

机器学习应用实践（实验三）—支持向量机

一、实验目的

- 1、熟悉支持向量机算法的原理。
- 2、掌握 Scikit-Learn 中支持向量机算法的使用。
- 3、掌握支持向量机四种核函数对比及使用场合。
- 4、掌握 Scikit-Learn 中提供的交叉验证与模型评估方法。
- 5、掌握网格搜索的原理及 Scikit-Learn 中 GridSearchCV 的使用。
- 6、熟悉 Scikit-Learn 自带分类数据集（乳腺癌威斯康星州数据集）的使用。
- 7、具备使用 python 实现支持向量机 SMO 算法的编程能力。

二、实验准备

- 1、回顾支持向量机算法的原理和线性 SVM 的 SMO 算法。
- 2、学习 Scikit-Learn 中支持向量机算法涉及到的相关知识。

三、相关知识介绍

1、Scikit-Learn 支持向量机相关 API 介绍

<https://scikit-learn.org/stable/modules/svm.html#svm-classification>

Scikit-Learn 中的支持向量机模块是 `sklearn.svm`，包括分类算法和回归算法。分类算法中包括 `LinearSVC`、`SVC`、`NuSVC` 和三个类。

（1）LinearSVC（线性可分支持向量机）

`LinearSVC` 是线性分类支持向量机，不能使用核函数方法处理非线性分类问题。

`LinearSVC` 的原问题公式如下：

$$\min_{\omega, b} \frac{1}{2} \omega^T \omega + C \sum_{i=0} \max(0, 1 - y_i (\omega^T \phi(x_i) + b))$$

`LinearSVC` 使用了 hinge 损失，不涉及样本之间的内积（无法应用著名的内核技巧），因此只支持线性内核。`LinearSVC` 是基于 `liblinear` 库实现的，比 `SVC` 和 `NuSVC` 算法在使用‘linear’核函数时的计算速度更快。`LinearSVC` 有多种惩罚参数和损失函数可供选择，可以应用于大样本集（大于 10000）训练。

`sklearn.svm.LinearSVC` 的主要参数：

- **penalty**：正则化的种类。包括 ‘l1’ 和 ‘l2’，默认为 ‘l2’ 正则项。
- **loss**：定义损失函数。有两种形式：‘hinge’ 和 ‘squared_hinge’，默认为 ‘squared_hinge’。‘hinge’ 是 standard SVM 的损失函数。不支持 `penalty='l1'` 且 `loss='hinge'` 设置。
- **C**：正则化系数，也是对错误分类的惩罚强度。大于 0 的浮点数，默认值 1.0。正则化强度与 C 成反比。

●**multi_class**: 多类别分类策略开关。对于多元分类问题, 选择‘ovr’将使用多类别策略(one-vs-the rest) 直接对多个类别进行分类(默认方法); 选择‘crammer_singer’将逐次进行二值分类。

●**class_weight**: 各类样本在进行损失函数计算时的数值权重(只能用于训练)。特征变量的加权系数。用于为某个特征变量设权重, 默认所有特征变量的权重相同。

sklearn.svm.LinearSVC 的主要属性:

●**coef_**: 特征权重(原问题 primal problem 的系数), 即线性模型参数。

●**intercept_**: 决策函数中的常数项, 即线性模型截距。

●**classes_**: 样本数据的分类标签。指分几类, 每一类如何表示。

示例程序:

```
>>> from sklearn.svm import LinearSVC
>>> from sklearn.pipeline import make_pipeline
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_features=4, random_state=0)
>>> clf = make_pipeline(StandardScaler(),
...                      LinearSVC(random_state=0, tol=1e-5))
>>> clf.fit(X, y)
Pipeline(steps=[('standardscaler', StandardScaler()),
                 ('linearsvc', LinearSVC(random_state=0, tol=1e-05))])
```

```
>>> print(clf.named_steps['linearsvc'].coef_)
[[0.141...  0.526... 0.679... 0.493...]]
```

```
>>> print(clf.named_steps['linearsvc'].intercept_)
[0.1693...]
>>> print(clf.predict([[0, 0, 0, 0]]))
[1]
```

(2) SVM (sklearn.svm.SVC)

SVC 和 NuSVC 都可以使用核函数方法实现非线性分类(也可以实现线性分类)。

SVC 分类器可使用核函数实现非线性分类, 是基于 libsvm 库实现的。使用 SVC 类并选择‘linear’线性核函数时, 模型训练结果与 LinearSVC 是一致的。但 SVM 的 SMO 算法不能像 LinearSVC 那样扩展到大量样本。此外, SVC 多类模式提供了“OvO”策略, 而 LinearSVC 使用“OvR”策略。

sklearn.svm.SVC 的原问题公式如下:

$$\begin{aligned} \min_{\omega, b, \zeta} \quad & \frac{1}{2} \omega^T \omega + C \sum_{i=1}^n \zeta_i \\ \text{s.t.} \quad & y_i (\omega^T \phi(x_i) + b) \geq 1 - \zeta_i \\ & \zeta_i \geq 0, i = 1, 2, \dots, n \end{aligned}$$

公式中的 C 为惩罚系数, 它控制惩罚的强度, 它充当了一个逆正则化参数的作用。 ζ_i 为与正确间隔边界的距离。

sklearn.svm.SVC 的对偶问题公式如下:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \\ \text{s.t.} \quad & y^T \alpha = 0 \\ & 0 \leq \alpha_i \leq C, i = 1, 2, \dots, n \end{aligned}$$

其中， Q 是一个 $n \times n$ 的半正定矩阵 $Q_{ij} = y_i y_j k(x_i, x_j)$ ， $k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ 为核函数。公式中的 α_i 叫做对偶系数(dual coefficients)，它的上限是惩罚项。对偶问题体现了训练集特征被核函数隐形映射到了更高维（可能是无限维）空间。

对偶问题优化求解后，给定样本的决策函数的表达式为：

$$\sum_{i \in SV} y_i \alpha_i K(x_i, x) + b$$

使用符号函数 `sign` 就可以预测类别，决策函数中只需要将支持向量（位于间隔上和间隔内的样本）进行加和，因为其他样本的对偶系数为 0。

`sklearn.svm.SVC` 中与 `LinearSVC` 不同的主要参数有：

●**kernel**: 设定核函数，包括：‘linear’，‘poly’，‘rbf’，‘sigmoid’，‘precomputed’，默认为‘rbf’。

‘linear’: 线性核函数。

‘poly’: 多项式核函数。

‘rbf’: 高斯核函数。

‘sigmoid’: S 形核函数。

‘precomputed’: 自定义核。

The kernel function can be any of the following:

- linear: $\langle x, x' \rangle$.
- polynomial: $(\gamma \langle x, x' \rangle + r)^d$, where d is specified by parameter `degree`, r by `coef0`.
- rbf: $\exp(-\gamma \|x - x'\|^2)$, where γ is specified by parameter `gamma`, must be greater than 0.
- sigmoid $\tanh(\gamma \langle x, x' \rangle + r)$, where r is specified by `coef0`.

●**degree**: 多项式核函数的次数，默认为 3。当核函数为‘poly’时使用，其它核函数时忽略此参数。

●**gamma**: ‘rbf’，‘poly’和‘sigmoid’三种核函数的核系数，包括‘scale’和‘auto’。

●**coef0**: ‘poly’和‘sigmoid’核函数中的独立项。

●**decision_function_shape**: 决策函数形状，分为‘ovo’和‘ovr’，默认为 `default='ovr'`。多分类时多用‘ovo’。当输入为‘ovr’时，决策函数形状为 $(n_samples, n_classes)$ ，当输入为‘ovo’时，decision function of shape 为 $(n_samples, n_classes * (n_classes - 1) / 2)$ 。

`sklearn.svm.SVC` 的主要属性：

●**dual_coef_**: 对偶系数，内积 $y_i \alpha_i$ 。

●**support_vectors_**: 支持向量。

●**support_**: 支持向量的索引。

- `n_support_`: 每个类别的支持向量的数量。
- `intercept_`: 截距 b 项。

示例程序:

```
>>> from sklearn import svm
>>> X = [[0, 0], [1, 1]]
>>> y = [0, 1]
>>> clf = svm.SVC()
>>> clf.fit(X, y)
SVC()
```

模型训练好之后, 就可以用来预测新的数据值:

```
>>> clf.predict([[2., 2.]])
array([1])
```

SVM 的决策函数依赖于训练数据的某些子集, 被称为支持向量。这些支持向量的一些特性可以在属性 `support_vectors_`, `support_` 和 `n_support_` 中找到:

```
>>> # get support vectors
>>> clf.support_vectors_
array([[0., 0.],
       [1., 1.]])
>>> # get indices of support vectors
>>> clf.support_
array([0, 1]...)
>>> # get number of support vectors for each class
>>> clf.n_support_
array([1, 1]...)
```

(3) NuSVC

NuSVC 相比 SVC 引入了一个新的参数 `nu`, 代替惩罚项 C , 控制支持向量的数量和间隔错误。 $nu \in (0,1]$ 是间隔错误(margin error)百分比的上限和支持向量百分比的下限。

`sklearn.svm.NuSVC` 的大部分参数和属性与 `SVC` 相似。独有参数为:

- `nu`: 训练错误率的上限, 也即支持向量的百分比下限。默认值 0.5, 取值范围(0,1]。

示例程序:

```
>>> import numpy as np
>>> X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
>>> y = np.array([1, 1, 2, 2])
>>> from sklearn.pipeline import make_pipeline
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.svm import NuSVC
>>> clf = make_pipeline(StandardScaler(), NuSVC())
>>> clf.fit(X, y)
Pipeline(steps=[('standardscaler', StandardScaler()), ('nusvc', NuSVC())])
>>> print(clf.predict([[-0.8, -1]]))
[1]
```

2、支持向量机 SMO 算法介绍

软间隔 SVM 的目标函数为:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j k(x_i, x_j) - \sum_{i=0}^n \alpha_i \\ \text{s.t.} \quad & \sum_{i=0}^n \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, i = 1, 2, \dots, n \end{aligned}$$

(1) SMO 算法公式及步骤

第一步：计算误差

$$\begin{aligned} E_i &= f(x_i) - y_i = \sum_{k=1}^n \alpha_k y_k \mathbf{x}_k^T \mathbf{x}_i + b - y_i \\ E_j &= f(x_j) - y_j = \sum_{k=1}^n \alpha_k y_k \mathbf{x}_k^T \mathbf{x}_j + b - y_j \end{aligned}$$

第二步：计算上下界 L 和 H

$$\begin{cases} L = \max(0, \alpha_j^{old} - \alpha_i^{old}), H = \min(C, C + \alpha_j^{old} - \alpha_i^{old}) & \text{if } y_i \neq y_j \\ L = \max(0, \alpha_j^{old} + \alpha_i^{old} - C), H = \min(C, \alpha_j^{old} + \alpha_i^{old}) & \text{if } y_i = y_j \end{cases}$$

第三步：计算 eta

$$\eta = \mathbf{x}_i^T \mathbf{x}_i + \mathbf{x}_j^T \mathbf{x}_j - 2\mathbf{x}_i^T \mathbf{x}_j$$

第四步：更新 alpha_j

$$\alpha_j^{new} = \alpha_j^{old} + \frac{y_j(E_i - E_j)}{\eta}$$

第五步：修剪 alpha_j

$$\alpha_j^{new,clipped} = \begin{cases} H & \text{if } \alpha_j^{new} > H \\ \alpha_j^{new} & \text{if } L \leq \alpha_j^{new} \leq H \\ L & \text{if } \alpha_j^{new} < L \end{cases}$$

第六步：更新 alpha_i

$$\alpha_i^{new} = \alpha_i^{old} + y_i y_j (\alpha_j^{old} - \alpha_j^{new,clipped})$$

第七步：更新阈值 bi 和 bj

$$\begin{aligned} b_i^{new} &= b^{old} - E_i - y_i(\alpha_i^{new} - \alpha_i^{old})\mathbf{x}_i^T \mathbf{x}_i - y_j(\alpha_j^{new} - \alpha_j^{old})\mathbf{x}_j^T \mathbf{x}_i \\ b_j^{new} &= b^{old} - E_j - y_i(\alpha_i^{new} - \alpha_i^{old})\mathbf{x}_i^T \mathbf{x}_j - y_j(\alpha_j^{new} - \alpha_j^{old})\mathbf{x}_j^T \mathbf{x}_j \end{aligned}$$

第八步：更新阈值 b

$$b = \begin{cases} b_i & 0 < \alpha_i^{new} < C \\ b_j & 0 < \alpha_j^{new} < C \\ \frac{b_i + b_j}{2} & \text{otherwise} \end{cases}$$

(2) SMO 算法伪代码 (详见参考文献)

target = desired output vector

point = training point matrix

procedure takeStep(i1,i2)

 if (i1 == i2) return 0

 alph1 = Lagrange multiplier for i1

 y1 = target[i1]

 E1 = SVM output on point[i1] - y1 (check in error cache)

 s = y1*y2

 Compute L, H

 if (L == H)

 return 0

 k11 = kernel(point[i1],point[i1])

 k12 = kernel(point[i1],point[i2])

 k22 = kernel(point[i2],point[i2])

 eta = k11+k22-2*k12

 if (eta > 0)

 {

 a2 = alph2 + y2*(E1-E2)/eta

 if (a2 < L) a2 = L

 else if (a2 > H) a2 = H

 }

 else

 {

 Lobj = objective function at a2=L

 Hobj = objective function at a2=H

 if (Lobj < Hobj-eps)

 a2 = L

 else if (Lobj > Hobj+eps)

 a2 = H

 else

 a2 = alph2

 }

 if ($|a2 - \alpha_2| < \epsilon * (a2 + \alpha_2 + \epsilon)$)

 return 0

 a1 = $\alpha_1 + s * (\alpha_2 - a2)$

 Update threshold to reflect change in Lagrange multipliers

 Update weight vector to reflect change in a1 & a2, if SVM is linear

 Update error cache using new Lagrange multipliers

```

Store a1 in the alpha array
Store a2 in the alpha array
return l
endprocedure

```

参考文献：

[1] John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, 1998.

[2] John C. Platt. Fast training of support vector machines using sequential minimal optimization. In Advances in Kernel Methods - Support Vector Learning. B. Scholkopf, C. J. C. Burges, and A. J. Smola, Eds. MIT Press, Cambridge, MA, 1999: 185-208.

3、交叉验证与模型评估

在机器学习领域，严格意义上的测试集是不能参与建模的，不仅在训练模型时不能带入测试集进行训练，还不能根据测试集上的泛化误差再进行模型修改（比如模型更换或超参数调整）。因此，我们可以将数据集划分成训练集、验证集、测试集，使用训练集进行模型训练，通过验证集评价模型参数选择的优劣，最终在测试集上对最优模型进行模型评估，机器学习的模型训练和评估流程如图 1 所示。官网：https://scikit-learn.org/stable/modules/cross_validation.html。

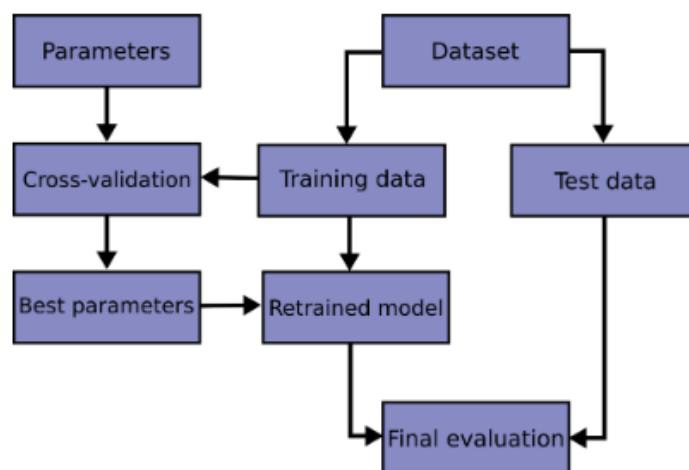


图 1 sklearn 官网提供模型训练和评估流程图

在不严格区分测试集的情况下，可以只划分训练集和验证集。在前两次实验中，我们使用了留出法 `train_test_split` 函数进行了一次数据集划分，将数据集分成训练集和验证集，一部分用于训练，一部分用于验证，存在两个弊端：一是最终模型与参数的选取极大程度依赖于数据集的划分，如果数据集划分得不够好，很可能无法选择到最好的模型与参数。二是只使用了部分数据进行模型训练，没有充分利用已有数据。基于此，提出了交叉验证(cross-validation)方法。

交叉验证（cross-validation，简称 CV）是一种能够提高数据利用率和模型评估可信度的误差估计方法。最基础最常用的一种交叉验证法是 K-fold cross-validation（K 折交叉验证），就是将数据集（或训练集）进行 K 份等比例划分，然后依次取出其中一份作为验证集、剩下全部用来训练，最终产生 K 个不同的数据集划分结果。例如，当我们使用 5 折交叉验证时，数据集划分情况如图 2 所示，蓝色表示验证集，绿色表示训练集，共产生 5 种数据集划分结果，对于同一组模型超参数设置，

基于 5 组不同的数据进行建模，得到 5 个模型在验证集上的表现，将 5 个验证集结果求平均，作为当前模型超参数的模型评价结果。通过交叉验证可以选择模型超参数（这部分在网格搜索中介绍）。

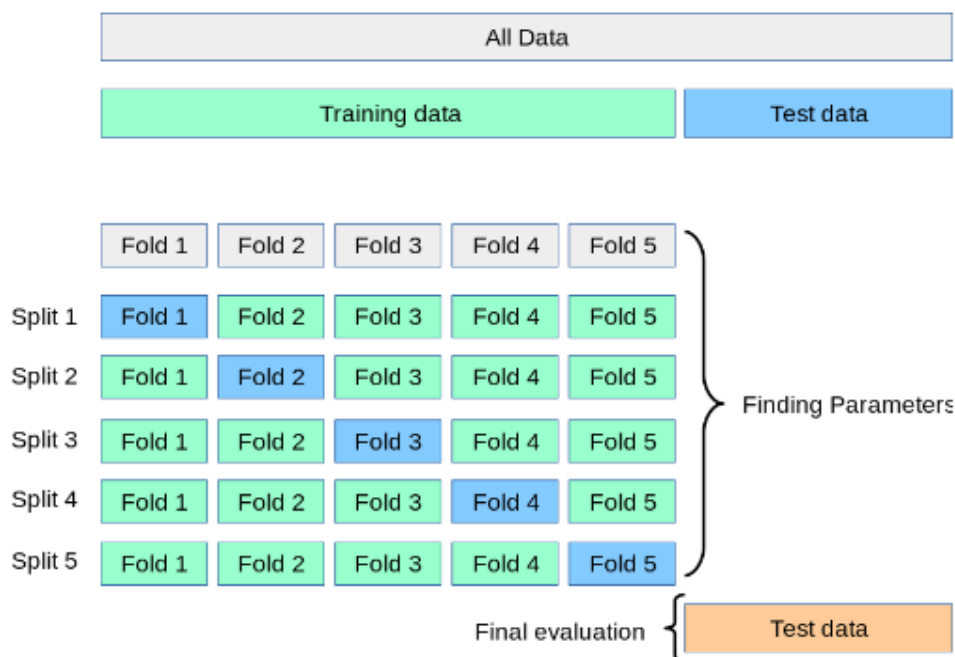


图 2 5 折交叉验证数据集划分图示

sklearn 中提供许多 K-fold 数据划分方法和交叉验证方法，这里介绍三种较常用的 `cross_val_score`、`cross_validate` 和 `validation_curve` 方法，可适用于不同的应用场合。

(1) `cross_val_score`

`cross_val_score` 可直接用于某一组超参数下的模型评估。

示例程序：

```
>>> from sklearn.model_selection import cross_val_score
>>> clf = svm.SVC(kernel='linear', C=1, random_state=42)
>>> scores = cross_val_score(clf, X, y, cv=5)
>>> scores
array([0.96..., 1. , 0.96..., 0.96..., 1. ])
```

可通过 `mean()` 和 `std()` 方法计算 K-fold 上的评分均值和标准差：

```
>>> print("%.2f accuracy with a standard deviation of %.2f" % (scores.mean(), scores.std()))
0.98 accuracy with a standard deviation of 0.02
```

也可根据需求设定不同评价指标：

```
>>> from sklearn import metrics
>>> scores = cross_val_score(
...     clf, X, y, cv=5, scoring='f1_macro')
>>> scores
array([0.96..., 1. ..., 0.96..., 0.96..., 1. ])
```

(2) `cross_validate`

`cross_validate` 相比 `cross_val_score`，有两个不同之处。其一是 `cross_validate` 支持超过一个性能指标的计算。其二是能够提供更多的模型训练和测试过程信息，如训练过程所用时间、测试过程所用时间、K-fold 训练集上的误差等。

示例程序：

```
>>> from sklearn.model_selection import cross_validate
>>> from sklearn.metrics import recall_score
>>> scoring = ['precision_macro', 'recall_macro']
>>> clf = svm.SVC(kernel='linear', C=1, random_state=0)
>>> scores = cross_validate(clf, X, y, scoring=scoring)
>>> sorted(scores.keys())
['fit_time', 'score_time', 'test_precision_macro', 'test_recall_macro']
>>> scores['test_recall_macro']
array([0.96..., 1. ..., 0.96..., 0.96..., 1. ...])
```

(3) validation_curve

`validation_curve` 可以得到单个超参数的取值不同，交叉验证后的训练分数和验证分数，可以此绘制单个超参数对训练分数和验证分数的影响曲线，有助于分析估计器对某些超参数值是过拟合还是欠拟合。详见官网：https://scikit-learn.org/stable/modules/learning_curve.html#validation-curve。

示例程序：

```
>>> import numpy as np
>>> from sklearn.model_selection import validation_curve
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import Ridge

>>> np.random.seed(0)
>>> X, y = load_iris(return_X_y=True)
>>> indices = np.arange(y.shape[0])
>>> np.random.shuffle(indices)
>>> X, y = X[indices], y[indices]

>>> train_scores, valid_scores = validation_curve(
...     Ridge(), X, y, param_name="alpha", param_range=np.logspace(-7, 3, 3),
...     cv=5)
>>> train_scores
array([[0.93..., 0.94..., 0.92..., 0.91..., 0.92...],
      [0.93..., 0.94..., 0.92..., 0.91..., 0.92...],
      [0.51..., 0.52..., 0.49..., 0.47..., 0.49...]])
>>> valid_scores
array([[0.90..., 0.84..., 0.94..., 0.96..., 0.93...],
      [0.90..., 0.84..., 0.94..., 0.96..., 0.93...],
      [0.46..., 0.25..., 0.50..., 0.49..., 0.52...]])
```

`validation_curve` 与 `cross_validate` 和 `cross_val_score` 的不同在于，`cross_validate` 和 `cross_val_score` 只用于一组超参数取值（`scores` 均是一维数组），而 `validation_curve` 能够得到多个超参数取值下的模型训练分数和验证分数（`scores` 均是二维数组）。可通过 `mean()` 和 `std()` 方法计算 K-fold 上的评分均值和标准差，用于绘制超参数曲线。

官网提供了曲线绘制示例程序，结果如图 3。图中是两条以超参数的不同取值为横坐标，模型的训练集分数和验证集分数为纵坐标的曲线。对于非常低的 `gamma` 值，可以看到训练分数和验证分数都很低，为欠拟合。中等 `gamma` 值的分类器性能刚好。`gamma` 值太高，训练分数高，验证分数低，为过拟合。由此可以确定适合的 `gamma` 值。

https://scikit-learn.org/stable/auto_examples/model_selection/plot_validation_curve.html#sphx-glr-auto-examples-model-selection-plot-validation-curve-py。

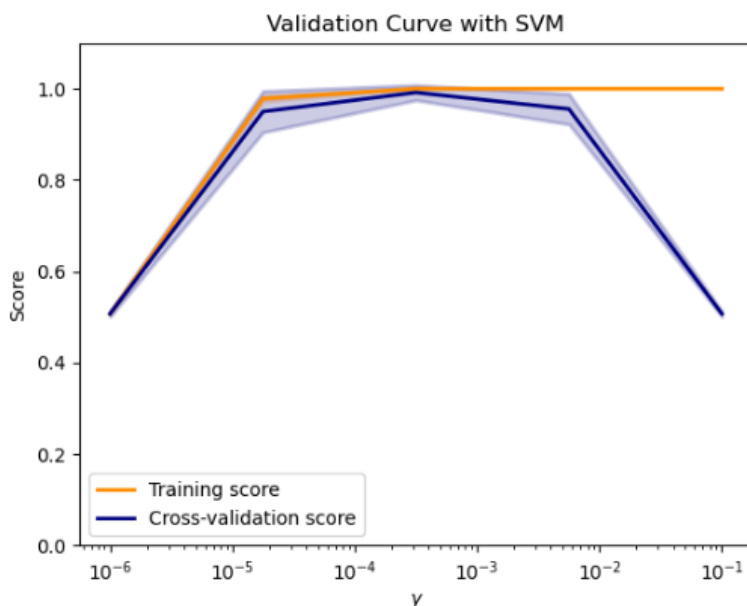


图 3 不同核参数 gamma 值的 SVM 模型的训练分数和验证分数曲线

4、基于网格搜索的模型超参数调整

提升模型泛化性能的手段有很多，如模型选择（甚至是模型创新）、特征工程、模型超参数（hyper-parameters）选择等，此处我们将要介绍的，是围绕某个具体的模型，通过选择适合的模型超参数，来提高模型的泛化能力。

在机器学习的过程中，超参数（hyper-parameter）是指在开始模型训练之前，就人为设置好的，无法在模型训练中直接学习的参数。而模型参数是指通过训练得到的参数数据（包括 `coef_`、`intercept_` 等）。虽然对于机器学习来说超参数众多，但能够对模型的建模结果产生决定性影响的超参数却不多，对于一部分超参数，我们采用“经验结合实际”的方式就可以确定其取值，如数据集划分比例、交叉验证的折数等（这些在 `sklearn` 中遵循默认设置即可），而对于一些如正则化系数、特征衍生阶数等直接影响模型泛化性能的超参数则需要一定的方法进行寻优。超参数寻优没有一个严谨的数学流程或公式，因此可以通过“经验+一定范围内枚举”的方法确定，也就是网格搜索（`gridsearch`）。

网格搜索是指将备选的超参数一一列出，多个不同参数的不同取值组成一个参数空间（parameter space），在这个参数空间中选取不同的值带入模型进行训练，最终选取一组最优的超参数组合作为模型的最终超参数。此处“最优”的超参数，应是尽可能使模型泛化能力更好的参数。在这个过程中，有两个核心问题需要注意，其一是构造参数空间，要选取能够控制模型的经验风险和结构风险（让模型既不过拟合也不欠拟合）的超参数。其二是选取适合的模型泛化能力的评估指标。

在大多数情况下，更为严谨的网格搜索的过程是将网格搜索（`gridsearch`）和交叉验证（CV）结合，这也是 `sklearn` 中 `GridSearchCV` 的命名由来。

该过程的具体步骤如下（也是 `sklearn` 中的 `GridSearchCV` 实现的功能）：

- （1）确定参数空间：选择网格搜索的超参数及其范围。
- （2）遍历参数空间中的所有参数组合，对于每一组超参数，采用交叉验证评估模型性能：

- a) 在训练集中进行验证集划分 (K-fold) ;
- b) 超参数带入训练集进行建模、带入验证集进行验证,并输出验证集上的模型评估指标;
- c) 计算多组验证集上的评估指标的均值, 作为该超参数下模型最终表现。

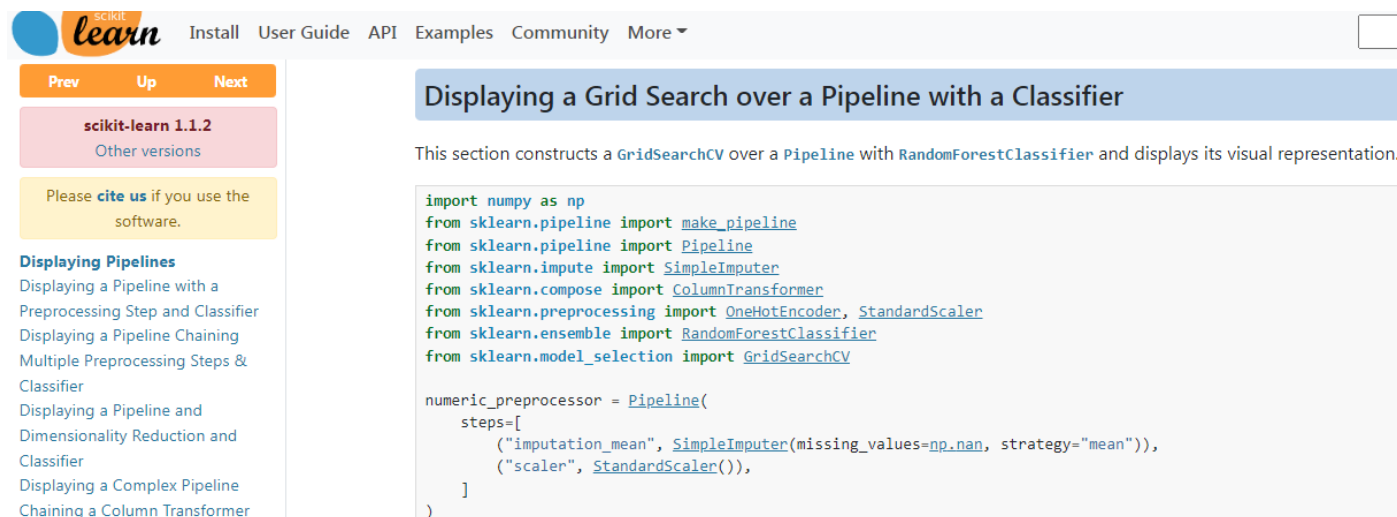
(3) 在所有超参数组合中选取模型泛化能力最好的一组参数, 以及数据集划分。

(4) 根据最好的数据集划分和超参数组合, 训练得到最优模型用于预测。

sklearn 中 GridSearchCV 网格搜索中的核心要素, 分别是: 评估器、参数空间、搜索策略、交叉验证以及评估指标。官网: https://scikit-learn.org/stable/modules/grid_search.html#grid-search。

sklearn 提供了多种搜索策略, 分别是 GridSearchCV、RandomizedSearchCV、HalvingGridSearchCV 和 HalvingRandomSearchCV 方法。GridSearchCV 会尝试参数空间内的所有组合, 其缺点就是搜索速度过慢。sklearn 中提供的各种搜索策略主要从两方面进行了优化: 一是调整搜索空间, 相应的方法是 RandomizedSearchCV(随机网格搜索), 它对参数空间内按某种分布随机抽样得到的子集进行搜索。二是调整每次训练的数据, 相应的方法就是 HalvingGridSearchCV(对半网格搜索)。还有二方面结合的 HalvingRandomSearchCV(随机对半网格搜索), HalvingGridSearchCV 和 HalvingRandomSearchCV 是 sklearn 0.24 版及以上才支持的功能, 该功能也是 0.24 版最大的改动之一, 能够进一步减少网格搜索所需计算资源、加快网格搜索的速度。本节重点介绍 GridSearchCV 策略, HalvingGridSearchCV 策略将在后续的章节中介绍。

GridSearchCV 中的评估器可以是模型, 也可以是 pipeline。网格搜索与 pipeline 的结合使用详见官网: https://scikit-learn.org/stable/auto_examples/miscellaneous/plot_pipeline_display.html#sphx-glr-auto-examples-miscellaneous-plot-pipeline-display-py。



The screenshot shows the scikit-learn website interface. The sidebar on the left contains navigation links: 'Prev', 'Up', 'Next', 'scikit-learn 1.1.2', 'Other versions', 'Please cite us if you use the software.', and a list of tutorials including 'Displaying Pipelines'. The main content area is titled 'Displaying a Grid Search over a Pipeline with a Classifier' and contains a code snippet for creating a GridSearchCV object over a pipeline.

```
import numpy as np
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

numeric_preprocessor = Pipeline(
    steps=[
        ("imputation_mean", SimpleImputer(missing_values=np.nan, strategy="mean")),
        ("scaler", StandardScaler()),
    ]
)
```

GridSearchCV 中的参数如表 1 所示。

表 1 GridSearchCV 超参数表

参数	解释
estimator	调参对象, 某评估器
param_grid	参数空间, 可以是字典或者字典构成的列表
scoring	评估指标, 支持同时输出多个参数

n_jobs	设置工作时参与计算的 CPU 逻辑核数
refit	挑选评估指标和最佳参数，并在完整数据集上进行训练，得到一个泛化性能最好的模型
cv	交叉验证的折数
verbose	输出工作日志形式
pre_dispatch	多任务并行时任务划分数量
error_score	当网格搜索报错时返回结果，选择'raise'时将直接报错并中断训练过程，其他情况会显示警告信息后继续完成训练
return_train_score	在交叉验证中是否显示训练集中参数得分

●**n_jobs**: 设置要并行运行的作业数，即参与计算的 CPU 逻辑核数，默认为 None，表示 1。用于设置使用 joblib 并行计算的程序要使用多少并发进程或线程。joblib 库是一个可以将 Python 代码转换为并行计算模式的软件包。sklearn 中就大量使用 joblib 进行并行加速，提高计算速度。在网格搜索中，在每个参数值和每个交叉验证分割的组合上，训练和预测的过程都是可以并行计算的。n_jobs 最多可设置为机器 CPU 逻辑核心数量，超出亦等价于使用全部逻辑核，设置 n_jobs = -1 也表示使用全部逻辑核，若不希望全部 CPU 资源均被并行任务占用，也可设置更小的负数来保留适当的空闲核，此时使用的逻辑核数=(n_cpus + 1 + n_jobs)，例如设置为-2 则使用(全部核-1)个，设置为-3 则使用(全部核-2)个。

●**verbose**: 设置输出工作日志形式，数值越大信息越详细。为‘1’表示显示交叉验证折数和参数候选的计算时间，如下图所示。为‘2’还会显示交叉验证分数。为‘3’表示每个交叉验证划分和候选参数索引也与计算的开始时间一起显示。

```
reg = RFR(random_state=1412)
cv = KFold(n_splits=5,shuffle=True,random_state=1412)

result_pre_adjusted = cross_validate(reg,X,y,cv=cv,scoring="neg_mean_squared_error",
                                     ,return_train_score=True
                                     ,verbose=True
                                     ,n_jobs=-1)

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:   1.1s finished
```

GridSearchCV 在使用时也需要实例化，使用 fit 函数完成网格搜索和模型训练的过程。训练完成后，可查看的属性和调用方法如表 2 所示。

表 2 GridSearchCV 的主要属性和方法

参数	解释
best_estimator_	最终挑选出的最优
best_score_	在最优参数情况下，训练集的交叉验证的平均得分
best_params_	最优参数组合
best_index_	CV 过程会对所有参数组合标号，该参数表示最优参数组合的标号
scorer	在最优参数下，计算模型得分的方法
n_splits_	交叉验证的折数
refit_time_	用于在整个数据集上重新训练最佳模型的时间

cv_results_

交叉验证过程中的重要结果（字典形式，可 DataFrame 显示）

示例程序：

```
>>> from sklearn import svm, datasets
>>> from sklearn.model_selection import GridSearchCV
>>> iris = datasets.load_iris()
>>> parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
>>> svc = svm.SVC()
>>> clf = GridSearchCV(svc, parameters)
>>> clf.fit(iris.data, iris.target)
GridSearchCV(estimator=SVC(),
              param_grid={'C': [1, 10], 'kernel': ('linear', 'rbf')})
>>> sorted(clf.cv_results_.keys())
['mean_fit_time', 'mean_score_time', 'mean_test_score', ...
 'param_C', 'param_kernel', 'params', ...
 'rank_test_score', 'split0_test_score', ...
 'split2_test_score', ...
 'std_fit_time', 'std_score_time', 'std_test_score']
```

四、实验内容

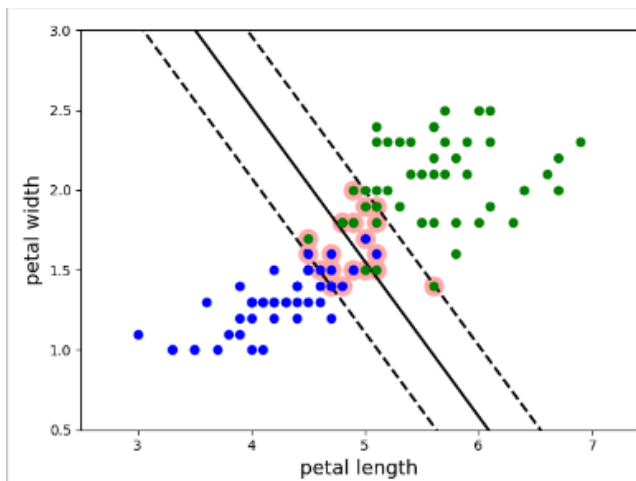
1、题目一：采用 scikit-learn 中的线性 SVM 对 iris 数据集进行二分类。

具体内容：

（1）选取两个特征和两类数据使用 scikit-learn 中的 SVM 进行二分类。

（2）输出：决策边界的参数和截距、支持向量等。

（3）可视化：通过散点图可视化数据样本（之前选择的两个特征），并画出决策边界和 2 个最大间隔边界，标出支持向量。



【讨论一】选取的两个特征能否线性可分？若线性可分，可选择 scikit-learn 中何种 SVM 进行建模？若线性不可分，可选择 scikit-learn 中何种 SVM 进行建模？

【讨论二】SVM 中的惩罚系数 C 对模型有何影响？

（1）尝试改变惩罚系数 C，分析其变化对应间隔宽度、支持向量数量的变化趋势，并解释原因。

（2）尝试改变惩罚系数 C，分析其对 iris 分类模型性能的影响，并解释原因。

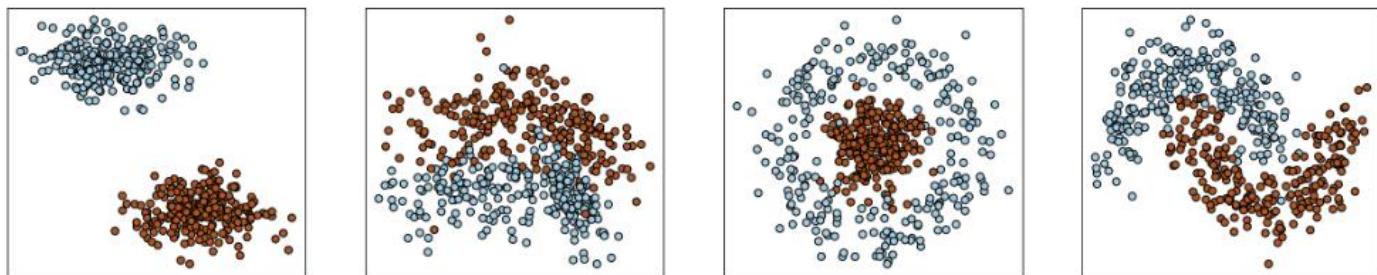
2、题目二（选做）：采用不同的 SVM 核函数对多种类型数据集进行二分类。

具体内容：

(1) 使用 `scikit-learn` 中提供的样本生成器 `make_blobs`、`make_classification`、`make_moons`、`make_circles` 生成一系列线性或非线性可分的二类别数据（数据量任取）。样本生成器的使用参考官网：<https://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets>

数据集生成代码和数据集散点图如下所示：

```
X, Y = datasets.make_blobs(n_samples = 500, centers = 2, cluster_std=0.4, random_state = 0)
X, Y = datasets.make_classification(n_samples=500, n_features=2, n_informative=2, n_redundant=0, random_state=20)
X, Y = datasets.make_circles(n_samples = 500, noise = 0.2, factor = 0.2, random_state=0)
X, Y = datasets.make_moons(n_samples = 500, noise = 0.2, random_state=0)
```



(2) 建模：分别将 **SVM** 中四种核函数（线性核、多项式核、高斯核、S 形核）用于上述四种数据集。

提示：对于每一种核函数，选择最适合的核参数（如 RBF 核中 `gamma`、多项式核中 `degree` 等）。可通过超参数曲线帮助选择超参数。

(3) 可视化：通过散点图可视化数据样本，并画出 **SVM** 模型的决策边界。

(4) 模型评价：分类准确率。

【讨论三】如何选择最优超参数？

为每种模型选择适合的核函数及核参数，参数寻优方式自选。

【讨论四】不同核函数在不同数据集上表现如何？

通过观察不同核函数在不同数据集上的决策边界和分类准确率，分析不同核函数的适用场合。

3、题目三：使用 `scikit-learn` 中的 **SVM** 分类器对乳腺癌威斯康星州数据集进行分类。

(1) 导入数据集：乳腺癌威斯康星州数据集是 `sklearn` 中自带的数据集（`load_breast_cancer`）。

通过查看数据量和维度、特征类型（离散 or 连续）、特征名、标签名、标签分布情况、数据集的描述等信息了解数据集。

(2) 建模：分别使用四种核函数对数据集进行分类。

(3) 模型评价：每种核函数下的分类准确率、计算时间等。

【讨论五】四种核函数在这个数据集上表现如何？

提示：不要求可视化，从准确率上判断即可。

【讨论六】**SVM** 是否需要数据进行归一化处理？数据归一化对核函数有何影响？

提示：尝试分析数据归一化对四种核函数的工作有何影响，从分类准确率、计算时间等角度对比。

4、题目四：编写 SMO 算法实现线性 SVM 分类器，对 iris 数据集进行二分类。

具体内容：

(1) 选取两个特征和两类数据进行二分类。

注意：二分类标签为 1 和-1。

(2) 划分数据（分成训练集和数据集）

(3) 数据归一化

(4) 训练模型（参考程序模板：SVM_numpy_template.py）

(5) 输出：SVM 对偶问题目标函数的最优解 α ，决策函数的参数和截距，支持向量等。

(6) 可视化：通过散点图可视化训练数据样本，并画出决策面和 2 个最大间隔面，标出支持向量（包括间隔上和间隔内的样本），能够帮助检验算法正确性。

(7) 测试集数据进行预测，评估模型性能。

【讨论七】描述软间隔 SVM 中的 C 参数、拉格朗日乘子 α 、支持向量与最优决策面和间隔区域之间的关系。

五、实验报告要求

每个题目请按照以下说明进行书写。

(1) 开发环境：jupyter notebook 或其他开发环境，请根据自身情况如实填写。

(2) 总结算法编写流程和库函数使用经验。

(3) 程序代码、运行结果：请截图，保证图片清晰能够反映实际运行结果（如果程序较长，可分段进行程序和结果展示）。

(4) 拓展探究：对函数参数不同设置的研究、不同算法的尝试等。

(5) 调试改进：针对每个题目，有选择性的列举在实验过程中遇到的典型问题及描述、原因分析排查以及解决方式说明等。

注意事项：

1、实验报告严禁相互抄袭，如发现实验报告雷同，则雷同的实验报告小项评分最高为及格。

2、每次实验需提交：实验报告+程序源码(均按“学号+姓名”形式命名)，报告与程序分开提交。