_2(2+1)} + \frac{2^1 - 1}{\log_2(3+1)} + \frac{2^0 - 1}{\log_2(4+1)} + \frac{2^2 - 1}{\log_2(5+1)} \approx 10.5538, \\
\mathrm{IDCG}_5 &= \mathrm{DCG}_5(\text{sorted rel} = [3, 2, 2, 1, 0]) \approx 10.8235, \\
\mathrm{NDCG}_5 &= \frac{\mathrm{DCG}_5}{\mathrm{IDCG}_5} \approx 0.975.
\end{aligned}$$

  (这一坨又是什么玩意？)

3.2 Learning to Rank(L2R) (排序学习)

Learning to Rank (L2R) trains models to order items by relevance, optimizing ranking-specific objectives such as pairwise or listwise losses. (你说得对，但是这里好像也没有什么公式啊 (雾))

- L2R is a supervised learning problem for ranking.
- Training data consists of:
  - A set of queries $Q = \{q_1, \ldots, q_m\}$
  - A set of documents D
  - For each query i, relevant documents $D_i = \{d_{i,1}, \ldots, d_{i,n_i}\} \subseteq D$
  - Relevance scores $\mathbf{y}_i = (y_{i,1}, \ldots, y_{i,n_i})$ for each $d_{i,j}$
- Goal: Given a new query q, output a sorted list of relevant documents.
- **Point-wise Modeling:** Predicts each (query, document) pair independently.
  Pro: Simple, can use regression/classification.
  Con: Ignores relative order between documents.
  (一句话：点对点建模简单但忽略了文档间的相对顺序。)
- **Pair-wise Modeling:** Predicts preference between document pairs for a query.
  Pro: Models relative order.
  Con: Cannot distinguish excellent-bad from fair-bad pairs.
  (一句话：成对建模能捕捉相对顺序，但无法区分优秀-差和一般-差的对比。)
- **List-wise Modeling:** Predicts for the whole ranked list of documents.
  Pro: Considers position in ranking, aligns with ranking metrics.
  Con: High training complexity.
  (一句话：列表建模考虑排名位置，但训练复杂度高。)
- **Evaluation for L2R:** Use benchmark datasets and ranking metrics (e.g., MAP, NDCG).

- **Algorithms for L2R:**
  - **Example: Ranking SVM (pairwise):**
    * **Goal:** Learn a scoring function $h(x) = w^\top x$ such that for any pair with labels $y_i > y_j$ we have $h(x_i) > h(x_j)$. (人话：让相关性更高的文档得分更高)
    * **Training pairs:** Construct pair set $\mathcal{P} = \{(i,j) : y_i > y_j\}$; $m = |\mathcal{P}|$ denotes number of pairs.
    * **Optimization (primal):**

$$\min_{w,\,\xi_{ij}\ge 0} \ \frac{1}{2}\|w\|^2 + \frac{C}{m}\sum_{(i,j)\in\mathcal{P}} \xi_{ij}$$
$$\text{s.t.} \qquad w^\top x_i \ge w^\top x_j + 1 - \xi_{ij}, \quad \forall (i,j) \in \mathcal{P}.$$

    * **Notes:** ξ_{ij} are hinge-loss slacks; C controls margin vs. training error. The objective is equivalent to minimizing the average pairwise hinge loss.
    * **Prediction:** Score each document by $h(x) = w^\top x$ and sort descending to produce a ranking.
    * **Remarks:** Works well for pairwise preferences; training can be expensive due to $O(n^2)$ pairs, so sampling or stochastic methods are often used. Kernel SVMs and regularization extend naturally.
    * (人话总结：Ranking SVM 通过学习一个线性评分函数来排序文档，优化目标是最大化正确排序对的间隔 + 最小化排序错误的惩罚。训练时需要处理大量文档对，复杂度高。)

(叽里咕噜说什么在)

4 Lec4-1: Graph Cut and Spectral Clustering(谱聚类)

4.1 Graph Paritition

A similarity graph G=(V,E,W) represents data points as vertices V, with an edge in E when the pairwise similarity is positive and weights W storing those affinities. The affinity matrix records these pairwise similarities. Graph partitioning (clustering) aims to split the graph so that edges inside a group have large weights while edges across groups have small weights. (人话：图划分就是把图分成若干部分，使得每个部分内的节点之间联系紧密 (组内权重大)，而不同部分之间的联系较弱 (组间权重小)。)

Given data points, a similarity graph can be constructed using methods such as **k-nearest neighbor or ϵ-neighborhood**. The edge weights are often defined by a **Gaussian kernel**:

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

4.2 Minimum Cut

- Minimum cut: partition the graph into two sets A and B minimizing $\mathrm{cut}(A,B) := \sum_{i\in A,\, j\in B} w_{ij}$.
- Solvable efficiently (e.g. via max-flow/min-cut algorithms, typical cost O(|V||E|)), but the minimum cut often yields unbalanced solutions (it may isolate vertices). (人话：最小割可以高效求解，但结果往往不平衡，可能会把一些节点孤立出来。)

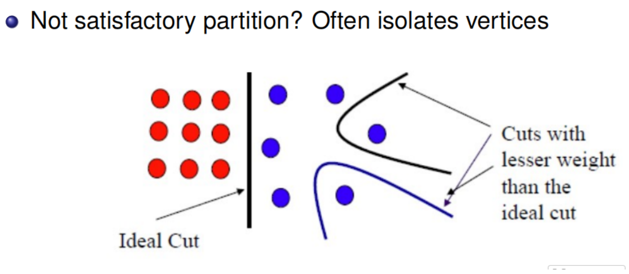

图 1: Minimum Cut Example

(画的不错的一张图)

- To address this, we can use **Normalized Cut (Ncut)**:

4.3 Normalized Cut

Normalized Cut balances cut weight with cluster sizes. For a partition (A,B), define

$$\mathrm{vol}(A) = \sum_{i \in A} d_i, \quad d_i = \sum_j w_{ij},$$

and

$$\mathrm{Ncut}(A, B) := \mathrm{cut}(A, B) \left(\frac{1}{\mathrm{vol}(A)} + \frac{1}{\mathrm{vol}(B)} \right).$$

- Minimizing Ncut favors balanced partitions but is **NP-hard**;
- **spectral clustering (谱聚类)** provides an efficient relaxation.

4.3.1 Degree Matrix and Graph Laplacian

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N1} & w_{N2} & \cdots & w_{NN} \end{bmatrix}$$

$$D = \begin{bmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_N \end{bmatrix}, \qquad d_j = \sum_{i=1}^{N} w_{ij}.$$

- D is the **degree matrix**. And the **graph Laplacian matrix** is defined as

$$L := D - W.$$

- Properties: L is symmetric positive semi-definite, $\mathbf{1}^\top L = 0$, and its eigenvectors are used in spectral clustering (relaxation of Ncut).

4.3.2 Normalized Cut and Graph Laplacian

Mathematical derivation (optional)

Recall $L = D - W$ and $D = \mathrm{diag}(d_1, \dots, d_N)$.
Let $u = [u_1, u_2, \dots, u_N]^\top$ with

$$u_i = \begin{cases} \dfrac{1}{\mathrm{vol}(A)}, & \text{if } i \in A, \\ -\dfrac{1}{\mathrm{vol}(B)}, & \text{if } i \in B. \end{cases}$$

Then

$$u^\top L u = \tfrac{1}{2} \sum_{i,j} w_{ij}(u_i - u_j)^2 = \sum_{i \in A, j \in B} w_{ij} \left(\frac{1}{\mathrm{vol}(A)} + \frac{1}{\mathrm{vol}(B)} \right)$$

and

$$u^\top D u = \sum_i d_i u_i^2 = \sum_{i \in A} \frac{d_i}{\mathrm{vol}(A)^2} + \sum_{j \in B} \frac{d_j}{\mathrm{vol}(B)^2} = \frac{1}{\mathrm{vol}(A)} + \frac{1}{\mathrm{vol}(B)}.$$

Therefore

$$\frac{u^\top L u}{u^\top D u} = \sum_{i \in A, j \in B} w_{ij} \left(\frac{1}{\mathrm{vol}(A)} + \frac{1}{\mathrm{vol}(B)} \right) = \mathrm{Ncut}(A, B).$$

(这几把都是什么玩意？好像是上面的推导吧)
(不管了，反正是 optional，摆在这图个吉利)

Conclusion

- Ncut is equivalent to minimizing the Rayleigh quotient $\dfrac{u^\top L u}{u^\top D u}$, i.e.,

$$\min_{A,B} \mathrm{Ncut}(A, B) \iff \min_u \frac{u^\top L u}{u^\top D u}, \quad u \in \mathbb{R}^N, \quad u_i = \begin{cases} \dfrac{1}{\mathrm{vol}(A)}, & i \in A, \\ -\dfrac{1}{\mathrm{vol}(B)}, & i \in B. \end{cases}$$

- Equivalent formulation: minimize the quotient subject to $u^\top D\mathbf{1} = 0$ and binary constraints $u_i \in \{1, -b\}$ (for some $b > 0$).

- Relaxation:

$$Lu = \lambda D u,$$

  taking the eigenvector corresponding to the **second smallest eigenvalue** as the relaxed solution.

- Equivalently, use the normalized Laplacian $\widetilde{L} = D^{-1} L = I - D^{-1} W$.
  Obtain a binary partition by thresholding u at 0: $i \in A$ if $u_i \geq 0$, else $i \in B$.

- Extend to k clusters by using the first k nontrivial eigenvectors and applying k-means (spectral clustering).

- (人话：2 类划分找第二小特征值对应的特征向量，k 类划分用前 k 个非平凡特征向量，然后 k-means 聚类。)

(事实上还是没看懂，插个眼以后复习的时候看看有没有什么需要补的)

4.4 Spectral Clustering Algorithm

- Input: data $X = \{x_1, x_2, \dots, x_N\}$ and number K of clusters.
- **Step 1**: Construct a similarity (affinity) matrix W (e.g. Gaussian kernel)
$$w_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$
and build either a k-nearest neighbor or ϵ-neighborhood graph.
- **Step 2**: Compute Laplacian L (or normalized variants):
$$L = D - W, \qquad \widetilde{L} = D^{-1}L = I - D^{-1}W, \qquad \hat{L}_{sym} = I - D^{-1/2}WD^{-1/2}.$$
- **Step 3**: Eigen-decompose (normalized) Laplacian and take first K nontrivial eigenvectors to form $Z = [v_1, \dots, v_K]^\top \in \mathbb{R}^{K\times N}$.(对 L 做特征值分解)
- **Step 4**: Normalize the columns (row-wise embedding) to unit ℓ_2 norm(归一化):
$$z_i \leftarrow z_i/\|z_i\|, \quad i = 1, \dots, N.$$
- **Step 5**: Run K-means on $\{z_1, \dots, z_N\}$ and output K clusters (assignments on Z or map back to X).
- Notes: use $\widetilde{L}_{sym}$ (symmetric normalized) for best numerical stability; thresholding the second eigenvector recovers a binary partition.
- **Properties of L:**
For $L = D - W$ or $\hat{L} = I - D^{-1/2}WD^{-1/2}$
  - L (and $\hat{L}$) are symmetric and positive semi-definite.(对称 + 半正定)
  - Eigenvalues satisfy $0 = \lambda_1 \le \lambda_2 \le \cdots \le \lambda_N$.
  - The multiplicity K of eigenvalue 0 equals the number of connected components of the graph (hence K clusters). (0 特征值的数量等于图的连通分量数，也就是聚类数)

5 Lec4-2: Semi-Supervised Learning

to be continued...