数学建模参考

Ray Xia

2025/07/05

目录

1 前言 *Forward*

本文作为数学建模的非权威性参考资料，主要包含了针对全国大学生数学建模大赛（国赛）的相关参考资料，以便翻阅查找。程序编写主要采用python语言，使用的主要库有：

numpy, pandas 无需多盐

sklearn 大部分经典算法

keras 神经网络

matplotlib 绘图

pymoo 多目标优化

pygad 遗传算法

程序的编写标准基于Google Python Style Guide进行调整：

1. 执行类文件（.ipynb）中禁止使用异常处理，所有函数在**输入正确的情况下**都用该成功返回。非执行类文件中可以简单使用（如检测参数是否正确等）。
2. 所有执行类文件以Jupyter Notebook(.ipynb)形式；非执行类文件（库，配置文件）都使用.py形式。
3. 所有执行类文件都应该能够从头到尾运行一次，并得到正确结果（不要依赖特殊的Jupyter Notebook的执行顺序）。使用Jupyter Notebook只是为了方便编写调试。

2 数学基础 *Mathemathical Fundamentals*

高等数学 包含一（多）元函数极限、一（多）元函数微分学，一（多）函数积分学、等

线性代数 矩阵、特征值、等

概率统计 作者也没有学过，但是马尔科夫链等模型应该和这个有点关系。只是理解模型的话，高中的知识就足够了。

3 符号与惯用标记 *Notations*

Notation	Meaning
x, y, z	Generic Variables
f, g, h	Generic Functions
$\mathbf{x_n}, \mathbf{y_n}, \mathbf{z_n}$	Vector Variables with a dimension of n
$\mathbf{f_n}, \mathbf{g_n}, \mathbf{h_n}$	Vector Functions with a dimension of n
$\mathbf{A}, \mathbf{B}, \mathbf{M}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	Matrices
$\mathbf{M}^T$	Transposed Matrices
$\mathbf{a} > (<, \geqslant, \leqslant) \mathbf{b}$	$\forall a_i > (<, \geqslant, \leqslant) b_i$
$\|\mathbf{x}\|_n$	$\mathbf{x}$的n范数，如果是二范数则可以省略n
$A := B$	定义A为B

如果没有特殊说明，所有向量都是列向量。

4 常用模型/算法 *Common Models / Algorithms*

此部分包含了数学建模中常用的一些模型或算法。
根据模型/算法的用途，可将其分为：

- 优化算法

$$\begin{aligned} &minimize\ \mathbf{x}, \\ s.t.\ &\mathbf{F}(\mathbf{x}) = 0 \end{aligned}$$

- 预测算法

$$\text{history data} \overset{algorithm}{\Rightarrow} \text{experience (model)} \Rightarrow \text{prediction}$$

- 评价算法

$$\text{data} \overset{\text{(standard)}}{\Rightarrow} \text{scoring}$$

- 其他 权重决策（重要性分析）

每个部分除特殊情况都应该包含一下要点：

4.1 示例 Example

这是该模型/算法的基本介绍。

- 原理

  这是更为详细的原理解释。

  可能还会附带一些数学推理/公式和插图。

  $$x = 1 + 1 \tag{1}$$
  $$x = 2$$

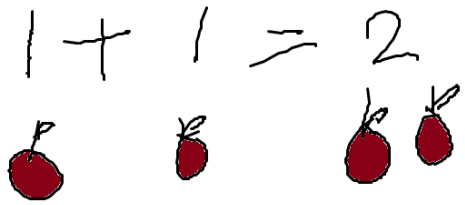

图 1: placeholder

- 特性

  这是算法特性的基本介绍。

  - （优点）
  - （缺点）
  - （适用场景）
  - （其他）

- python 代码实现

```
if __name__ == "__main__":
    print("hello world")
```

根据模型/算法的用途，可将其分为：

优化算法

$$minimize\ \mathbf{x},$$
$$s.t.\ \mathbf{F}(\mathbf{x}) = 0$$

预测算法

$$\text{history data} \overset{algorithm}{\Rightarrow} \text{experience (model)} \Rightarrow \text{prediction}$$

评价算法

$$\text{data} \overset{\text{(standard)}}{\Rightarrow} \text{scoring}$$

其他 权重决策（重要性分析）

需要注意的是：

1. 每个部分的算法都大致按照算法的使用率/优先程度从高到低排列。
2. 部分算法可能会有多个用途，为了避免重复，这些算法在全文中只会出现一次。

4.2 优化算法

4.2.1 线性规划（单目标） Linear Programming

`https://en.wikipedia.org/wiki/Linear_programming`
最基本的有约束的优化问题，更加复杂的形式包含整形规划（Integer Programming）此部分只讨论单目标规划的情况。

- 原理

  限制条件的并集会会形成一个可行域，最优解必然出现在该可行域的边界上。

$$\begin{aligned}&\min \mathbf{c}^T\mathbf{x}\\ &s.t.\ \mathbf{A}\mathbf{x} \leqslant \mathbf{b}\end{aligned} \overset{or}{\Leftrightarrow} find\ \min\{\mathbf{c}^T\mathbf{x} | \mathbf{A}\mathbf{x} \leqslant \mathbf{b}\} \tag{2}$$

- 特性

  线性规划可以说是最简单的一个优化算法了。

  - 简单，找到的必定是全局最优解。
  - 需要把条件转化成可以计算的数学条件，只能用于线性的条件。
  - 适用于简单，孤立的问题。

- python 代码实现

  线性规划使用的是python的pulp库。

  示例可见`src/linear_programming/main.ipynb` g'g

4.2.2 遗传算法 Genetic Algorithm

通过变异、选择的方式优化问题。

- 原理

  目标：

$$\begin{aligned} &\min \Phi(\mathbf{x}) \\ &\quad \text{, where } \Phi \text{ is an unknown or overly complex function} \\ &s.t.\ \mathbf{Ax} \leqslant \mathbf{b} \end{aligned} \tag{3}$$

  1. 初始化一个参数随机的种群。
  2. 用函数检测每个个体的适应程度。
  3. 适应程度高的个体被选为父本。
  4. 父本之间结合生成新个体。
  5. 新个体发生变异。
  6. 新个体取代种群中的部分个体。
  7. 重复 2-6 步。

- 特性

  遗传算法比较简单粗暴，效率比较低下，说不出来什么名道，不建议使用。

  - 黑盒化对待，不需要知道目标函数的具体形式，只需要输入和输出。
  - 得到的可能是局部最优；参数选择对最后结果有很大影响。

- python 代码实现

  使用的是pygad库。

  `https://pygad.readthedocs.io/en/latest/index.html`

  示例见`src/genetic_algorithm/main.ipynb`

4.2.3 多目标遗传优化算法 NSGA-II

GA + 多目标优化来获得一个区间内的 Pareto Front.

- 原理

  多目标优化中遗传算法的选择标准（non-dominated sorting）：

  Given an objective function of multiple dimensions to **minimize**: $\mathbf{F}$.

  And X is the set of the whole population.

$$\begin{aligned}
&\widetilde{F_1} = \{\mathbf{x} | \mathbf{x} \in X \wedge (\exists \mathbf{x}_0 \in X \, (\mathbf{F}(\mathbf{x}) > \mathbf{F}(\mathbf{x}_0)))\}, \\
&\widetilde{F_i} = \{\mathbf{x} | \mathbf{x} \in X - \bigcup_{k=1}^{i-1} F_k \wedge (\exists \mathbf{x}_0 \in X - \bigcup_{k=1}^{i-1} F_k \, (\mathbf{F}(\mathbf{x}) > \mathbf{F}(\mathbf{x}_0)))\}.
\end{aligned} \tag{4}$$

- 特性

  说不出来什么名道，不建议使用。

  - 黑盒化对待，不需要知道目标函数的具体形式，只需要输入和输出。
  - 复杂；得到的是一组Pareto Front，还需要进一步选出最优方案。

- python 代码实现

  使用的是pymoo库。`https://pymoo.org/algorithms/moo/nsga2.html`

4.2.4 蚁群算法 Ant Colony Optimization

4.2.5 模拟退火 Simulated Annealing

4.3 预测算法

4.3.1 线性回归 Linear Regression

用一个一次函数拟合多个样本点，目标是使预测结果和样本点的方差最小。

- 原理

$$\begin{aligned} &\text{假设样本点为}\mathbf{x}_0 \mapsto \mathbf{y}_0 \\ &\min_{\mathbf{A},\mathbf{b}} \|\mathbf{A}\mathbf{x}_0 + \mathbf{b} - \mathbf{y}_0\| \end{aligned} \tag{5}$$

- 特性

  对线性特征明显，无大幅度波动的数据表现良好。

  - 简单、可解释。
  - 只适用于线性性良好的数据。

- python 代码实现

  用的是scikit-learn库
  示例见`linear_models\linear_regression.ipynb` 注意reshape。

4.3.2 线性回归ultra（岭、Lasso、弹性网回归） Ridge, Lasso, Elastic-Net Regression

岭回归、Lasso、弹性网回归是为了减少数据波动对得回归函数的影响。

- 原理

  这三种回归在线性回归的基础上增加了不同的规范函数（α, ρ都是参数）：

线性回归：

$$\min \|\mathbf{Ax} + \mathbf{b} - \mathbf{y_0}\|_2$$

岭回归：

$$\min \|\mathbf{Ax} + \mathbf{b} - \mathbf{y_0}\|_2^2 + \alpha\|\mathbf{A}\|^2$$

Lasso回归：

$$\min \frac{\|\mathbf{Ax} + \mathbf{b} - \mathbf{y_0}\|_2^2}{2n_{\text{samples}}} + \alpha\|\mathbf{A}\|_1 \tag{6}$$

弹性网回归：

$$\min \frac{\|\mathbf{Ax} + \mathbf{b} - \mathbf{y_0}\|_2^2}{2n_{\text{samples}}} + \alpha\rho\|\mathbf{A}\|_1 + \frac{\alpha(1-\rho)}{2}\|\mathbf{A}\|_2^2$$

α, ρ确定方法：

1. Cross Validation
2. Akaike Information Criterion / Bayes Information Criterion

- 特性
  - 适合拟合波动较大的数据。
  - 要确定合适的规范参数（α, ρ）。
  - 建议在普通线性回归无法良好拟合的时候使用。
- python 代码实现

  建议先将数据正规化（normalize）或标准化（standardize）再进行拟合。

  示例见`src/linear_models/ridge_regression.ipynb`
  `src/linear_models/Lasso_regression.ipynb`
  `src/linear_models/elastic-net_regression.ipynb`

4.3.3 随机森林 Random Forest

bagging算法+决策树+voting

- 原理

  将数据随机分组，用来训练多个单独的决策树，预测的时候所有决策树的投票。

  procedure TRAIN(TrainingData)
  - **for all** DecisionTree ∈ Forest **do**
    - TrainingDataSet ⊂ TrainingData
    - DecisionTree → FIT(TrainingDataSet)

  procedure PREDICT(Input)
  - Votes
  - **for all** DecisionTree ∈ Forest **do**
    - DecisionTree → VOTE(Input)
  - **return** max Votes

- 特性

  主要是解决了决策树容易过拟合的问题。

  – 快。

  – 不适合维度很高的数据集；只建议用来预测在输入数据范围内的数据点，不太能预测显著在样本范围之外的数据点。

- python 代码实现

  示例见`src\decision_tree\random_forest.ipynb`

4.3.4 自回归综合移动平均 ARIMA

时间序列模型：自回归+综合+移动平均。

- 原理

  自回归、综合、移动平均分别的原理：

AutoRegression(**AR)(p):**

$$y_t = \boldsymbol{\phi}^T \mathbf{y}_{t-1\sim t-p} + c + \epsilon_t$$

Integrated(**I)(d):**

$$\begin{aligned} &\text{find } d \in \mathbb{N} \text{ so that} \\ &\{y_n\} \text{ is a } d^{th} \text{ order arithmatic sequence.} \\ &\text{（d阶等差数列）} \end{aligned} \tag{7}$$

MovingAverage(**MA)(q)**

$$y_t = \mu + \boldsymbol{\theta}^T \boldsymbol{\epsilon}_{t-1\sim t-q} + \epsilon_t$$

其中$\mathbf{y}_{i\sim j}$表示从第i个到第j个的样本点； $\boldsymbol{\phi}, \boldsymbol{\theta}, c, \mu$表示有待优化的变量； ϵ_t表示第t次预测的误差。

所以整个ARIMA时间序列模型有三个参数：p, d, q需要在拟合之前确定，也可以用Auto ARIMA让算法自动确定这些参数。

- 特性

  结合了三种算法，能够辨识数据中的整体变化趋势、周期性。

  - 准确度高，可解释性强。
  - 手动确定参数有一定技术含量。
  - 时间序列模型的标杆。

- python 代码实现

  示例见`src\time_series\ARIMA.ipynb`，
  `src\time_series\AutoARIMA.ipynb`

4.3.5 马尔科夫链/预测 Markov Prediction Model

使用（多次）条件概率的方式预测未来某个状体的可能性。

- 原理

  假设某个事件总共有有限个可能状态：$\{A_1, A_2, A_3, \ldots, A_n\}$，且同时有且仅有一个状态发生。

现在有T次历史发生过的状态的记录：$\{A_{i_1}, A_{i_2}, \ldots, A_{i_T}\}$。

那么A_i发生之后紧接着发生A_j的频率为：

$$f_{j,i} := \mathbf{f}(A_j \mid A_i) = \frac{N_{A_i \to A_j}}{N_{A_i \to any}}$$

显然有：

$$(\forall i \in [1, n]) \left(\sum_{j=1}^{n} f_{j,i} = 1 \right)$$

如果**把频率作为概率**，那么就得到了状态转移的概率矩阵：

$$\mathbf{P}_{n \times n} = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,n} \end{bmatrix} = [f_{i,j}] = \begin{bmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,n} \\ f_{2,1} & f_{2,2} & \cdots & f_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{n,1} & f_{n,2} & \cdots & f_{n,n} \end{bmatrix}$$

假设最近一次的状态为A_i，那么下一次每个状态发生的概率为：

$$\mathbf{p}_1 := \begin{bmatrix} \mathbf{P}(A_1) \\ \mathbf{P}(A_1) \\ \vdots \\ \mathbf{P}(A_1) \end{bmatrix} = \begin{bmatrix} \mathbf{P}(A_1 \mid A_i) \\ \mathbf{P}(A_1 \mid A_i) \\ \vdots \\ \mathbf{P}(A_1 \mid A_i) \end{bmatrix} = \mathbf{P} \times \begin{bmatrix} 0 \\ \vdots \\ 1_{\text{(i-th element)}} \\ \vdots \\ 0 \end{bmatrix} = \mathbf{P} \times \mathbf{e}_i$$

后面第k次，每个状态发生的概率为：

$$\mathbf{p}_k := \begin{bmatrix} \mathbf{P}(A_1) \\ \mathbf{P}(A_1) \\ \vdots \\ \mathbf{P}(A_1) \end{bmatrix} = normalized(\mathbf{P}^k \times \mathbf{e}_i)$$

让$k \to \infty$可以得到整体的发展趋势：

$$\mathbf{p} := \lim_{k \to \infty} \mathbf{p}_k = \mathbf{P}^{\infty} \times \mathbf{e}_i$$

- 特性

  就业场景较为狭窄。

  - 可解释性强，使用简单。
  - 非常粗糙地假设每次状体只与前一次的状态有关。

- python 代码实现

  实现见：`src/probabilities/markov_model.py`
  示例见：`src/probabilities/markov_model.ipynb`

4.3.6 梯度提升树 Gradient Boosting Trees

除了随机森林的另一种基于决策树的拟合算法。

- 原理

  用一个新的决策树去拟合已有模型的误差，在整合到模型当中。

  procedure TRAIN(TrainingData)
  repeat
  Errors ← (Model → ERRORS(TrainingData))
  NewTree → FIT(Errors)
  Model += NewTree
  until N Trees in Model

- 特性

  - 准确度高，可解释性强。
  - 拟合比较缓慢；要确定合适的参数（决策树的数量）来防止过拟合。

- python 代码实现

  对样本数量比较少的情况（$< 10,000$）建议使用 GradientBoosting-Classifier / GradientBoostingRegressor

对样本数量比较多的情况（$> 10,000$）建议使用 HistGradientBoostingClassifier / HistGradientBoostingRegressor

示例见`src/decision_tree/gradient_boosting.ipynb`

4.3.7 灰色预测 Grey Method

一种通过累加使数据趋势更加明显后，再进行预测的预测算法。

- 原理

  通过求和（或积分）使数据趋势更加明显。

  假设数据x是连续的，X是其原函数（为了方便这里讨论的是一维数据）：

  $$X = \int_0^t x \,\mathrm{d}t$$

  假设$\mathbf{X}$满足常微分方程，也就是满足指数关系：

  $$\begin{aligned} \frac{\mathrm{d}X}{\mathrm{d}t} + aX &= b \\ \Leftrightarrow X &= \frac{b}{a} + Ce^{-ax} \\ \Rightarrow x = \frac{\mathrm{d}X}{\mathrm{d}t} &= -aCe^{-ax} \end{aligned}$$

  现在的问题就是如何得到合适的参数a, b, C。

  假设有x在$t_0, t_1, \ldots, t_n$时刻的数据$x(t_0), x(t_1), \ldots, x(t_n)$。那么可以估计$X$在对应时刻的数值（假设$t_0 = 0$）：

  $$\begin{aligned} X(t_i) &= \int_0^{t_i} x \,\mathrm{d}t \\ &= \sum_{k=0}^{i-1} \int_{t_k}^{t_{k+1}} x \,\mathrm{d}t \\ &\approx \sum_{k=0}^{i-1} \frac{1}{2}(x(t_k) + x(t_{k+1}))(t_{k+1} - t_k) \end{aligned}$$

有了x, X就可以用最小二乘法求出常微分方程中的a, b，再对X的解析式用一次最小二乘，来求得合适的C。

- 特性
  - 非常简单，可解释性强；对噪声有一定屏蔽能力。
  - 对异常值应对能力差；长期预测能力差。
  - 适用于样本点较少的情况。
- python 代码实现

  `src/grey_model/gm_1_1.py`

  示例见`src/decision_tree/main.ipynb`

4.3.8 朴素贝叶斯 Naive Bayes

一种基于贝叶斯理论的预测模型。

- 原理

  贝叶斯模型假设自变量之间都是独立的。

$$
\begin{aligned}
P(y \mid x_1 x_2 \dots x_n) &= \frac{P(y \wedge x_1 x_2 \dots x_n)}{P(x_1 x_2 \dots x_n)} \\
&= \frac{P(y) P(x_1 x_2 \dots x_n \mid y)}{P(x_1 x_2 \dots x_n)}
\end{aligned}
$$

因为x_i之间都是独立的，所以有：

$$
P(y \mid x_1 x_2 \dots x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1 x_2 \dots x_n)}
$$

$$
\begin{aligned}
\Rightarrow E(y) &= \arg\max_y P(y \mid x_1 x_2 \dots x_n) \\
&= \arg\max_y \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1 x_2 \dots x_n)} \\
&\overset{P(x_1 x_2 \dots x_n) = C(y)}{=} \arg\max_y \left[P(y) \prod_{i=1}^{n} P(x_i \mid y) \right]
\end{aligned}
$$

- 特性
  - 非常简单，可解释性强；速度很快。
  - 精度较低；在实际情况中假设不一定成立。
- python 代码实现

  示例见`src/probabilities/naive_bayes.ipynb`

4.3.9 决策树 Decision Tree

- 原理

  通过增加分支和判断条件得出预测结果。

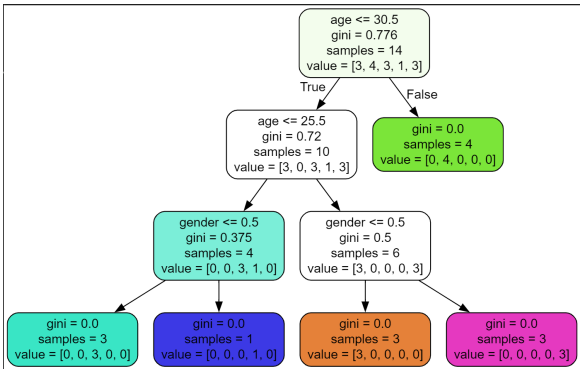

图 2: 决策树示意图

- 特性
  - 非常简单，可解释性强；速度很快。
  - 容易受噪声影响；容易过拟合。
- python 代码实现

  示例见`src/decision_tree/decision_tree.ipynb`

4.3.10 K近邻 K-Nearest Neighbours

根据预测点周围最近的K个样本点的标签确定预测标签。

- 原理

  通过增加分支和判断条件得出预测结果。

  对于数据集S中的样本点和对应的标签：(x, y)。对于预测点$\hat{x}$：

$$N : (N \subset S) \wedge (|N| = k) \wedge$$
$$(\forall (x, y) \in N, (x', y') \in S \backslash N)(\|x - \hat{x}\| < \|x' - \hat{x}\|)$$

  预测结果$\hat{y}$为N中出现次数最多的数据点（也可以用距离作为权重进行累加）。

- 特性

  – 参数k的确定对模型的准确性影响很大；对噪声和异常值敏感；容易过拟合或者拟合不良；对较大的数据集速度慢。

- python 代码实现

  示例见`src/k_nearest_neighbors/main.ipynb`

4.3.11 K均值聚类 K-Means Clustering

非监督学习聚类算法。

- 原理

  procedure TRAIN(DataPoints)
  - Randomly select Centroids
  - **repeat**
    - DataPoints → closest Centroid
    - Centroids = MEAN(data points)
  - **until** Centroids stop changing

- 特性

  – 聚类数k必须预先确定；对于线性性较差的数据集表现较差。

- python 代码实现

  示例见`src/k_means_clustoring/main.ipynb`

4.3.12 支持向量机 Support Vector Machine

通过构造一个超平面，再通过计算样本点到超平面的距离来确定预测结果。

- 原理

  假设样本是n维的数据集，预测标签只有两种类型。可以尝试在样本的n维空间中找出一个超平面，使该平面将不同标签的样本点分布在平面的两侧，并使不同样本点离平面的距离最大。

  几个常见的问题和解决方法：

  - 样本点不是线性可分的
    - 近似线性可分 设置一定宽容度，使得距离在这个范围内的样本点可以分布在超平面的任意一侧。
    - 完全不线性可分 添加核函数（Kernel Function）改变样本点在空间中的分布形态。
  - 有多种（> 2）预测标签
    假设有m种不同的标签，可以使用以下策略：
    - One versus One 创建$\frac{m(m-1)}{2}$个SVM模型。
    - One versus Rest 创建m个SVM模型。

- 特性

  - 可解释性强；找到的支持向量**可能**有实际意义；适用于样本维度大于样本数的情况。
  - 对样本数很大的数据集表现较差。

- python 代码实现

  示例见`src/support_vector_machine/main.ipynb`

4.3.13 神经网络 Neural Network

基本的神经网络（MLP）。

- 原理

一个神经网络由多个层构成，每一层由多个节点组成，每个结点的值都是前一层所有结点分别的仿射变换套上一个激活函数。

记$n_{i,j}$为第i层的第j个节点，那么有：

$$n_{i+1,j} = \sum_j \left[a_{(i,k)\to(i+1,j)} n_{i,k} + b_{(i-1,k)\to(i+1,j)}\right]$$

- 特性
  - 可解释性强；找到的支持向量**可能**有实际意义；适用于样本维度大于样本数的情况。
  - 对样本数很大的数据集表现较差。
- python 代码实现

  示例见`src/support_vector_machine/main.ipynb`

4.3.14 逻辑斯谛回归 Logistic Regression

4.3.15 密度聚类 DBSCAN

4.3.16 卷积神经网络 Convolutional Neural Network

4.4 评价算法

4.4.1 优劣解距离法 TOPSIS

基于相对最优&最差方案的距离进行评价。

- 原理

  注意：以下步骤省略了将数据标准化的过程

  假设有n个方案：$\{\mathbf{x}_n\}$，每个方案有m个指标：

  $$\mathbf{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,m}]$$

  每个指标的最优与最差情况分别为：

$$\mathbf{x}^+ = [x_1^+, x_2^+, \ldots, x_m^+]$$
$$\mathbf{x}^- = [x_1^-, x_2^-, \ldots, x_m^-]$$

可以得到每个方案与最优与最差方案之间的距离：（**w**表示每个指标的权重）

$$d_i^+ = \|\mathbf{w} * (\mathbf{x}_j - \mathbf{x}^+)\|$$
$$d_i^- = \|\mathbf{w} * (\mathbf{x}_j - \mathbf{x}^-)\|$$
where $*$ means multiplying the elements individually.

由此得到每个方案的评分：

$$\text{score}_i = \frac{d_i^-}{d_i^+ + d_i^-}$$

- 特性

  标准的评价算法。

  - 可解释性强。
  - 要手动确定权重。

- python 代码实现

  使用numpy和pandas库实现：`src/topsis/topsis.py`
  示例见`src/topsis/main.ipynb`

4.4.2 模糊综合评价法 Fuzzy Comprehensive Evaluation

4.5 其他

4.5.1 主成分分析 Principle Component Analysis

一种数据降维的方法。

- 原理

将一个高维的线性空间降到低维的线性空间，同时尽量保留数据的信息。其关键就是找出一个合适的高维到低维的线性变换。

以下步骤省略了去平均值

假设原始数据为n维，要降维到m维，可以记线性变换的矩阵为：$\boldsymbol{\Phi}_{m\times n}$

降维前后的数据应该为：

$$\mathbf{X}_{n\times k} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k] \text{where } \mathbf{x}_i \text{ is a n-dimensional vector.}$$
$$\mathbf{X}'_{m\times k} = \boldsymbol{\Phi} \times \mathbf{X}$$

为了使降维后的数据信息量最大化，应该使$\mathbf{X}'$中的**行向量**之间的协方差最小化。

$$\begin{aligned}
&\min Cov\mathbf{X}' \\
\Rightarrow &\text{使} \begin{bmatrix} Cov(r_1,r_1) & Cov(r_1,r_2) & \ldots & Cov(r_1,r_m) \\ Cov(r_2,r_1) & Cov(r_2,r_2) & \ldots & Cov(r_2,r_m) \\ \vdots & \vdots & \ddots & \vdots \\ Cov(r_m,r_1) & Cov(r_m,r_2) & \ldots & Cov(r_1,r_m) \end{bmatrix} \text{接近对角矩阵} \\
&= \frac{1}{n}\mathbf{X}'\mathbf{X}'^T \\
&= \frac{1}{n}(\boldsymbol{\Phi}^T\mathbf{X})(\boldsymbol{\Phi}^T\mathbf{X})^T \\
&= \frac{1}{n}\boldsymbol{\Phi}^T\mathbf{X}\mathbf{X}^T\boldsymbol{\Phi} \\
\Rightarrow &\text{对角化}\mathbf{X}\mathbf{X}^T \\
&\text{方法：特征值 + 特征向量}
\end{aligned}$$

因为进行的是降维，所以矩阵$\boldsymbol{\Phi}$不能包含所有特征向量，应该从特征值最大的开始选。

- 特性

  比较简单的数据降维方法。

  – 可解释性强。

  - 降维后的数据的每个维度没有什么现实意义。
  - 可以用来数据降维，降低数据噪声，防止后续过拟合。

- python 代码实现

4.5.2 熵权法 Entropy Weight Method

4.5.3 层次分析法 Analytic Hierarchy Process

5 绘图

此部分主要简单介绍matplotlib绘图中的较为常用和通用的功能。处非额外区分，此部分的功能应该同时使用2维图表和3维图表。

5.1 图表结构

5.2 图标内容

6 致谢

I don't know why this vidoe is made private:
https://www.youtube.com/watch?v=BT6Aw6Q75Yg
Here is a reupload from Bilibili
https://www.bilibili.com/video/BV1MS411A72D