



# SVM

## 大纲



### 1. 线性SVM分类

- **回顾线性模型分类超平面：** 线性模型、分类边界、分类规则
- **线性可分情况：** 间隔、最优超平面、支持向量
- **硬间隔分类算法：** 优化目标、求解方法
- **SVM的特征缩放敏感性：** 特征缩放对决策边界的影响
- **硬间隔分类的局限性：** 只适用于线性可分数据，对离群点敏感
- **软间隔分类：** 引入松弛变量、优化目标、支持向量
- **梯度下降算法：** 转换为无约束优化问题、损失函数、梯度计算



### 2. 非线性SVM分类

- **特征变换：** 通过添加特征使线性不可分数据变为线性可分
- **核函数：** 核技巧、常用核函数（线性核、多项式核、高斯核、sigmoid核）
- **核技巧与SVM：** 利用核函数避免高维映射



### 3. scikit-learn中的SVM

- **SVM算法实现**： 分类算法（SVC、LinearSVC、SGDClassifier）、回归算法（SVR、LinearSVR、SGDRegressor）、异常值检测（OneClassSVM）
- **Scikit-learn中SVC类**： 主要参数、主要方法、主要属性
- **Scikit-learn中LinearSVC类**： 主要参数、主要方法、主要属性
- **不同SVM分类器决策面**： 线性模型、非线性模型



### 4. 超参数优化

- **超参数**： 定义、影响
- **网格搜索交叉验证**： GridSearchC类、主要参数、主要方法、主要属性
- **随机搜索交叉验证**： RandomizedSearchCV类、主要参数、主要方法、主要属性、RandomizedSearchCV特点



### 5. SVM回归

- **$\epsilon$ -SVR原问题**： 目标、优化目标、求解方法
- **sklearn中的SVM回归类**： SVR、LinearSVR、SGDRegressor
- **sklearn中的SVR类**： 主要参数、主要方法、主要属性
- **SVM用于加州房价预测**： 数据集介绍、随机搜索交叉验证寻找最优参数组合、最优模型的RMSE



### 6. 保存模型

- **joblib工具包**： 保存模型、加载模型

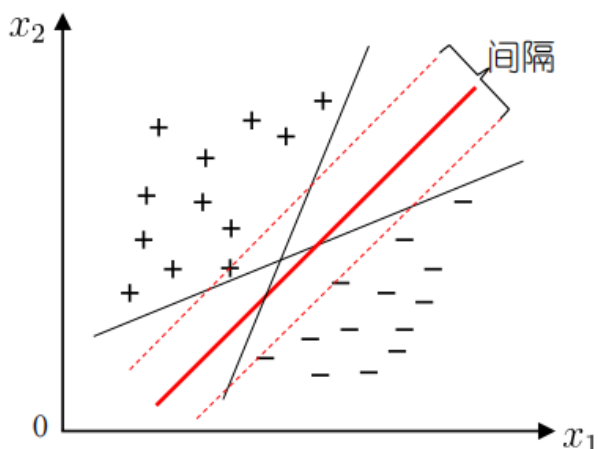


### 7. SVM优缺点和参数w

- **优点**： 最大间隔准则、核函数技术、适用范围广、预测速度快、内存消耗少
- **缺点**： 对样本规模缩放敏感、需要预处理数据、调参复杂
- **参数**： gamma、C、Kernel

## 原理

## 支持向量与间隔



## 间隔(margin)

这个正中间的线离两类中最靠近样本的距离同时达到最大值。这里，这个距离被称为**分类间隔**。

间隔越大，分类器的泛化性能越好。因此，**具有最大间隔的分类超平面是最优分类超平面。**

支持向量是用来确定**间隔**的样本，它们对寻找最优超平面起到决定性作用。SVM分类算法依靠支持向量，寻找使得分类间隔最大的那个最优超平面。两类中与最优超平面最靠近的样本，被称为**支持向量**。

## 求解超平面

**输入：**线性可分训练数据集 $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ ,  $y^{(i)} \in \{+1, -1\}$ ,  $i=1, \dots, m$ 。

**输出：**最大间隔分离超平面和分类决策函数。

(1) 通过求解约束最优化问题：

**硬间隔线性SVM**  
**分类目标**

$$\begin{aligned} \min_{w, b} \quad & \frac{1}{2} w^T w \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) - 1 \geq 0, \quad i = 1, 2, \dots, m \end{aligned}$$

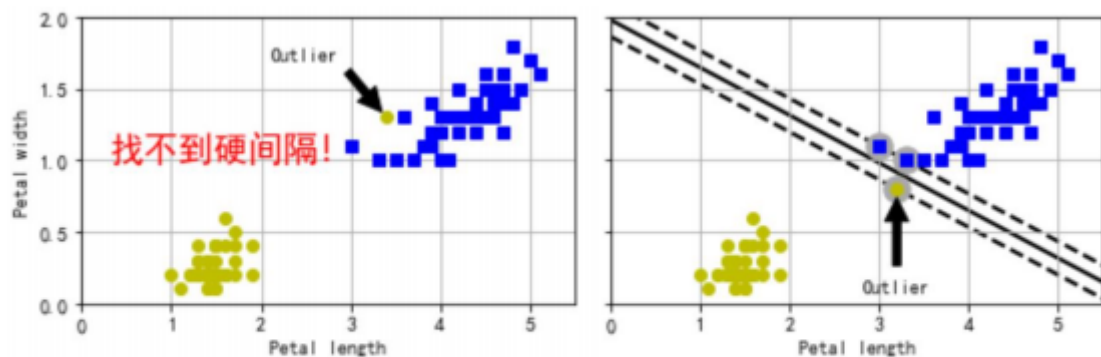
属于线性约束的凸二次优化问题。  
这类问题被称为二次规划。  
有许多现成的解法。

求得最优解 $w^*, b^*$ 。

(2) 得到分类超平面(即决策边界):  $w^* \cdot x + b^* = 0$

这种严格要求所有样本都在间隔边界的正确一侧的分类，被称为硬间隔最大化分类，简称**硬间隔分类**。

**硬间隔的缺点：**对离群点，或者叫噪声点极度敏感



有异常值时

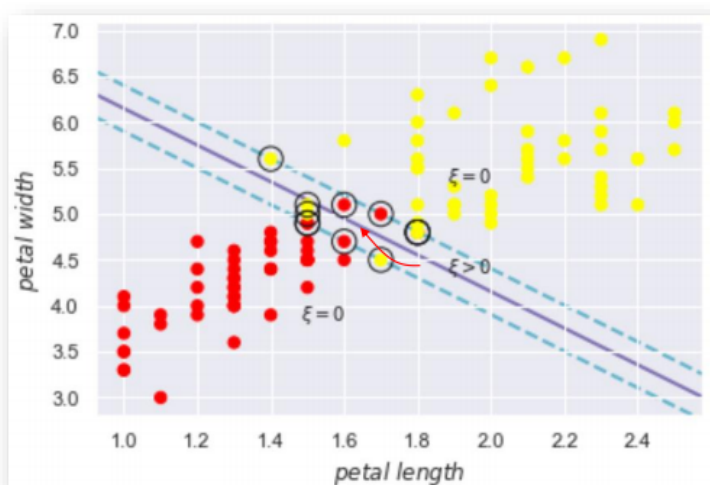
🎁 所以我们可以引入松弛变量，来对超平面附近的点去除硬间隔约束，从而达到更好地分割效果

对存在少数离群点的情况，SVM引入  
**松弛变量  $\xi^{(i)}$**

- 不能与超平面完美分离时，允许一些样本离它们正确一侧的间隔边界有一定距离  $\xi^{(i)}$ ，即

$$y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi^{(i)}。$$

- 优化目标：**最大化间隔的同时，当样本被错分或在边界内时 ( $\xi^{(i)} > 0$ ) 给予惩罚。用超参数C控制惩罚强度。



位于间隔边界内或被错分的样本(违例)  $i$ ， $\xi^{(i)} > 0$

位于间隔边界上或外正确一侧的样本点  $i$ ， $\xi^{(i)} = 0$

☂ C是正则化参数，也称惩罚因子，用C来控制对间隔违例的惩罚强度。

算法对应的约束优化问题为：

$$\begin{aligned} \min_{\mathbf{w}, b, \xi} & \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^m \xi^{(i)} \\ \text{s.t. } & \xi^{(i)} \geq 0 \\ & y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq 1 - \xi^{(i)}, i = 1, 2, \dots, m \end{aligned}$$

将其近似转为无约束优化问题，得**线性SVM损失函数**：

$$L(\mathbf{w}, b) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^m \max(0, 1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b))$$

Hinge损失函数

计算梯度：

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} + \begin{cases} 0, & \text{若 } y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) > 1 \\ -C y^{(i)} \mathbf{x}^{(i)}, & \text{若 } y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) < 1 \end{cases}$$

$$\frac{\partial L}{\partial b} = \begin{cases} 0, & \text{若 } y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) > 1 \\ -C y^{(i)}, & \text{若 } y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) < 1 \end{cases}$$

🍒 这里的方式就是用高等数学当中的拉格朗日条件，求条件极值；这个条件就是偏导等于0的时候的情况，也就是求梯度等于0的情况。

## 损失函数

**合页损失**是一种用于训练分类器的损失函数。常用于“最大间隔分类”中，尤其是SVM。对于预期输出  $y^{(i)} = \pm 1$  和分类器输出  $f(\mathbf{x}^{(i)})$ ，预测  $f(\mathbf{x}^{(i)})$  的合页损失定义为：

$$l(y^{(i)}, f(\mathbf{x}^{(i)})) = \max(0, 1 - y^{(i)} f(\mathbf{x}^{(i)}))$$

平方hinge损失函数：  $l(s) = \max(0, (1 - s)^2) = \max(0, (1 - y^{(i)} f(\mathbf{x}^{(i)}))^2)$



对于给定的数据点  $(x, y)$ ，其中  $x$  是特征向量， $y$  是标签（取值为 +1 或 -1），以及模型的预测  $f(x) = \text{sign}(w^T x + b)$ ，合页损失的数学表达式为：

$$L(y, f(x)) = \max(0, 1 - y \cdot f(x))$$

其中， $w$  是权重向量， $b$  是偏置项， $\text{sign}(\cdot)$  是符号函数，当内部表达式大于0时返回 +1，小于0时返回 -1。

合页损失的解释如下：

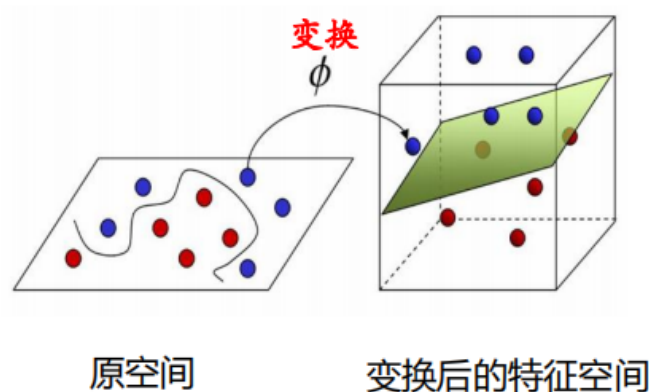
1. 当  $y \cdot f(x) \geq 1$  时，意味着模型正确分类了数据点，并且分类边界的间隔至少为1。在这种情况下，损失为 0，因为模型已经正确地处理了这个数据点。
2. 当  $y \cdot f(x) < 1$  时，意味着模型没有正确地分类数据点，或者分类边界的间隔小于1。在这种情况下，损失为  $1 - y \cdot f(x)$ ，这是一个正值，表示模型在分类这个数据点时犯了错误，或者分类边界的间隔不够大。

## 非线性分类方法



当数据集线性不可分的时候，增强特征映射到高维空间可以线性可分

**存在的问题：**线性不可分的数据集，通过添加特征，映射到高维空间，以便生成的数据集线性可分，这种思路非常好。但是，添加的特征少对复杂数据集无效，而添加大量特征会导致模型变得很慢，**会耗费计算资源、也会导致泛化困难！！**



## 线性模型决策函数的对偶式：

$$f(x) = \underbrace{w \cdot x + b}_{\text{模型的原始形式}} = \underbrace{\sum_{i \in sv} \alpha_i y_i (x_i \cdot x)}_{\text{模型的对偶形式}} + b$$

特征向量的点积

其中， $x$ 是测试样本， $x_i$ 是支持向量。 $sv$ 是训练集中所有支持向量的集合。

通过特征变换  $\phi$  将数据映射到一个更高维空间中，新决策函数的对偶式：

$$f(x) = \sum_{i \in sv} \alpha_i y_i (\phi(x_i) \cdot \phi(x)) + b$$

映射后向量的点积

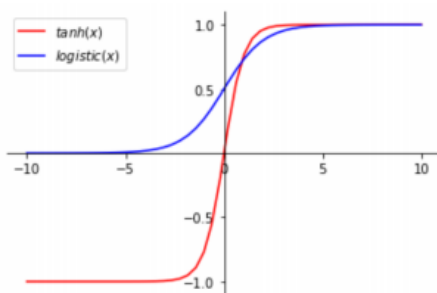
这个时候我们就可以更换我们的决策函数，使得有更多分割方式，但是这个函数一定是连续的

## 常用的核函数

常用的核函数：线性核、多项式核、高斯核、sigmoid核。

- 线性核：  $K(a, b) = a^T b$
- 多项式核：  $K(a, b) = (\gamma a^T b + r)^d$ ， $d > 1$  多项式次数
- 高斯RBF核：  $K(a, b) = \exp(-\gamma \|a - b\|^2)$ ， $\gamma > 0$  高斯核的核宽倒数
- sigmoid核：  $K(a, b) = \tanh(\gamma a^T b + r) = \frac{\exp(\gamma a^T b + r) - \exp(-(\gamma a^T b + r))}{\exp(\gamma a^T b + r) + \exp(-(\gamma a^T b + r))}$

在Scikit-learn的核SVM中  
 $\gamma$ ：由超参数 gamma指定；  
 $d$ ：由超参数degree指定；  
 $r$ ：由超参数coef0指定。



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$$

### 1. 线性核 $K(a, b) = a^T b$ :

- 应用场景：当数据是线性可分的时候，即数据点可以通过一条直线或超平面完全分开时，线性核是最合适的选择。它计算快速且易于实现，适合大规模数据集和高维数据。

### 2. 多项式核 $K(a, b) = (\gamma a^T b + r)^d$ , 其中 $d > 1$ 是多项式次数:

- 应用场景：多项式核可以捕捉到数据的非线性关系，特别适用于那些在原始特征空间中不是线性可分的数据。通过调整参数  $d$ ,  $\gamma$  和  $r$ , 可以实现不同复杂度的决策边界。然而，随着  $d$  的增加，模型的计算复杂度和风险也会增加。

### 3. 高斯径向基函数 (RBF) 核 $K(a, b) = \exp(-\gamma \|a - b\|^2)$ :

- 应用场景：RBF核是非常灵活的，能够处理几乎任何类型的非线性问题。它在局部保持数据的结构信息，因此非常适合于具有复杂、非线性的决策边界的分类任务。RBF核的主要优点是不需要预先知道数据的分布情况，但它的性能很大程度上依赖于参数  $\gamma$  的选择。

### 4. sigmoid核 $K(a, b) = \tanh(\gamma a^T b + r)$ :

- 应用场景：sigmoid核模仿了神经网络的激活函数，可以在某些情况下模拟多层感知器的效果。它可以用于解决非线性分类问题，但在实践中不如RBF核常用。sigmoid核同样依赖于参数  $\gamma$  和  $r$  的调优。



总结，先对数据进行分析，线性可分用线性核，反之用RBF核。

## 代码介绍与案例

### 主要使用的函数

#### sklearn.svm中的SVC类:

```
sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='scale',  
                 coef0=0.0, probability=False, ...)
```

##### • 主要参数:

- C: 正则化参数。float型正数，默认1.0，一般取 $10^t$ ，如1.0、0.1、10等。高C旨在正确分类所有训练样本；低C使决策表面平滑，模型更关注间隔最大化。
- kernel: 设置算法用的核函数。字符串，默认'rbf'。可取'linear', 'poly', 'rbf', 'sigmoid'。
- gamma: 核('rbf', 'poly', 'sigmoid')的系数 $\gamma$ 。float型，默认'scale'=1/(n\_features\*X.var()); 若'auto', 用1/n\_features。gamma越大，模型越复杂。
- coef0: 核函数中的独立项。它仅在“poly”和“sigmoid”中有意义。

##### • 主要方法:

- decision\_function(X): 计算X中样本的决策函数值。
- fit(X\_train, y\_train)、predict(X\_new)、score(X\_test, y\_test)、predict\_proba(X\_new)

##### • 主要属性:

- support\_vectors\_: 支持向量。
- coef\_: kernel="linear"时分配给特征的权重。
- intercept\_: 决策函数中的独立项b。



sklearn.model\_selection的GridSearchCV元评估器，自动寻找验证得分最大的最优超参数和模型。

**GridSearchCV**(*estimator*, *param\_grid*, *scoring*=None, *cv*=None,...)

穷尽搜索指定估计器的超参数组合，以优化超参数。

- 四样
- **主要参数:**
    - *estimator*: 待优化超参数的估计器对象，如 SVC 分类器。
    - *param\_grid*: 为搜索提供空间，是字典或字典列表。字典键为超参数名、值是参数可取值的列表。
    - *scoring*: 评分参数，默认None，这时使用估计器默认的性能评价指标。分类常用 ‘accuracy’。
    - *cv*: int 型或交叉验证生成器。设置交叉验证拆分策略。默认为None，取5折。
  - **主要方法:**
    - *fit*(X\_train, y\_train): 运行所有参数集来拟合数据。
    - *predict*(X\_new): 用找到的最佳参数配置评估器，以预测新数据。
    - *score*(X\_test, y\_test): 计算评估器在给定数据上的得分。
  - **主要属性:**
    - *best\_estimator\_*: 通过搜索选出的评估器，即在验证集上得分最高的评估器。
    - *best\_params\_*: 在验证集上得分最高的参数组合。
    - *best\_score\_*: *best\_estimator\_* 平均交叉验证分数

估计器的超参数名查找:  
`estimator.get_params()`

## 案例：乳腺癌分类

```
1 from sklearn.datasets import load_breast_cancer
2 from sklearn.model_selection import train_test_split
3 from sklearn.svm import LinearSVC
4
5 # 加载乳腺癌数据集
6 cancer = load_breast_cancer()
7 X = cancer.data
8 y = cancer.target
9
10 # 拆分数据集为75%训练集和25%测试集
11 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
12                                                    random_state=42)
13
14 # 初始化LinearSVC分类器
15 lin_svc = LinearSVC(C=1, loss="hinge", max_iter=1000, random_state=42)
16
17 # 训练模型
18 lin_svc.fit(X_train, y_train)
19
20 # 输出训练集得分
21 train_score = lin_svc.score(X_train, y_train)
22 print("训练集得分:", train_score)
23
24 # 输出测试集得分
25 test_score = lin_svc.score(X_test, y_test)
26 print("测试集得分:", test_score)
```

输出：

```
1 训练集得分：0.9225352112676056
2 测试集得分：0.9370629370629371
3 d:\py\Anaconda3\envs\pytorch_gpu\lib\site-packages\sklearn\svm\_base.py:985:
  ConvergenceWarning: Liblinear failed to converge, increase the number of
    iterations.
4 warnings.warn("Liblinear failed to converge, increase "
```



显示模型无法收敛，可能存在模型特征尺度差异很大，特征相关性太强，数据噪声太多，这里不存在模型太复杂或者迭代次数不够的问题)

这里加入make\_pipeline模块，对特征进行缩放（标准化处理）就可以。

```
1 lin_svc2 = make_pipeline(MinMaxScaler(), LinearSVC(C=1, loss='hinge',
  max_iter=10000, random_state=42))
2 lin_svc2.fit(X_train, y_train)
```

我们继续优化，选用不同的核以及不同的超参数进行优化。

```
1 from sklearn.pipeline import make_pipeline
2 from sklearn.preprocessing import MinMaxScaler
3 from sklearn.model_selection import cross_val_score
4 from sklearn.svm import SVC
5 rbf_svc2=make_pipeline(MinMaxScaler(),
6 SVC(kernel='rbf',C=100,gamma=0.1,random_state=42))
7 rbf_svc2.fit(X_train,y_train) #模型训练
8 print("交叉验证得分：
  {:.4f}".format(cross_val_score(rbf_svc2,X_train,y_train).mean()))
```

类	时间复杂度	支持核外训练否	需要缩放	核技巧
LinearSVC	$O(m \times n)$	否	是	否
SGDClassifier	$O(m \times n)$	是	是	否
SVC	$O(m^2 \times n)$ 到 $O(m^3 \times n)$	否	是	是

Scikit-learn提供了搜索最优超参数的函数：

## — GridSearchCV()

**网格搜索**交叉验证，穷举所有组合。

— **RandomizedSearchCV()** **随机搜索**交叉验证，在搜索空间随机采样，常用于超参数取值空间维度很高的情况。

我们这里使用网格搜索找到最优超参数：

```
1 from sklearn.svm import SVC
2 from sklearn.model_selection import GridSearchCV
3 from sklearn.metrics import classification_report
4 # 设置交叉验证参数
5 tuned_parameters = [
6 {"kernel": ["rbf"], "gamma": [1,0.1,1e-2,1e-3, 1e-4], "C": [1, 10, 100,
7 1000,10_000]},
8 {"kernel": ["linear"], "C": [0.1, 1, 10, 100, 1000]},
9 {"kernel": ["poly"], "gamma": [10,1,0.1,1e-2,1e-3,1e-4], "C": [1e-4,1e-3,1e-
10 2,0.1,1,10],
11 "degree": [2,3,4,5,6]}]
12 clf = GridSearchCV(SVC(), tuned_parameters)
13 clf.fit(X_train_scaled, y_train)
14 print(f"最优超参数组合:{clf.best_params_}")
15 print(f"最优模型交叉验证得分: {clf.best_score_:.4f}")
```

```
1 最优超参数组合: {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
2 最优模型交叉验证得分: 0.9789
```


打印分类报告：

```
1 best_clf = clf.best_estimator_
2 best_clf.fit(X_train_scaled, y_train)
3 print(f'最优svc的测试得分: {best_clf.score(X_test_scaled, y_test):.4f}')
4 print("\n详细的分类报告:\n")
5 y_true, y_pred = y_test, best_clf.predict(X_test_scaled)
6 print(classification_report(y_true, y_pred))
```

```
1 最优svc的测试得分: 0.9720
2 详细的分类报告:
```

3					
4		precision	recall	f1-score	support
5					
6	0	0.96	0.96	0.96	54
7	1	0.98	0.98	0.98	89
8					
9	accuracy			0.97	143
10	macro avg	0.97	0.97	0.97	143
11	weighted avg	0.97	0.97	0.97	143

## 案例：手写数字识别

 在机器学习中，通常不建议在拆分数据集之前进行缩放，因为这样可能会导致数据泄漏（data leakage）。数据泄漏是指在模型训练过程中使用了测试集的信息，从而导致模型的评估结果不准确。

### 原因

1. 数据泄漏：如果在拆分之前对整个数据集进行缩放，测试集的信息会影响训练集的缩放参数（如最小值和最大值），这会导致模型在测试集上的表现过于乐观。

真实场景：在实际应用中，您通常会在模型部署后接收到新的数据。在这种情况下，您只会对新数据进行缩放，而不会使用整个数据集的信息。

### 正确的做法

正确的做法是先拆分数据集，然后对训练集进行缩放，最后使用相同的缩放参数对测试集进行转换。

```

1 from sklearn.datasets import load_breast_cancer
2 from sklearn.model_selection import train_test_split
3 from sklearn.preprocessing import MinMaxScaler
4
5 # 加载乳腺癌数据集
6 cancer = load_breast_cancer()
7 X = cancer.data
8 y = cancer.target
9
10 # 拆分数据集为75%训练集和25%测试集
11 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
12                                                    random_state=42)
13 # 创建MinMaxScaler实例，设置缩放范围为[-1, 1]
```

```

14 scaler = MinMaxScaler(feature_range=(-1, 1))
15
16 # 拟合并转换训练集
17 X_train_scaled = scaler.fit_transform(X_train)
18
19 # 仅转换测试集
20 X_test_scaled = scaler.transform(X_test)

```

## 使用随机搜索以对超平面进行搜索

```

1 from sklearn.svm import SVC
2 from sklearn.model_selection import RandomizedSearchCV
3 from sklearn.utils.fixes import loguniform
4
5 clf = SVC(probability = True, random_state=1)
6 search_dict = {'kernel': ['linear', 'rbf'],
7               'C': loguniform(1, 1000), 'gamma': loguniform(0.0001, 0.1)}
8 random_search = RandomizedSearchCV(estimator = clf, param_distributions =
9                                   search_dict, scoring='accuracy', cv=5)
10 random_search.fit(X_train, y_train)
11
12 print('最优得分: %0.3f' % random_search.best_score_)
13 print("最优超参数组合:", random_search.best_params_)
14 print('测试得分: %0.3f' % random_search.score(X_test, y_test))

```

## 输出:

```

1 最优得分: 0.960
2 最优超参数组合: {'C': 34.7622250644291, 'gamma': 0.031433819838160494, 'kernel':
   'linear'}
3 测试得分: 0.944

```

## 再输出分类报告:

```

1 from sklearn.metrics import classification_report
2 best_clf = random_search.best_estimator_
3 best_clf.fit(X_train, y_train)
4 print(f'最优svc的测试得分: {best_clf.score(X_test, y_test):.4f}')
5 print("\n详细的分类报告:\n")
6
7 y_true, y_pred = y_test, best_clf.predict(X_test)

```



```
8 print(classification_report(y_true, y_pred))
```

## SVM优缺点和参数

SVM	<ul style="list-style-type: none"><li>• 最大间隔准则，以防止过拟合，提高泛化性能</li><li>• 核函数技术，以避免因变换导致的计算困难</li></ul>
优点	<ul style="list-style-type: none"><li>• 线性/非线性分类、回归都可，特别适用于中小型复杂数据集（高维度）。</li><li>• 依赖的支持向量比较少，消耗内存少。</li><li>• 预测速度快</li><li>• 与核方法配合及其通用性，使其在各种数据集（低维数据和高维数据、线性和非线性）上的表现都很好。</li></ul>
缺点	<ul style="list-style-type: none"><li>• 对样本规模的缩放表现不好。当样本数高达10万甚至更大时，计算资源面临挑战。</li><li>• SVM的另一个缺点是需要预处理数据，且调参需要十分小心。</li></ul>
参数	<ul style="list-style-type: none"><li>• <b>gamma</b>：核函数系数，控制决策边界的复杂程度。gamma越大，模型越复杂。</li><li>• <b>C</b>：对分类错误的惩罚。C控制每个数据点对模型的影响程度。C越大，对错分的惩罚越大，使得模型尽可能对样本做出正确分类。C和gamma的设定是强相关的，应该同时调节。</li><li>• <b>Kernel</b>：核函数选择。</li></ul>
预处理	<ul style="list-style-type: none"><li>• SVM对尺度变化敏感，各特征取值范围需大致相同。</li></ul>

## SVR回归

### 原理


给定训练数据集 $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ ,  $y^{(i)} \in \mathbb{R}$ ,  $i = 1, \dots, m$ 。

### $\epsilon$ -SVR原问题的目标：

$$\begin{aligned} \min_{w, b, \xi, \xi^*} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^m (\xi^{(i)} + \xi^{*(i)}) \\ \text{s.t.} \quad & y^{(i)} - (\mathbf{w}^T \phi(x^{(i)}) + b) \leq \epsilon + \xi^{(i)}, \\ & (\mathbf{w}^T \phi(x^{(i)}) + b) - y^{(i)} \leq \epsilon + \xi^{*(i)}, \\ & \xi^{(i)}, \xi^{*(i)} \geq 0, \quad i = 1, 2, \dots, m. \end{aligned}$$

也通过解决对偶问题，  
利用核技巧来预测。

惩罚那些预测值  $\mathbf{w}^T \phi(x^{(i)}) + b$  至少有  $\epsilon$  远离他们真实值  $y^{(i)}$  的样本。  
这些样本通过  $\xi^{(i)}$  或者  $\xi^{*(i)}$  惩罚目标。

 任务：在加州住房数据集上训练并微调一个SVM回归器，来预测加州地区房价。测试多个超参数组合，以找出最优超参数及其对应模型，输出最优模型的RMSE。

## 代码

### 导入房价数据，并且划分数据集

```
1 from sklearn.datasets import fetch_california_housing
2 housing = fetch_california_housing(as_frame=True)
3 X = housing.data
4 y = housing.target
5 from sklearn.model_selection import train_test_split
6 X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)
```

### 使用SVR，进行随机搜索并且进行交叉验证

```
1 from sklearn.svm import SVR
2 from sklearn.model_selection import RandomizedSearchCV
3 from sklearn.utils.fixes import loguniform
4 from sklearn.pipeline import make_pipeline
5 from sklearn.preprocessing import StandardScaler
6 pipe = make_pipeline(StandardScaler(), SVR())
7 param_distrib = {'svr__kernel': ['linear', 'rbf'], "svr__gamma":
    loguniform(0.0001, 0.1), "svr__C": loguniform(1, 1000) }
8 rnd_search_cv = RandomizedSearchCV(estimator =
    pipe, param_distributions=param_distrib, n_iter=10, cv=3, random_state=42)
9 rnd_search_cv.fit(X_train[:2000], y_train[:2000])
```

### 输出训练参数和MSE

```
1 from sklearn.model_selection import cross_val_score
2 from sklearn.metrics import mean_squared_error
3 best_svr = rnd_search_cv.best_estimator_
4 print('最优超参数', rnd_search_cv.best_params_)
5 print(f"在训练集上的交叉验证分数(R^2): {rnd_search_cv.best_score_:.4f}")
6 print(f"在测试集上的得分(R^2): {best_svr.score(X_test, y_test):.4f}")
7 train_rmse = -cross_val_score(best_svr, X_train, y_train,
8     scoring="neg_root_mean_squared_error").mean()
9 print(f"在训练集上的rmse: {train_rmse:.4f}")
10 y_pred = best_svr.predict(X_test)
```

```
11 test_rmse = mean_squared_error(y_test, y_pred, squared=False)
12 print(f"在测试集上的rmse:{test_rmse:.4f}")
```

## 保存模型并且进行预测

```
1 import joblib
2 joblib.dump(best_svr, 'california_housing_svm.pkl')
3 model=joblib.load('california_housing_svm.pkl')
4 Ypred=model.predict(X_test[0:4]);
5 print(Ypred)
```

## SVM标准分类流程

### 加载wine数据集：

```
1 wine = load_wine()
2 X, y = wine.data, wine.target
```

### 划分数数据集为训练集和测试集：

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
    random_state=42)
```

### 定义SVC分类器：

```
1 svc = SVC(probability=True, random_state=42)
```

### 定义超参网格：

```
1 param_grid = {
2     'svc__C': [0.1, 1, 10, 100],      # 正则化参数
3     'svc__gamma': [0.001, 0.01, 0.1, 1], # Gamma参数
4     'svc__kernel': ['rbf', 'linear']    # SVM核
5 }
```

## 创建一个管道，包括特征缩放和SVC：

```
1 pipeline = Pipeline([
2     ('scaler', StandardScaler()),
3     ('svc', svc)
4 ])
```

## 网格搜索找到最优超参：

```
1 grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy')
2 grid_search.fit(X_train, y_train)
```

## 使用grid\_search中找到的最优参数来训练模型：

```
1 best_svc_clf = grid_search.best_estimator_
```

## 使用最优分类器预测测试集：

```
1 y_pred = best_svc_clf.predict(X_test)    # 返回预测的类别标签。
2 y_pred_proba = best_svc_clf.predict_proba(X_test) # 返回预测的概率值。
```

## 输出性能以及报告

```
1 # 计算测试准确率
2 test_accuracy = accuracy_score(y_test, y_pred)
3
4 # 为了计算ROC AUC值，我们需要将标签二值化
5 y_test_bin = label_binarize(y_test, classes=[0, 1, 2])
6
7 # 计算ROC AUC值
8 roc_auc = roc_auc_score(y_test_bin, y_pred_proba, multi_class='ovr')
9
10 # 输出分类性能报告
11 report = classification_report(y_test, y_pred, target_names=wine.target_names)
12
13 print(classification_report(y_test, y_pred))
14 print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))
15 print("ROC AUC: {:.2f}%".format(roc_auc * 100))
```

