第12讲 支持向量机

- □ SVM简介
- □ 线性-SVM
- □ 非线性-SVM
- □ 模型评估与优化

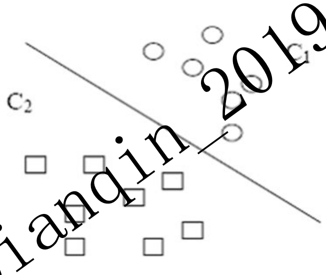
支持向量机-SVM

- SVM(Support Vector Machine):
  Cortes和Vapnik于1995年提出
  一种基于分类边界的二分类模型
- 线性可分-SVM
- 非线性可分-SVM

线性可分-SVM

- 例如, 对于二维数据的二分类，利用线性函数：

$$g(x) = wx + b$$



- 设阈值为0，当样本x_i需判别时，计算$g(x_i)$值：$g(x_i) > 0$，则判别为类别C_1；$g(x_i) < 0$，则判别为类别C_2；$g(x_i) = 0$，可拒绝判断 。

  等价于给函数g(x)附加一个符号函数*sign*()：$f(x) = sign(g(x))$

线性可分-SVM

- 多维线性函数：$g(x) = wx+b$

  x为样本特征向量，如样本点(3,8)，则x=(3,8)

  w为系数向量

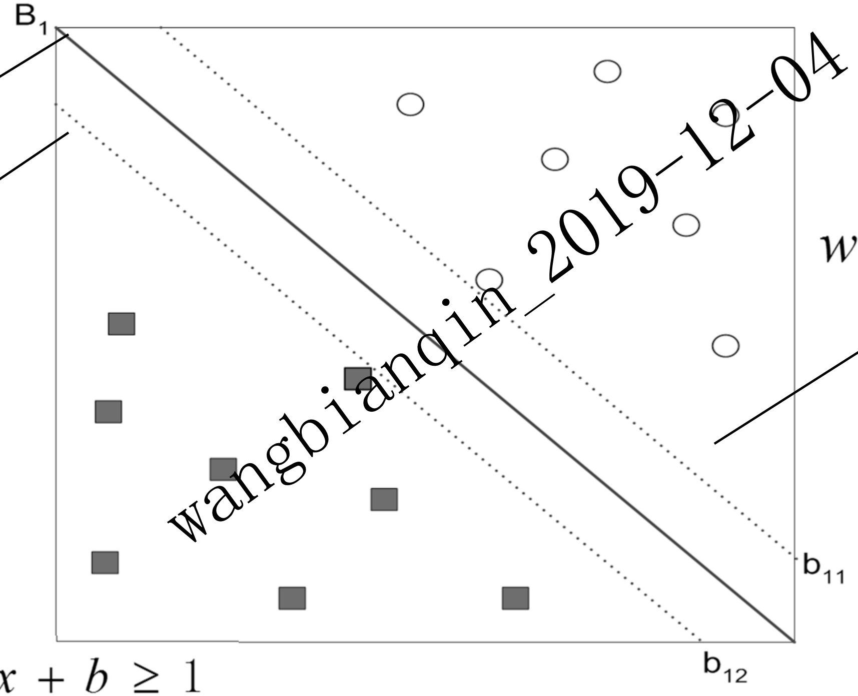
- 如何求得最优g(x)？

  即$g(x) = wx + b$中的w和b如何确定？

线性可分-SVM

$$w \bullet x + b = 0$$

$$w \bullet x + b = -1$$



$$w \bullet x + b = +1$$

$$\text{margin} = \frac{2}{\|w\|}$$

$$f(x) = \begin{cases} 1 & \text{if } w.x + b \geq 1 \\ -1 & \text{if } w.x + b \leq -1 \end{cases}$$

线性可分-SVM

- **||w||称为向量w的范数，是对向量长度的一种度量**
  范数最一般的表示形式为p-范数
  向量w = (w_1, w_2, w_3,…… w_n)，它的p-范数：

$$\|w\| = \sqrt[p]{w_1^p + w_2^p + w_3^p + \ldots + w_n^p}$$

常用2-范数，当不指明*p*时，意味着不关心*p*的值，用几范数都可以

线性可分-SVM

- 支持向量(support vector)：距离超平面距离最近的样本点，唯一决定了超平面

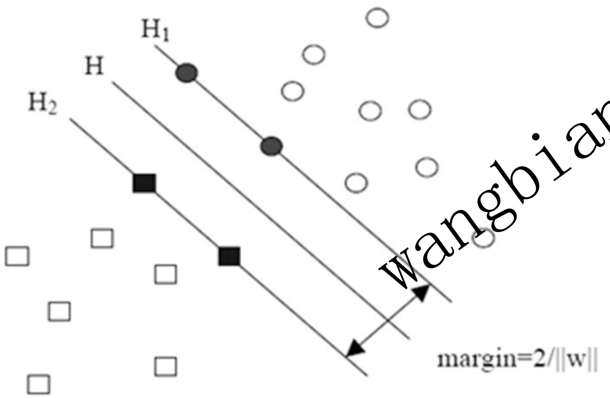


图2 线性可分情况下的最优分类线

线性可分-SVM

- 最大化：

$$\text{margin} = \frac{2}{\|w\|}$$

- 转化为：

$$\min \frac{1}{2}\|w\|$$

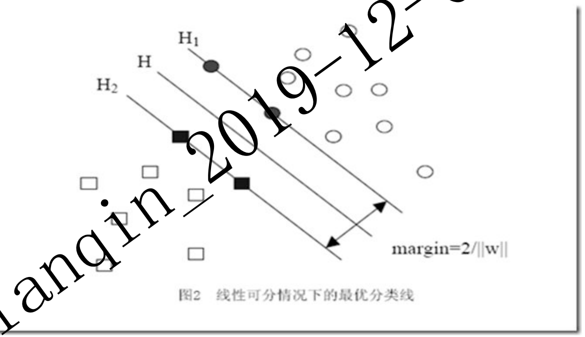


图2 线性可分情况下的最优分类线

- 当 $\|w\| = 0$ 时，目标函数获得最小值，反映在图中，则H1与H2两条直线间的距离无限大，所有样本进入无法分类的区域
- 解决方法：加一个约束条件

条件最优问题

- 规定距离超平面间隔最小的点的间隔定为1，则其它点间隔都不会小于1，则有不等式：
  $y_i(w\cdot x_i+b) \geq 1$ (i=1,2,…,n)， **$y_i \in$ {-1,1}**表示x_i的类标签

- 目标函数转化成求条件最优问题：

$$\min \frac{1}{2}\|w\|^2,$$

$$s.t.\ \ y_i(w\cdot x_i+b)-1\geq 0,\ i=1,\cdots,n$$

s.t. = subject to

- 一个凸优化问题，即约束优化，理论上存在全局最优解
- 解法：拉格朗日乘子方法（**lagrange Multiplier**）和 **KKT**约束条件

最优化问题

- 无约束最优化：***min f(x)***
- 等式约束最优化

$$
\begin{aligned}
&\min f(x) \\
&s.t.\quad h_i(x)=0 \quad i=1,2,\ldots,n
\end{aligned}
\qquad\qquad
\begin{cases}
\min\ f(x)=x_1+x_2 \\
s.t.\quad h(x)=x_1^2+x_2^2\text{-}2
\end{cases}
$$

引入拉格朗日（**Lagrange**）函数***F(x)*:**

$$F(x,\lambda)=f(x)+\sum_{i=1}^{n}\lambda_i h_i(x)$$

其中λ_i是各个约束条件的待定系数

解变量的偏导方程：$\dfrac{\partial F}{\partial x_i}=0,\ldots,\dfrac{\partial F}{\partial \lambda_i}=0$

- 不等式约束最优化：**KKT**约束条件

拉格朗日函数

- 拉格朗日函数：新目标函数（改写目标函数，考虑施加在解上的约束）

$$\mathcal{L}(w,b,\alpha)=\frac{1}{2}\|w\|^2-\sum_{i=1}^{n}\alpha_i\left(y_i(w^Tx_i+b)-1\right)$$

参数a_i称为拉格朗日乘子，拉格朗日函数的第1项与原目标函数相同，第2项则捕获了不等式约束。

- 最小化拉格朗日函数，须对此函数关于**w**和**b**求偏导，并令它们等于零：

$$\frac{\partial L}{\partial w}=0\Rightarrow w=\sum_{i=1}^{n}a_iy_ix_i$$

$$\frac{\partial L}{\partial b}=0\Rightarrow \sum_{i=1}^{n}a_iy_i=0$$

对偶拉格朗日函数

- 代入 L(w,b,a):

$$
\begin{aligned}
\mathcal{L}(w,b,\alpha) &= \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j - \sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j - b\sum_{i=1}^{n}\alpha_i y_i + \sum_{i=1}^{n}\alpha_i \\
&= \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j
\end{aligned}
$$

- 对偶拉格朗日函数：仅包含拉格朗日乘子

$$
\begin{aligned}
&\max_{\alpha} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j \\
&s.t.,\ \alpha_i \geq 0, i = 1,\ldots,n \\
&\qquad \sum_{i=1}^{n}\alpha_i y_i = 0
\end{aligned}
$$

拉格朗日乘子约束，称作**Karuch-Kuhn-Tucher(KTT)**约束

- 由凸二次规划的性质能保证这样最优的向量*a*是存在的

对偶最优化问题

- 对偶最优化：由求极大转换为求极小

$$\min_{a} \ \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^{n}\alpha_i$$

$$s.t. \quad \sum_{i=1}^{n}\alpha_i y_i = 0,$$

$$\alpha_i \geq 0,\ i{=}1,\ldots,n$$

解得 $\alpha^* = (\alpha_1, \ldots, \alpha_l)^T$

$$w^* = \sum_{i=1}^{n} y_i \alpha_i^* x_i,$$

$$b^* = y_j - \sum_{i=1}^{n} y_i \alpha_i^* (x_i \cdot x_j), \quad \forall j \in \{j \mid \alpha_j^* > 0\}$$

线性可分-SVM

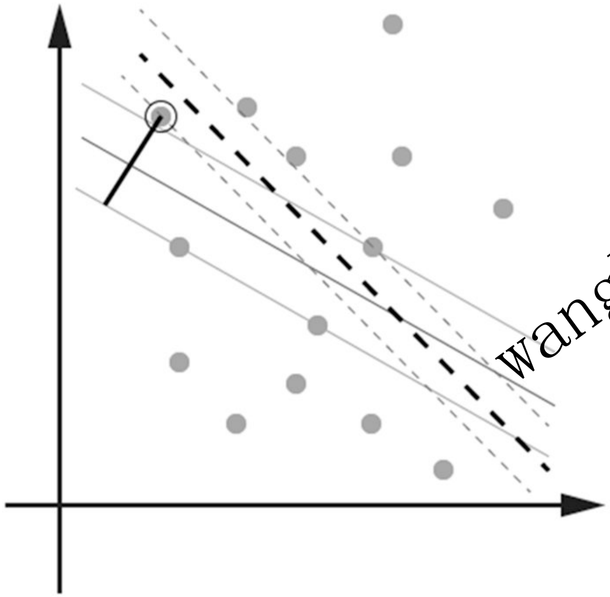
- 分类决策函数：

$$f(x) = sign\{w^* \cdot x + b^*\} = sign\{\sum_{i=1}^{n} \alpha_i^* y_i (x_i \cdot x) + b^*\}$$

只包含待分类样本与训练样本中的支持向量的内积运算

线性可分-SVM

- 个别样本的不容错性，使原本线性可分问题变成线性不可分



线性可分-SVM

- 当最优分类面无法完全分开两类点时，引入松弛因子$\varsigma_i(\varsigma_i \ge 0)$，允许错分样本存在，即近似线性可分

- 分类界面： $w \cdot x + b = 0$

  约束条件弱化：

$$y_i(w \cdot x_i) + b) \ge 1 - \varsigma_i \ , \quad i = 1, \cdots, n$$

$$\varsigma_i \ge 0, \quad i = 1, \cdots, n$$

线性可分-SVM

- 目标函数加入惩罚因子，原优化问题变为：

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\varsigma_i$$

$$s.t. \quad y_i(w\cdot x_i)+b) \geq 1-\varsigma_i\ ,\ \ \varsigma_i \geq 0,\ \ i=1,\cdots,n$$

- 对偶问题：

$$\max_{a} \ \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j (x_i\cdot x_j)$$

$$s.t. \quad \sum_{i=1}^{n}\alpha_i y_i = 0,$$

$$0 \leq \alpha_i \leq C,\ i=1,\ldots,n$$

线性可分-SVM

- 线性支持向量机原始最优化问题:

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i$$

$$\text{s.t.} \quad y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad i = 1,2,\cdots,N$$

$$\xi_i \geq 0, \quad i = 1,2,\cdots,N$$

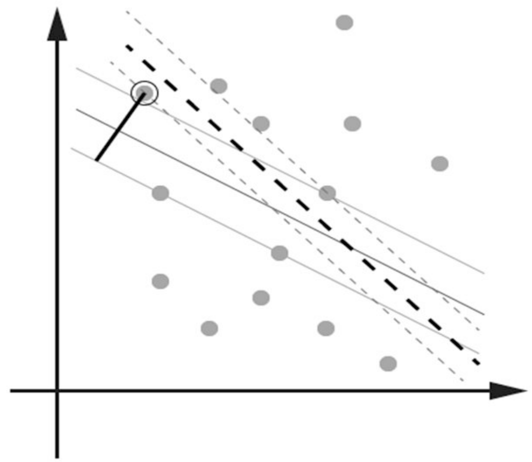
$$\Updownarrow$$

- 等价于:

$$\min_{w,b} \quad \sum_{i=1}^{N}[1 - y_i(w \cdot x_i + b)]_+ + \lambda\|w\|^2$$

线性可分-SVM

- 对线性可分的二值分类数据集，通过引入跨分割线的数据点的损失函数。对于n个数据点，引入**soft margin**损失函数：



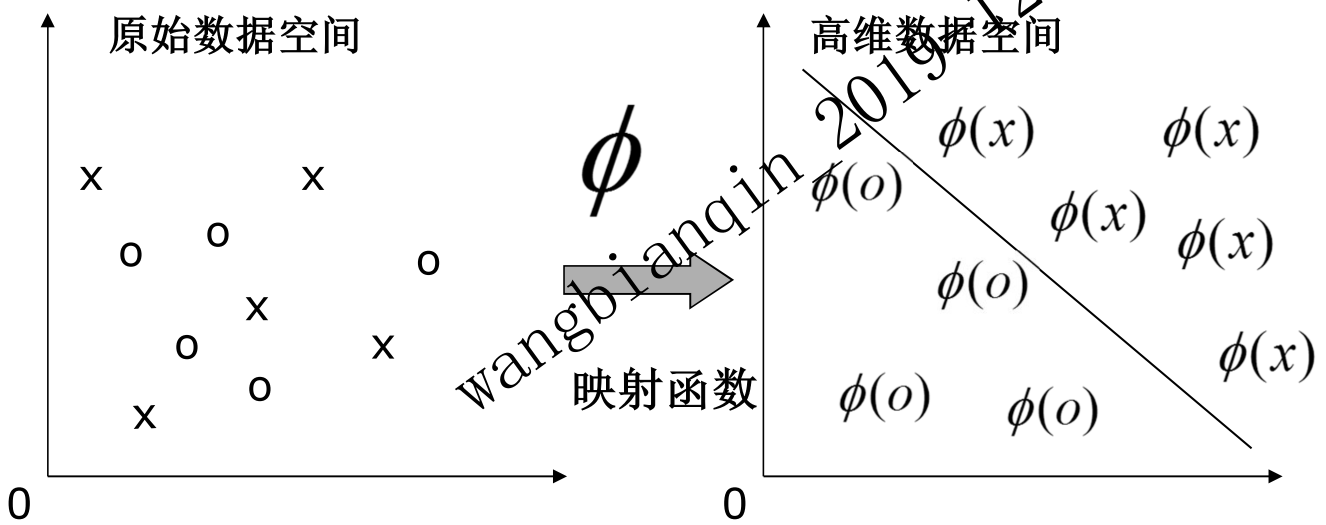
$$\frac{1}{N}\sum_{i=1}^{N}\max(0,1-y_i(wx_i-b))+\lambda\|w\|^2$$

注意：如果数据点分割正确，乘积$y_i(wx_i-b)$总是大于**1**

意味着损失函数左边项等于零，这时对损失函数有影响的仅仅只有间隔大小

非线性-SVM

- 假设存在一个适合的函数ϕ ,变换给定的数据集



- 变换后的空间中，线性决策边界的形式： $w \cdot \phi(x) + b = 0$

非线性-SVM

- 目标函数： $\min \frac{1}{2}\|w\|^2,$

$$s.t.\ \ y_i((w\cdot\phi(x_i)+b)-1)\geq 0,\ i=1,\cdots,n$$

- 对偶问题： $\max_{a}\ \sum_{i=1}^{n}\alpha_i-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{\mathrm{n}}\alpha_i\alpha_j y_i y_j(\phi(x_i)\cdot\phi(x_j))$

$$s.t.\quad \sum_{i=1}^{n}\alpha_i y_i=0$$

$$0\leq\alpha_i\leq C,\ i=1,\ldots,\mathrm{n}$$

- 分类决策函数：

$$f(x)=sign\{w^*\cdot x+b^*\}=sign\{\sum_{i=1}^{n}\alpha_i^* y(\phi(x_i)\cdot\phi(x))+b^*\}$$

数据被映射到高维空间，$\phi(x_i)\cdot\phi(x_j)$计算量比 $x_i\cdot x_j$ 大很多

非线性-SVM

- 核函数(kernel function)：

$$K(x, z) = \phi(x) \cdot \phi(z)$$

函数 $\boldsymbol{K(x, z)}$ 称为点积核函数

- 核函数作用：接受两个低维空间向量，计算出经过某个变换后在高维空间的向量内积。使计算量回归到 $x_i \cdot x_j$ 的量级

非线性-SVM

□ 常用核函数

线性核 $K(x_i, x_j) = x_i \cdot x_j$

多项式核 $K(x_i, x_j) = (x_i \cdot x_j + 1)^d$，$d$为多项式的次数

高斯核 $K(x_i, x_j) = \exp\left(\dfrac{\|x_i - x_j\|^2}{2\sigma^2}\right)$，$\sigma$为高斯核的带宽

拉普拉斯核 $K(x_i, x_j) = \exp\left(\dfrac{\|x_i - x_j\|}{\sigma}\right)$，$\sigma > 0$

双曲正切核 $K(x_i, x_j) = \tanh(\beta x_i^T x_j + \theta)$，双曲正切函数，$\beta>0$，$\theta < 0$

非线性-SVM

- 对偶问题：

$$\max_{a} \ \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{\mathrm{n}}\alpha_i\alpha_j y_i y_j K(x_i, x_j)$$

$$s.t. \quad \sum_{i=1}^{n}\alpha_i y_i = 0,$$

$$0 \le \alpha_i \le C,\ i{=}1,\ldots,\mathrm{n}$$

- 分类决策函数：

$$f(x) = sign\{w^{*}\cdot x + b^{*}\} = sign\{\sum\nolimits_{i=1}^{n}\alpha_i^{*} y_i K(x_i, x) + b^{*}\}$$

SVM实现--sklearn.SVM

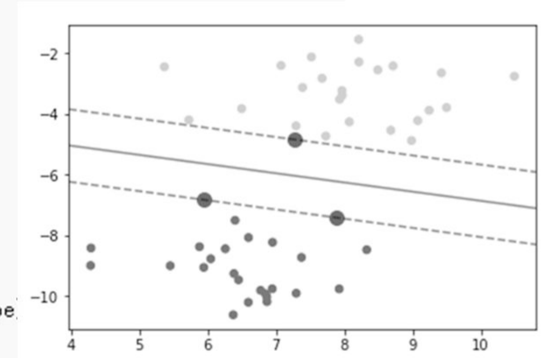
- 分类：SVC, NuSVC, LinearSVC, OneClassSVM
- 回归：SVR, NuSVR, LinearSVR
- SVM实现线性：
  - LinearSVC
  - SVC中的kernel设置为linear
- 实现多分类：SVC, NuSVC, LinearSVC
  - “one-against-one”和“one-vs-the-rest”多类策略
- SVM参数：参数C和高斯核参数gamma

➢ 如何寻找参数的最佳组合？

线性核-SVM

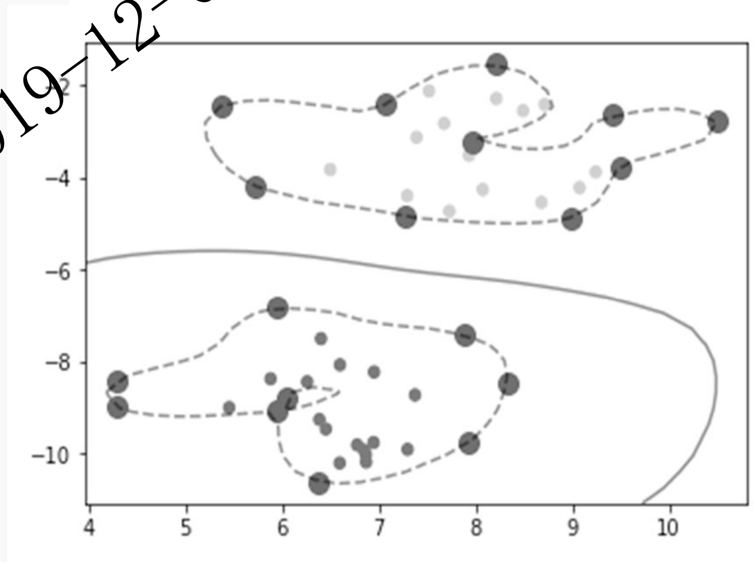
```python
# 导入 numpy
import numpy as np
# 导入绘图工具
import matplotlib.pyplot as plt
%matplotlib inline
# 导入支持向量机SVM
from sklearn import svm
# 导入数据集生成工具
from sklearn.datasets import make_blobs

# 创建50个数据点，并分成两类
X, y = make_blobs(n_samples=50, centers=2, random_state=6)

# 创建一个线性内核的支持向量机模型
clf = svm.SVC(kernel='linear', C=1000)
clf.fit(X, y)

# 可视化数据点
plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)
```

```python
# 建立图像坐标
ax = plt.gca() # get the current polar axes on the current figure
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# 生成两个等差数列
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)

YY, XX = np.meshgrid(yy, xx) # 生成坐标矩阵

xy = np.vstack([XX.ravel(), YY.ravel()]).T

Z = clf.decision_function(xy).reshape(XX.shape

# 绘制分类决定边界
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])

ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
           linewidth=1,c='g',facecolors='none')
```



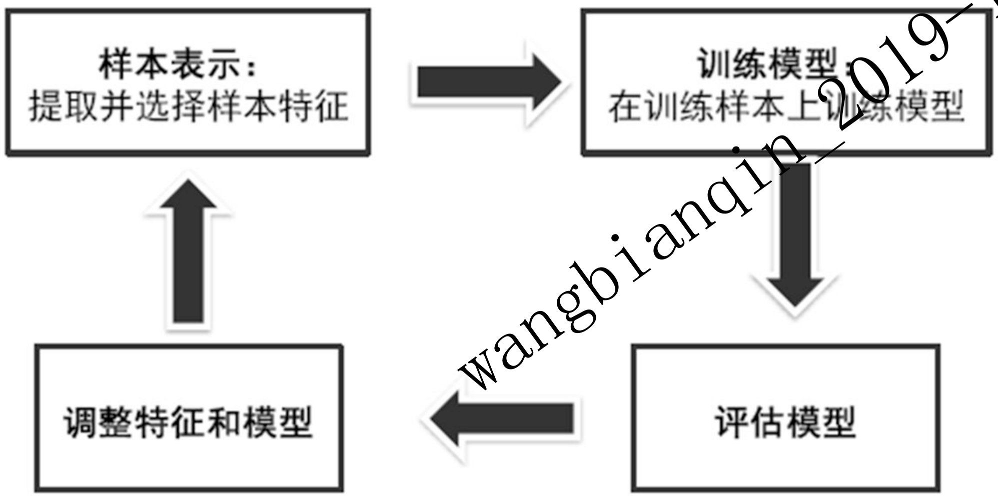
高斯核-SVM

```
# 创建一个RBF内核的支持向量机模型
clf_rbf = svm.SVC(kernel='rbf', C=1000)
clf_rbf.fit(X, y)

# 数据点散点图
plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)

# 建立图像坐标
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# 生成两个等差数列
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)

YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T

Z = clf_rbf.decision_function(xy).reshape(XX.shape)

# 绘制分类决定边界
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])

ax.scatter(clf_rbf.support_vectors_[:, 0], clf_rbf.support_vectors_[:, 1], c='g', s=100,
           linewidth=1, facecolors='none')
```



模型评估与优化

建模过程



交叉验证法

- 留出法：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

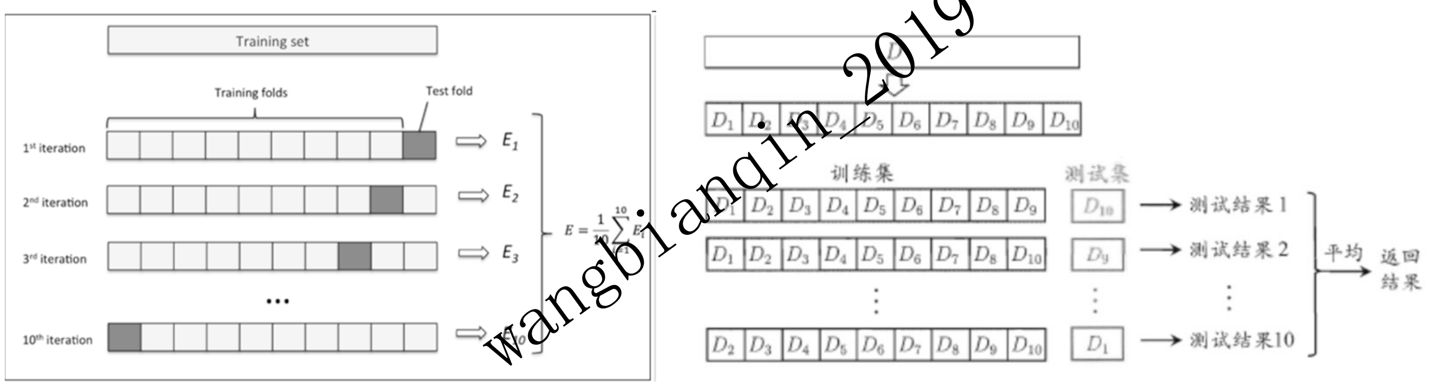
$$D = S \cup T,\ S \cap T = \Phi$$

在*S*上训练模型，用*T*评估测试误差，作为对泛化误差的评估

通常，2/3数据做训练集，其余1/3数据做测试集

交叉验证法

- *S*折交叉验证（S-fold cross validation）：



10折验证示意图

from sklearn.model_selection import cross_val_score

交叉验证法

```python
# scikit-learn的交叉验证法

# 导入红酒数据集
from sklearn.datasets import load_wine
wine = load_wine()

# 导入SVC模型
from sklearn.svm import SVC
# 设置SVC的核函数为线性核
svc = SVC(kernel='linear')

# 导入交叉验证工具
from sklearn.model_selection import cross_val_score
# 默认, cv = 3
scores = cross_val_score(svc, wine.data, wine.target)

# 显示结果
print('交叉验证得分: {}'.format(scores))
print('交叉验证平均分: {:.3f}'.format(scores.mean()))
```

```
交叉验证得分: [0.83333333 0.95       1.        ]
交叉验证平均分: 0.928
```

随机拆分交叉验证

```
# 随机拆分交叉验证（shuffle-split cross-validation）：先从数据集中随机抽一部分数据集作为训练集，
# 再从其余的部分随机抽一部分作为测试集，进行评分后再迭代，重复上一步的动作，直到把希望迭代的次数全部跑完

# 导入随机拆分工具
from sklearn.model_selection import ShuffleSplit
# 设置拆分的份数为10个
shuffle_split = ShuffleSplit(test_size=.2, train_size=.7,n_splits = 10)

# 对拆分好的数据集进行交叉验证
scores = cross_val_score(svc, wine.data, wine.target, cv=shuffle_split)

print('随机拆分交叉验证模型得分：\n{}'.format(scores))
```

```
随机拆分交叉验证模型得分：
[0.94444444 0.83333333 0.94444444 0.94444444 0.91666667 0.86111111
 0.97222222 0.97222222 0.97222222 0.94444444]
```

留一法交叉验证

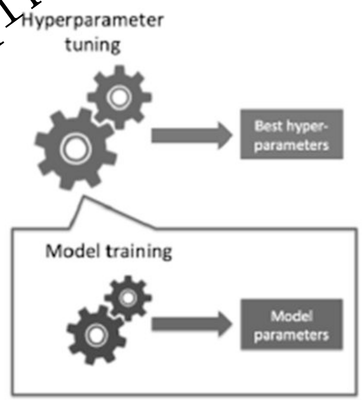
```python
# 留一法交叉验证（Leave One Out cross-validation）：它其实有点像k 折交叉验证，
# 不同的是，它把每一个数据点都当成一个测试集，数据集里有多少样本，它就要迭代多少次

# 导入留一法LeaveOneOut
from sklearn.model_selection import LeaveOneOut
# 设置cv参数为LeaveOneOut()
cv = LeaveOneOut()

# 对拆分好的数据集进行交叉验证
scores = cross_val_score(svc, wine.data, wine.target, cv=cv)

print('迭代次数:{}'.format(len(scores)))
print("模型平均分: {:.3f}".format(scores.mean()))
```

```
迭代次数:178
模型平均分: 0.955
```

模型参数调整

参数调整

- 模型参数包括两种
  1. 模型自身参数，通过样本学习得到的参数。如：逻辑回归及神经网络中的权重及偏置的学习等
  2. 超参数，模型框架的参数，如kmeans中的k，神经网络中的网络层数及每层的节点个数。通常由手工设定
- 如何调整参数
  1. 交叉验证
     sklearn.model_selection.cross_val_score()
  2. 网格搜索(Grid Search)
     sklearn.model_selection.GridSearchCV()



https://scikit-learn.org/stable/model_selection.html#model-selection

网格搜索

- 网格搜索（**grid search**）：搜索参数的所有可能组合

- **Scikit-learn**中的内置类：**GridSearchCV**

  from sklearn.model_selection import GridSearchCV

SVC参数调优

```python
# 利用GridSearchCV寻求最优参数组合

# 导入红酒数据集
from sklearn.datasets import load_wine
wine = load_wine()

# 拆分数据集
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(wine.data,wine.target,random_state=0)

# 构建SVC模型
from sklearn.svm import SVC
svc = SVC()

# 导入网格搜索工具
from sklearn.model_selection import GridSearchCV
# 将需要遍历的参数定义为字典
params = [ {'kernel': ['linear'], 'C': [1, 10, 50, 600]},
           {'kernel': ['poly'], 'degree': [2, 3]},
           {'kernel': ['rbf'], 'gamma': [0.01, 0.001], 'C': [1, 10, 50, 600]},
          ]
# 网格搜索中使用的模型和参数
grid_search = GridSearchCV(svc,params,cv=3)
# 网格搜索模型拟合数据
grid_search.fit(X_train, y_train)

print('模型最高分: {:.3f}'.format(grid_search.score(X_test, y_test)))
print('最优参数: {}'.format(grid_search.best_params_))
print('交叉验证最高得分: {:.3f}'.format(grid_search.best_score_))
```

```
模型最高分: 0.978
最优参数: {'kernel': 'linear', 'C': 1}
交叉验证最高得分: 0.947
```

随机搜索

- 随机搜索类-**RandomizedSearchCV**
- 方法其实和**GridSearchCV**一致，但它在参数空间中采用随机采样的方式代替了**GridSearchCV**对于参数的网格搜索
- 连续变量值的参数，**RandomizedSearchCV**会将其当作一个分布进行采样，这是网格搜索做不到的，它的搜索能力取决于设定的**n_iter**参数

➢ 如何利用**RandomizedSearchCV**搜索**SVC**的最佳组合？