

AI1811 Machine Learning

Xiyuan Yang

2025.10.26

Machine Learning Courses for AI1811, simple machine learning.

目录

1. Introduction	2
1.1. Course Content	2
1.2. Introduction to ML	3
1.2.1. Basic Concepts	3
1.2.2. No Free Lunch Theorem	3
2. Chapter 2 Model Selection	4
2.1. OverFitting	4
2.1.1. Cross Validation	4
2.1.2. Boot Strapping	4
2.2. Evaluation	4
2.2.1. Precision and Recall	4
2.2.2. ROC and AUC	5
3. Chapter 3 Linear Regression	5
3.1. Introduction to linear regression	5
3.2. Hyperparameter Space	6
3.3. MSE	6
3.4. KNN	6
3.5. Least Square	6
3.6. Regularization	7
3.6.1. Ridge Regression	7
3.6.2. Lasso (Least Absolute Shrinkage and Selection Operator)	7
3.6.3. Elastic Net	8
3.7. Classification	8
3.7.1. Logistic Regression	8
3.8. LDA	9
3.9. 多分类学习	10
3.9.1. OvO	10
3.9.2. OvR	10
3.9.3. MvM	10
3.9.3.1. ECOC	10
3.10. 类别不平衡问题	11
4. Chapter 4 Decision Tree	11
4.1. Partition Selection	12
4.2. Pruning	13
4.2.1. PrePruning	13
4.2.2. PostPruning	13
4.3. 连续值	13
4.4. 缺失值	13
4.5. 多变量决策树	14

5. Chapter 6: SVM (Support accenttor Machine)	14
6. Chapter 7: Naive Bayes	14
7. Ensemble Learning	14
8. Clustering	14
8.1. Evaluation for Clustering	14
8.1.1. External Index	14
8.1.2. Internal Index	15
8.2. Prototype Based Clustering	16
8.2.1. K-Means	16
8.2.2. K-Means++	16
8.2.3. Expectation–Maximization	17
8.2.4. GMM	18
8.2.4.1. Continuity and Soft Assignment	19
8.2.4.2. Source Code	20
8.3. Density-Based Clustering	27
8.3.1. DBS-CAN	27
9. Dimension Reduction	28
9.1. Linear Dimension Reduction	28
9.1.1. Multiple Dimensional Scaling	28
9.1.2. PCA	30
9.1.2.1. Prof1 for PCA	30
9.1.2.2. Prof 2 for PCA	33
9.2. Non-Linear Dimension Reduction	33
9.2.1. Why Non-Linear?	33
9.2.2. Kernelized PCA	33
9.2.2.1. Kernelization and Kernel Function	34
9.2.3. Manifold Learning	35
9.2.3.1. Isomap	35
9.2.3.2. Locally Linear Embeddings (LLE)	35
10. Metric Learning	36
11. Conclusion	36

§ 1. Introduction

§ 1.1. Course Content

- Supervised Learning
 - ▶ Regression
 - KNN-based regression
 - linear regression
 - least squares
 - gradient descent
 - ▶ Classification
 - Decision Trees
 - Logistic regression
 - Support accenttor Machine
- Dimensional Reduction
 - ▶ PCA

- ▶ Locally Linear Embeddings
- Clustering
 - ▶ K-means
 - ▶ Expectation-maximization
 - ▶ K-means++
- Model Selection and evaluation
 - ▶ Overfitting
 - ▶ L1/L2 regularization
 - ▶ K-fold cross-validation
- MLE / MAP
 - ▶ Likelihood
 - Given the output, predict the likelihood of the parameters.
 - $P(x|\theta)$: probability
 - $L(\theta|x)$: Likelihood
 - $P(x|\theta) = L(\theta|x)$
 - generated from the probability distribution: $f(x, \theta)$
 - ▶ Maximum likelihood estimation (MLE)
 - 观测问题，通过统计学来反推结果。

$$\hat{\theta} = \operatorname{argmax}_{\theta} L(\theta|x)$$

- ▶ Maximum a posteriori (MAP)
 - Add a prior

$$\hat{\theta} = \operatorname{argmax}_{\theta} L(\theta|x)P(\theta)$$

§ 1.2. Introduction to ML

§ 1.2.1. Basic Concepts

Definition 1.2.1.1 Basic Concepts for ML.

- feature/attribute
- feature/attribute space
- sample space for the data set
- label space for output
- generalization
 - ▶ learning process is the searching and evaluating process during the hypothesis space.
 - ▶ inductive bias
 - 机器学习算法对一个假设空间中假设的偏好
 - 学习的关键在于形成正确的假设偏好
 - 奥卡姆剃刀
 - 归纳偏好很容易陷入局部过拟合中
 - 具体问题具体分析

§ 1.2.2. No Free Lunch Theorem

“奥卡姆剃刀”本身在理论上并不严格成立，即期望性能相同。

$$\sum_f E_{\text{ote}}(\mathcal{L}_a|X, f) = \sum_f E_{\text{ote}}(\mathcal{L}_b|X, f) = 2^{|\mathcal{X}|-1} \sum_{x \in \mathcal{X}-X} P(x)$$

- \mathcal{X} 代表着样本空间
- \mathcal{L}_a 代表着不同的算法
- X 代表着训练数据
- f 代表着目标函数

§ 2. Chapter 2 Model Selection

§ 2.1. OverFitting

§ 2.1.1. Cross Validation

Given the dataset D , splitting the dataset into:

$$D = D_1 \cup D_2 \cup D_3 \cup \dots \cup D_k, D_i \cap D_j = \emptyset$$

- 注意分层采样
- 使用 k 折交叉验证，分别使用不同的 D_i 作为最终的测试集，返回平均的测试结果。

§ 2.1.2. Boot Strapping

返回采样 m 次，概率保证有一定量的数据不会被采样到。这样的估计方法被称作 out-of-bag estimate.

§ 2.2. Evaluation

Definition 2.2.1 MSE Error.

$$E(f; D) = \frac{1}{m} \sum_{i=1}^m (f(x_i) - y_i)^2$$

§ 2.2.1. Precision and Recall

Definition 2.2.1.1 Precision and Recall.

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

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

We need a tradeoff!

- P-R graph
 - ▶ 改变分类阈值而画出的曲线
- BEP Point: where $P = R$
- F1 Score

$$F_1 = \frac{2PR}{P + R}$$

$$F_\beta = \frac{(1 + \beta^2)PR}{(\beta^2 P) + R}$$

- $\beta > 1$: focus more on Recall (查全率)
- $\beta < 1$: focus more on precision

§ 2.2.2. ROC and AUC

在预测问题场景下，模型往往输出 $[0, 1]$ 之间的一个浮点数来表示输出的概率大小，最终设置一个阈值来表达具体的分类。阈值的选取往往和实际场景和期望性能有关。例如如果更关心查全率，阈值就会更低，如果更关心查准率，阈值就会更高。因此，单纯阈值的设置往往需要根据 P-R 曲线来做具体的 tradeoff，无法表示该模型在一般任务下的泛化性能。

如何衡量一般任务下的泛化性能？我们给出 ROC 和 AUC 的定义指标。

Definition 2.2.2.1 ROC.

Receiver Operating Characteristic

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}}$$

和 PR 曲线类似，ROC 曲线也是根据实际的混淆矩阵和设置不同的阈值就可以得到 ROC 曲线。根据公式，TPR 和 FPR 的值本身就反应了分类器是否成功进行了分类。我们希望 TPR 尽可能的大，而 FPR 尽可能的小，即图上的 $(0,1)$ 点。

§ 3. Chapter 3 Linear Regression

§ 3.1. Introduction to linear regression

- 最基本的多维度线性回归，就是两个向量的点积操作。
- 考虑并行化处理，我们可以使用矩阵实现并行计算。

$$y = Xw^\top$$

- 其中矩阵 $X \in \text{number of samples} \times \text{dimension}$
- 权重向量被每一个样本点所复用

Recordings Regression is the basis.

- 回归和分类的关系：
 - 可以使用单热编码编码成高维的向量
 - softmax
- 回归和排序的关系
 - 直接将排序问题转化为回归，最小化真实相关度分数 y_i 与预测分数 \hat{y}_i 之间的均方误差
 - 使用 pointwise 的方法或者 pairwise 的方法。
- 回归于密度估计的关系
 - 在高斯假设下的回归模型，本质上是对条件概率分布的参数建模：

$$y|x \sim \mathcal{N}(x^\top w, \sigma^2)$$

- 此处最大似然估计等价于最小均方化误差

§ 3.2. Hyperparameter Space

对于一般化的线性回归模型，我们可以通过偏置矩阵消去偏置项，方便训练和计算。

$$\hat{y} = Xw + b_1$$

$$X' = [1, X] \in \mathbb{R}^{n \times (d+1)}$$

$$w' = \begin{pmatrix} b \\ w \end{pmatrix}$$

$$\hat{y} = X'w'$$

§ 3.3. MSE

在线性回归中，最常见的优化目标是最小化均方误差 (Mean Squared Error, MSE)：

$$\mathcal{L}(w) = \frac{1}{n} \|y - Xw\|^2$$

MSE 可等效写为：

$$\mathcal{L}(w) = \frac{1}{n} (y - Xw)^T (y - Xw)$$

通过对损失函数求梯度并令其为零，可以得到闭式解 (普通最小二乘 OLS)：

$$w^* = ((X^T X)^{-1} X^T y)$$

(当 $X^T X$ 可逆时)

Recordings MAE and MSE.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

- MAE 对异常值不敏感
- MSE 对异常值非常敏感，使用于 Loss 比较大的雨花算法求解。

§ 3.4. KNN

Definition 3.4.1 Intuition for KNN.

- 一种非训练的基于实例的非参数化方法。
- 在预测阶段开始计算，懒惰学习
- 最终根据距离加权 K 个最近的样本点。
- 计算复杂度高（对于多次查询），并且对异常值敏感。

§ 3.5. Least Square

在线性回归中，最常见的优化目标是最小化均方误差 (Mean Squared Error, MSE):

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$$

MSE 可等效写为:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n} (\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w})$$

通过对损失函数求梯度并令其为零，可以得到闭式解 (普通最小二乘 OLS):

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = \frac{2}{n} \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

$$\mathbf{X}^\top \mathbf{X}\mathbf{w} = \mathbf{X}^\top \mathbf{y}$$

$$\mathbf{w}^* = ((\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y})$$

(当 $\mathbf{X}^\top \mathbf{X}$ 可逆时)

Recordings 几何意义.

- 最小二乘法是在参数空间找寻找一个 d 维度的超平面，并且保证所有样本点在做投影时尽可能的接近该平面。
- $\mathbf{X}\beta$ 是 \mathbf{X} 的列向量的线性组合，或者说张成了 \mathbf{X} 的列空间上。
- 或者说，最小二乘法等价于将目标向量 \mathbf{y} 投影到 \mathbf{X} 的列空间上，投影向量正是 $\mathbf{X}\beta$ 而残差向量与列空间正交

$$\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\beta) = 0$$

§ 3.6. Regularization

§ 3.6.1. Ridge Regression

增加 L2 正则化，惩罚参数量过大的情况。

$$\min_{\beta} \| \mathbf{y} - \mathbf{X}\beta \|_2^2 + \lambda \|\beta\|_2^2$$

在增加 L2 正则化之后，其对应的理论最优解也发生了偏移：

$$\hat{\beta} = (\mathbf{X}^\top \mathbf{X} + \lambda I_p)^{-1} \mathbf{X}^\top \mathbf{y}$$

Recordings Explanation.

在 OLS 中，我们最小化的是残差平方和，其等值线是椭圆形。而 正则项 $\|\beta\|^2 \leq t$ 对应的约束区域是一个超球体（在二维为圆形）。Ridge 回归等价于在椭圆等值线与圆形约束的交线上寻找最优点，由于圆形约束会均匀压缩系数，使得 Ridge 回归倾向于得到较小但非零的系数。

§ 3.6.2. Lasso (Least Absolute Shrinkage and Selection Operator)

- 使用 L1 正则化

Definition 3.6.2.1 Lasso.

$$\hat{\beta} = \arg \min_{\beta} \left(\frac{1}{2n} \|y - X\beta\|_2^2 + \lambda \|\beta\|_1 \right)$$

Recordings Lasso.

在 L2 正则化是，约束区域是一个球形区域，但是 L1 正则化约束是 L1 范数的菱形区域。

§ 3.6.3. Elastic Net

设训练数据为

$$\mathbf{X} \in \mathbb{R}^{n \times p}, \mathbf{y} \in \mathbb{R}^n$$

其中 n 为样本数， p 为特征数，参数向量 β 为 \mathbb{R}^p ，截距为 β_0 。

Elastic Net 的优化目标函数为：

$$\hat{\beta} = \arg \min_{\beta} \left(\frac{1}{2n} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \left(\alpha \|\beta\|_1 + \frac{1-\alpha}{2} \|\beta\|_2^2 \right) \right)$$

其中：

- $\|\beta\|_1 = \sum_{j=1}^p |\beta_j|$ 为 L1 范数 (Lasso);
- $\|\beta\|_2^2 = \sum_{j=1}^p \beta_j^2$ 为 L2 范数平方 (Ridge);
- $\lambda \geq 0$ 为整体正则化强度超参数;
- $\alpha \in [0, 1]$ 控制 L1 与 L2 的比例:
 - $\alpha = 1$ 时退化为 Lasso;
 - $\alpha = 0$ 时退化为 Ridge;
 - $0 < \alpha < 1$ 时为 Elastic Net 的混合正则化。

§ 3.7. Classification

Recordings Regression and Classification.

- Classification 是输出到离散的标签点
- Regression 是输出到连续的实数值
- 从输入维度来讲，都是有标签数据

$$\text{Acc} = \frac{1}{N} \sum_{i=1}^N \mathbb{L}(\hat{p}_i = y_i)$$

但是在模型训练的过程中，需要更精确的损失函数，使用二进制交叉熵损失。

§ 3.7.1. Logistic Regression

有关逻辑回归的基本算法内容跳过，下面证明其函数在可导性上的良好性质：

考虑一个基本的逻辑回归：

$$y = \frac{1}{1 + e^{-w^\top x + b}}$$

考虑上面式子中的 y 为一个后验概率：

$$p(y = 1|x) = \frac{1}{1 + e^{-w^\top x + b}}$$

可以得到：

$$\ln \frac{p(y = 1|x)}{y = 0|x} = w^\top x + b$$

在概率中使用极大似然法来估计参数：

$$\ell(w, b) = \sum_{i=1}^m \ln p(y_i|x_i; w, b)$$

即令每个样本属于其真实标记的概率越大越好。为便于讨论，令 $\beta = (w; b)$, $\hat{x} = (x; 1)$, 则 $w^\top x + b$ 可简写为 $\beta^\top \hat{x}$. 再令 $p_1(\hat{x}; \beta) = p(y = 1 | \hat{x}; \beta)$, $p_0(\hat{x}; \beta) = p(y = 0 | \hat{x}; \beta) = 1 - p_1(\hat{x}; \beta)$, 则式(3.25)中的似然项可重写为

$$p(y_i | x_i; w, b) = y_i p_1(\hat{x}_i; \beta) + (1 - y_i) p_0(\hat{x}_i; \beta)$$

将式(3.26)代入(3.25)，并根据式(3.23)和(3.24)可知，最大化式(3.25)等价于最小化

$$\ell(\beta) = \sum_{i=1}^m (-y_i \beta^\top \hat{x}_i + \ln(1 + e^{\beta^\top \hat{x}_i}))$$

该式子根据经典的凸优化理论，可以求得其理论解，从理论上证明了其可导性！

§ 3.8. LDA

经典的使用投影的思想解决问题：

给定高维数据集 $D = ((x_i, y_i))_{i=1}^m$:

- 将一个数据点投影到一条直线 w :
 - 高维空间下的直线: $r(t) = r_0 + t\vec{v}$
 - 其中 r_0 代表的固定点 P , 为一个高维向量
 - \vec{v} 代表一个方向向量, 也是高维的
 - 在这里默认直线经过原点 (为了后续简化计算), 因此可以使用一个向量代表直线
 - 投影点的坐标 $w^\top \mu_0$
- 在上文的例子中 μ_0 代表着均值向量
 - 考虑协方差 $w^\top \sum_0 w$

因此得到最大化的目标:

$$J = \frac{\|w^\top \mu_0 - w^\top \mu_1\|_2^2}{w^\top \sum_0 w + w^\top \sum_1 w}$$

$$S_b = (\mu_0 - \mu_1)(\mu_0 - \mu_1)^\top$$

$$S_w = \sum_0 + \sum_1$$

Recordings S_b and S_w.

- 我们的直觉是同类数据点之间尽可能靠近
 - 在分母上放协方差作为惩罚项
- 不同类别的中心尽可能远离
 - 以均值中心为代表，衡量投影的二范数（距离）作为分子

对于多分类任务同样如此：

- 类内散度矩阵被定义为所有类别散度矩阵的和，每一个类别散度矩阵通过协方差衡量数据点的分布关系：

$$S_W = \sum_{k=1}^K S_k$$

$$S_k = \sum_{x \in C_k} (x - \mu_k)(x - \mu_k)^\top$$

- 类间散度矩阵为每个类别的均值相对于所有样本的全局均值的分散程度。它反映了不同类别中心之间的分离程度。

§ 3.9. 多分类学习

多分类学习可以从二分类学习的方法中进行升维并且一般化，但是也可以有一些更通用的策略：

基本思路：将多分类学习的任务拆解为若干个二分类任务求解。

- Splitter
- Classifier

§ 3.9.1. OvO

对于 N 个类别的多分类任务，采用 $\frac{N(N-1)}{2}$ 个二分类任务进行组合。我们训练这么多分类器之后就可以实现分类任务就可以通过正反例的方式进行分类

§ 3.9.2. OvR

训练 N 个分类器，将一个类别标记为正类，其余类别标记为反类

§ 3.9.3. MvM

相当于分治法划归为子问题，但是其正反类需要特殊的设计

§ 3.9.3.1. ECOC

纠错输出码 Error Correcting Output Codes

- 编码：对于 N 个类别做 M 次划分
 - 形成编码矩阵 $\in N \times M$
 - 编码矩阵可以是二元码也可以是三元码
- 解码：分别对输出样本做预测
 - 因此解码的时候可以得到一个预测输出向量
 - 计算输出向量和编码矩阵的每一行（代表对一个训练样本数据每一个预测器的判断）的向量比较并计算距离
 - Hamming Distance
 - Euclidean Distance

ECOC 的关键在于选择的 M 会很多，远大于 $\log_2 N$ 的最低下限值。因此这会带来比较空间的冗余（因为维度上去了），但是这也提升了鲁棒性。

- 基础二分类器的选择可以自由选择。
- OvR 可以看做是一种特殊的 ECOC，但是 OvR 对应的编码矩阵的鲁棒性很小

§ 3.10. 类别不平衡问题

对于实际 GT 分布存在严重不平衡的样本，对于二分类器的训练带来了难度：

可以使用除了 Precision 之外的其他指标。（这是 evaluation 阶段做的修改）

对于二分类器而言，也可以在 inference 阶段实现优化：

- 对于均分样本，我们一般比较输出的概率大小来输出最终的值，即阈值设定为 0.5

$$\frac{y}{1-y} > 1$$

- 对于非均衡样本，可以使用偏斜的预测阈值

$$\frac{y}{1-y} > \frac{m^+}{m^-}$$

具体而言，我们可以对模型的输出值再加一个 Rescaling，即：

$$\frac{y'}{1-y'} = \frac{m^-}{m^+} \frac{y}{1-y}$$

Recordings Rescaling.

- 在理想情况下，这个方法是理论可行的，但是这个的前提是 训练集必须是真实样本的无偏估计的理想假设
- 这一个假设在偏斜数据集的场景下更难满足
- 具体的，也可以使用欠采样或者过采样的方式使模型的正反例的样本数基本相同
 - 欠采样：使用集成学习机制，将反例划分为不同的集合供学习器使用，类似于 Kfold 的思想
 - 过采样：插值增加采样数据

§ 4. Chapter 4 Decision Tree

决策树的 Intuition 非常的简单，通过树状结构来模拟人类决策的 Options，是一种很自然的模拟机制。最终，当决策过程达到最大深度的叶子结点时，就代表一个最终的分类结果。同时，这也是一种简单而直观的“分而治之”的策略。但是决策树非常容易陷入在训练数据的过拟合中，我们希望产生一颗泛化能力强的决策树。

输入: 训练集 $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$;
 属性集 $A = \{a_1, a_2, \dots, a_d\}$.
 过程: 函数 TreeGenerate(D, A)
 1: 生成结点 node;
 2: if D 中样本全属于同一类别 C then
 3: 将 node 标记为 C 类叶结点; return
 4: end if
 5: if $A = \emptyset$ OR D 中样本在 A 上取值相同 then
 6: 将 node 标记为叶结点, 其类别标记为 D 中样本数最多的类; return
 7: end if
 8: 从 A 中选择最优划分属性 a_* ;
 9: for a_* 的每一个值 a_*^v do
 10: 为 node 生成一个分支; 令 D_v 表示 D 中在 a_* 上取值为 a_*^v 的样本子集;
 11: if D_v 为空 then
 12: 将分支结点标记为叶结点, 其类别标记为 D 中样本最多的类; return
 13: else
 14: 以 TreeGenerate($D_v, A \setminus \{a_*\}$) 为分支结点
 15: end if
 16: end for
 输出: 以 node 为根结点的一棵决策树

图 4.2 决策树学习基本算法

图 1 Simple algorithm for decision trees

上述算法是一个经典的递归算法, 他的基本思想如下, 递归的终点就是叶子结点的生成位置:

- 当输入样本全部属于一个标签类别的时候, 说明无需继续分类
 - 直接根据该标签类别生成一个叶节点
- 所有样本在属性集合上的取值相同 (或者样本的属性集合为空): 此时无法做出有效的划分, 停止递归
 - 需要标记为叶节点, 属性是输入样本中样本数最多的类
- 划分的样本集合为空, 无法进行划分, 停止递归。
 - 根据父节点的样本数最多的类

因此, 该算法的关键在于在每一次计算过程中找到 the best partition properties 代表着树进行一次分叉的属性。

§ 4.1. Partition Selection

我们可以把基于决策树的分类任务当成是一种提纯的操作。如何定义纯度? 信息熵! 这就是决策树的基本原理:

$$\text{Ent}(D) = - \sum_{k=1}^{|Y|} p_k \log_2 p_k$$

从简单情况出发, 考虑某一个离散属性 $a \in A$, which has several possible values: (a^1, a^2, \dots, a^V) , then it will split the current tree into V new branches, then the input D is split into V partitions: D_1, D_2, \dots, D_V .

我们希望这是一个有效的划分而不是一个随机的划分, 因此我们定义信息增益并希望最大化信息增益:

$$\text{Gain}(D, a) = \text{Ent}(D) - \sum_{v=1}^V \frac{|D^v|}{|D|} \text{Ent}(D^v)$$

因为我们希望 Gain 是正相关于我们的选择好坏的，而选择越好，信息熵越小，因此其实这是一个信息熵的减少量。

Definition 4.1.1 增益率.

- 信息增益确实很好的衡量了熵的变化，但是以为这是一个加权平均数，因此这对可取值数目较多的属性有所偏好。
- 因此，我们需要定义一个更准确的增益率。

$$\text{IV}(a) = - \sum_{v=1}^V \frac{|D^v|}{|D|} \log_2 \frac{|D^v|}{|D|}$$

$$\text{Gain Ratio}(D, a) = \frac{\text{Gain}(D, a)}{\text{IV}(a)}$$

§ 4.2. Pruning

树算法一般都会涉及到剪枝操作。这也是决策树学习算法主要应对过拟合的手段之一。

§ 4.2.1. PrePruning

预剪枝的处理操作发生在树的生成过程中，我们需要评估模型的泛化性，如果模型的这一次 selection 没有办法带来泛化性能的提升，就停止剪枝，作为叶子结点。

如何评判泛化性？可以考虑验证集，评判划分和不划分在验证集的精度上是否有上升或者下降的趋势。

- 这样的预剪枝处理可以有效的除了训练数据的偏差，并且防止树的层数过深导致问题
- 过度粗暴的预剪枝也会带来欠拟合的风险。

§ 4.2.2. PostPruning

后剪枝是在决策树已经生成之后再实现的，如果减去这个节点，是否会带来泛化性能的提升？如果有，就一切从简。

- 因为后剪枝是在决策树生成之后进行的，因此所需要计算的节点更多
- 但是往往不会像预剪枝一样落入贪心的陷阱而进入欠拟合。

§ 4.3. 连续值

我们希望保证连续属性的离散化，在经典的决策树 C4.5 中，采用二分法实现连续属性的决策树处理。

因为最终的样本数据肯定是离散的，因此选择划分点可以选择每两个相邻训练数据点的中点。

§ 4.4. 缺失值

- 我们只能使用不包含缺失值的数据进行计算
- 因此我们需要引入合适的系数来修正信息的增益

在决策树学习开始阶段，
根结点中各样本的权重初
始化为 1。

给定训练集 D 和属性 a , 令 \tilde{D} 表示 D 中在属性 a 上没有缺失值的样本子集。对问题(1), 显然我们仅可根据 \tilde{D} 来判断属性 a 的优劣。假定属性 a 有 V 个可取值 $\{a^1, a^2, \dots, a^V\}$, 令 \tilde{D}^v 表示 \tilde{D} 中在属性 a 上取值为 a^v 的样本子集, \tilde{D}_k 表示 \tilde{D} 中属于第 k 类 ($k = 1, 2, \dots, |\mathcal{Y}|$) 的样本子集, 则显然有 $\tilde{D} = \bigcup_{k=1}^{|\mathcal{Y}|} \tilde{D}_k$,

$$\tilde{D} = \bigcup_{v=1}^V \tilde{D}^v.$$

假定我们为每个样本 \mathbf{x} 赋予一个权重 $w_{\mathbf{x}}$, 并定义

$$\rho = \frac{\sum_{\mathbf{x} \in \tilde{D}} w_{\mathbf{x}}}{\sum_{\mathbf{x} \in D} w_{\mathbf{x}}}, \quad (4.9)$$

$$\tilde{p}_k = \frac{\sum_{\mathbf{x} \in \tilde{D}_k} w_{\mathbf{x}}}{\sum_{\mathbf{x} \in \tilde{D}} w_{\mathbf{x}}} \quad (1 \leq k \leq |\mathcal{Y}|), \quad (4.10)$$

$$\tilde{r}_v = \frac{\sum_{\mathbf{x} \in \tilde{D}^v} w_{\mathbf{x}}}{\sum_{\mathbf{x} \in \tilde{D}} w_{\mathbf{x}}} \quad (1 \leq v \leq V). \quad (4.11)$$

图 2 对缺失值的处理

§ 4.5. 多变量决策树

让我们从高维空间的视角看看决策树到底在干什么?

假设每一个数据点 (x_i, y_i) , 其中 y_i 代表输出的离散分类, 那这其实是一个在高维度的空间下的数据点, 而分类问题的关键在于找到某个约束方法或者约束条件, 实现对未见过的数据点的正确分类。

单变量决策树保证在每一个节点只会考虑一个属性作为切分, 这就导致了单变量决策树的分类边界是和坐标轴平行的。通过这一道道的分类边界将高维空间切割成正正方方的若干块。

因此, 对于多变量决策树, 每一次寻找的目标不再是最优划分的属性, 而是建立一个合适的 linear classifier, 这样的分类器可以实现更复杂的阈值划分, 进而简化决策树的深度。

$$\text{Gain}(D, a) = \rho \text{ Gain}(\tilde{D}, a)$$

§ 5. Chapter 6: SVM (Support Vector Machine)

§ 6. Chapter 7: Naive Bayes

§ 7. Ensemble Learning

§ 8. Clustering

聚类需要把样本集 D 划分成若干互不相交的子集, 即样本簇。

§ 8.1. Evaluation for Clustering

Simply, for K-means clustering:

$$\{C_1, \dots, C_k\} = \operatorname{argmin}_{k=1}^K \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

§ 8.1.1. External Index

在给定参考模型的聚类结果，我们可以检查每一个数据点的聚类结果是否和 references model 相同来判断具体聚类的优劣。具体来说，两两配对得到 $\frac{n(n+1)}{2}$ 的点对，考虑每个点对是否被正确的分类到相同或者不同的聚类中。

对数据集 $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$, 假定通过聚类给出的簇划分为 $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$, 参考模型给出的簇划分为 $\mathcal{C}^* = \{C_1^*, C_2^*, \dots, C_s^*\}$. 相应地, 令 λ 与 λ^* 分别表示与 \mathcal{C} 和 \mathcal{C}^* 对应的簇标记向量. 我们将样本两两配对考虑, 定义

$$a = |SS|, \quad SS = \{(\mathbf{x}_i, \mathbf{x}_j) \mid \lambda_i = \lambda_j, \lambda_i^* = \lambda_j^*, i < j\}, \quad (9.1)$$

$$b = |SD|, \quad SD = \{(\mathbf{x}_i, \mathbf{x}_j) \mid \lambda_i = \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j\}, \quad (9.2)$$

$$c = |DS|, \quad DS = \{(\mathbf{x}_i, \mathbf{x}_j) \mid \lambda_i \neq \lambda_j, \lambda_i^* = \lambda_j^*, i < j\}, \quad (9.3)$$

$$d = |DD|, \quad DD = \{(\mathbf{x}_i, \mathbf{x}_j) \mid \lambda_i \neq \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j\}, \quad (9.4)$$

其中集合 SS 包含了在 \mathcal{C} 中隶属于相同簇且在 \mathcal{C}^* 中也隶属于相同簇的样本对, 集合 SD 包含了在 \mathcal{C} 中隶属于相同簇但在 \mathcal{C}^* 中隶属于不同簇的样本对, ……由于每个样本对 $(\mathbf{x}_i, \mathbf{x}_j)$ ($i < j$) 仅能出现在一个集合中, 因此有 $a + b + c + d = m(m - 1)/2$ 成立.

根据这些, 可以推导一些聚类性能的外部指标。

§ 8.1.2. Internal Index

Define these metrics:

$$\text{avg}(C) = \frac{2}{|C|(|C| - 1)} \sum_{1 \leq i < j \leq |C|} \text{dist}(x_i, x_j)$$

$$\text{diam}(C) = \max_{1 \leq i < j \leq |C|} \text{dist}(x_i, x_j)$$

$$d_{\min}(C_i, C_j) = \min_{x_i \in C_i, x_j \in C_j} \text{dist}(x_i, x_j)$$

$$d_{\text{cen}}(C_i, C_j) = \text{dist}(\mu_i, \mu_j)$$

Recordings All these metrics.

- 这些指标无外乎实现两个优化目标:
 - 不同簇之间要相隔尽量明显: inter-cluster similarity
 - 相同簇之间要尽可能紧密: intra-cluster similarity
- 对于数据点的距离度量很重要。
 - 使用公理化的方法定义 distance measure

$$\text{dist}_{\text{mk}}(x_i, x_j) = \left(\sum_{u=1}^n |x_{iu} - x_{ju}|^p \right)^{\frac{1}{p}}$$

- distance metric learning

§ 8.2. Prototype Based Clustering

也称为基于中心点的聚类方法，每个簇的代表向量都有原型（中心点）的位置向量来表示。

§ 8.2.1. K-Means

$$\{C_1, \dots, C_k\} = \operatorname{argmin} \sum_{k=1}^K \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

输入: 样本集 $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$;
聚类簇数 k .

过程:

```

1: 从  $D$  中随机选择  $k$  个样本作为初始均值向量  $\{\mu_1, \mu_2, \dots, \mu_k\}$ 
2: repeat
3:   令  $C_i = \emptyset$  ( $1 \leq i \leq k$ )
4:   for  $j = 1, 2, \dots, m$  do
5:     计算样本  $\mathbf{x}_j$  与各均值向量  $\mu_i$  ( $1 \leq i \leq k$ ) 的距离:  $d_{ji} = \|\mathbf{x}_j - \mu_i\|_2$ ;
6:     根据距离最近的均值向量确定  $\mathbf{x}_j$  的簇标记:  $\lambda_j = \arg \min_{i \in \{1, 2, \dots, k\}} d_{ji}$ ;
7:     将样本  $\mathbf{x}_j$  划入相应的簇:  $C_{\lambda_j} = C_{\lambda_j} \cup \{\mathbf{x}_j\}$ ;
8:   end for
9:   for  $i = 1, 2, \dots, k$  do
10:    计算新均值向量:  $\mu'_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}$ ;
11:    if  $\mu'_i \neq \mu_i$  then
12:      将当前均值向量  $\mu_i$  更新为  $\mu'_i$ 
13:    else
14:      保持当前均值向量不变
15:    end if
16:  end for
17: until 当前均值向量均未更新
输出: 簇划分  $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ 
```

为避免运行时间过长，通常设置一个最大运行轮数或最小调整幅度阈值，若达到最大轮数或调整幅度小于阈值，则停止运行。

图 9.2 k 均值算法

Recommended Websites:

<https://www.naftaliharris.com/blog/visualizing-k-means-clustering/>

Kmeans 算法最大的局限性在于其最终的聚类结果和初始点的分布高度相关

Recordings Initialization for KMeans Algorithms.

The target optimization for K-means is:

$$\text{SSE} = \sum_{j=1}^K \sum_{x \in C_j} \|x - \mu_j\|^2$$

- This will let KMeans falling into local minima, although it converges in the end!

§ 8.2.2. K-Means++

K-means++ 算法专门用来解决 K-means 初始化陷入局部最优解的问题，他希望使用一定计算方法构造出初始点，而不是依赖随机生成。

- 随机选择第一个簇中心
- 在后续的选择中，希望保证初始化的点尽可能远离

$$P(x_i) = \frac{D(x_i)^2}{\sum_j D(x_j)^2}, D(x_i) = \min_k \|x_i - \mu_k\|$$

按照上面式子的概率选择后续的质心分布。很显然，如果某个点距离已经被选择的质心距离都相对比较大，则他被选择成为新质心的概率也会很大。

上述解释是基于直觉的，在数学上可证明 K-means++ 很好的缓解了 K-means 初始化带来的问题：

<https://theory.stanford.edu/~sergei/papers/kMeansPP-soda.pdf>

Theorem 8.2.2.1 Theory for the Upper Bound.

$$\mathbb{E}[\varphi] \leq 8(\ln k + 2)\varphi_{\text{OPT}}$$

§ 8.2.3. Expectation – Maximization

EM 算法是一种对隐变量进行参数推断的方法。

定义 X 为已观测变量， Z 隐变量，我们希望对模型参数 Θ 进行估计。

使用最大化对数似然：

$$\text{LL}(\Theta|X, Z) = \ln P(X, Z|\Theta)$$

注意， Z 是隐变量，因此上面的式子难以直接求解。因此，我们希望通过观测期望来近似估计隐变量，相当于最大化已观测数据的对数边际似然。

$$\text{LL}(\Theta|X) = \ln P(X|\Theta) = \ln \sum_Z P(X, Z|\Theta)$$

Definition 8.2.3.1 EM.

- E steps:
 - If Θ is known, then we can calculate the expectations of Z : Z_t
- Maximization:
 - 做极大似然估计

Define Q-function: 对数似然函数的条件期望(后验概率分布)

$$Q(\Theta, \Theta_t) = \mathbb{E}_{Z|X, \Theta_t} \text{LL}(\Theta|X, Z) = \mathbb{E}_{Z|X, \Theta_t} \ln P(X, Z|\Theta)$$

Using Bayes:

$$P(Z|X, \Theta^t) = \frac{P(X, Z|\Theta^t)}{P(X|\Theta^t)}$$

$$P(X, Z|\Theta) = P(Z|\Theta)P(X|Z, \Theta)$$

Then, we can calculate:

$$Q(\Theta, \Theta_t) = \sum_Z \ln(P(Z|\Theta)P(X|Z, \Theta)) \frac{P(X, Z|\Theta^t)}{P(X|\Theta^t)}$$

在 E 步中，需要计算根据现有参数估计的后验概率，得到若干隐变量的概率分布，即计算

$$P(Z|X, \Theta^t) = \frac{P(X, Z|\Theta^t)}{P(X|\Theta^t)}$$

在 M 步，根据隐变量的分布，使用下面的公式最大化 Theta (这也是 Q 函数在确定 Z 分布之后的唯一变量)

$$\Theta^{t+1} = \operatorname{argmax}_{\Theta} Q(\Theta|\Theta^t)$$

这是一种非梯度的优化方法。

§ 8.2.4. GMM

K-means 分配方法是一种硬分配的聚类，每个点被分配到最近的簇中心。但是这会导致分类模型并不鲁棒，因此，EM 聚类提出了一种基于概率分配的软化聚类方法。数据点根据不同的概率属于所有的 K 个簇。

Gaussian Distribution:

$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)} = p(x|\mu, \Sigma)$$

Mixed Gaussian Distribution:

$$p_M(x) = \sum_{i=1}^k \alpha_i p\left(x|\mu_i, \sum_i\right)$$

在这里，每一个聚类可以看作是有某一个高斯成分 j 生成，计算后验概率：

$$\gamma_{ij} = P(z_i = j|x_i, \Theta^t) = \frac{\alpha_j p(x_i|\mu_j, \Sigma_j)}{\sum_{k=1}^K p(x_i|\mu_k, \Sigma_k) \alpha_k}$$

上面这个式子的意义是样本 x_i 是有第 j 个高斯混合成分生成的。

$$\lambda_i = \operatorname{argmax}_{j \in \{1, 2, 3, \dots, k\}} \gamma_{ij}$$

在 M 步，需要最大化更新似然函数的参数：

使用极大似然估计：

$$\text{LL}(D) = \ln\left(\prod_{j=1}^m p_M(x_j)\right) = \sum_{j=1}^m \ln\left(\sum_{i=1}^k \alpha_i p\left(x_j|\mu_i, \sum_i\right)\right)$$

这也是 EM 算法中对应的 Q function。

因为高斯分布的良好性质，可以求出参数的解析解：

$$\alpha_j^{t+1} = \frac{\sum_{i=1}^N \gamma_{ij}}{N}$$

$$\mu_j^{t+1} = \frac{\sum_{i=1}^N \gamma_{ij} x_i}{\sum_{i=1}^N \gamma_{ij}}$$

$$\Sigma_k = \frac{\sum_i \gamma_{ik} (x_i - \mu_k)(x_i - \mu_k)^T}{\sum_i \gamma_{ik}}$$

输入: 样本集 $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$;
高斯混合成分个数 k .

过程:

```

1: 初始化高斯混合分布的模型参数  $\{(\alpha_i, \mu_i, \Sigma_i) \mid 1 \leq i \leq k\}$ 
2: repeat
3:   for  $j = 1, 2, \dots, m$  do
4:     根据式(9.30)计算  $\mathbf{x}_j$  由各混合成分生成的后验概率, 即
       $\gamma_{ji} = p_M(z_j = i \mid \mathbf{x}_j)$  ( $1 \leq i \leq k$ )
5:   end for
6:   for  $i = 1, 2, \dots, k$  do
7:     计算新均值向量:  $\mu'_i = \frac{\sum_{j=1}^m \gamma_{ji} \mathbf{x}_j}{\sum_{j=1}^m \gamma_{ji}}$ ;
8:     计算新协方差矩阵:  $\Sigma'_i = \frac{\sum_{j=1}^m \gamma_{ji} (\mathbf{x}_j - \mu'_i)(\mathbf{x}_j - \mu'_i)^T}{\sum_{j=1}^m \gamma_{ji}}$ ;
9:     计算新混合系数:  $\alpha'_i = \frac{\sum_{j=1}^m \gamma_{ji}}{m}$ ;
10:    end for
11:    将模型参数  $\{(\alpha_i, \mu_i, \Sigma_i) \mid 1 \leq i \leq k\}$  更新为  $\{(\alpha'_i, \mu'_i, \Sigma'_i) \mid 1 \leq i \leq k\}$ 
12: until 满足停止条件
13:  $C_i = \emptyset$  ( $1 \leq i \leq k$ )
14: for  $j = 1, 2, \dots, m$  do
15:   根据式(9.31)确定  $\mathbf{x}_j$  的簇标记  $\lambda_j$ ;
16:   将  $\mathbf{x}_j$  划入相应的簇:  $C_{\lambda_j} = C_{\lambda_j} \cup \{\mathbf{x}_j\}$ 
17: end for
输出: 簇划分  $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ 
```

图 9.6 高斯混合聚类算法

§ 8.2.4.1. Continuity and Soft Assignment

More about GMM...

$$P(x) = \sum_{k=1}^K \alpha_k \mathcal{N}(x | \Theta^{(k)}) = \sum_{k=1}^K \alpha_k \mathcal{N}\left(x | \mu_k, \sum_k\right)$$

\sum_k 代表第 k 个混合成分的协方差矩阵:

Definition 8.2.4.1.1 Cov.

$$\sum = \begin{pmatrix} \text{Cov}(X_1, X_1) & \text{Cov}(X_1, X_2) & \dots & \text{Cov}(X_1, X_D) \\ \text{Cov}(X_2, X_1) & \text{Cov}(X_2, X_2) & \dots & \text{Cov}(X_2, X_D) \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}(X_D, X_1) & \text{Cov}(X_D, X_2) & \dots & \text{Cov}(X_D, X_D) \end{pmatrix}$$

协方差矩阵本身保证对称并且半正定。

- 协方差矩阵的特征值 λ_i 决定簇的方差和形状
- 特征向量决定了椭球体的主轴方向

协方差矩阵实际上刻画了高斯分布在高维数据下是一个椭球分布。(例如, 在二维情况, 即只有两个特征值的情况下, 是一个椭圆), 而特征值就是高维椭球的每一个维度半轴的长

度，而具体的特征向量的指向则决定了具体的角度。（而且这个是确定的！因为不同特征向量之间是正交的关系）

§ 8.2.4.2. Source Code

In the following code, we will see that compared with K-means, GMM can fit data with more flexible shape!

```

import numpy as np
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import silhouette_score, homogeneity_score
from sklearn.datasets import make_blobs, make_moons
import matplotlib.pyplot as plt
import os

class ClusterModel:
    """
    An abstract/factory class for initializing, training, and predicting
    with different clustering models (GMM or KMeans++).

    This module is designed to be independent of data loading and evaluation logic.
    """

    def __init__(
        self, n_clusters: int, model_type: str = "GMM", random_state: int = 42
    ):
        """
        Initializes the clustering model instance.

        Args:
            n_clusters (int): The expected number of clusters (K).
            model_type (str): The type of model to use ('GMM' or 'KMeans').
            random_state (int): The random seed for reproducibility.

        Raises:
            ValueError: If an unsupported model_type is provided.
        """
        self.n_clusters = n_clusters
        self.model_type = model_type
        self.model = self._initialize_model(random_state)

    def _initialize_model(self, random_state):
        """
        Internal method to create and return the scikit-learn model instance.
        ! now only supports GMM and KMeans for clustering
        """

        if self.model_type == "GMM":
            return GaussianMixture(
                n_components=self.n_clusters, random_state=random_state
            )
        elif self.model_type == "KMeans":
            return KMeans(n_clusters=self.n_clusters, random_state=random_state)
        else:
            raise ValueError(f"Unsupported model type: {self.model_type}")
    """

```

```

        return KMeans(
            n_clusters=self.n_clusters,
            # use k-means++ initialization
            init="k-means++",
            random_state=random_state,
            n_init="auto",
        )
    else:
        raise ValueError("Unsupported model type. Please choose 'GMM' or
'KMeans'.")
}

def fit(self, X: np.ndarray):
    """
    Trains the underlying clustering model.

    Args:
        X (np.ndarray): The training data features.
    """
    print(f"--- Training {self.model_type} model (K={self.n_clusters}) ---")
    self.model.fit(X)

def predict(self, X: np.ndarray) -> np.ndarray:
    """
    Predicts the cluster labels for the given data points.

    Args:
        X (np.ndarray): The data to predict on.

    Returns:
        np.ndarray: The predicted cluster labels.
    """
    return self.model.predict(X)

def get_model(self):
    """
    Returns the underlying scikit-learn model instance for inspection.

    Returns:
        self.model
    """

class DataProcessor:
    """
    Utility class for loading, splitting, and preprocessing data,
    specifically handling feature scaling for clustering models.

    It is decoupled from the model training and prediction logic.
    """

    def __init__(self):
        """
        Initializes the data processor, including the standard scaler.

        self.scaler = StandardScaler()
        self.X_train = None
        self.X_test = None
        self.y_test_true = None
        """

```

```

def load_and_preprocess_data(
    self, X: np.ndarray, y: np.ndarray = None, test_size: float = 0.3
):
    """
    Splits data into training/testing sets and performs feature scaling.

    Args:
        X (np.ndarray): The raw feature data.
        y (np.ndarray, optional): The true labels (if available, used for
            external evaluation).
        test_size (float): The proportion of data to use for the test set.

    Returns:
        tuple: (X_train_scaled, X_test_scaled)
    """
    # Split data; includes true labels if provided for later evaluation
    if y is not None:
        X_train, X_test, _, self.y_test_true = train_test_split(
            X, y, test_size=test_size, random_state=42
        )
    else:
        X_train, X_test = train_test_split(X, test_size=test_size,
random_state=42)

    # Fit scaler on training data and transform both sets
    X_train_scaled = self.scaler.fit_transform(X_train)
    X_test_scaled = self.scaler.transform(X_test)

    self.X_train = X_train_scaled
    self.X_test = X_test_scaled

    print(f"Train set shape: {X_train_scaled.shape}")
    print(f"Test set shape: {X_test_scaled.shape}")
    return X_train_scaled, X_test_scaled

def get_test_data(self) -> tuple[np.ndarray, np.ndarray | None]:
    """
    Retrieves the processed test data and true labels.

    Returns:
        tuple: (X_test, y_test_true)
    """
    return self.X_test, self.y_test_true

class ClusteringVisualizer:
    """
    Handles the visualization of clustering results.
    """

    def __init__(self, output_dir="images"):
        """
        Initializes the visualizer and ensures the output directory exists.
        """
        self.output_dir = output_dir

```

```

        if not os.path.exists(self.output_dir):
            os.makedirs(self.output_dir)

    def plot_clusters(
        self,
        X: np.ndarray,
        y_pred: np.ndarray,
        model_name: str,
        y_true: np.ndarray = None,
    ):
        """
        Generates and saves a scatter plot of the clustered data.
        """
        plt.figure(figsize=(12, 5))

        # Plot predicted clusters
        plt.subplot(1, 2, 1)
        plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap="viridis", s=50, alpha=0.7)
        plt.title(f"{model_name} - Predicted Clusters")
        plt.xlabel("Feature 1")
        plt.ylabel("Feature 2")

        # Plot true clusters if available
        if y_true is not None:
            plt.subplot(1, 2, 2)
            plt.scatter(X[:, 0], X[:, 1], c=y_true, cmap="viridis", s=50, alpha=0.7)
            plt.title("Ground Truth")
            plt.xlabel("Feature 1")
            plt.ylabel("Feature 2")

        # Save the combined plot
        plot_path = os.path.join(self.output_dir, f"{model_name}_clusters.png")
        plt.savefig(plot_path)
        plt.close()
        print(f"Saved plot to {plot_path}")

    class Prediction:
        """
        The main driver responsible for coordinating data, models, execution, and
        evaluation.
        It remains agnostic to the specific model implementation or data scaling method.
        """

        def __init__(self, processor: DataProcessor):
            """
            Initializes the framework with a DataProcessor instance.

            Args:
                processor (DataProcessor): The data processing module instance.
            """
            self.processor = processor
            self.models = {}

        def train_and_predict(

```

```

        self, model_name: str, model_type: str, n_clusters: int
    ) -> np.ndarray:
        """
        Instantiates, trains a model, and returns test set predictions.

        Args:
            model_name (str): A unique name for the model instance.
            model_type (str): The type of model ('GMM' or 'KMeans').
            n_clusters (int): The number of clusters (K).

        Returns:
            np.ndarray: The predicted labels for the test set.
        """
        X_train, X_test = self.processor.X_train, self.processor.X_test

        if X_train is None:
            raise RuntimeError(
                "Data not loaded or preprocessed. Please run load_and_preprocess_data first."
            )

        # Instantiate and train the model
        model_instance = ClusterModel(n_clusters, model_type)
        model_instance.fit(X_train)

        self.models[model_name] = model_instance

        # Predict on the test set
        y_pred = model_instance.predict(X_test)
        print(f"--- {model_name} prediction complete ---")
        return y_pred

    def evaluate(
        self,
        model_name: str,
        X_test: np.ndarray,
        y_pred: np.ndarray,
        y_true: np.ndarray = None,
    ):
        """
        Evaluates the clustering results using internal and external metrics.

        Args:
            model_name (str): The name of the model being evaluated.
            X_test (np.ndarray): The feature data of the test set.
            y_pred (np.ndarray): The predicted cluster labels.
            y_true (np.ndarray, optional): The true labels (if available).
        """
        print(f"\n===== {model_name} Evaluation Results =====")

        # 1. Internal Metric: Silhouette Score
        try:
            score = silhouette_score(X_test, y_pred)
            print(f"Silhouette Score: {score:.4f} (Internal Consistency)")
        except Exception as e:
            print(f"Could not calculate Silhouette Score: {e}")

```

```

# 2. External Metric: Homogeneity Score (Requires true labels)
if y_true is not None:
    score = homogeneity_score(y_true, y_pred)
    print(f"Homogeneity Score: {score:.4f} (External Metric)")
else:
    print("True labels not provided, skipping external metrics.")

def run_clustering_framework():
"""
Runs the complete clustering prediction framework to demonstrate GMM's
flexibility
with different covariance structures, as compared to KMeans.
"""
n_samples = 500
random_state = 170

# 1. Isotropic (Spherical) Cluster
X_iso, y_iso = make_blobs(
    n_samples=n_samples,
    centers=[[-3, -3]],
    cluster_std=0.5,
    random_state=random_state,
)

# 2. Diagonal (Axis-aligned Elliptical) Cluster
X_diag_raw, _ = make_blobs(
    n_samples=n_samples,
    centers=[[0, 5]],
    cluster_std=0.5,
    random_state=random_state,
)
transform_diag = [[2.5, 0], [0, 0.5]]
X_diag = np.dot(X_diag_raw, transform_diag)
y_diag = np.full(n_samples, 1)

# 3. Full Covariance (Rotated Elliptical) Cluster
X_full_raw, _ = make_blobs(
    n_samples=n_samples,
    centers=[[5, 0]],
    cluster_std=0.5,
    random_state=random_state,
)
transform_full = [[0.8, 0.6], [-0.7, 0.9]]
X_full = np.dot(X_full_raw, transform_full)
y_full = np.full(n_samples, 2)

# Combine the datasets
X = np.vstack((X_iso, X_diag, X_full))
y = np.concatenate((y_iso, y_diag, y_full))
K_TRUE = 3

# 2. Initialize and run the Data Processor
data_processor = DataProcessor()
data_processor.load_and_preprocess_data(X, y=y, test_size=0.3)

```

```

X_test, y_true = data_processor.get_test_data()

# 3. Initialize the Prediction Framework
framework = Prediction(data_processor)
K = K_TRUE

# Initialize the visualizer
visualizer = ClusteringVisualizer()

# --- Run GMM Model ---
GMM_NAME = "GMM_Model_Full_Covariance"
y_pred_gmm = framework.train_and_predict(GMM_NAME, "GMM", K)
framework.evaluate(GMM_NAME, X_test, y_pred_gmm, y_true)
visualizer.plot_clusters(X_test, y_pred_gmm, GMM_NAME, y_true)

# --- Run K-Means++ Model ---
KMEANS_NAME = "KMeansPP_Model"
y_pred_kmeans = framework.train_and_predict(KMEANS_NAME, "KMeans", K)
framework.evaluate(KMEANS_NAME, X_test, y_pred_kmeans, y_true)
visualizer.plot_clusters(X_test, y_pred_kmeans, KMEANS_NAME, y_true)

if __name__ == "__main__":
    run_clustering_framework()

```

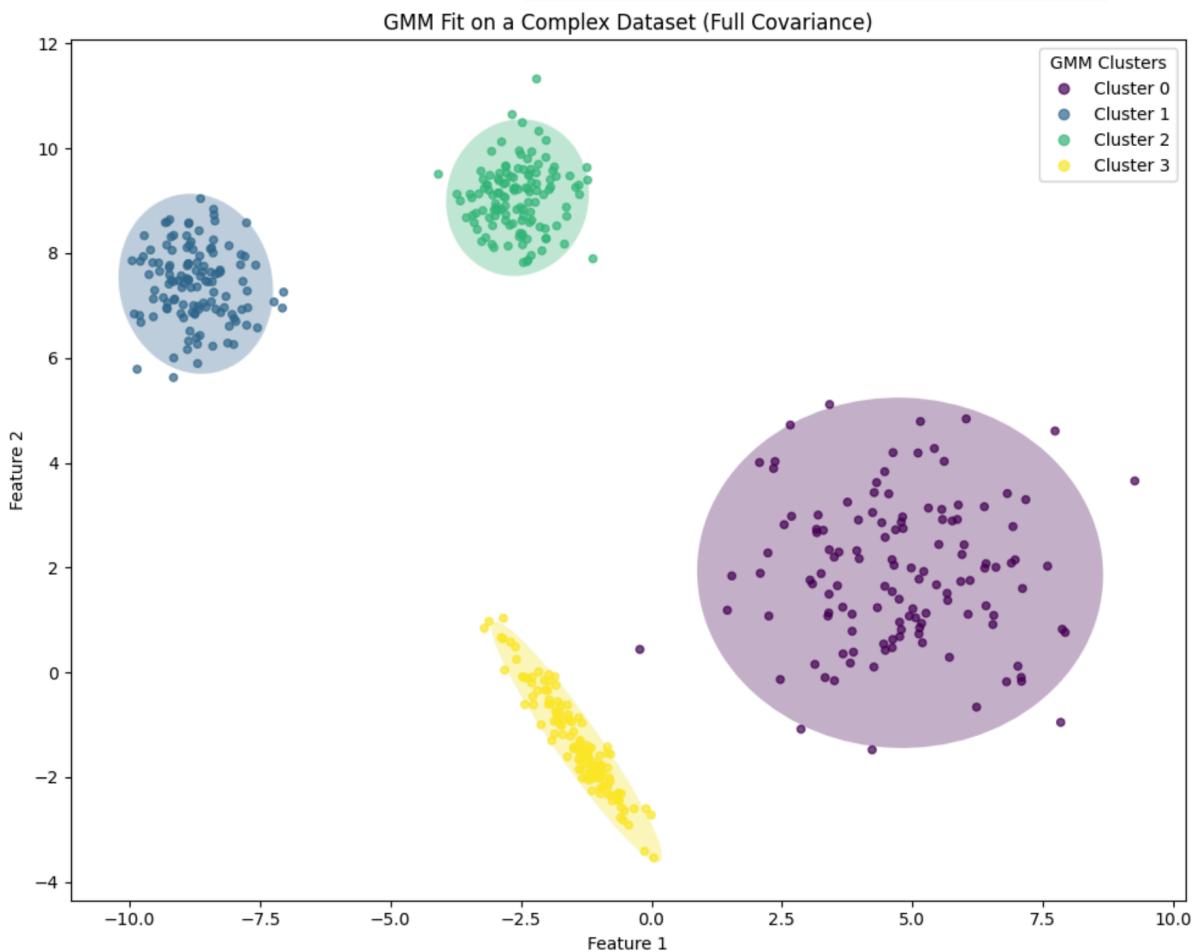


图 6 GMM Visualization

§ 8.3. Density-Based Clustering

原型聚类的方法显而易见存在局限性：

- 对于非凸的簇难以识别，因为收到欧式度量的影响，算法更倾向寻找到凸型或者球形的簇
- 原型聚类高度依赖质心，因此会导致鲁棒性很低
 - 这一点在 GMM 算法的软分隔中得到缓解

密度聚类的方法基于样本分布的紧密程度判断聚类的效果。这从根源上避免了原型聚类带来的问题！

§ 8.3.1. DBS-CAN

Definition 8.3.1.1 DBSCAN.

簇被定义为数据空间中高密度的区域，这些区域被低密度的区域（噪声）隔开

- MinPts (Minimum Points): 密度阈值。定义了一个区域要成为高密度区所需要的最小点数。
- 核心点 (Core Point): 在其 ϵ 半径内，至少有 MinPts 个数据点（包括自身）。是高密度区域的核心，能“生成”簇。
- 边界点 (Border Point) 在其 ϵ 半径内的数据点少于 MinPts 个，但它位于某个核心点的 ϵ 半径内。是簇的边缘，依附于核心点，但本身密度不够高。
- 噪声点 (Noise Point) 既不是核心点，也不是边界点。异常值，不属于任何高密度区域。

Definition 8.3.1.2 Density Reachable.

- 密度可达 (Density-Reachable): 如果一个点 p 沿着一系列核心点的路径可以到达点 q ，那么 q 相对于 p 是密度可达的。
- 密度相连 (Density-Connected): 如果两个点 p 和 q 都可以从同一个核心点 o 密度可达，那么 p 和 q 是密度相连的。

DBSCAN 的算法非常类似于寻找联通分量的 Tarjan 算法，实际上，这个问题本身也可以看做寻找联通分量。形式化来说，DBSCAN 实际上要找到由密度关系导出的最大的密度相连样本集合。

输入: 样本集 $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$;
邻域参数 $(\epsilon, MinPts)$.

过程:

- 1: 初始化核心对象集合: $\Omega = \emptyset$
- 2: **for** $j = 1, 2, \dots, m$ **do**
- 3: 确定样本 \mathbf{x}_j 的 ϵ -邻域 $N_\epsilon(\mathbf{x}_j)$;
- 4: **if** $|N_\epsilon(\mathbf{x}_j)| \geq MinPts$ **then**
- 5: 将样本 \mathbf{x}_j 加入核心对象集合: $\Omega = \Omega \cup \{\mathbf{x}_j\}$
- 6: **end if**
- 7: **end for**
- 8: 初始化聚类簇数: $k = 0$
- 9: 初始化未访问样本集合: $\Gamma = D$
- 10: **while** $\Omega \neq \emptyset$ **do**
- 11: 记录当前未访问样本集合: $\Gamma_{old} = \Gamma$;
- 12: 随机选取一个核心对象 $\mathbf{o} \in \Omega$, 初始化队列 $Q = <\mathbf{o}>$;
- 13: $\Gamma = \Gamma \setminus \{\mathbf{o}\}$;
- 14: **while** $Q \neq \emptyset$ **do**
- 15: 取出队列 Q 中的首个样本 \mathbf{q} ;
- 16: **if** $|N_\epsilon(\mathbf{q})| \geq MinPts$ **then**
- 17: 令 $\Delta = N_\epsilon(\mathbf{q}) \cap \Gamma$;
- 18: 将 Δ 中的样本加入队列 Q ;
- 19: $\Gamma = \Gamma \setminus \Delta$;
- 20: **end if**
- 21: **end while**
- 22: $k = k + 1$, 生成聚类簇 $C_k = \Gamma_{old} \setminus \Gamma$;
- 23: $\Omega = \Omega \setminus C_k$
- 24: **end while**

输出: 簇划分 $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$

图 9.9 DBSCAN 算法

§ 9. Dimension Reduction

§ 9.1. Linear Dimension Reduction

考虑更加一般的线性降维方法, 对于待降维的矩阵 $X = (x_1, x_2, \dots, x_m) \in \mathbb{R}^{d \times m}$

我们需要构造一个线性变换矩阵 $W \in \mathbb{R}^{d \times d'}$

而降维的过程就是简单的线性变换的矩阵乘法: $Y = W^T X \in \mathbb{R}^{d' \times m}$

Recordings Why Linear?.

- 线性变换矩阵的列向量是降维后新坐标系的基向量
- 数据之间的线性关系得以保留
- 更特殊的情况, 如果施加的是正交变换:
 - 基向量构成了一组标准正交基
 - 形成了低维的标准正交子空间
 - 而正交矩阵本质上就是 rotation 操作! 因此带来了保持距离和角度的良好性质。

下面将逐一介绍一些很经典的线性降维和非线性降维的方法。

§ 9.1.1. Multiple Dimensional Scaling

定义原始距离矩阵 D 代表 m 个样本的原始空间的距离矩阵，每个元素代表两个特定数据点的距离表示。我们构建的目标是构建降维后的 m 个样本点的坐标，而保持任意两个样本的欧氏距离保证等于原始空间的距离。（距离矩阵作为不变量）

考虑样本在低维空间中的表示： $Z \in \mathbb{R}^{d' \times m}$ ，计算内积矩阵 $B = Z^T Z \in \mathbb{R}^{m \times m}$

$$b_{i,j} = z_i^T z_j$$

考虑降维后的样本被中心化，则内积矩阵具有良好的性质：每一行和每一列都是和为 0 的。

$$\sum_{i=1}^m dist_{ij}^2 = \text{tr}(\mathbf{B}) + mb_{jj}, \quad (10.4)$$

$$\sum_{j=1}^m dist_{ij}^2 = \text{tr}(\mathbf{B}) + mb_{ii}, \quad (10.5)$$

$$\sum_{i=1}^m \sum_{j=1}^m dist_{ij}^2 = 2m \text{tr}(\mathbf{B}), \quad (10.6)$$

其中 $\text{tr}(\cdot)$ 表示矩阵的迹(trace)， $\text{tr}(\mathbf{B}) = \sum_{i=1}^m \|z_i\|^2$. 令

$$dist_{i\cdot}^2 = \frac{1}{m} \sum_{j=1}^m dist_{ij}^2, \quad (10.7)$$

$$dist_{\cdot j}^2 = \frac{1}{m} \sum_{i=1}^m dist_{ij}^2, \quad (10.8)$$

$$dist^2 = \frac{1}{m^2} \sum_{i=1}^m \sum_{j=1}^m dist_{ij}^2, \quad (10.9)$$

由式(10.3)和式(10.4)~(10.9)可得

$$b_{ij} = -\frac{1}{2}(dist_{ij}^2 - dist_{i\cdot}^2 - dist_{\cdot j}^2 + dist^2), \quad (10.10)$$

由此即可通过降维前后保持不变的距离矩阵 \mathbf{D} 求取内积矩阵 \mathbf{B} .

图 8 MDS Proof

Recordings MDS.

- 求解 MDS 的过程首先求解内积矩阵
- 保证距离的不变形这就会导致内积矩阵的很多完美性质
 - 最终可以保证内积矩阵的每一个元素都被不变量的距离所表示
 - 这是内积所带来的！
- 求出内积矩阵之后再使用特征值分解

求得内积矩阵 B 之后，很显然这并不是一个满秩矩阵，因此可以做矩阵的满秩分解。在这里我们做特征值分解：

$$B = V \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_d) V^\top$$

假设其中有 d^* 个非零特征值（或者在实际运算，可以手动截断前几个最大的特征值），那么 Z 就可以表达为：

$$Z = \sqrt{\text{diag}(\lambda_1, \lambda_2, \dots, \lambda_{d^*})} V_*^\top$$

§ 9.1.2. PCA

我们希望寻找一个超平面：

- 最近重构性：样本点到这个超平面的距离足够近
- 最大可分性：样本在超平面上的投影离的足够远

§ 9.1.2.1. Prof1 for PCA

假定数据样本进行了中心化，即 $\sum_i \mathbf{x}_i = \mathbf{0}$ ；再假定投影变换后得到的新坐标系为 $\{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_d\}$ ，其中 \mathbf{w}_i 是标准正交基向量， $\|\mathbf{w}_i\|_2 = 1$ ， $\mathbf{w}_i^\top \mathbf{w}_j = 0$ ($i \neq j$)。若丢弃新坐标系中的部分坐标，即将维度降低到 $d' < d$ ，则样本点 \mathbf{x}_i 在低维坐标系中的投影是 $\mathbf{z}_i = (z_{i1}; z_{i2}; \dots; z_{id'})$ ，其中 $z_{ij} = \mathbf{w}_j^\top \mathbf{x}_i$ 是 \mathbf{x}_i 在低维坐标系下第 j 维的坐标。若基于 \mathbf{z}_i 来重构 \mathbf{x}_i ，则会得到 $\hat{\mathbf{x}}_i = \sum_{j=1}^{d'} z_{ij} \mathbf{w}_j$ 。

考虑整个训练集，原样本点 \mathbf{x}_i 与基于投影重构的样本点 $\hat{\mathbf{x}}_i$ 之间的距离为

$$\begin{aligned} \sum_{i=1}^m \left\| \sum_{j=1}^{d'} z_{ij} \mathbf{w}_j - \mathbf{x}_i \right\|_2^2 &= \sum_{i=1}^m \mathbf{z}_i^\top \mathbf{z}_i - 2 \sum_{i=1}^m \mathbf{z}_i^\top \mathbf{W}^\top \mathbf{x}_i + \text{const} \\ &\propto -\text{tr} \left(\mathbf{W}^\top \left(\sum_{i=1}^m \mathbf{x}_i \mathbf{x}_i^\top \right) \mathbf{W} \right). \end{aligned} \quad (10.14)$$

根据最近重构性，式(10.14)应被最小化，考虑到 \mathbf{w}_j 是标准正交基， $\sum_i \mathbf{x}_i \mathbf{x}_i^\top$ 是协方差矩阵，有

$$\begin{aligned} \min_{\mathbf{W}} \quad &-\text{tr} (\mathbf{W}^\top \mathbf{X} \mathbf{X}^\top \mathbf{W}) \\ \text{s.t.} \quad &\mathbf{W}^\top \mathbf{W} = \mathbf{I}. \end{aligned} \quad (10.15)$$

这就是主成分分析的优化目标。

下面我们来对上式进行具体证明：

考虑降维前已经归一化的样本 \vec{x}_i ，我们希望降维后的样本为 $\vec{\hat{x}}_i$

For the linear transformation:

$$\hat{\mathbf{X}} = \mathbf{W}^\top \mathbf{X}$$

- 其中 \mathbf{W}^\top 是有标准正交基组成的，并且维度为无损压缩的最大上限，这也是矩阵的秩。
- 但是在实际建模的过程中，我们选择 $d' \leq d$

$$\vec{z}_i = (z_{i1}, z_{i2}, \dots, z_{id'}) = \mathbf{W}^\top \vec{x}_i$$

$$\vec{\hat{x}}_i = \sum_{j=1}^{d'} z_{ij} \vec{w}_j = \sum_{j=1}^{d'} \vec{w}_j^\top \vec{x}_i \vec{w}_j = \mathbb{W}^\top \vec{x}_i$$

$$x_i \in \mathbb{R}^d$$

$$W = (w_1, w_2, \dots, w_{d'}) \in \mathbb{R}^{d \times d'}$$

(新坐标系基向量), 且

$$W^\top W = I_{d'}$$

(标准正交基)。

$$z_i = W^\top x_i \in \mathbb{R}^{d'}$$

(降维坐标), 其中

$$z = w_j^\top x_i$$

◦

$$\hat{x}_i = \sum_{j=1}^{d'} z_{ij} w_j = W z_i = W W^\top x_i$$

重构误差平方和 L :

$$L = \sum_{i=1}^m \|\hat{x}_i - x_i\|_2^2 = \sum_{i=1}^m \left\| \sum_{j=1}^{d'} z_{ij} w_j - x_i \right\|_2^2$$

利用二范数展开式 根据 $\|a - b\|_2^2 = \|a\|_2^2 - 2a^\top b + \|b\|_2^2$, 其中 $a = \hat{x}_i$ 且 $b = x_i$ ◦

$$L = \sum_{i=1}^m \|\hat{x}_i\|_2^2 - 2\hat{x}_i^\top x_i + \|x_i\|_2^2$$

最后一项 $\sum_{i=1}^m \|x_i\|_2^2$

$$\sum_{i=1}^m \|x_i\|_2^2 = \text{const}$$

第二项 $\sum_{i=1}^m \hat{x}_i^\top x_i$ 代入 $\hat{x}_i = W W^\top x_i$ 并利用

$$(W W^\top)^\top = W W^\top$$

(对称性):

$$\sum_{i=1}^m \hat{x}_i^\top x_i = \sum_{i=1}^m (W W^\top x_i)^\top x_i = \sum_{i=1}^m x_i^\top (W W^\top)^\top x_i = \sum_{i=1}^m x_i^\top W W^\top x_i$$

第一项 $\sum_{i=1}^m \|\hat{x}_i\|_2^2$ 代入 $\hat{x}_i = W W^\top x_i$ 并利用

$$W^\top W = I$$

(标准正交性):

$$\begin{aligned}
\sum_{i=1}^m \|\hat{x}_i\|_2^2 &= \sum_{i=1}^m \hat{x}_i^\top \hat{x}_i \\
&= \sum_{i=1}^m (WW^\top x_i)^\top (WW^\top x_i) \\
&= \sum_{i=1}^m x_i^\top (WW^\top)^\top (WW^\top) x_i \\
&= \sum_{i=1}^m x_i^\top (WW^\top WW^\top) x_i \\
&= \sum_{i=1}^m x_i^\top W \left(\underbrace{W^\top W}_I \right) W^\top x_i \\
&= \sum_{i=1}^m x_i^\top WW^\top x_i
\end{aligned}$$

将 A、B、C 三项组合回 L :

$$L = \sum_{i=1}^m \left(\underbrace{x_i^\top WW^\top x_i}_{\text{项 C}} \right) - 2 \left(\underbrace{x_i^\top WW^\top x_i}_{\text{项 B}} \right) + \text{const}$$

$$\text{合并: } L = \sum_{i=1}^m -x_i^\top WW^\top x_i + \text{const} = -\sum_{i=1}^m x_i^\top WW^\top x_i + \text{const}$$

利用迹的性质:

1. 标量 $a = \text{tr}(a)$ 。
2. 循环置换性质: $\text{tr}(ABC) = \text{tr}(BCA) = \text{tr}(CAB)$ 。

对于每一项 $x_i^\top WW^\top x_i$ (它是一个标量):

$$x_i^\top WW^\top x_i = \text{tr}(x_i^\top WW^\top x_i)$$

利用循环置换性质 (将 x_i^\top 搬到最后, 将 x_i 搬到最前):

$$\text{tr}(x_i^\top WW^\top x_i) = \text{tr}(W^\top x_i x_i^\top W)$$

因此, 总和可以写成:

$$\sum_{i=1}^m x_i^\top WW^\top x_i = \sum_{i=1}^m \text{tr}(W^\top x_i x_i^\top W)$$

由于迹和求和运算的线性性质, 可以交换顺序:

$$= \text{tr} W^\top \sum_{i=1}^m x_i x_i^\top W$$

将此结果代回 L :

$$L = -\text{tr} W^\top \sum_{i=1}^m x_i x_i^\top W + \text{const}$$

因此, 最小化重构误差 L 等价于:

$$\min_W - \text{tr} W^\top \sum_{i=1}^m x_i x_i^\top W$$

这等价于最大化负号后面的项：

$$\max_W \text{tr} W^\top \sum_{i=1}^m x_i x_i^\top W \quad s.t. W^\top W = I$$

§ 9.1.2.2. Prof 2 for PCA

从第二个角度来推导主成分分析，我们可以考虑投影样本点的方差最大化，而这个的数据表征就是协方差矩阵。

$$\sum = (\text{Cov}(x_i, x_j))_{i,j}$$

对于原来的矩阵，投影到一个方向上可以得到一个向量，向量的每一个元素代表该方向的值：

$$y = w^\top X$$

投影后数据的方差计算，注意样本已经被归一化：

$$\begin{aligned} \text{Var}(y) &= \mathbb{E}[(y - \mathbb{E}[y])^2] = \mathbb{E}[y^2] \\ \text{Var}(y) &= \frac{1}{m} \sum_{i=1}^m y_i^2 = \frac{1}{m} \sum_{i=1}^m (w^\top x_i)^2 = \frac{1}{m} \sum_{i=1}^m (w^\top x_i)(w^\top x_i)^\top \\ \text{Var}(y) &= \frac{1}{m} \sum_{i=1}^m w^\top (x_i x_i^\top) w = w^\top \left(\frac{1}{m} \sum_{i=1}^m x_i x_i^\top \right) w \\ C &= \frac{1}{m} \sum_{i=1}^m x_i x_i^\top = \text{Cov}(X) \end{aligned}$$

- 注意上面的是指只对中心化的矩阵成立！

因此，从两种优化目标出发，PCA 最终得到了同一个一般的约束条件和最小化目标！

对于具体的求解过程，我们只是需要求解协方差矩阵 $X^\top X$ 的特征值分解并取最大的若干特征值对应的归一化的特征向量。对于实对称矩阵，其特征向量天然的保证正交性。

§ 9.2. Non-Linear Dimension Reduction

下面介绍若干非线性降维的方法，即从高维空间向低维空间的函数映射是非线性的。

§ 9.2.1. Why Non-Linear?

理论上，降维是一个高度泛化的任务，数据点的分布会影响函数映射的具体形式，在某些场景下简单的线性降维难以压缩到低维空间。

线性降维本质是通过一个矩阵 W 施加线性变化，在空间上即寻找数据的最佳线性投影（一个超平面或者子空间），他的基本假设是数据在高维空间中是近似线性的或可以被一个直线/平面很好地表示。

经典的例子，线性降维无法通过投影的手段把一个瑞士卷数据展开，而是会发生严重的重叠和混淆。因此，下文介绍若干非线性的函数映射。

§ 9.2.2. Kernelized PCA

§ 9.2.2.1. Kernelization and Kernel Function

The intuition of **Kernelization** is to find the non-linear mapping from original data samples into a new space \mathcal{F} , ensuring the new data samples $\varphi(x)$ can be split by a linear bound!

Definition 9.2.2.1.1 kernel function.

$$K(x_i, x_j) = \varphi(x_i)^\top \varphi(x_j)$$

- 直接计算内积相似度，无需耗费时间复杂度计算原始的映射
- 核函数最大的用处是将线性算法核化为非线性算法

对于线性投影：

$$\left(\sum_{i=1}^m z_i z_i^\top \right) w_j = \lambda_j w_j$$

$$w_j = \frac{1}{\lambda_j} \left(\sum_{i=1}^m z_i z_i^\top \right) w_j = \sum_{i=1}^m z_i \frac{z_i^\top w_j}{\lambda_j} = \sum_{i=1}^m z_i \alpha_i^j$$

α_i^j 代表着第 i 个数据点在构造第 j 个主成分的时候的权重。

- 对于一般的线性 PCA， z_i 就代表着样本点在高维特征中的像，需要做一次归一化。
- 对于核化 PCA，相当于加了一层函数映射映射到好被线性映射的空间 \mathcal{F} ， $z_i = \varphi(x_i)$ 。

因此：

$$\left(\sum_{i=1}^m \varphi(x_i) \varphi(x_i)^\top \right) w_j = \lambda_j w_j$$

$$\left(\sum_{i=1}^m K(x_i, x_i) \right) w_j = \lambda_j w_j$$

$$w_j = \sum_{i=1}^m \varphi(x_i) \alpha_i^j$$

下面，我们来推导其矩阵形式：

$$\left(\sum_{i=1}^m K(x_i, x_i) \right) \left(\sum_{k=1}^m \varphi(x_k) \alpha_k^j \right) = \lambda_j \left(\sum_{k=1}^m \varphi(x_k) \alpha_k^j \right)$$

展开：

$$\sum_{i=1}^m \sum_{k=1}^m \varphi(x_i) \varphi(x_i)^\top \varphi(x_k) \alpha_k^j = \lambda_j \left(\sum_{k=1}^m \varphi(x_k) \alpha_k^j \right)$$

合并到核函数并且左乘 $\varphi(x_l)^\top$ ：

$$\sum_{i=1}^m \sum_{k=1}^m K(x_i, x_k) K(x_l, x_i) \alpha_k^j = \lambda_j \sum_{k=1}^m K(x_l, x_k) \alpha_k^j$$

转化为矩阵形式：

$$(K^2 \alpha^j)_l = \lambda_j (K \alpha^j)_l$$

因为 l 的选择是任意的，因此可以推广：

$$K^2 \alpha^j = \lambda_j K \alpha^j$$

在实际情况下，为了更方便，我们求解下面的简洁形式：

$$K \alpha^j = \lambda_j \alpha^j$$

回到最终 KPCA 的求解目标：

$$z_j = w_j^\top \varphi(x) = \sum_{i=1}^m \alpha_i^j \varphi(x_i)^\top \varphi(x) = \sum_{i=1}^m \alpha_i^j K(x_i, x)$$

Recordings Kernel Function.

- 核化方法的最精彩的地方就在于不求解复杂的函数映射 φ ，而是转化为求解核化矩阵，因为最终还是划归到内积求解相似度的场景上！
- 具体的核函数可以手动选择一些非线性的核函数，例如高斯核

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$$

§ 9.2.3. Manifold Learning

虽然数据在高维空间中看起来非常复杂，但它很可能内嵌（嵌入）在一个低维的、弯曲的“表面”上，这个“表面”就是流形（Manifold）。

流形学习中，我们希望解卷这个高维的流型，类似于把一个瑞士卷拉直，找出数据的低维内在结构。

§ 9.2.3.1. Isomap

等度量映射的关键在于在流型中使用欧式空间度量距离会产生较大的误差。但是，我们可以利用流型在局部上与欧式空间同胚的性质，把距离的度量转化为近邻连接图上最短路径的问题。具体实现可以使用 Dijkstra Algorithms。

在得到距离后，在利用 MDS 算法保持距离不变形进行线性压缩。

§ 9.2.3.2. Locally Linear Embeddings (LLE)

保持每个坐标点可以被领域样本线性组合而重构。

$$x_i = w_{ij} x_j + w_{ik} x_k + w_{il} x_l$$

并且这些权重的和为 1。

具体来说，可以变成这个凸优化问题：

$$\min_{w_1, w_2, \dots, w_m} \sum_{i=1}^m \left\| \left(x_i - \sum_{j \in Q_i} w_{ij} x_j \right) \right\| \text{ s.t. } \sum_{i=1}^m w_{ij} = 1$$

上述问题存在闭式解。

LLE 在低维空间中保持 w_i 不变，并求解下面的优化目标：

$$\min_{z_1, z_2, \dots, z_m} \sum_{i=1}^m \left\| \left(z_i - \sum_{j \in Q_i} w_{ij} z_j \right) \right\|$$

矩阵形式表达:

$$\begin{aligned} \text{令 } \mathbf{Z} = (\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_m) \in \mathbb{R}^{d' \times m}, & (\mathbf{W})_{ij} = w_{ij}, \\ \mathbf{M} = (\mathbf{I} - \mathbf{W})^T (\mathbf{I} - \mathbf{W}), & \end{aligned} \quad (10.30)$$

则式(10.29)可重写为

$$\begin{aligned} \min_{\mathbf{Z}} \text{tr}(\mathbf{Z} \mathbf{M} \mathbf{Z}^T), \\ \text{s.t. } \mathbf{Z} \mathbf{Z}^T = \mathbf{I}. \end{aligned} \quad (10.31)$$

式(10.31)可通过特征值分解求解: \mathbf{M} 最小的 d' 个特征值对应的特征向量组成的矩阵即为 \mathbf{Z}^T .

图 10 LLE Algorithms

§ 10. Metric Learning

§ 11. Conclusion