```
-> def svmTrain_SMO(X, y, C, kernelFunction='linear', tol=1e-3, max_iter=5, **kargs):

    start = time.clock()

    m,n = X.shape
    X = np.mat(X)
    y = np.mat(y, dtype='float64')

    y[np.where(y==0)] = -1

    alphas = np.mat(np.zeros((m,1)))
    b = 0.0
    E = np.mat(np.zeros((m,1)))
    iters = 0
    eta = 0.0
    L = 0.0
    H = 0.0

    if kernelFunction =='linear':
        K = X*X.T
    elif kernelFunction == 'gaussian':
        K = kargs['K_matrix']
    else :
```

函数参数：
X, y为**loadData**()的返回值，numpy.ndarray类型。
X为m×n的数组，m是样本数，n是特征维度。
y为m×1的数组，是对应样本的标签值，其中正例用1表示，反例用0表示。

C为惩罚参数，C越大，模型越接近硬间隔SVM。

kernelFunction指定了核函数类型，有 ‘linear’ 和 ‘gaussian’ 两种可选。默认为 ‘linear’。选择 ‘Gaussian’ 时，需要添加预先计算好的核矩阵作为额外参数。

tol为容错率。max_iter为最大迭代次数。
以上两参数对于不同的模型有不同的最佳取值，可通过模型调参过程调试取优。

**kargs为额外参数。

```
def svmTrain_SMO(X, y, C, kernelFunction='linear', tol=1e-3, max_iter=5, **kargs):

->      start = time.clock()                    记录训练开始时间；

        m,n = X.shape
        X = np.mat(X)
        y = np.mat(y, dtype='float64')

        y[np.where(y==0)] = -1

        alphas = np.mat(np.zeros((m,1)))
        b = 0.0
        E = np.mat(np.zeros((m,1)))
        iters = 0
        eta = 0.0
        L = 0.0
        H = 0.0

        if kernelFunction =='linear':
            K = X*X.T
        elif kernelFunction == 'gaussian':
            K = kargs['K_matrix']
        else :
```

```python
def svmTrain_SMO(X, y, C, kernelFunction='linear', tol=1e-3, max_iter=5, **kargs):

    start = time.clock()

->  m,n = X.shape
    X = np.mat(X)
    y = np.mat(y, dtype='float64')

    y[np.where(y==0)] = -1

    alphas = np.mat(np.zeros((m,1)))
    b = 0.0
    E = np.mat(np.zeros((m,1)))
    iters = 0
    eta = 0.0
    L = 0.0
    H = 0.0

    if kernelFunction =='linear':
        K = X*X.T
    elif kernelFunction == 'gaussian':
        K = kargs['K_matrix']
    else :
```

记录训练开始时间

m为样本数，n为特征维度

将X,y转换为numpy.matrix类型，且将y的数据类型转换为float64

```
def svmTrain_SMO(X, y, C, kernelFunction='linear', tol=1e-3, max_iter=5, **kargs):

    start = time.clock()
```

记录训练开始时间

```
    m,n = X.shape
    X = np.mat(X)
    y = np.mat(y, dtype='float64')
```

m为样本数，n为特征维度

将X,y转换为numpy.matrix类型，且将y的数据类型转换为float64

```
->  y[np.where(y==0)] = -1
```

将反例改为用-1表示

```
    alphas = np.mat(np.zeros((m,1)))
    b = 0.0
    E = np.mat(np.zeros((m,1)))
    iters = 0
    eta = 0.0
    L = 0.0
    H = 0.0

    if kernelFunction =='linear':
        K = X*X.T
    elif kernelFunction == 'gaussian':
        K = kargs['K_matrix']
    else :
```

```
def svmTrain_SMO(X, y, C, kernelFunction='linear', tol=1e-3, max_iter=5, **kargs):

    start = time.clock()

    m,n = X.shape
    X = np.mat(X)
    y = np.mat(y, dtype='float64')

    y[np.where(y==0)] = -1

->  alphas = np.mat(np.zeros((m,1)))
    b = 0.0
    E = np.mat(np.zeros((m,1)))
    iters = 0
    eta = 0.0
    L = 0.0
    H = 0.0

    if kernelFunction =='linear':
        K = X*X.T
    elif kernelFunction == 'gaussian':
        K = kargs['K_matrix']
    else :
```

记录训练开始时间

m为样本数，n为特征维度

将X,y转换为numpy.matrix类型，且将y的数据类型转换为float64

将反例改为用-1表示

1.Initialize alphas向量为0

alphas为m×1的列向量

b为SVM模型中的参数b

E存储当前预测值与标签值的差距（error）

iters指示当前迭代次数

eta为alpha[j]的最优修改量

L,H分别指示alpha修改时的上下边界

```
-> def svmTrain_SMO(X, y, C, kernelFunction='linear', tol=1e-3, max_iter=5, **kargs):

        start = time.clock()

        m,n = X.shape
        X = np.mat(X)
        y = np.mat(y, dtype='float64')

        y[np.where(y==0)] = -1

        alphas = np.mat(np.zeros((m,1)))
        b = 0.0
        E = np.mat(np.zeros((m,1)))
        iters = 0
        eta = 0.0
        L = 0.0
        H = 0.0

        if kernelFunction =='linear':
            K = X*X.T
        elif kernelFunction == 'gaussian':
            K = kargs['K_matrix']
        else :
```

核函数类型选择，输入非法值时，返回None.

```
    H = 0.0

    if kernelFunction =='linear':
        K = X*X.T
    elif kernelFunction == 'gaussian':
        K = kargs['K_matrix']
    else :
        print('Kernel Error')
        return None


->  print('Training ...', end='')
    dots = 12

    while iters < max_iter:

        num_changed_alphas = 0

        for i in range(m):
            E[i] = b + np.sum(np.multiply(np.multiply(alphas, y), K[:,i])) - y[i]


            if (y[i]*E[i] < -tol and alphas[i] < C) or (y[i]*E[i] > tol and alphas[i] > 0):

                j = np.random.randint(m)
```

核函数类型选择；输入非法值时，返回None.

窗口输出内容

```python
    H = 0.0

->  if kernelFunction =='linear':
        K = X*X.T
    elif kernelFunction == 'gaussian':
        K = kargs['K_matrix']
    else :
        print('Kernel Error')
        return None


    print('Training ...', end='')
    dots = 12

    while iters < max_iter:

        num_changed_alphas = 0

        for i in range(m):
            E[i] = b + np.sum(np.multiply(np.multiply(alphas, y), K[:,i])) - y[i]


            if (y[i]*E[i] < -tol and alphas[i] < C) or (y[i]*E[i] > tol and alphas[i] > 0):

                j = np.random.randint(m)
```

2.while 迭代次数小于最大迭代次数:

```
    dots = 12

    while iters < max_iter:

->          num_changed_alphas = 0

        for i in range(m):
            E[i] = b + np.sum(np.multiply(np.multiply(alphas, y), K[:,i])) - y[i]


            if (y[i]*E[i] < -tol and alphas[i] < C) or (y[i]*E[i] > tol and alphas[i] > 0):

                j = np.random.randint(m)
                while j == i:
                    j = np.random.randint(m)

                E[j] = b + np.sum(np.multiply(np.multiply(alphas, y), K[:,j])) - y[j]

                alpha_i_old = alphas[i].copy()
                alpha_j_old = alphas[j].copy()

                if y[i] == y[j]:
                    L = max(0, alphas[j] + alphas[i] - C)
                    H = min(C, alphas[j] + alphas[i])
                else:
                    L = max(0, alphas[j] - alphas[i])
```

2.while 迭代次数小于最大迭代次数:

标识改变的alpha，每次外循环初始化为0

```
    dots = 12

    while iters < max_iter:

        num_changed_alphas = 0

->      for i in range(m):
            E[i] = b + np.sum(np.multiply(np.multiply(alphas, y), K[:,i])) - y[i]


            if (y[i]*E[i] < -tol and alphas[i] < C) or (y[i]*E[i] > tol and alphas[i] > 0):

                j = np.random.randint(m)
                while j == i:
                    j = np.random.randint(m)

                E[j] = b + np.sum(np.multiply(np.multiply(alphas, y), K[:,j])) - y[j]

                alpha_i_old = alphas[i].copy()
                alpha_j_old = alphas[j].copy()

                if y[i] == y[j]:
                    L = max(0, alphas[j] + alphas[i] - C)
                    H = min(C, alphas[j] + alphas[i])
                else:
                    L = max(0, alphas[j] - alphas[i])
```

2.while 迭代次数小于最大迭代次数:

标识改变的alpha，每次外循环初始化为0

3. for each alpha[i] ***in*** alphas: (对于每个样本)

```
dots = 12

while iters < max_iter:

    num_changed_alphas = 0

    for i in range(m):
        E[i] = b + np.sum(np.multiply(np.multiply(alphas, y), K[:,i])) - y[i]

        if (y[i]*E[i] < -tol and alphas[i] < C) or (y[i]*E[i] > tol and alphas[i] > 0):

            j = np.random.randint(m)
            while j == i:
                j = np.random.randint(m)

            E[j] = b + np.sum(np.multiply(np.multiply(alphas, y), K[:,j])) - y[j]

            alpha_i_old = alphas[i].copy()
            alpha_j_old = alphas[j].copy()

            if y[i] == y[j]:
                L = max(0, alphas[j] + alphas[i] - C)
                H = min(C, alphas[j] + alphas[i])
            else:
                L = max(0, alphas[j] - alphas[i])
```

2.while 迭代次数小于最大迭代次数:

标识改变的alpha，每次外循环初始化为0

3. *for* each alpha[i] ***in*** alphas: (对于每个样本)

->

$E_i = b + \sum_{k=1}^{m} \alpha_k y_k \kappa(\boldsymbol{x}_k, \boldsymbol{x}_i) - y_i$

```
dots = 12

while iters < max_iter:

    num_changed_alphas = 0

    for i in range(m):
        E[i] = b + np.sum(np.multiply(np.multiply(alphas, y), K[:,i])) - y[i]

->      if (y[i]*E[i] < -tol and alphas[i] < C) or (y[i]*E[i] > tol and alphas[i] > 0):

            j = np.random.randint(m)
            while j == i:
                j = np.random.randint(m)

            E[j] = b + np.sum(np.multiply(np.multiply(alphas, y), K[:,j])) - y[j]

            alpha_i_old = alphas[i].copy()
            alpha_j_old = alphas[j].copy()

            if y[i] == y[j]:
                L = max(0, alphas[j] + alphas[i] - C)
                H = min(C, alphas[j] + alphas[i])
            else:
                L = max(0, alphas[j] - alphas[i])
```

2.while 迭代次数小于最大迭代次数:

标识改变的alpha，每次外循环初始化为0

3. for each alpha[i] ***in*** alphas: (对于每个样本)

$E_i = b + \sum_{k=1}^{m} \alpha_k y_k \kappa(\boldsymbol{x}_k, \boldsymbol{x}_i) - y_i$

4.if alpha[i]可优化:

可优化即意味着违反KKT条件。同时，alpha大于C小于0时在后面会被调整为C或0，if后的条件即为在tol精度下违反KKT条件的形式。

```
    dots = 12

->  while iters < max_iter:                5.随机选择另一个alpha[j],同时优化这两个向量。

        num_changed_alphas = 0

        for i in range(m):
            E[i] = b + np.sum(np.multiply(np.multiply(alphas, y), K[:,i])) - y[i]


            if (y[i]*E[i] < -tol and alphas[i] < C) or (y[i]*E[i] > tol and alphas[i] > 0):

                j = np.random.randint(m)
                while j == i:
                    j = np.random.randint(m)

                E[j] = b + np.sum(np.multiply(np.multiply(alphas, y), K[:,j])) - y[j]

                alpha_i_old = alphas[i].copy()
                alpha_j_old = alphas[j].copy()

                if y[i] == y[j]:
                    L = max(0, alphas[j] + alphas[i] - C)
                    H = min(C, alphas[j] + alphas[i])
                else:
                    L = max(0, alphas[j] - alphas[i])
```

```
if (y[i]*E[i] < -tol and alphas[i] < C) or (y[i]*E[i] > tol and alphas[i] > 0):
```

5.随机选择另一个alpha[j],同时优化这两个向量。

```
j = np.random.randint(m)
while j == i:
    j = np.random.randint(m)
```

同上计算E[j]

->

```
E[j] = b + np.sum(np.multiply(np.multiply(alphas, y), K[:,j])) - y[j]

alpha_i_old = alphas[i].copy()
alpha_j_old = alphas[j].copy()

if y[i] == y[j]:
    L = max(0, alphas[j] + alphas[i] - C)
    H = min(C, alphas[j] + alphas[i])
else:
    L = max(0, alphas[j] - alphas[i])
    H = min(C, C + alphas[j] - alphas[i])

if L == H:
    continue

eta = 2*K[i,j] - K[i,i] -K[j,j]
if eta >= 0:
    continue

alphas[j] = alphas[j] - (y[j]*(E[i] - E[j]))/eta
```

```
if (y[i]*E[i] < -tol and alphas[i] < C) or (y[i]*E[i] > tol and alphas[i] > 0):

    j = np.random.randint(m)   5.随机选择另一个alpha[j],同时优化这两个向量。
    while j == i:
        j = np.random.randint(m)
                                     同上计算E[j]
    E[j] = b + np.sum(np.multiply(np.multiply(alphas, y), K[:,j])) - y[j]

->  alpha_i_old = alphas[i].copy()   保留alphas[i]和alphas[j]的初始值
    alpha_j_old = alphas[j].copy()

    if y[i] == y[j]:
        L = max(0, alphas[j] + alphas[i] - C)
        H = min(C, alphas[j] + alphas[i])
    else:
        L = max(0, alphas[j] - alphas[i])
        H = min(C, C + alphas[j] - alphas[i])

    if L == H:
        continue

    eta = 2*K[i,j] - K[i,i] -K[j,j]
    if eta >= 0:
        continue

    alphas[j] = alphas[j] - (y[j]*(E[i] - E[j]))/eta
```

```python
if (y[i]*E[i] < -tol and alphas[i] < C) or (y[i]*E[i] > tol and alphas[i] > 0):
```

5.随机选择另一个alpha[j],同时优化这两个向量。

```python
    j = np.random.randint(m)
    while j == i:
        j = np.random.randint(m)
```

同上计算E[j]

```python
    E[j] = b + np.sum(np.multiply(np.multiply(alphas, y), K[:,j])) - y[j]
```

保留alphas[i]和alphas[j]的初始值

```python
    alpha_i_old = alphas[i].copy()
    alpha_j_old = alphas[j].copy()
```

->

```python
    if y[i] == y[j]:
        L = max(0, alphas[j] + alphas[i] - C)
        H = min(C, alphas[j] + alphas[i])
    else:
        L = max(0, alphas[j] - alphas[i])
        H = min(C, C + alphas[j] - alphas[i])
```

$$\begin{cases} y_1 = y_2 \\ L = \max\left(0, \alpha_2^{old} + \alpha_1^{old} - C\right) \\ H = \min\left(C, \alpha_2^{old} + \alpha_1^{old}\right) \\ y_1 \neq y_2 \\ L = \max\left(0, \alpha_2^{old} - \alpha_1^{old}\right) \\ H = \min\left(C, \mathrm{C} + \alpha_2^{old} + \alpha_1^{old}\right) \end{cases}$$

```python
    if L == H:
        continue

    eta = 2*K[i,j] - K[i,i] -K[j,j]
    if eta >= 0:
        continue

    alphas[j] = alphas[j] - (y[j]*(E[i] - E[j]))/eta
```

```
if (y[i]*E[i] < -tol and alphas[i] < C) or (y[i]*E[i] > tol and alphas[i] > 0):
```

5.随机选择另一个alpha[j],同时优化这两个向量。

```
        j = np.random.randint(m)
        while j == i:
            j = np.random.randint(m)
```

同上计算E[j]

```
        E[j] = b + np.sum(np.multiply(np.multiply(alphas, y), K[:,j])) - y[j]
```

保留alphas[i]和alphas[j]的初始值

```
        alpha_i_old = alphas[i].copy()
        alpha_j_old = alphas[j].copy()

        if y[i] == y[j]:
            L = max(0, alphas[j] + alphas[i] - C)
            H = min(C, alphas[j] + alphas[i])
        else:
            L = max(0, alphas[j] - alphas[i])
            H = min(C, C + alphas[j] - alphas[i])
```

$$
\begin{cases}
y_1 = y_2 \\
L = \max\left(0, \alpha_2^{old} + \alpha_1^{old} - C\right) \\
H = \min\left(C, \alpha_2^{old} + \alpha_1^{old}\right) \\
y_1 \neq y_2 \\
L = \max\left(0, \alpha_2^{old} - \alpha_1^{old}\right) \\
H = \min\left(C, \mathrm{C} + \alpha_2^{old} + \alpha_1^{old}\right)
\end{cases}
$$

->

```
        if L == H:
            continue
```

L=H，则不做任何修改，退出内循环寻找下一个可优化值

```
        eta = 2*K[i,j] - K[i,i] -K[j,j]
        if eta >= 0:
            continue
```

eta=0，说明alpha[a]最优修改量为0。（根据eta定义，eta<=0)）

```
        alphas[j] = alphas[j] - (y[j]*(E[i] - E[j]))/eta
```

```
if (y[i]*E[i] < -tol and alphas[i] < C) or (y[i]*E[i] > tol and alphas[i] > 0):
```

->

```
j = np.random.randint(m)
while j == i:
    j = np.random.randint(m)

E[j] = b + np.sum(np.multiply(np.multiply(alph

alpha_i_old = alphas[i].copy()
alpha_j_old = alphas[j].copy()
if y[i] == y[j]:
    L = max(0, alphas[j] + alphas[i] - C)
    H = min(C, alphas[j] + alphas[i])
else:
    L = max(0, alphas[j] - alphas[i])
    H = min(C, C + alphas[j] - alphas[i])

if L == H:
    continue

eta = 2*K[i,j] - K[i,i] -K[j,j]
if eta >= 0:
    continue

alphas[j] = alphas[j] - (y[j]*(E[i] - E[j]))/eta
```

$$\alpha_2^{\text{new,unc}} = \alpha_2^{\text{old}} + \frac{y_2(E_1 - E_2)}{\eta}$$

$$\alpha_2^{\text{new}} = \begin{cases} H, & \alpha_2^{\text{new,unc}} > H \\ \alpha_2^{\text{new,unc}}, & L \leqslant \alpha_2^{\text{new,unc}} \leqslant H \\ L, & \alpha_2^{\text{new,unc}} < L \end{cases}$$

```
            continue

        alphas[j] = alphas[j] - (y[j]*(E[i] - E[j]))/eta

        alphas[j] = min(H, alphas[j])
        alphas[j] = max(L, alphas[j])
```

$$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\eta}$$

$$\alpha_2^{new} = \begin{cases} H, & \alpha_2^{new,unc} > H \\ \alpha_2^{new,unc}, & L \leqslant \alpha_2^{new,unc} \leqslant H \\ L, & \alpha_2^{new,unc} < L \end{cases}$$

```
->      if abs(alphas[j] - alpha_j_old) < tol:
            alphas[j] = alpha_j_old
            continue
```

如果改变量小于容许误差精度，那本次改变没有任何意义。退出内循环，寻找下一个可优化值。

```
        alphas[i] = alphas[i] + y[i]*y[j]*(alpha_j_old - alphas[j])

        b1 = b - E[i]\
         - y[i] * (alphas[i] - alpha_i_old) *  K[i,j]\
         - y[j] * (alphas[j] - alpha_j_old) *  K[i,j]

        b2 = b - E[j]\
         - y[i] * (alphas[i] - alpha_i_old) *  K[i,j]\
         - y[j] * (alphas[j] - alpha_j_old) *  K[j,j]

        if (0 < alphas[i] and alphas[i] < C):
            b = b1
        elif (0 < alphas[j] and alphas[j] < C):
```

```
            continue

        alphas[j] = alphas[j] - (y[j]*(E[i] - E[j]))/eta
```

$$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\eta}$$

```
        alphas[j] = min(H, alphas[j])
        alphas[j] = max(L, alphas[j])
```

$$\alpha_2^{new} = \begin{cases} H, & \alpha_2^{new,unc} > H \\ \alpha_2^{new,unc}, & L \leqslant \alpha_2^{new,unc} \leqslant H \\ L, & \alpha_2^{new,unc} < L \end{cases}$$

```
        if abs(alphas[j] - alpha_j_old) < tol:
            alphas[j] = alpha_j_old
            continue
```

如果改变量小于容许误差精度，那本次改变没有任何意义。退出内循环，寻找下一个可优化值。

```
->      alphas[i] = alphas[i] + y[i]*y[j]*(alpha_j_old - alphas[j])
```

$$\alpha_1^{new} = \alpha_1^{old} + y_1 y_2 (\alpha_2^{old} - \alpha_2^{new})$$

```
        b1 = b - E[i]\
         - y[i] * (alphas[i] - alpha_i_old) *  K[i,j]\
         - y[j] * (alphas[j] - alpha_j_old) *  K[i,j]

        b2 = b - E[j]\
         - y[i] * (alphas[i] - alpha_i_old) *  K[i,j]\
         - y[j] * (alphas[j] - alpha_j_old) *  K[j,j]

        if (0 < alphas[i] and alphas[i] < C):
            b = b1
        elif (0 < alphas[j] and alphas[j] < C):
```

```python
            continue
```

->

```python
        alphas[j] = alphas[j] - (y[j]*(E[i] - E[j]))/eta

        alphas[j] = min(H, alphas[j])
        alphas[j] = max(L, alphas[j])
```

$$b_1^{new} = -E_1 - y_1 K_{11}(\alpha_1^{new} - \alpha_1^{old}) - y_2 K_{21}(\alpha_2^{new} - \alpha_2^{old}) + b^{old}$$

```python
        if abs(alphas[j] - alpha_j_old`
            alphas[j] = alpha_j_old
            continue
```

$$b_2^{new} = -E_2 - y_1 K_{12}(\alpha_1^{new} - \alpha_1^{old}) - y_2 K_{22}(\alpha_2^{new} - \alpha_2^{old}) + b^{old}$$

$$b^{new} = \begin{cases} b_1^{new} & 0 \le \alpha_1^{new} \le C \\ b_2^{new} & 0 \le \alpha_2^{new} \le C \\ (b_1 + b_2)/2 & otherwise \end{cases}$$

```python
        alphas[i] = alphas[i] + y[i]*y[j]*(alpha_j_old - alphas[j])

        b1 = b - E[i]\
          - y[i] * (alphas[i] - alpha_i_old) *  K[i,j]\
          - y[j] * (alphas[j] - alpha_j_old) *  K[i,j]

        b2 = b - E[j]\
          - y[i] * (alphas[i] - alpha_i_old) *  K[i,j]\
          - y[j] * (alphas[j] - alpha_j_old) *  K[j,j]

        if (0 < alphas[i] and alphas[i] < C):
            b = b1
        elif (0 < alphas[j] and alphas[j] < C):
```

```
alphas[i] = alphas[i] + y[i]*y[j]*(alpha_j_old - alphas[j])

b1 = b - E[i]\
 - y[i] * (alphas[i] - alpha_i_old) *  K[i,j]\
 - y[j] * (alphas[j] - alpha_j_old) *  K[i,j]
```

$$b_1^{new} = -E_1 - y_1 K_{11}(\alpha_1^{new} - \alpha_1^{old}) - y_2 K_{21}(\alpha_2^{new} - \alpha_2^{old}) + b^{old}$$

```
b2 = b - E[j]\
 - y[i] * (alphas[i] - alpha_i_old) *  K[i,j]\
 - y[j] * (alphas[j] - alpha_j_old) *  K[i,j]
```

$$b_2^{new} = -E_2 - y_1 K_{12}(\alpha_1^{new} - \alpha_1^{old}) - y_2 K_{22}(\alpha_2^{new} - \alpha_2^{old}) + b^{old}$$

```
if (0 < alphas[i] and alphas[i] < C):
    b = b1
elif (0 < alphas[j] and alphas[j] < C):
    b = b2
else:
    b = (b1+b2)/2.0

num_changed_alphas = num_changed_alphas + 1
```

$$b^{new} = \begin{cases} b_1^{new} & 0 \le \alpha_1^{new} \le C \\ b_2^{new} & 0 \le \alpha_2^{new} \le C \\ (b_1 + b_2)/2 & otherwise \end{cases}$$

-> 程序运行至此，一对alphas已完成更新，故修改指示。

```
if num_changed_alphas == 0:
    iters = iters + 1
else:
    iters = 0

print('.', end='')
dots = dots + 1
```

```
alphas[i] = alphas[i] + y[i]*y[j]*(alpha_j_old - alphas[j])

b1 = b - E[i]\
 - y[i] * (alphas[i] - alpha_i_old) *  K[i,j]\
 - y[j] * (alphas[j] - alpha_j_old) *  K[i,j]
```

$$b_1^{\text{new}} = -E_1 - y_1 K_{11}(\alpha_1^{\text{new}} - \alpha_1^{\text{old}}) - y_2 K_{21}(\alpha_2^{\text{new}} - \alpha_2^{\text{old}}) + b^{\text{old}}$$

```
b2 = b - E[j]\
 - y[i] * (alphas[i] - alpha_i_old) *  K[i,j]\
 - y[j] * (alphas[j] - alpha_j_old) *  K[i,j]
```

$$b_2^{\text{new}} = -E_2 - y_1 K_{12}(\alpha_1^{\text{new}} - \alpha_1^{\text{old}}) - y_2 K_{22}(\alpha_2^{\text{new}} - \alpha_2^{\text{old}}) + b^{\text{old}}$$

```
if (0 < alphas[i] and alphas[i] < C):
    b = b1
elif (0 < alphas[j] and alphas[j] < C):
    b = b2
else:
    b = (b1+b2)/2.0

num_changed_alphas = num_changed_alphas + 1
```

$$b^{new} = \begin{cases} b_1^{new} & 0 \le \alpha_1^{new} \le C \\ b_2^{new} & 0 \le \alpha_2^{new} \le C \\ (b_1 + b_2)/2 & otherwise \end{cases}$$

程序运行至此，一对alphas已完成更新，故修改指示。

```
->      if num_changed_alphas == 0:
            iters = iters + 1
        else:
            iters = 0

        print('.', end='')
        dots = dots + 1
```

最大迭代次数是指在没有alpha值发生改变时的最大次数

```
            alphas[i] = alphas[i] + y[i]*y[j]*(alpha_j_old - alphas[j])

            b1 = b - E[i]\
             - y[i] * (alphas[i] - alpha_i_old) *  K[i,j]\
             - y[j] * (alphas[j] - alpha_j_old) *  K[i,j]

            b2 = b - E[j]\
             - y[i] * (alphas[i] - alpha_i_old) *  K[i,j]\
             - y[j] * (alphas[j] - alpha_j_old) *  K[j,j]

            if (0 < alphas[i] and alphas[i] < C):
                b = b1
            elif (0 < alphas[j] and alphas[j] < C):
                b = b2
            else:
                b = (b1+b2)/2.0

            num_changed_alphas = num_changed_alphas + 1

        if num_changed_alphas == 0:
            iters = iters + 1
        else:
            iters = 0

        print('.', end='')
        dots = dots + 1
```

->

窗口显示内容

记录程序结束时间，并显示

```python
            iters = 0

        print('.', end='')
        dots = dots + 1
        if dots > 78:
            dots = 0
            print()

    print('Done',end='')
    end = time.clock()
    print('( '+str(end-start)+'s )')
    print()

->  idx = np.where(alphas > 0)
    model = {'X':X[idx[0],:], 'y':y[idx], 'kernelFunction':str(kernelFunction), \
             'b':b, 'alphas':alphas[idx], 'w':(np.multiply(alphas,y).T*X).T}
    return model
```

窗口显示内容

记录程序结束时间，并显示

确定支持向量的索引

```
            iters = 0

        print('.', end='')
        dots = dots + 1
        if dots > 78:
            dots = 0
            print()
```

窗口显示内容

```
    print('Done',end='')
    end = time.clock()
    print('( '+str(end-start)+'s )')
    print()
```

记录程序结束时间，并显示

确定支持向量的索引

```
    idx = np.where(alphas > 0)
->  model = {'X':X[idx[0],:], 'y':y[idx], 'kernelFunction':str(kernelFunction), \
             'b':b, 'alphas':alphas[idx], 'w':(np.multiply(alphas,y).T*X).T}
    return model
```

存储模型，返回模型。