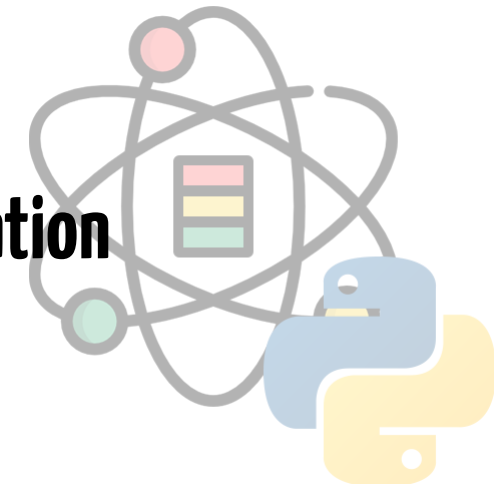
Python 数据科学导论
Data Science Introduction with Python

模型评估 & 超参数优化
Model Evaluation & Hyperparameter Optimization

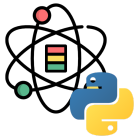
范叶亮



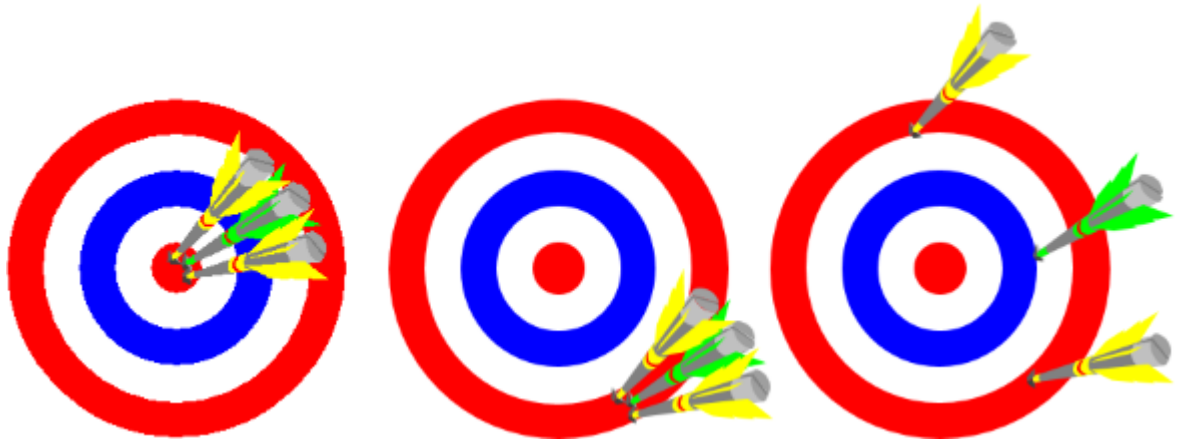
目录

模型性能评估

模型性能评估

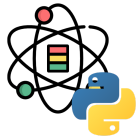


对学习器的泛化性能进行评估，不仅需要有效可行的实验评估方法，还需要有衡量模型泛化能力的评价标准，也就是性能度量（performance measure）。性能度量反映了任务需求，在对比不同模型的能力时，使用不同的性能度量往往会导致不同的评判结果，这意味着模型的“好坏”是相对的，什么样的模型时好的，不仅仅取决于算法和数据，还决定于任务需求。



[1] 图片来源：https://steemit.com/science/@cryptotrendz/the-difference-between-industry-and-academia

回归问题性能评估



平均绝对误差（mean absolute error, MAE）

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|f\left(x_i\right) - y_i\right| \tag{1}$$

平均绝对百分比误差（mean absolute percentage error, MAPE）

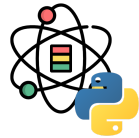
$$\mathrm{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{f\left(x_i\right) - y_i}{y_i}\right| \tag{2}$$

均方误差（mean squared error, MSE）

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(f\left(x_i\right) - y_i\right)^2 \tag{3}$$

其中， $SSE = \sum_{i=1}^{n}\left(f\left(x_i\right) - y_i\right)^2$，称之为残差平方和。

回归问题性能评估



均方根误差（root-mean-square error, RMSE）

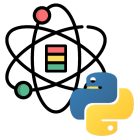
$$\mathrm{RMSE} = \sqrt{\mathrm{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(f\left(x_i\right) - y_i\right)^2} \tag{4}$$

误差标准差

$$\sigma_e = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(e_i - \bar{e}\right)^2} \tag{5}$$

其中， e_i表示使用的误差， $\bar{e} = \frac{1}{n}\sum_{i=1}^{n} e_i$

回归问题性能评估



R^2 称为确定或多重确定（在多重线性回归中）的系数，一般而言， R^2 越大，模型与数据拟合得越好，其值在 0 与 100% 之间变动。

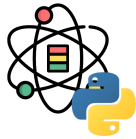
$$SSR = \sum_{i=1}^{n} \left(f\left(x_i\right) - \bar{y}\right)^2 \qquad SST = \sum_{i=1}^{n} \left(y_i - \bar{y}\right)^2 \tag{6}$$

$$R^2 = \frac{SSR}{SST} \qquad \bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i \tag{7}$$

R^2 不只是模型误差的函数，它的定义中包含了两个模型的比较：一个是当前被分析的模型，一个是所谓的常数模型，即只利用因变量均值进行预测的模型。因此， R^2 回答的是“我的模型是否比一个常数模型更好？”这样一个问题，然而我们通常想要回答的是另一个完全不同的问题：“我的模型是否比真实的模型更差？”

除了 R^2 以外，通常还有一种称之为调整后的 R^2，因为 R^2 表示了自变量对因变量的解释程度，随着自变量的增加，模型的 R^2 总是会增大，因此需要根据自变量的个数对 R^2 进行相应的调整。调整后的 R^2 包含了自变量个数对模型的影响，从而可以帮助我们更好的选择模型。

回归问题性能评估



```
sklearn.metrics.mean_absolute_error(y_true, y_pred, sample_weight=None, multioutput='uniform_average')
sklearn.metrics.mean_squared_error(y_true, y_pred, sample_weight=None, multioutput='uniform_average', squared=True
)
```

```
from sklearn.metrics import \
  mean_absolute_error, mean_squared_error

y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
mean_absolute_error(y_true, y_pred)
```

```
## 0.5
```

```
y_true = [[0.5, 1], [-1, 1], [7, -6]]
y_pred = [[0, 2], [-1, 2], [8, -5]]
mean_absolute_error(y_true, y_pred)
```

```
## 0.75
```

```
mean_squared_error(y_true, y_pred, squared=False)
```

```
## 0.8227486121839513
```

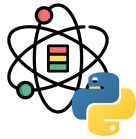
```
mean_squared_error(
  y_true, y_pred, multioutput='raw_values')
```

```
## array([0.41666667, 1.        ])
```

```
mean_squared_error(
  y_true, y_pred, multioutput=[0.3, 0.7])
```

```
## 0.825
```

回归问题性能评估



```
sklearn.metrics.r2_score(y_true, y_pred, sample_weight=None, multioutput='uniform_average')
```

```
from sklearn.metrics import r2_score

y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
r2_score(y_true, y_pred)
```

```
## 0.9486081370449679
```

```
y_true = [[0.5, 1], [-1, 1], [7, -6]]
y_pred = [[0, 2], [-1, 2], [8, -5]]
r2_score(
  y_true, y_pred, multioutput='variance_weighted')
```

```
## 0.9382566585956417
```

```
y_true, y_pred = [1, 2, 3], [1, 2, 3]
r2_score(y_true, y_pred)
```

```
## 1.0
```

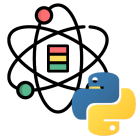
```
y_true, y_pred = [1, 2, 3], [2, 2, 2]
r2_score(y_true, y_pred)
```

```
## 0.0
```

```
y_true, y_pred = [1, 2, 3], [3, 2, 1]
r2_score(y_true, y_pred)
```

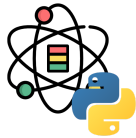
```
## -3.0
```

回归问题性能评估



方法	描述
metrics.explained_variance_score	Explained variance regression score function
metrics.max_error	max_error metric calculates the maximum residual error.
metrics.mean_absolute_error	Mean absolute error regression loss
metrics.mean_squared_error	Mean squared error regression loss
metrics.mean_squared_log_error	Mean squared logarithmic error regression loss
metrics.median_absolute_error	Median absolute error regression loss
metrics.r2_score	R^2 (coefficient of determination) regression score function.
metrics.mean_poisson_deviance	Mean Poisson deviance regression loss.
metrics.mean_gamma_deviance	Mean Gamma deviance regression loss.
metrics.mean_tweedie_deviance	Mean Tweedie deviance regression loss.

分类问题性能评估



分类问题可以划分为两类：二分类问题和多分类问题，两种不同的分类问题的性能评估方法略有不同。

误差和精度

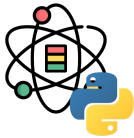
误差和精度是性能评估的两个最基本的指标。这两个指标具有很好的普适性，同时适用于二分类和多分类问题。类似于值预测问题中的各种误差，误差是指分类错误的样本数占样本总数的比例，相反的精度是指分类正确的样本数占样本总数的比例，误差的精度的定义如下：

$$err = \frac{1}{n}\sum_{i=1}^{n}\text{sign}\left(f\left(x_i\right) \neq y_i\right) \tag{8}$$

$$acc = \frac{1}{n}\sum_{i=1}^{n}\text{sign}\left(f\left(x_i\right) = y_i\right) \tag{9}$$

其中， $f\left(x\right)$ 表示模型的预测值， sign 函数当其内部条件满足是为 1 不满足时为 0。

分类问题性能评估

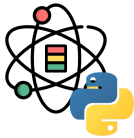


准确率，召回率和 F Score

准确率，召回率和 F Score 是评价二分类问题的重要评价指标。二分类问题是日常生活中一种常见的分类问题，例如预测明天是否会下雨，一个贷款客户在未来是否会发生违约等等。对于二分类分为，其目标变量分为两种，我们将其标记为正样本（1）和负样本（0）。样本通过分类器的预测会出现 4 种不同情况，分别是：真正例（True Positive），即将正样本预测为了正样本；假正例（False Positive），即将负样本预测为了正样本；真反例（True Negative），即将负样本预测为了负样本；假反例（False Negative），即将正样本预测为了负样本。对于 4 种不同的情况，我们可以利用一个混淆矩阵来表示分类器的分类结果：

真实情况 \ 预测结果	正样本	负样本
正样本	真正例（TP）	假反例（FN）
负样本	假正例（FP）	真反例（TN）

分类问题性能评估



根据 4 种不同的预测结果，可以定义准确率（Precision）和召回率（Recall）如下：

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{10}$$

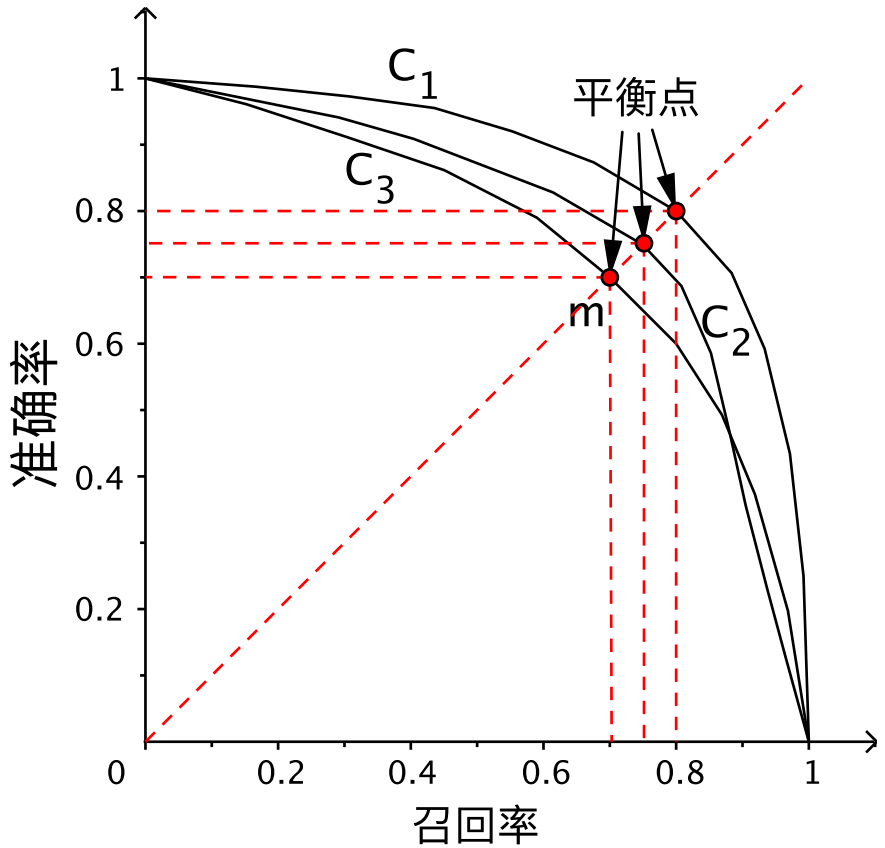
准确率和召回率是两个互相矛盾的评价指标，一个值越大往往另一个值就越小。不难想象两个极端情况，我们一个都不预测为正样本，因为准确率的分子和分母均为 0，也可以说我们的准确率为 100%，但此时的召回率却为 0；或是我们将所有样本都预测为正样本，则召回率为 100%，但此时准确率却很低，仅为正样本占样本总数的比例。

对分类问题，我们往往会构建一个概率模型，即对样本产生一个数值型的概率预测，同时我们会设置一个阈值 θ，如果预测值大于阈值，则属于正类，如果预测值小于阈值，则属于负类。因此，我们将所有的样本按照属于属于正样本的概率从大到小进行排序，逐步改变阈值 θ，则可以绘制出一张准确率和召回率的关系曲线，简称“P-R 曲线”。

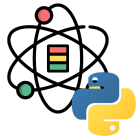
分类问题性能评估

从图中可以看出曲线 C_1 完全“包住”了曲线 C_2 和 C_3，此时我们可以断言 C_1 对应的分类器的性能要优于 C_2 和 C_3 对应的分类器。但对于 C_2 和 C_3 对应的分类器而言，由于两条曲线有交叉，因此仅通过曲线我们很难判定孰优孰劣。

同种我们还标注了 3 个点，当准确率等于召回率时，我们称这个点为“平衡点”(Break-Even Point，BEP)，因此如果用平衡点来衡量 C_2 $(BEP = 0.75)$ 和 C_3 $(BEP = 0.7)$ 对应的分类器，则 C_2 对应的分类器性能更优。



分类问题性能评估



F Score

相比于准确率和召回率，F Score 综合考虑了两个评价指标：

$$\frac{1}{F_\beta} = \frac{1}{1+\beta^2}\left(\frac{1}{P} + \frac{\beta^2}{R}\right) \tag{11}$$

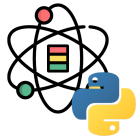
$$F_\beta = \frac{\left(1+\beta^2\right)\cdot P \cdot R}{\beta^2 \cdot P + R} \tag{12}$$

其中， β 用于控制准确率和召回率的相对重要程度。

- 若 $\beta = 1$ 则为标准的 F1
- 若 $0 < \beta < 1$ 时，则准确率更重要
- 若 $\beta > 1$ 时，则召回率更重要。

因此，用户可以根据实际的业务需求决定 β 值，从而控制准确率和召回率的相对重要程度。

分类问题性能评估

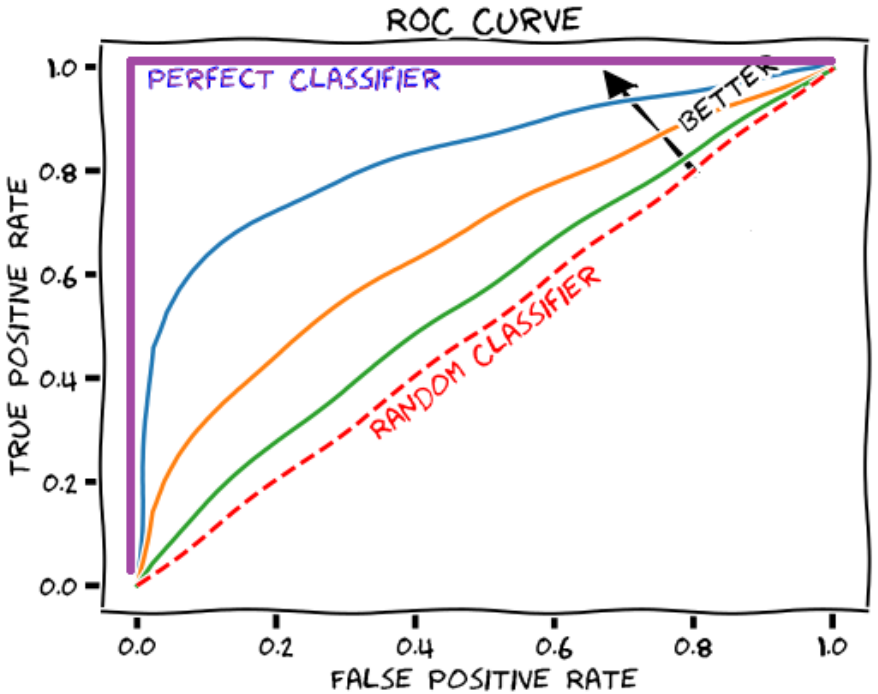


ROC 和 AUC

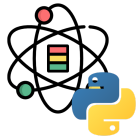
ROC（Receiver Operation Characteristic，接收者操作特征）曲线，首先是由二战中的电子工程师和雷达工程师发明的，用来侦测战场上的敌军载具，后来逐渐被用于医学、无线电、生物学、犯罪心理学以及数据挖掘和机器学习领域。类似 P-R 曲线，但 ROC 曲线的横轴为“假正例率”（False Positive Rate， FPR），纵轴为“真正例率”（True Positive Rate，TPR）：

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{13}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{TN} + \mathrm{FP}} \tag{14}$$



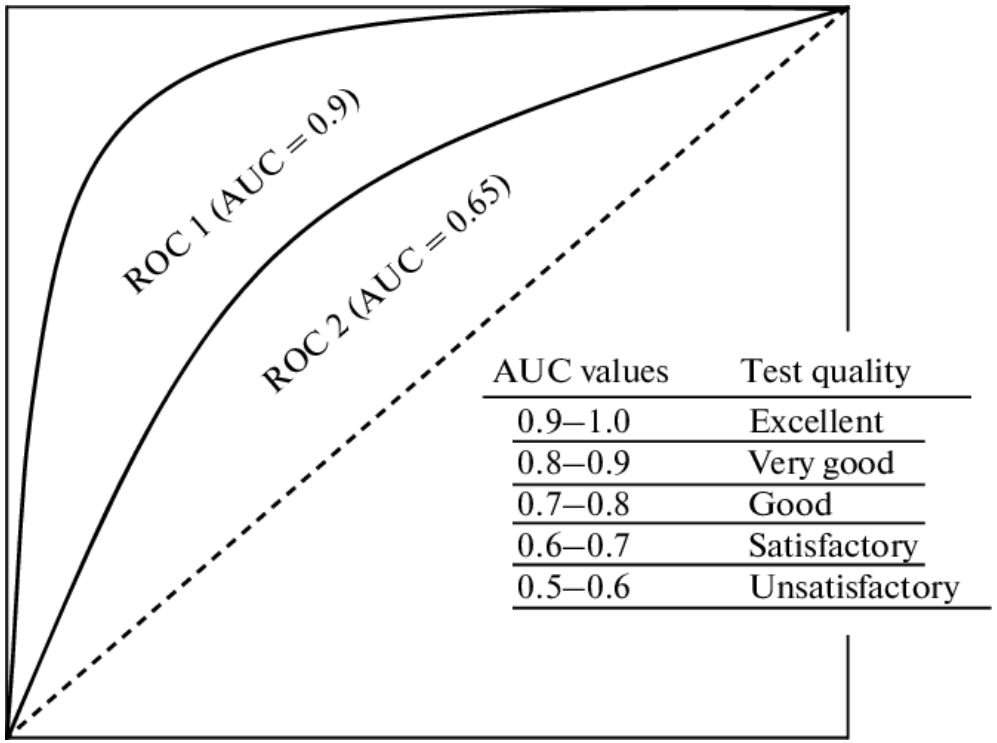
分类问题性能评估



ROC 和 AUC

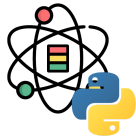
在进行学习器的比较式，与 P-R 图相似，若一个学习器的 ROC 曲线被另一个学习器的曲线完全“包住”，则可断言后者的性能优于前者，若两个学习器的 ROC 曲线发生交叉，则难以一般性地断言两者孰优孰劣。此时如果一定要进行比较，则较为合理的依据是比较 ROC 曲线下的面积，及 AUC（Area Under ROC Curve）。

$$\mathrm{AUC} = \frac{1}{2}\sum_{i=1}^{n-1}(x_{i+1}-x_i)(y_i+y_{i+1}) \qquad (15)$$



AUC values	Test quality
0.9–1.0	Excellent
0.8–0.9	Very good
0.7–0.8	Good
0.6–0.7	Satisfactory
0.5–0.6	Unsatisfactory

分类问题性能评估

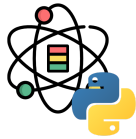


多分类 Log Loss

$$\text{LogLoss} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{i,j}\log\left(p_{i,j}\right) \tag{16}$$

其中， n 为数据集个数； m 为标签类个数； 如果样本 i 的分类为 j，则 $y_{i,j}$ 为 1，否则为 0； $p_{i,j}$ 为样本 i 为类型 j 的概率。

分类问题性能评估



```
sklearn.metrics.precision_score(
  y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None, zero_division='warn')
```

```
from sklearn.metrics import precision_score

y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]
precision_score(y_true, y_pred, average='macro')
```

```
## 0.2222222222222222
```

```
precision_score(y_true, y_pred, average='micro')
```

```
## 0.3333333333333333
```

```
precision_score(y_true, y_pred, average='weighted')
```

```
## 0.2222222222222222
```

```
precision_score(y_true, y_pred, average=None)
```

```
## array([0.66666667, 0.        , 0.        ])
```

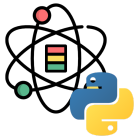
```
y_pred = [0, 0, 0, 0, 0, 0]
precision_score(y_true, y_pred, average=None)
```

```
## array([0.33333333, 0.        , 0.        ])
```

```
precision_score(
  y_true, y_pred, average=None, zero_division=1)
```

```
## array([0.33333333, 1.        , 1.        ])
```

分类问题性能评估



```
sklearn.metrics.recall_score(
  y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None, zero_division='warn')
```

```
from sklearn.metrics import recall_score

y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]
recall_score(y_true, y_pred, average='macro')
```

```
## 0.3333333333333333
```

```
recall_score(y_true, y_pred, average='micro')
```

```
## 0.3333333333333333
```

```
recall_score(y_true, y_pred, average='weighted')
```

```
## 0.3333333333333333
```

```
recall_score(y_true, y_pred, average=None)
```

```
## array([1., 0., 0.])
```

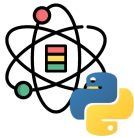
```
y_pred = [0, 0, 0, 0, 0, 0]
recall_score(y_true, y_pred, average=None)
```

```
## array([1., 0., 0.])
```

```
recall_score(
  y_true, y_pred, average=None, zero_division=1)
```

```
## array([1., 0., 0.])
```

分类问题性能评估



```
sklearn.metrics.f1_score(
  y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None, zero_division='warn')
sklearn.metrics.fbeta_score(
  y_true, y_pred, beta, labels=None, pos_label=1, average='binary', sample_weight=None, zero_division='warn')
```

```
from sklearn.metrics import f1_score, fbeta_score

y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]
f1_score(y_true, y_pred, average='macro')
```

```
## 0.26666666666666666
```

```
f1_score(y_true, y_pred, average='micro')
```

```
## 0.3333333333333333
```

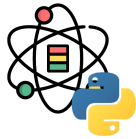
```
fbeta_score(
  y_true, y_pred, average='weighted', beta=0.5)
```

```
## 0.2380952380952381
```

```
fbeta_score(
  y_true, y_pred, average=None, beta=0.5)
```

```
## array([0.71428571, 0.        , 0.        ])
```

分类问题性能评估



```
sklearn.metrics.confusion_matrix(y_true, y_pred, labels=None, sample_weight=None, normalize=None)
```

```
from sklearn.metrics import confusion_matrix

y_true = [2, 0, 2, 2, 0, 1]
y_pred = [0, 0, 2, 2, 0, 2]
confusion_matrix(y_true, y_pred)
```

```
## array([[2, 0, 0],
##        [0, 0, 1],
##        [1, 0, 2]])
```

```
y_true = ["cat", "ant", "cat", "cat", "ant", "bird"]
y_pred = ["ant", "ant", "cat", "cat", "ant", "cat"]
```

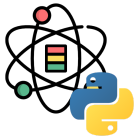
```
confusion_matrix(y_true, y_pred,
                 labels=["ant", "bird", "cat"])
```

```
## array([[2, 0, 0],
##        [0, 0, 1],
##        [1, 0, 2]])
```

```
tn, fp, fn, tp = confusion_matrix(
  [0, 1, 0, 1], [1, 1, 1, 0]).ravel()
(tn, fp, fn, tp)
```

```
## (0, 2, 1, 1)
```

分类问题性能评估



```
sklearn.metrics.roc_auc_score(
  y_true, y_score, average='macro', sample_weight=None, max_fpr=None, multi_class='raise', labels=None)
sklearn.metrics.roc_curve(
  y_true, y_score, pos_label=None, sample_weight=None, drop_intermediate=True)
sklearn.metrics.auc(x, y)
```

```
from sklearn.metrics import roc_auc_score, roc_curve
, auc

y_true = np.array([0, 0, 1, 1])
y_scores = np.array([0.1, 0.4, 0.35, 0.8])
roc_auc_score(y_true, y_scores)
```

```
## 0.75
```

```
fpr, tpr, thresholds = roc_curve(
  y_true, y_scores, pos_label=1)
```

```
fpr
```

```
## array([0. , 0. , 0.5, 0.5, 1. ])
```

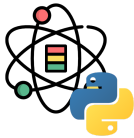
```
tpr
```

```
## array([0. , 0.5, 0.5, 1. , 1. ])
```

```
auc(fpr, tpr)
```

```
## 0.75
```

分类问题性能评估



```
sklearn.metrics.classification_report(
  y_true, y_pred, labels=None, target_names=None, sample_weight=None, digits=2, output_dict=False,
  zero_division='warn')
```

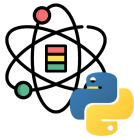
```
from sklearn.metrics import classification_report

y_true = [0, 1, 2, 2, 2]
y_pred = [0, 0, 2, 2, 1]
target_names = ['class 0', 'class 1', 'class 2']
```

```
classification_report(
  y_true, y_pred, target_names=target_names)
```

```
##               precision  recall  f1-score  support
##
##      class 0       0.50    1.00      0.67        1
##      class 1       0.00    0.00      0.00        1
##      class 2       1.00    0.67      0.80        3
##
##     accuracy                         0.60        5
##    macro avg       0.50    0.56      0.49        5
## weighted avg       0.70    0.60      0.61        5
```

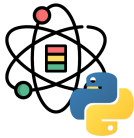
分类问题性能评估



方法	描述
metrics.accuracy_score	Accuracy classification score.
metrics.auc	Compute Area Under the Curve (AUC) using the trapezoidal rule
metrics.average_precision_score	Compute average precision (AP) from prediction scores
metrics.balanced_accuracy_score	Compute the balanced accuracy
metrics.brier_score_loss	Compute the Brier score.
metrics.classification_report	Build a text report showing the main classification metrics
metrics.cohen_kappa_score	Cohen's kappa: a statistic that measures inter-annotator agreement.
metrics.confusion_matrix	Compute confusion matrix to evaluate the accuracy of a classification.

(接下表)

分类问题性能评估

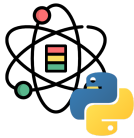


(接上表)

方法	描述
metrics.dcg_score	Compute Discounted Cumulative Gain.
metrics.f1_score	Compute the F1 score, also known as balanced F-score or F-measure
metrics.fbeta_score	Compute the F-beta score
metrics.hamming_loss	Compute the average Hamming loss.
metrics.hinge_loss	Average hinge loss (non-regularized)
metrics.jaccard_score	Jaccard similarity coefficient score
metrics.log_loss	Log loss, aka logistic loss or cross-entropy loss.
metrics.matthews_corrcoef	Compute the Matthews correlation coefficient (MCC)

(接下表)

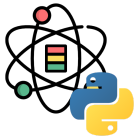
分类问题性能评估



(接上表)

方法	描述
metrics.multilabel_confusion_matrix	Compute a confusion matrix for each class or sample
metrics.ndcg_score	Compute Normalized Discounted Cumulative Gain.
metrics.precision_recall_curve	Compute precision-recall pairs for different probability thresholds
metrics.precision_recall_fscore_support	Compute precision, recall, F-measure and support for each class
metrics.precision_score	Compute the precision
metrics.recall_score	Compute the recall
metrics.roc_auc_score	Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
metrics.roc_curve	Compute Receiver operating characteristic (ROC)

聚类问题性能评估



聚类性能度量大致有两类：

- 一类是将聚类结果与某个“参考模型”（reference model）进行比较，称为“外部指标”（external index）。
- 另一类是直接考查聚类结果而不利用任何参考模型，称为“内部指标”（internal index）。

对数据集 $D = \{x_1, x_2, \ldots, x_n\}$，假定通过聚类给出的簇划分为 $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$，参考模型给出的簇划分为 $\mathcal{C}^* = \{C_1^*, C_2^*, \ldots, C_k^*\}$。相应地，令 λ 与 λ^* 分别表示与 C 和 C^* 对应的簇标记向量。

我们将样本两两配对考虑，定义：

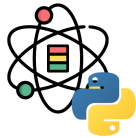
$$a = |SS|, \quad SS = \{(x_i, x_j) | \lambda_i = \lambda_j, \lambda_i^* = \lambda_j^*, i < j\} \tag{17}$$

$$b = |SD|, \quad SD = \{(x_i, x_j) | \lambda_i = \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j\} \tag{18}$$

$$c = |DS|, \quad DS = \{(x_i, x_j) | \lambda_i \neq \lambda_j, \lambda_i^* = \lambda_j^*, i < j\} \tag{19}$$

$$d = |DD|, \quad DD = \{(x_i, x_j) | \lambda_i \neq \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j\} \tag{20}$$

聚类问题性能评估



外部指标

- Jaccard 系数（Jaccard Coefficient，JC）

$$\mathrm{JC} = \frac{a}{a+b+c} \tag{21}$$

- FM 指数（Fowlkes and Mallows Index，FMI）

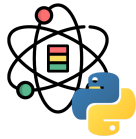
$$\mathrm{FMI} = \sqrt{\frac{a}{a+b} \cdot \frac{a}{a+c}} \tag{22}$$

- Rand 指数（Rand Index，RI）

$$\mathrm{RI} = \frac{2\left(a+d\right)}{n\left(n-1\right)} \tag{23}$$

上述三值均在 $[0,1]$ 区间，值越大越好。

聚类问题性能评估



内部指标：考虑聚类结果的簇划分 $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$，定义：

- 簇 C 内样本间的平均距离：

$$\operatorname{avg}(C) = \frac{2}{|C|(|C|-1)} \sum_{1 \le i \le j \le |C|} \operatorname{dist}(x_i, x_j) \tag{24}$$

- 簇 C 内样本间最远距离：

$$\operatorname{diam}(C) = \max_{1 \le i \le j \le |C|} \operatorname{dist}(x_i, x_j) \tag{25}$$

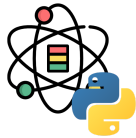
- 簇 C_i 与 C_j 最近样本间的距离

$$d_{\min}(C_i, C_j) = \min_{x_i \in C_i, x_j \in C_j} \operatorname{dist}(x_i, x_j) \tag{26}$$

- 簇 C_i 与 C_j 中心点间的距离

$$d_{\text{cen}}(C_i, C_j) = \operatorname{dist}(\mu_i, \mu_j) \tag{27}$$

聚类问题性能评估



- DB 指数（Davies-Bouldin Index，DBI）类比簇间相似度

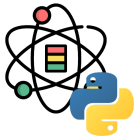
$$\mathrm{DBI} = \frac{1}{k}\sum_{i=1}^{k}\max_{j\neq i}\left(\frac{\mathrm{avg}\left(C_i\right)+\mathrm{avg}\left(C_j\right)}{d_{\mathrm{cen}}\left(\mu_i,\mu_j\right)}\right) \tag{28}$$

- Dunn 指数（Dunn Index，DI）类比簇内相似度

$$\mathrm{DI} = \min_{1\leq i\leq k}\left\{\min_{j\neq i}\left(\frac{d_{\min}\left(C_i,C_j\right)}{\max_{1\leq l\leq k}\mathrm{diam}\left(C_l\right)}\right)\right\} \tag{29}$$

  DBI 的值越小越好，DI 相反，越大越好。

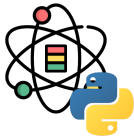
聚类问题性能评估



方法	描述
metrics.adjusted_mutual_info_score	Adjusted Mutual Information between two clusterings.
metrics.adjusted_rand_score	Rand index adjusted for chance.
metrics.calinski_harabasz_score	Compute the Calinski and Harabasz score.
metrics.davies_bouldin_score	Computes the Davies-Bouldin score.
metrics.completeness_score	Completeness metric of a cluster labeling given a ground truth.
metrics.cluster.contingency_matrix	Build a contingency matrix describing the relationship between labels.
metrics.fowlkes_mallows_score	Measure the similarity of two clusterings of a set of points.

(接下表)

聚类问题性能评估

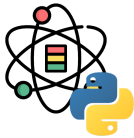


(接上表)

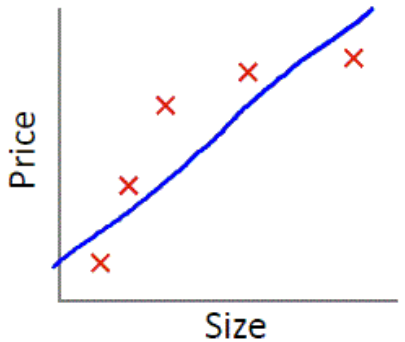
方法	描述
metrics.homogeneity_completeness_v_measure	Compute the homogeneity and completeness and V-Measure scores at once.
metrics.homogeneity_score	Homogeneity metric of a cluster labeling given a ground truth.
metrics.mutual_info_score	Mutual Information between two clusterings.
metrics.normalized_mutual_info_score	Normalized Mutual Information between two clusterings.
metrics.silhouette_score	Compute the mean Silhouette Coefficient of all samples.
metrics.silhouette_samples	Compute the Silhouette Coefficient for each sample.
metrics.v_measure_score	V-measure cluster labeling given a ground truth.

模型生成和选择

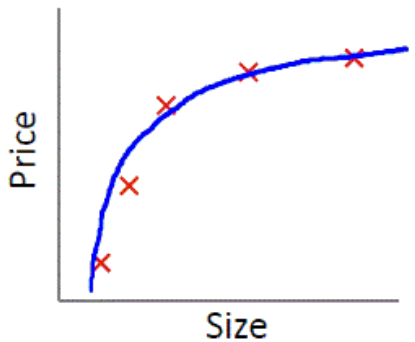
过拟合问题



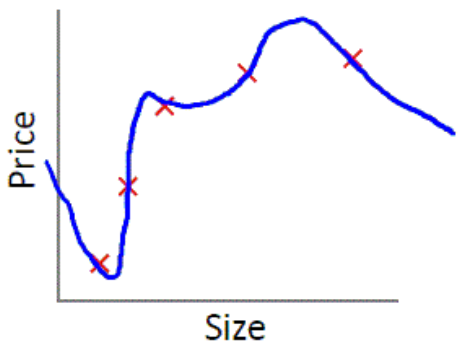
在构建机器学习模型的时候，我们希望新样本也能够表现的很好。为了达到这个目的，应该从训练样本中尽可能学出适用于所有潜在样本的“普遍规律”，这样才能在遇到新样本时做出正确的判别。 然而，当学习器把训练样本学得“太好”了的时候，很可能已经把训练样本自身的一些特点当作了所有潜在样本都会具有的普适性质，这样就会导致泛化性能下降。这种现象在机器学习中称为**“过拟合”（overfitting)**。与“过拟合”相对的是**“欠拟合”（underfitting)**，这是指对训练样本的一般性质尚未学好。



$\theta_0 + \theta_1 x$

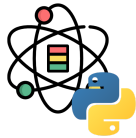


$\theta_0 + \theta_1 x + \theta_2 x^2$



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

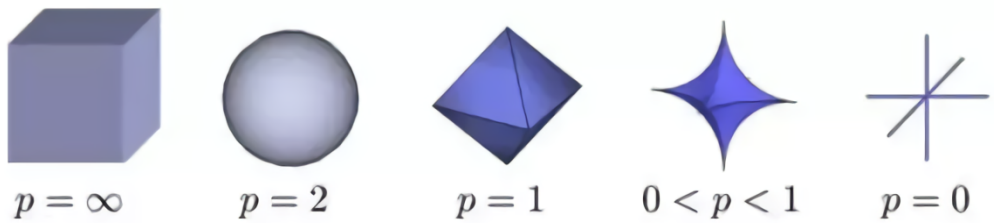
正则化



一般来说，监督学习可以看做最小化下面的目标函数 和 LP 范数定义如下：

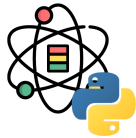
$$w^* = arg\,min \sum_i L\left(y_i, f\left(x_i, w\right)\right) + \lambda \Omega\left(w\right) \qquad L_p = \|w\|_p = \left(\sum_{i=1} n|w_i|^p\right)^{\frac{1}{p}} \tag{30}$$

其中，$\sum_i L\left(y_i, f\left(x_i, w\right)\right)$ 为损失函数，$\Omega\left(w\right)$ 为正则项，常见的正则化包括 L1 正则化和 L2 正则化。根据 P 值的变化，范数也随之变化，一个三维的 LP 范数变化示意图如下：



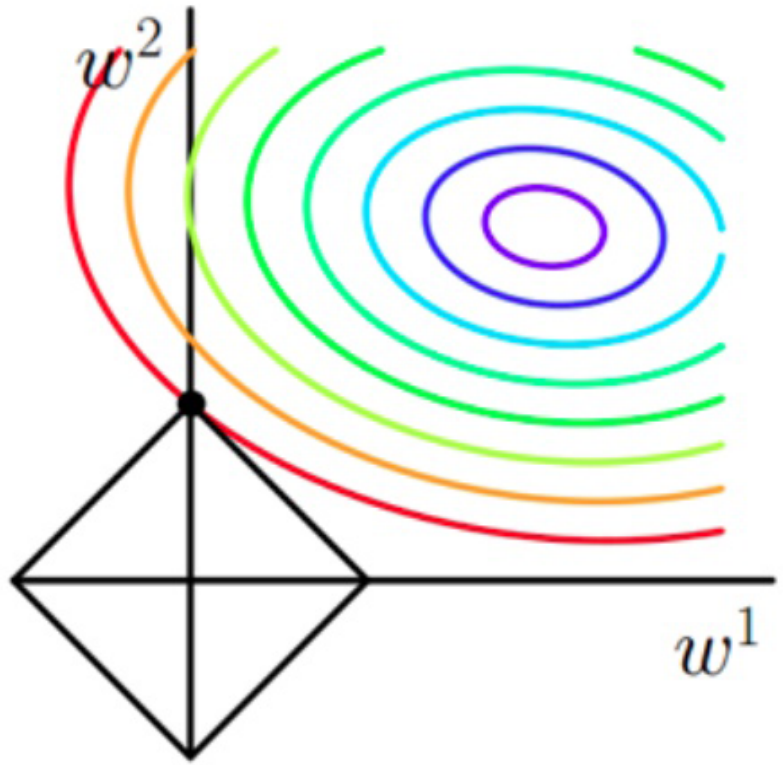
L2 正则化是通过惩罚权重大的个体来降低模型复杂度的一种方法。L1 正则化则是通过会产生大量部分特征值权重为 0 的系数特征向量来降低模型复杂度。如果高维数据集的样本包含许多不相关的特征，特别是在有更多不相关维度的情况下，稀疏性很有实用价值。

L1 正则化

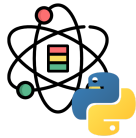


L1 惩罚的是绝对权重之和，我们可以用菱形区域来表示，如右图所示。

从右图中可以看出，代价函数的轮廓与 L1 的菱形在 $w_1 = 0$ 处相交。由于 L1 正则化系统的轮廓是尖锐的，所以更可能的是代价函数的椭圆与 L1 另行边界的交点位于轴上，从而促进了稀疏性。

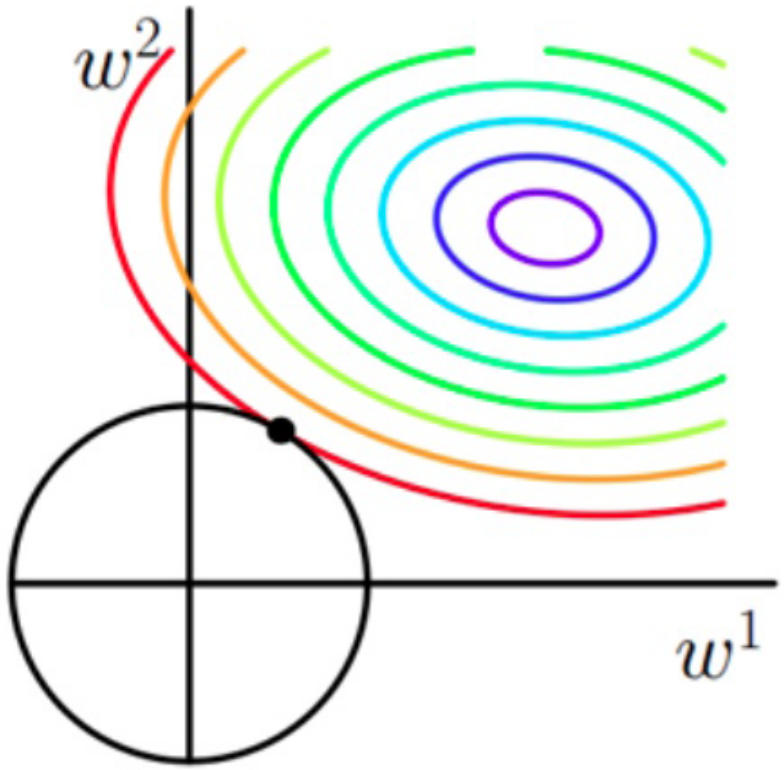


L2 正则化

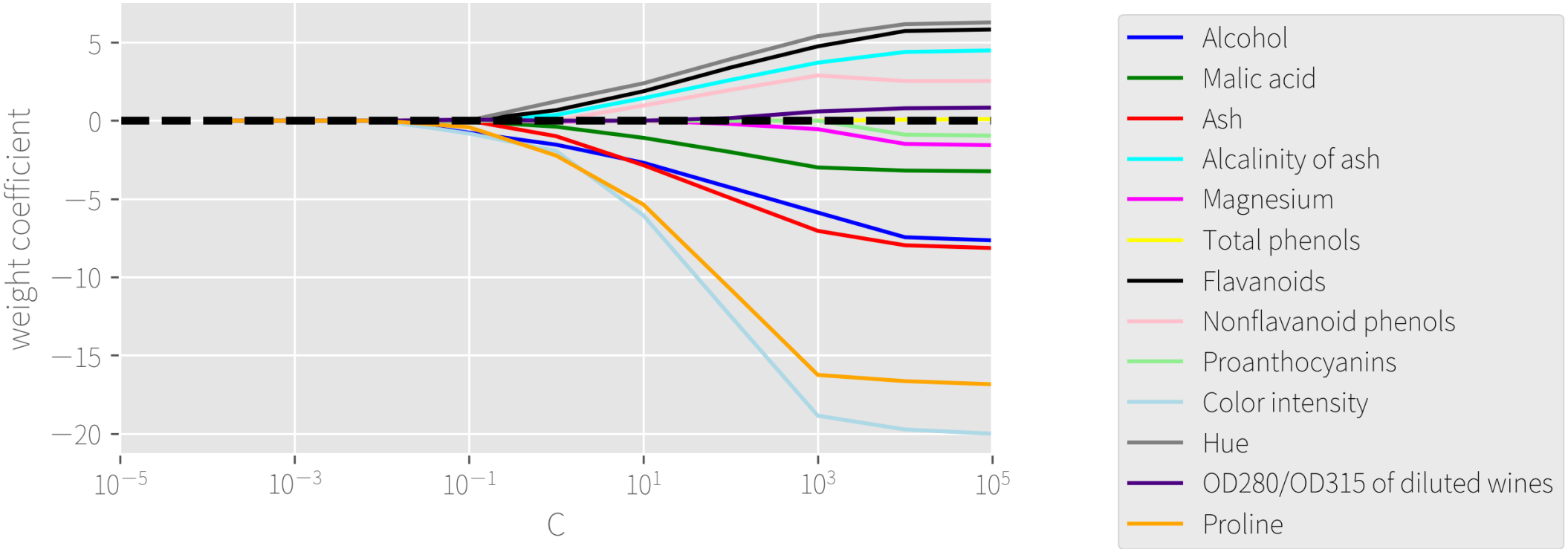


L2 惩罚的是权重较大的参数，我们可以用圆形区域来表示，如右图所示。

在惩罚的约束下，尽可能确定 L2 圆形与无惩罚代价函数轮廓的交叉点。 λ 正则化参数越大，惩罚成本的增速越快，导致 L2 圆形变窄。例如：正则参数趋于无穷大，则权重系数将快速变为 0，即 L2 圆形的中心。总之，目标是最小化无惩罚成本与惩罚项的总和，可以理解为增加偏置和偏好简单模型，以最小化模型在缺乏足够训练数据拟合情况下的方差。

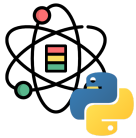


正则化



红酒分类正则化（C 是正则化参数 λ 的逆，越小表示正则化强度越大）

评估方法

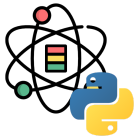


通常，我们可以通过实验测试来对学习器的泛化误差进行评估并进而做出选择。为此，需使用一个测试集（testing set）来测试学习器对新样本的判别能力，然后以测试集上的测试误差（testing error）作为泛化误差的近似。

通常，我们假设测试样本也是从样本真实分布中独立同分布采样而来。但需要注意，测试集应该尽可能与训练集互斥，即测试样本尽量不在训练集中出现、未在训练过程中使用过。

若测试样本被用作训练了，则得到的将是过于“乐观”的估计结果。

留出法

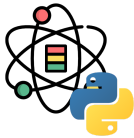


留出法（hold-out）直接将数据集 D 划分为两个互斥的集合，其中一个集合作为训练集 S，另一个作为测试集 T，即 $D = S \cup T$，$S \cap T = \varnothing$。在 S 上训练出模型后，用 T 来评估其测试误差，作为对泛化误差的估计。

以二分类任务为例，假定 D 包含 1000个样本，将其划分为包含 700 样本的 S，和包含 300 样本的 T。用 S 进行训练后，如果模型在 T 上有 90 个样本分类错误，那么其准确率为 $(210/300)\times 100\% = 70\%$，相应的，错误率为 $(90/300)\times 100\% = 30\%$。

需要注意的是，训练/测试集的划分要尽可能保持数据分布的一致性，避免因数据划分过程引入额外的偏差而对最终结果产生影响，例如在分类任务中至少要保持样本的类别比例相似。如果从采样（sampling）的角度来看待数据集的划分过程，则保留类别比例的采样方式通常称为分层采样（stratified sampling）。另一个问题是，即便给定训练/测试集的样本比例后，对数据集的不同划分将导致不同的训练/测试集，相应的，模型评估的结果也会有差别。因此，单次使用留出法得到的估计结果往往不够稳定可靠，在使用留出法时，一般要采用若干次随机划分、重复进行实验评估后取平均值作为留出法的评估结果。常见做法是将大约 2/3 ~ 4/5 的样本用于训练，剩余样本用于测试。

交叉验证

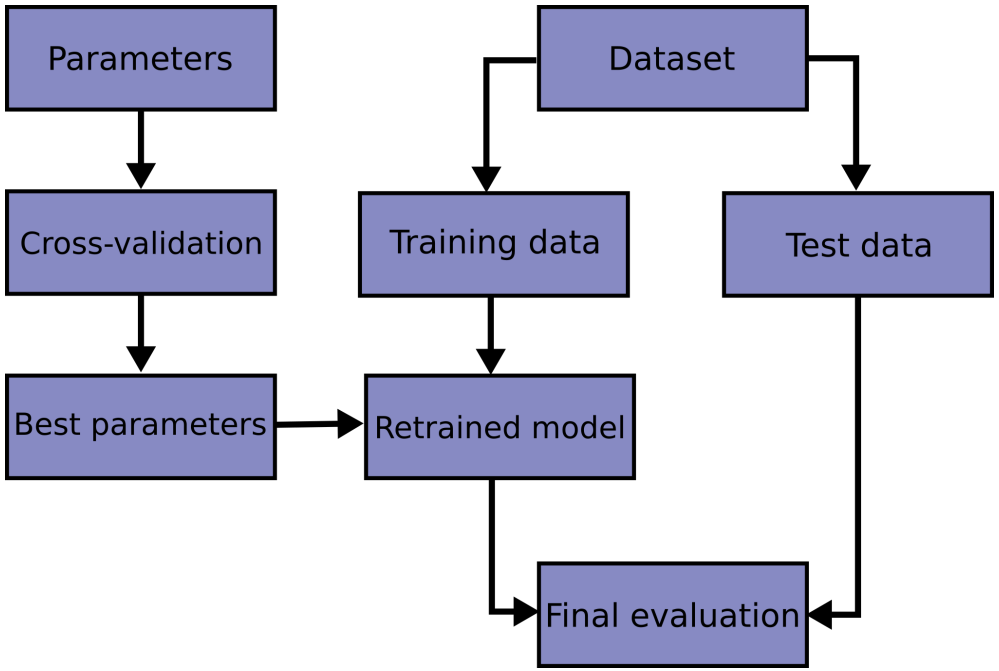


交叉验证法（cross validation）先将数据集 D 划分为 k 个大小相似的互斥子集，即 $D = D_1 \cup D_2 \cup \ldots \cup D_k$，$D_i \cap D_j = \varnothing (i \neq j)$。每个子集 D_i 都尽可能保持数据分布的一致性，即从 D 中通过分层采样得到。然后，每次用 $k-1$ 个子集的并集作为训练集，余下的那个子集作为测试集；这样就可获得 k 组训练/测试集，从而可进行 k 次训练和测试，最终返回的是这 k 个测试结果的均值。

与留出法相似，把数据集 D 划分为 k 个子集同样存在多种划分方式。为减小因样本划分不同而引入的差别， k 折交叉验证通常要随机使用不同的划分重复 p 次，最终的评估结果是这 p 次 k 折交叉验证结果的均值，例如常见的有“10 次 10 折交叉验证”。

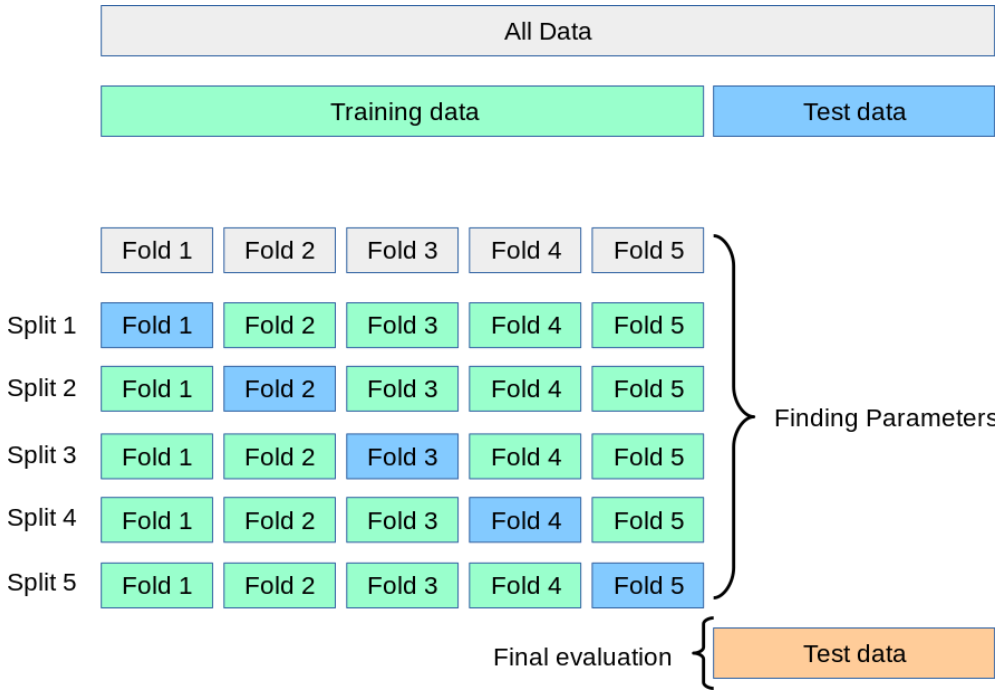
假定数据集 D 包含 m 个样本，若令 $k = m$，则得到了交叉验证法的一个特例：留一法（Leave-One-Out）。显然，留一法不受随机样本划分方式的影响，因为只有一种划分方式。而且留一法的训练集只比数据集少了一个样本，因而被实际评估的模型与期望评估的用 D 训练出的模型很相似，因而留一法的评估结果往往被认为比较准确。但是当数据集比较大时，训练 m 个模型的计算开销可能是难以忍受的。

交叉验证



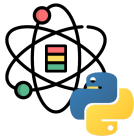
模型训练中交叉检验流程

[1] 图片来源：https://scikit-learn.org/stable/modules/cross_validation.html



K 折交叉检验

交叉验证



```
sklearn.model_selection.cross_validate(
  estimator, X, y=None, groups=None, scoring=None, cv=None, n_jobs=None, verbose=0, fit_params=None,
  pre_dispatch='2*n_jobs', return_train_score=False, return_estimator=False, error_score=nan)
```

```
from sklearn import datasets
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit

X, y = datasets.load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
  X, y, test_size=0.4, random_state=0)
clf = svm.SVC(kernel='linear', C=1).fit(
  X_train, y_train)
clf.score(X_test, y_test)
```

```
## 0.9666666666666667
```

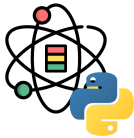
```
clf = svm.SVC(kernel='linear', C=1)
cross_val_score(clf, X, y, cv=5)
```

```
## array([0.96666667, 1.        , 0.96666667, 0.966666
67, 1.        ])
```

```
cv = ShuffleSplit(n_splits=5, test_size=0.3, random_
state=0)
cross_val_score(clf, X, y, cv=cv)
```

```
## array([0.97777778, 0.97777778, 1.        , 0.955555
56, 1.        ])
```

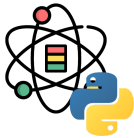
自助法



留出法和交叉验证法由于保留了一部分样本用于测试，因而实际评估的模型所使用的训练集比 D 小，这必然会引入一些因训练样本规模不同而导致的估计偏差。留一法受训练样本规模变化的影响较小，但计算复杂度又太高了。

自助法（bootstrapping）是一个比较好的解决方案，它直接以自助采样法（boostrap sampling）为基础。给定包含 m 个样本的数据集 D，我们对它进行采样产生数据集 $D\prime$：每次随机从 D 中挑选一个样本，将其拷贝放入 $D\prime$，然后再将该样本放回初始数据集 D 中，使得该样本在下次采样时仍可能被采到；这个过程重复执行 m 次之后，我们就得到了包含 m 个样本的数据集 $D\prime$，这就是自助采样的结果。

自助法



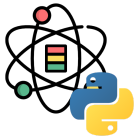
显然， D 中有一部分样本会在 $D\prime$ 中多次出现，而另一部分样本不出现。可以做一个简单的估计，样本在 m 次采样中始终不被采到的概率是 $(1-\frac{1}{m})^m$，取极限得到：

$$\lim_{m\to\infty}\left(1-\frac{1}{m}\right)^m \to \frac{1}{e} \approx 0.368 \tag{31}$$

即通过自助采样，初始数据集 D 中约有 36.8% 的样本未出现在采样数据集 $D\prime$ 中。于是我们可以把 $D\prime$ 用作训练集，把 $D-D\prime$ 用作测试集；这样，实际评估的模型与期望评估的模型都使用 m 个训练样本，而我们仍有数据总量的约 $1/3$ 的没在训练集中出现的样本用于测试。这样的测试结果，也称为包外估计（out-of-bag estimate）。

自助法在数据集较小、难以有效划分训练/测试集时很有用；然而，自助法产生的数据集改变了初始数据集的分布，这会引入估计偏差，因此，在初始数据量足够时，留出法和交叉验证更常用一些。

偏差和方差



对学习算法除了通过实验估计其泛化性能，我们往往还希望了解它“为什么”具有这样的性能。**偏差-方差分解（bias-variance decomposition）**是解释学习算法泛化性能的一种重要工具，它试图对学习算法的期望泛化错误率进行拆解。

算法在不同训练集上学得的结果很可能不同，即便这些训练集是来自同一个分布。对测试样本 $\boldsymbol{x}$，令 y_D 为 $\boldsymbol{x}$ 在数据集中的标记， y 为 $\boldsymbol{x}$ 的真实标记（有可能出现噪声使得 $y_D \neq y$）， $f(\boldsymbol{x};D)$ 为训练集 D 上学得模型 f 在 $\boldsymbol{x}$ 上的预测输出。以回归任务为例，学习算法的期望预测为：

$$\bar{f}(\boldsymbol{x}) = \mathbb{E}_D[f(\boldsymbol{x};D)] \tag{32}$$

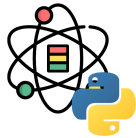
使用样本数相同的不同训练集产生的方差为：

$$var(\boldsymbol{x}) = \mathbb{E}_D\left[\left(f(\boldsymbol{x};D) - \bar{f}(\boldsymbol{x})\right)^2\right] \tag{33}$$

噪声为：

$$\varepsilon^2 = \mathbb{E}_D\left[(y_D - y)^2\right] \tag{34}$$

偏差和方差



泛化误差可以分解为偏差、方差与噪声之和：

$$E = bias^2(\boldsymbol{x}) + var(\boldsymbol{x}) + \varepsilon^2 \tag{35}$$

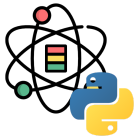
偏差 $bias^2(\boldsymbol{x}) = \left(\bar{f}(\boldsymbol{x}) - y\right)^2$ 度量了学习算法的期望预测与真实结果的偏离程度，即刻画了学习算法本身的拟合能力。

方差 $var(\boldsymbol{x}) = \mathbb{E}_D\left[\left(f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x})\right)^2\right]$ 度量了同样大小的训练集的变动所导致的学习性能的变化，即刻画了数据扰动所造成的影响。

噪声 $\varepsilon^2 = \mathbb{E}_D\left[(y_D - y)^2\right]$ 则表达了在当前任务上任何学习算法所能达到的期望泛化误差的下界，即刻画了学习问题本身的难度。

偏差-方差分解说明，泛化性能是由学习算法的能力、数据的充分性以及学习任务本身的难度所共同决定的。给定学习任务，为了取得好的泛化性能，需要使偏差较小，即能够充分拟合数据，且使方差较小，即使数据扰动产生的影响较小。

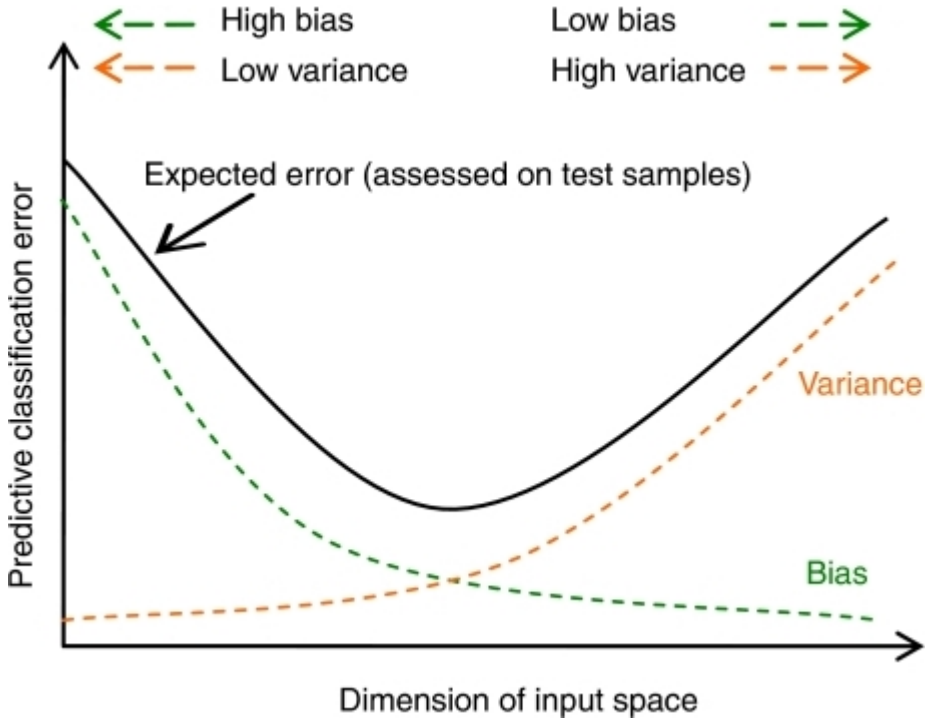
偏差和方差



一般来说，偏差与方差是有冲突的，这称为偏差-方差窘境（bias-variance dilemma）。

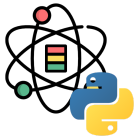
训练不足时，学习器拟合能力不够强，训练数据的扰动不足以使学习器产生显著变化，此时偏差主导泛化错误率。随着训练程度加深，学习器拟合能力逐渐增强，训练数据的扰动逐渐被学习器学到，方差逐渐主导泛化错误率。

在训练程度充足后，学习器的拟合能力已非常强，训练数据发生的轻微扰动都会导致学习器发生显著变化，若训练数据自身的、非全局的特性被学习器学到了，则将发生过拟合。



超参数优化

超参数优化

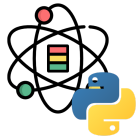


模型的**参数**和**超参数**二者有着本质上的区别：模型参数是模型内部的配置变量，可以用数据估计模型参数的值，例如：回归中的权重，决策树分类点的阈值等；模型超参数是模型外部的配置，必须手动设置参数的值，例如：随机森林树的个数，聚类方法里面类的个数，或者主题模型里面主题的个数等。

常用的超参数优化方法有：

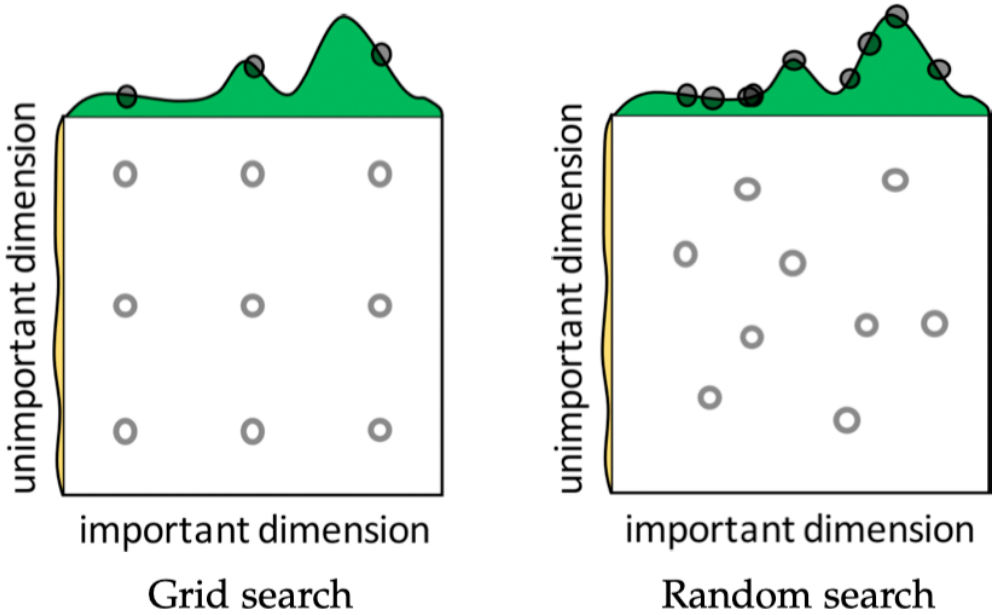
- **人工调参**：炼丹术
- **搜索算法**：网格搜索，随机搜索等
- **启发式算法**：遗传算法，粒子群算法等
- **贝叶斯优化**：高斯过程，TPE 等

搜索算法



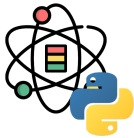
Grid Search 是一个暴力解法，通过所有需要测试的超参数，找出所有可能的超参数组合，最根据验证集的损失找出最好的一组超参数。这是一个非常消耗资源的方法。如果有 m 个超参数，每个超参数选最多 n 个可能值，那么该算法的时间复杂度是 $O\left(n^m\right)$。

Random Search 的使用方法 Grid Search 基本一致，区别在于 Random Search 会在超参数的组合空间内随机采样搜索，其搜索能力取决于设定的抽样次数，最重要的是收敛更快 [1]。



[1] Bergstra, James, and Yoshua Bengio. "Random search for hyper-parameter optimization." *Journal of machine learning research* 13.Feb (2012): 281-305.

搜索算法

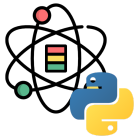


```
sklearn.model_selection.GridSearchCV(
  estimator, param_grid, scoring=None,
  n_jobs=None, refit=True, cv=None,
  verbose=0, pre_dispatch='2*n_jobs',
  error_score=nan,
  return_train_score=False)

sklearn.model_selection.RandomizedSearchCV(
  estimator, param_distributions,
  n_iter=10, scoring=None, n_jobs=None,
  refit=True, cv=None, verbose=0,
  pre_dispatch='2*n_jobs',
  random_state=None, error_score=nan,
  return_train_score=False)
```

参数	说明
estimator	估计器，需要提供 score 方法或传入 scoring
param_grid	参数字典或列表
param_distributions	参数分布字典或列表
scoring	用于评估预测性能的方法
cv	交叉检验的策略
refit	是否利用全部数据和最优参数重训练

搜索算法



```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()
parameters = {
  'C': [0, 1, 2, 3, 4],
  'penalty': ['l2', 'l1']
}

lr = LogisticRegression(solver='saga', tol=1e-2, max
_iter=200, random_state=0)
clf = GridSearchCV(lr, parameters).fit(
  iris.data, iris.target)
(clf.best_params_, clf.best_score_)
```

```
## ({'C': 1, 'penalty': 'l2'}, 0.9800000000000001)
```

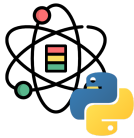
```
from scipy.stats import uniform
from sklearn.model_selection import RandomizedSearch
CV

distributions = {
  'C': uniform(loc=0, scale=4),
  'penalty': ['l2', 'l1']
}

lr = LogisticRegression(solver='saga', tol=1e-2, max
_iter=200, random_state=0)
clf = RandomizedSearchCV(lr, distributions, random_s
tate=0).fit(
  iris.data, iris.target)
(clf.best_params_, clf.best_score_)
```

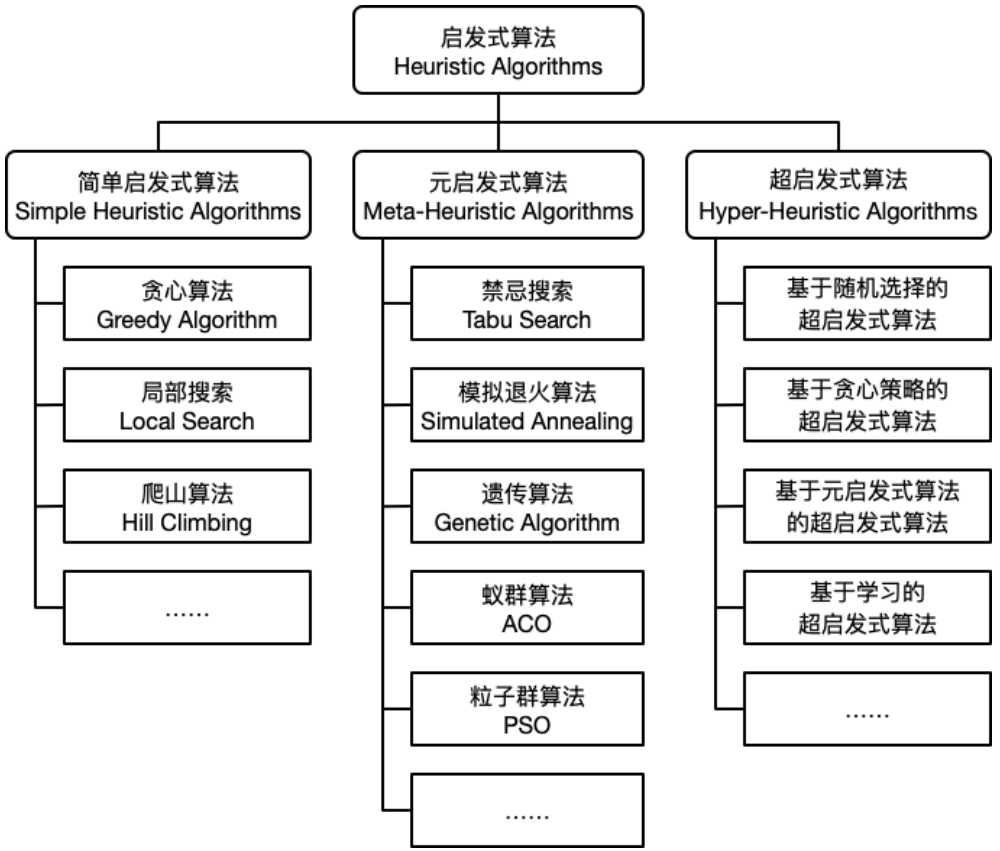
```
## ({'C': 2.195254015709299, 'penalty': 'l1'}, 0.98000
00000000001)
```

启发式算法



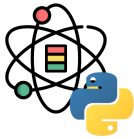
启发式算法（Heuristic Algorithms）是相对于最优算法提出的。一个问题的最优算法是指求得该问题每个实例的最优解。启发式算法可以这样定义[1]：一个基于直观或经验构造的算法，在可接受的花费（指计算时间、占用空间等）下给出待解决组合优化问题每一个实例的一个可行解，该可行解与最优解的偏离程度不一定事先可以预计。

在某些情况下，特别是实际问题中，最优算法的计算时间使人无法忍受或因问题的难度使其计算时间随问题规模的增加以指数速度增加，此时只能通过启发式算法求得问题的一个可行解。



[1] 邢文训，& 谢金星. (2005). *现代优化计算方法*. 清华大学出版社
[2] 图片来源：https://leovan.me/cn/2019/04/heuristic-algorithms/

贝叶斯优化



Grid Search 和 Randomized Search 可以让整个调参过程自动化，但无法从之前的调参结果中获取信息，可能会尝试很多无效的参数空间。而贝叶斯优化会对上一次的评估结果进行追踪，建立一个概率模型，反应超参数在目标函数上表现的概率分布用于指导下一次的参数选择。贝叶斯优化适用于随机、非凸、不连续方程的优化。Sequential Model-Based Optimization (SMBO) 是贝叶斯优化更具体的表现形式，一般包含如下过程：

1. 给定要搜索的超参数空间
2. 定义一个目标函数用于评估优化
3. 建立目标函数的 Surrogate Model
4. 建立一个选择超参数的标准的评估 Surrogate Model
5. 获取评分和超参数的样本用于更新 Surrogate Model

贝叶斯优化模型主要的区分是代理函数（Surrogate Function）的差异。Surrogate Model 一般有 Gaussian Process, Random Forest 和 Tree Parzen Estimator (TPE) 这几种。常见的框架有 Spearmint, Hyperopt, SMAC, MOE, BayesianOptimization, skopt 等，它们的对比如下表：

Library	Surrogate Function
Spearmit	Gaussian Process
Hyperopt	Tree Parzen Estimator (TPE)
SMAC	Random Forest

常用的自动化机器学习库有 auto-sklearn，nni，autokeras，adanet，autogluon，Auto-PyTorch。

感谢倾听