

电子科技大学信息与软件工程学院

# 实 验 报 告

学 号 2018091618008

姓 名 袁昊男

(实验) 课程名称 机器学习技术与应用

理论教师 黄 俊

实验教师 杨 珊

# 电子科技大学

## 实验报告

学生姓名：袁昊男      学号：2018091618008      指导教师：杨珊

实验地点：在线实验      实验时间：2020.05.20

一、实验室名称：信息与软件工程学院实验中心

二、实验名称：手写数字识别的支持向量机学习算法模型

三、实验学时：4 学时

四、实验原理：

把原始最优化问题转化成了其对偶问题，因为对偶问题是一个凸二次规划问题，这样的凸二次规划问题具有全局最优解。

$$\begin{aligned}\min_{\alpha} \Psi(\vec{\alpha}) &= \min_{\alpha} \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j K(\vec{x}_i, \vec{x}_j) \alpha_i \alpha_j - \sum_{i=1}^N \alpha_i, \\ 0 &\leq \alpha_i \leq C, \forall i, \\ \sum_{i=1}^N y_i \alpha_i &= 0.\end{aligned}$$

其中 $(x_i, y_i)$ 表示训练样本数据， $x_i$ 为样本特征， $C$ 为惩罚系数由自己设定。上述问题是要求解  $N$  个参数，其他参数均为已知，有多种算法可以对上述问题求解，但是算法复杂度均很大。由 Platt 提出的序列最小最优化算法（SMO）可以高效的求解上述 SVM 问题，它把原始求解  $N$  个参数二次规划问题分解成很多个子二次规划问题分别求解，每个子问题只需要求解 2 个参数，方法类似于坐标上升，节省时间成本和降低了内存需求。每次启发式选择两个变量进行优化，不断循环，直到达到函数最优值。

五、实验目的：

- 1、理解多项式核函数、高斯核函数和字符核函数。
- 2、掌握 SMO 算法原理。

## 六、实验内容：

- 1、收集数据，使用 MNIST 数据集。
- 2、准备数据，基于二值图像构造向量。
- 3、分析数据，对图像向量进行预测。
- 4、训练算法，采用一对一法进行多分类,采用两种不同的核函数，来运行 SMO 算法。
- 5、测试算法，编写一个函数来测量不同的核函数，并计算错误率。
- 6、使用算法，一个图像识别的完整应用，要求学生准备一些图像处理知识。

## 七、实验器材（设备、元器件）：

- 1、PC 电脑。
- 2、Python。

## 八、实验步骤：

### 1、读取数据

```
1. def load_data():  
2.     f = gzip.open('mnist.pkl.gz', 'rb')  
3.     training_data, validation_data, test_data = pickle.load(f, encoding='bytes')  
4.     f.close()  
5.     return (training_data, validation_data, test_data)
```

说明：gzip 文件读写的时候需要用到 Python 的 gzip 模块。pickle 库的 load 函数从 file 中读取一个字符串，并将它重构为原来的 Python 对象。参数 file 为类文件对象，有 read()和 readline()接口。MNIST 数据集来自美国国家标准与技术研究所。训练集由来自 250 个不同人手写的数字构成，其中 50%是高中学生，50%来自人口普查局的工作人员。测试集也是同样比例的手写数字数据。

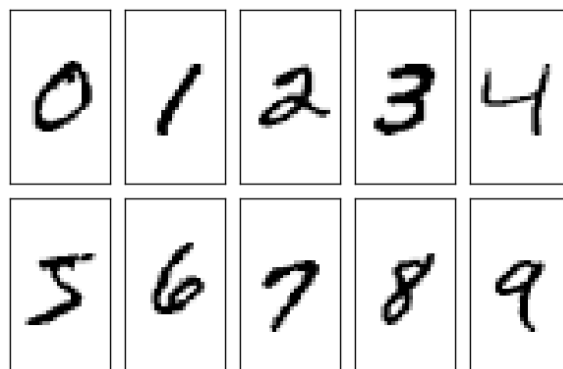


图 1.1 手写数字示例

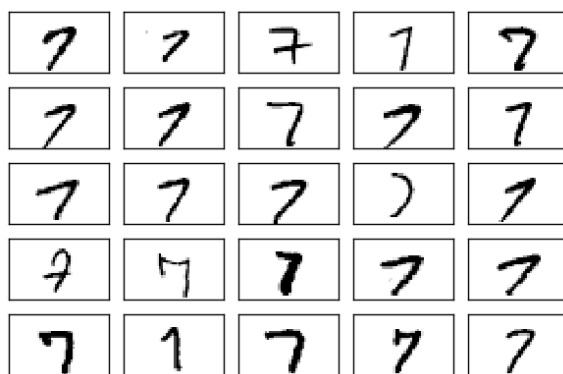


图 1.2 不同形态的数字 7

`load_data()`函数返回的是 `training_data` 训练集、`validation_data` 验证集、`test_data` 测试集。

## 2、训练模型

```
1. def svm_baseline():
2.     print(time.strftime('%Y-%m-%d %H:%M:%S'))
3.
4.     training_data, validation_data, test_data = load_data()
5.     clf = svm.SVC(C=100.0, kernel='rbf', gamma=0.03)
6.     clf.fit(training_data[0], training_data[1])
7.
8.     predictions = [int(a) for a in clf.predict(test_data[0])]
9.     num_correct = sum(int(a == y) for a, y in zip(predictions, test_data[1]))
10.
11.    print("%s of %s test values correct." % (num_correct, len(test_data[1])))
12.
13.    print(time.strftime('%Y-%m-%d %H:%M:%S'))
```

**说明：**使用 `sklearn` 库中有关 SVM 的相关函数。首先调用 `load_data()` 将训练集、验证集以及测试集数据调入，传递训练模型的参数，这里用默认的参数。SVM 关键是选取核函数的类型，主要有线性内核、多项式内核、径向基内核（RBF）、sigmoid 核。这些函数中应用最广的是 RBF 核了，无论是小样本还是大样本，高维还是低维等情况，RBF 核函数均适用。它相比其他的函数有以下优点：

- RBF 核函数可以将一个样本映射到一个更高维的空间，而且线性核函数是 RBF 的一个特例，也就是说如果考虑使用 RBF，那么就没有必要考虑线性核函数了。
- 与多项式核函数相比，RBF 需要确定的参数要少，核函数参数的多少直接影响函数的复杂程度。另外，当多项式的阶数比较高时，核矩阵的元素值将趋于无穷大或无穷小，而 RBF 则在上，会减少数值的计算困难。

- 对于某些参数，RBF 和 sigmoid 具有相似的性能。

svm.SVC()函数的完整调用格式如下：

```
sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0,
shrinking=True, probability=False, tol=0.001, cache_size=200,
class_weight=None, verbose=False, max_iter=-1, decision_function_shape=None, random_state=None), 其中：
```

- **C**: C-SVC 的惩罚参数 C，默认值是 1.0。C 越大，相当于惩罚松弛变量，希望松弛变量接近 0，即对误分类的惩罚增大，趋向于对训练集全分对的情况，这样对训练集测试时准确率很高，但泛化能力弱。C 值小，对误分类的惩罚减小，允许容错，将他们当成噪声点，泛化能力较强。
- **kernel**: 核函数，默认是 rbf，可以是 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'—线性:  $u \cdot v$ —多项式:  $(\gamma u \cdot v + \text{coef0})^{\text{degree}}$ —RBF 函数:  $\exp(-\gamma |u-v|^2)$ —sigmoid:  $\tanh(\gamma u \cdot v + \text{coef0})$ 。
- **degree**: 多项式 poly 函数的维度，默认是 3，选择其他核函数时会被忽略。
- **gamma**: 'rbf', 'poly' 和 'sigmoid' 的核函数参数。默认是 'auto'，则会选择  $1/n_{\text{features}}$ 。
- **coef0**: 核函数的常数项。对于 'poly' 和 'sigmoid' 有用。
- **probability**: 是否采用概率估计。默认为 False，可选。在训练 fit() 模型时需要加上这个参数，之后才能用相关的方法: predict\_proba 和 predict\_log\_proba。
- **shrinking**: 是否采用 shrinking heuristic 方法，默认为 true。
- **tol**: 停止训练的误差值大小，默认为  $1e-3$ 。
- **cache\_size**: 核函数 cache 缓存大小，默认为 200。
- **class\_weight**: 类别的权重，字典形式传递。设置第几类的参数 C 为  $\text{weight} \cdot C$  (C-SVC 中的 C)。
- **verbose**: 允许冗余输出。
- **max\_iter**: 最大迭代次数。-1 为无限制。
- **decision\_function\_shape**: 'ovo', 'ovr' or None, default=None3。
- **random\_state**: 数据洗牌时的种子值，int 值。

完成模型训练后使用测试集测试预测结果。

### 3、调整参数

```
1. import numpy as np
```

```

2. from sklearn import svm
3. from sklearn.model_selection import GridSearchCV
4.
5. parameters={'kernel':['linear','rbf','sig-
   moid','poly'],'C':np.linspace(0.1,20,50),'gamma':np.lin-
   space(0.1,20,20)}
6.
7. svc = svm.SVC()
8.
9. model = GridSearchCV(svc,parameters,cv=5,scoring='accuracy')
10. model.fit(X_train,y_train)
11. model.best_params_
12. model.score(X_test,y_test)

```

**说明：**SVM 模型有两个重要的参数：C 与 gamma。C 是惩罚系数，即对误差的宽容度。C 越高，说明越不能容忍出现误差，容易过拟合；C 越小，容易欠拟合；C 过大或过小，泛化能力变差。gamma 是选择 RBF 函数作为 kernel 后，该函数自带的一个参数，隐含地决定了数据映射到新的特征空间后的分布。gamma 越大，支持向量越少；gamma 越小，支持向量越多。支持向量的个数影响训练与预测的速度。Grid Search 是一种调参手段，即穷举搜索：在所有候选的参数选择中，通过循环遍历，尝试每一种可能性，表现最好的参数就是最终的结果。其原理就像是在数组里找最大值。使用 Grid Search 比较简单，有两个优点：全局最优；(C, gamma)相互独立，便于并行化进行。GridSearchCV()函数的完整调用格式如下：

GridSearchCV(estimator, param\_grid, scoring=None, fit\_params=None, n\_jobs=1, iid=True, refit=True, cv=None, verbose=0, pre\_dispatch='2\*n\_jobs', error\_score='raise', return\_train\_score='warn'), 其中：

- **estimator:** scikit-learn 分类器接口。
- **param\_grid:** 优化的参数的取值，值为字典或者列表，例如：  
param\_grid = {'n\_estimators':range(10,71,10)}。
- **scoring=None:** 模型评价标准，默认 None，根据所选模型不同，评价准则不同。字符串（函数名），形如：scorer(estimator, X, y)；如果是 None，则使用 estimator 的误差估计函数。
- **n\_jobs:** 并行数，int: 个数，-1: 跟 CPU 核数一致，1: 默认值。
- **cv=None:** 交叉验证参数，默认使用三折交叉验证。

#### 4、图像预处理

```

1. def getnumc(img):
2.     height_ = 8
3.     ratio_ = float(img.shape[1]) / float(img.shape[0])
4.     # 1 是宽 width, 2 是高 height

```

```

5.     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
6.     # 颜色空间转换函数 R, 第二个参数将 BG 格式转换为灰度图
7.     gray = cv2.resize(gray, (int(ratio_ * height_), height_))
8.     # 尺寸缩放, 参数分别为(源图片, (尺寸)), 指定高度按比例缩放
9.     gray = cv2.equalizeHist(gray)
10.    # 直方图均值化, 使得明暗更加平衡
11.    _, binary = cv2.threshold(gray, 20, 255, cv2.THRESH_BINARY)
12.    # 图像阈值处理函数, 将灰度图片转换为黑白图片
13.
14.    img_ = 255 - binary
15.    # 反转: 文字置为白色, 背景置为黑色
16.    return img_

```

**说明:** 使用 OpenCV 库对待识别数字图片进行预处理。首先使用 `resize()` 函数将图像大小统一为 8×8。后将图片进行灰度化、均值化、二值化处理, 使其能送入 SVM 模型进行识别。

## 5、手写数字识别测试

```

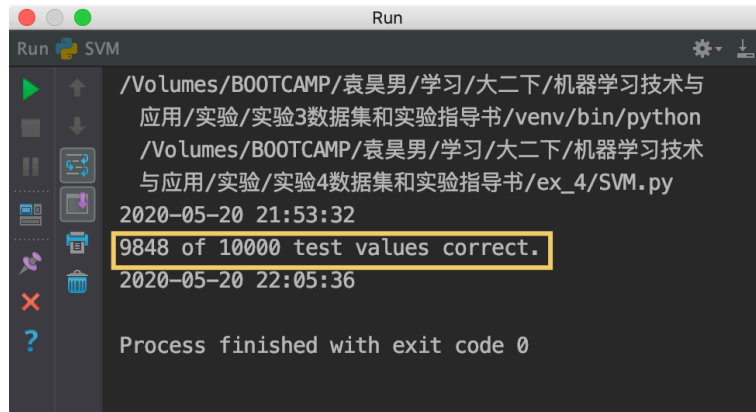
1. x = []
2. y = []
3.
4. for numi in xrange(1, 10):
5.     for numij in xrange(1, 5):
6.         fn = 'nums/' + str(numi) + '-' + str(numij) + '.png'
7.         x.append(getnumc(fn))
8.         y.append(numi)
9.
10. x = np.array(x)
11. y = np.array(y)
12.
13. svm = mlpy.LibSvm(svm_type='c_svc', kernel_type='poly', gamma=10)
14. svm.learn(x, y)
15.
16. print("训练样本测试: ")
17. print(svm.pred(x))
18. print("未知图像测试: ")
19.
20. for iii in xrange(1, 10):
21.     testfn = 'nums/test/' + str(iii) + '-test.png'
22.     testx = []
23.     testx.append(getnumc(testfn))
24.
25.     print(testfn + ":")
26.     print(svm.pred(testx))

```

**说明:** 将待识别的手写数字图片读入, 调用 `getnumc()` 函数对图像进行预处理。将处理后的图片转化为 Numpy 中的 `array` 对象送入 SVM 模型中进行预测, 输出预测结果。

## 九、实验数据及结果分析

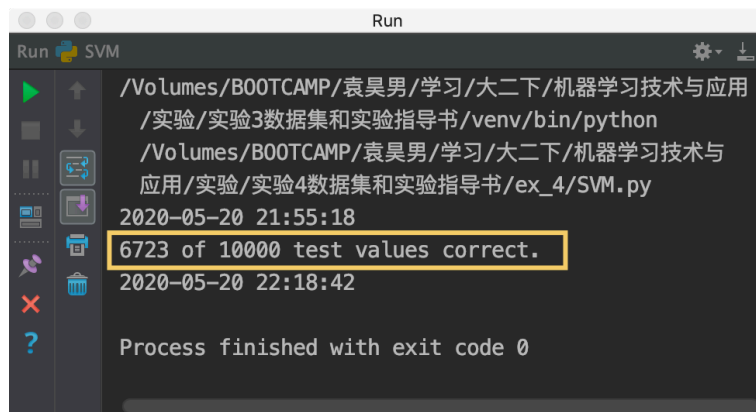
### 1、模型训练结果



```
Run SVM
/Volumes/BOOTCAMP/袁昊男/学习/大二下/机器学习技术与应用/实验/实验3数据集和实验指导书/venv/bin/python
/Volumes/BOOTCAMP/袁昊男/学习/大二下/机器学习技术与应用/实验/实验4数据集和实验指导书/ex_4/SVM.py
2020-05-20 21:53:32
9848 of 10000 test values correct.
2020-05-20 22:05:36
Process finished with exit code 0
```

图 9.1 模型训练结果一

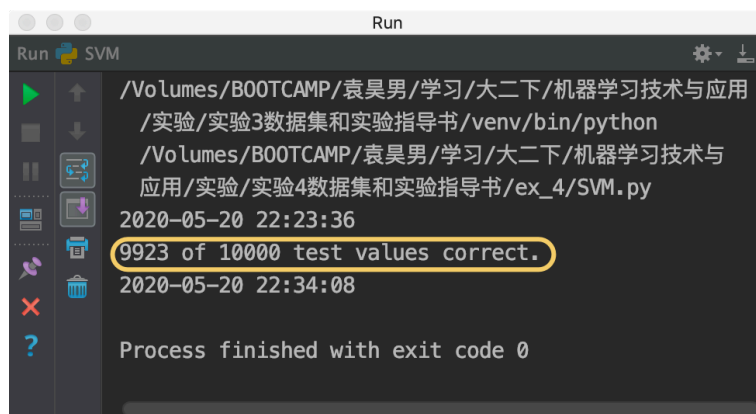
说明：采用的 kernel 为 RBF 函数。可以看出识别准确率较高，为 98.48%。



```
Run SVM
/Volumes/BOOTCAMP/袁昊男/学习/大二下/机器学习技术与应用/实验/实验3数据集和实验指导书/venv/bin/python
/Volumes/BOOTCAMP/袁昊男/学习/大二下/机器学习技术与应用/实验/实验4数据集和实验指导书/ex_4/SVM.py
2020-05-20 21:55:18
6723 of 10000 test values correct.
2020-05-20 22:18:42
Process finished with exit code 0
```

图 9.2 模型训练结果二

说明：采用的 kernel 为 linear 函数。可以看出识别准确率较低为 67.23%。



```
Run SVM
/Volumes/BOOTCAMP/袁昊男/学习/大二下/机器学习技术与应用/实验/实验3数据集和实验指导书/venv/bin/python
/Volumes/BOOTCAMP/袁昊男/学习/大二下/机器学习技术与应用/实验/实验4数据集和实验指导书/ex_4/SVM.py
2020-05-20 22:23:36
9923 of 10000 test values correct.
2020-05-20 22:34:08
Process finished with exit code 0
```

图 9.3 模型训练结果三

说明：经过 GridSearchCV()函数进行参数调整后，模型较采用 RBG 作为 kernel 时识别准确率更高，为 99.23%。



## 2、手写数字识别测试

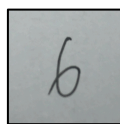


图 9.4 待识别手写数字图片

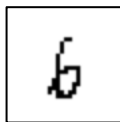


图 9.5 经过图像预处理后的手写数字图片

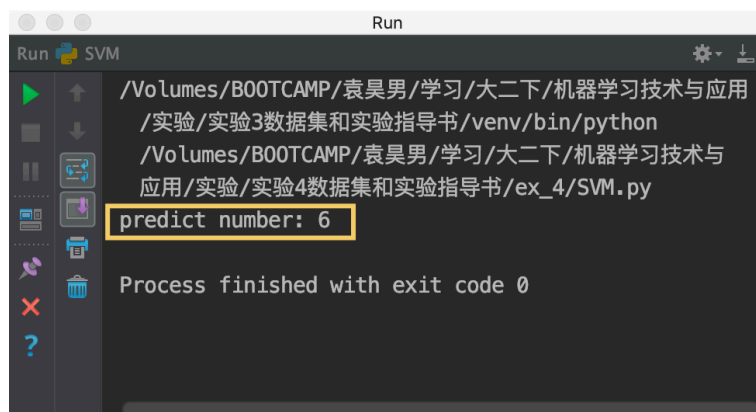


图 9.6 SVM 模型识别结果

## 十、实验结论

本实验通过 SVM 支持向量机学习算法模型，结合 MNIST 数据集及 sklearn 库，完成了手写数字识别的 SVM 模型训练与数字预测。调用 `svm.SVC()` 函数时，使用不同的参数值得出的模型识别准确率不同，还可以使用 `GridSearchCV()` 函数对模型进行参数调整。最差的准确率为 67.23%，较高的准确率为 99.23%。使用该模型对任意手写数字进行识别，识别结果准确。

## 十一、总结及心得体会

思考题：理解 SVM 中关键的 SMO 算法代码实现的具体流程，并绘制算法流程图。

解答：

序列最小优化(Sequential Minimal Optimization, SMO)是一种公认流行的用于训练 SVM 的算法，其本质是将一个优化问题分解为多个小优化问题来求解。这些被分解的小问题往往更容易计算，而且对其按既定顺序求解后的结果与将其作为一个整体求解的结果是完全等价的，因此一般情况下 SMO 有着更低的时间

和空间复杂度。SMO 算法的原理是每次循环中选择 2 个  $\alpha$  ( $\alpha_1, \alpha_2$ ) 进行优化处理，如果  $\alpha_1$  确定， $\alpha_2$  也随即可得；这 2 个  $\alpha$  必须在间隔边界之外，或者还未经过区间化处理或者不在边上。SMO 算法的步骤执行过程可表述如下：

- (1) 输入训练集  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ ，其中  $x_i \in X = R^n$ ， $y_i \in Y = \{-1, 1\}$ ， $i = 1, 2, \dots, N$ ，精度  $\varepsilon$ 。
- (2) 取初值  $\alpha^{(0)} = 0$ ，令  $k = 0$ 。
- (3) 选取优化变量  $\alpha_1^{(k)}$ ， $\alpha_2^{(k+1)}$ ，更新  $\alpha^k$  为  $\alpha^{(k+1)}$ ；最优解问题的内容定义如下所示：

$$\begin{aligned} \min_{\alpha_1, \alpha_2} W(\alpha_1, \alpha_2) &= \frac{1}{2} K_{11} \alpha_1^2 + \frac{1}{2} K_{22} \alpha_2^2 + y_1 y_2 K_{12} \alpha_1 \alpha_2 - (\alpha_1 + \alpha_2) \\ &\quad + y_1 \alpha_1 \sum_{i=3}^N y_i \alpha_i K_{i1} + y_2 \alpha_2 \sum_{i=3}^N y_i \alpha_i K_{i2} \\ \text{s.t. } \alpha_1 y_1 + \alpha_2 y_2 &= - \sum_{i=3}^N y_i \alpha_i = \xi \quad (0 \leq \alpha_i \leq C, i = 1, 2) \end{aligned}$$

其中， $K_{ij} = k(x_i, x_j)$ ， $i, j = 1, 2, \dots, N$ ， $\xi$  是常数。

- (4) 若在精度  $\varepsilon$  范围内满足停机条件，即：

$$\sum_{i=1}^N \alpha_i y_i = 0 \quad (0 \leq \alpha_i \leq C, i = 1, 2, \dots, N),$$

$$y_i g(x_i) = \begin{cases} \geq 1 & \{x_i | \alpha_i = 0\} \\ = 1 & \{x_i | 0 < \alpha_i < C\} \\ \leq 1 & \{x_i | \alpha_i = C\} \end{cases},$$

其中， $g(x_i) = \sum_{j=1}^N \alpha_j y_j K(x_i, x_j) + b$ ，则转(5)，否则令  $k = k + 1$ ，转(3)。

- (5) 取  $\alpha = \alpha^{(k+1)}$ ，输出近似解  $\alpha$ 。

基于前述给出的 SMO 算法分析，就可以得出 SMO 算法实现的一般方法。进一步，将推得该通用方法的伪代码设计逻辑流程如图 11.1 所示。对于线性分类问题，一般的线性分类支持向量机已经是一种成功有效的学习方法。但是，对于数据集分类为非线性的时候，这种线性分类的支持向量机将不再适用。SVM 优化中，所有的运算都可以写成内积的形式，即 2 个向量相乘之后得到的单个标量或者数值。此时，可以使用核方法，利用核函数表示将输入从输入空间映射到特征空间得到的特征向量之间的内积。通过使用核函数学习的非线性支持向量机，等价于隐式地在高维的特征空间中学习线性支持向量机。

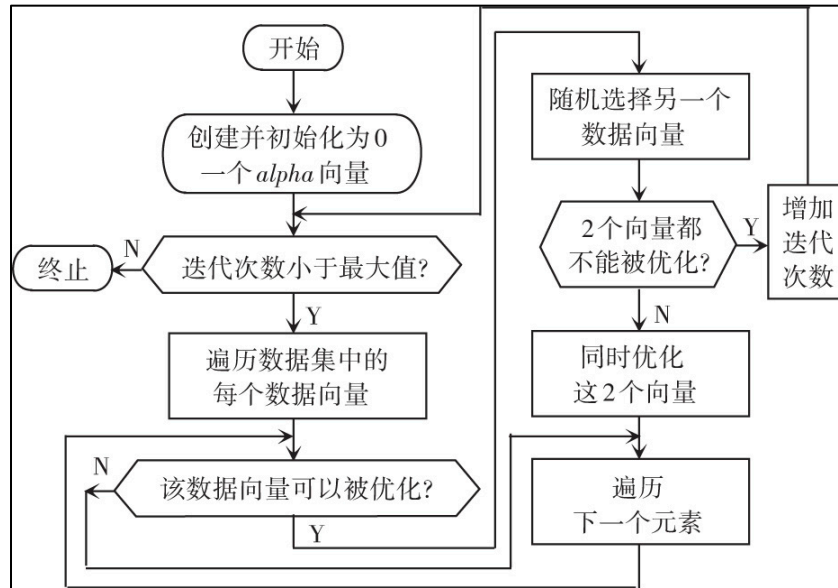


图 11.1 SMO 算法流程图

SMO 算法代码:

```

1. from numpy import *
2.
3. #加入核函数
4. def kernelTrans(X,A,kTup):
5.     m,n = shape(X)
6.     k = mat(zeros((m,1)))
7.     if kTup[0] == 'lin':
8.         k = X * A.T
9.     elif kTup[0] == 'rbf':
10.        for j in range(m):
11.            deltaRow = X[j,:] - A #每一行减去 A, 在自己乘
12.            k[j] = deltaRow * deltaRow.T
13.            k = exp(k/(-1 * kTup[1] ** 2)) #就是利用的公式
14.        return k
15.
16. #存储变量的类
17. class opStruct():
18.     def __init__(self,dataMatIn,classLabels,C,toler,kTup):
19.         self.X = dataMatIn #数据
20.         self.labelMat = classLabels #标签
21.         self.C = C #容忍度
22.         self.toler = toler #误差的容忍度
23.         self.m = shape(dataMatIn)[0] #数据的个数
24.         self.alphas = mat(zeros((self.m,1))) #alpha 值, 每个数据对应一
            个 alpha
25.         self.b = 0 # 常数项
26.         self.eCache = mat(zeros((self.m,2))) #保存误差和下标
27.         self.k = mat(zeros((self.m,self.m)))
28.         for i in range(self.m):
29.             self.k[:,i] = kernelTrans(self.X,self.X[i,:],kTup) #书上 128 页
                的核矩阵
30.
31. #保证 alpha 必须在范围内
32. def clipAlpha(ai,H,L):
33.     if ai > H:
34.         ai = H

```

```

35.     elif ai < L :
36.         ai = L
37.     return ai
38.
39. #随机选择第二个不同的 alpha
40. def selectJrand(i,oS):
41.     j = i
42.     while i == j:
43.         j = int(np.random.uniform(0,oS.m))
44.     return j
45.
46. #计算误差 书上 124 页
47. def calcEk(oS,k):
48.     #fXk = float(multiply(oS.alphas * oS.labelMat).T*oS.X * oS.X[k,:].T + oS.b) #预测值
49.     #利用核函数
50.     fXk = float(multiply(oS.alphas * oS.labelMat).T*oS.k[:,k] + oS.b) #预测值
51.     Ek = fXk - oS.labelMat[k] #误差值
52.     return Ek
53. #选择第二个 alpha 并且相差最大的
54. def selectJ(i,oS,Ei):
55.     maxK = -1
56.     maxDelaE = 0
57.     Ej = 0
58.     oS.eCache[i] = [1,Ei]
59.     validEcacheList = nonzero(oS.eCache[:,0].A)[0]
60.     if len(validEcacheList) > 0:
61.         for k in validEcacheList:
62.             if k == i: #取不同的 alpha
63.                 continue
64.             Ek = calcEk(oS,k) #计算 k 的与测试与真实值之间的误差
65.             deltaE = abs(Ei - Ek) #找与 Ei 距离最远的
66.             if maxDelaE < deltaE:
67.                 maxDelaE = deltaE #
68.                 maxK = k #与 Ei 差别最大的 K
69.                 Ej = Ek #K 的误差
70.         return maxK,Ej
71.     else:
72.         j = selectJrand(i,oS)
73.         Ej = calcEk(oS,j) #计算预测值和真实值的误差
74.     return j,Ej
75.
76. #更新误差
77. def updateEk(oS,k):
78.     Ek = calcEk(oS,k)
79.     oS.eCache[k] = [1,Ek]
80.
81. #优化
82. def innerL(i,oS):
83.     Ei = calcEk(oS,i)
84.     #在误差允许的范围外, 如果小于规定的误差, 就不需要更新了
85.     if ((oS.labelMat[i] * Ei) < -oS.toler and oS.alphas[i] < oS.C) or \
86.        ((oS.labelMat[i] * Ei) > oS.toler and oS.alphas[i] > 0):
87.         j,Ej = selectJ(i,oS,Ei) #选择另一个 alphaj 和预测值与真实值的差
88.         alphaIold = oS.alphas[i].copy() #复制 alpha, 因为后边会用到
89.         alphaJold = oS.alphas[j].copy()
90.

```

```

91.         if (oS.labelMat[i] != oS.labelMat[j]): #两个类别不一样 一个正类 一个
    负类
92.             L = max(0, oS.labelMat[j] - oS.labelMat[i]) # 约束条件 博客里
    有
93.             H = min(oS.C, oS.C + oS.alphas[j] - oS.alphas[i])
94.         else:
95.             L = max(0, oS.alphas[j] + oS.alphas[i] - oS.C)
96.             H = min(oS.C, oS.alphas[j] + oS.alphas[i])
97.
98.         if L == H:
99.             print('L == H')
100.            return 0
101.            # eta = 2.0 * oS.X[i,:] * oS.X[j,:].T - oS.X[i,:] * oS.X[i,:].T\
102.                - oS.X[j,:]* oS.X[j,:].T
103.            #利用核函数
104.            eta = 2.0 * oS.k[i,j] - oS.k[i,i] - oS.k[j,j]
105.            if eta > 0:
106.                return 0
107.            oS.alphas[j] -= oS.labelMat[j] * (Ei - Ej)/eta #就是按最后的公式
    求解
108.            oS.alphas[j] = clipAlpha(oS.alphas[j], H, L) #在 L, H 范围内
109.            updateEk(oS, j)
110.
111.            if (oS.alphas[j] - alphaJold) < 0.0001:
112.                return 0
113.
114.            oS.alphas[i] += oS.labelMat[j] * oS.labelMat[i] * (alpha-
    Jold - oS.alphas[j])
115.            updateEk(oS, i)
116.            #也是用最后的求解 b 的公式
117.            # b1 = oS.b - Ei - oS.label-
    Mat[i] * oS.X[i,:] * oS.X[i,:].T * (oS.alphas[i] - alphaIold)\
118.                # - oS.label-
    Mat[j] * oS.X[i,:] * oS.X[j,:].T * (oS.alphas[j] - alphaJold)
119.            # b2 = oS.b - Ej - oS.label-
    Mat[i] * oS.X[i,:] * oS.X[j,:].T * (oS.alphas[i] - alphaIold)\
120.                # - oS.label-
    Mat[j] * oS.X[j,:] * oS.X[j,:].T * (oS.alphas[j] - alphaJold)
121.            #利用核函数的
122.            b1 = oS.b - Ei - oS.labelMat[i] * oS.k[i,i] * (oS.alphas[i] - al-
    phaIold) - oS.labelMat[j] * oS.k[i,j] * (oS.alphas[j] - alphaJold)
123.            b2 = oS.b - Ej - oS.labelMat[i] * oS.k[i,j] * (oS.alphas[i] - al-
    phaIold) - oS.labelMat[j] * oS.k[j,j] * (oS.alphas[j] - alphaJold)
124.
125.
126.            #跟新 b
127.            if oS.alphas[i] < oS.C and oS.alphas[i] > 0:
128.                oS.b = b1
129.            elif oS.alphas[j] < oS.C and oS.alphas[j] > 0:
130.                oS.b = b2
131.            else:
132.                oS.b = (b1 + b2)/2.0
133.            return 1
134.        else:
135.            return 0
136.
137. #完整版的外循环

```

```

138. def smoP(dataMatIn,classLabels,C,toler,maxIter,kTup = ('lin',0)):
139.     oS = opStruct(mat(dataMatIn),mat(classLabels).T,C,toler)
140.     iter = 0
141.     entireSet = True
142.     alphaPairedChanged = 0
143.     while (iter < maxIter) and ((alphaPairedChanged > 0) or (entire-
Set)):
144.         alphaPairedChanged = 0
145.         if entireSet:
146.             # 遍历所有的数据 进行更新
147.             for i in range(oS.m):
148.                 alphaPairedChanged += innerL(i,oS)
149.             iter += 1
150.         else:
151.             nonBoundIs = nonzero((oS.alphas.A > 0) * (oS.al-
phas.A < oS.C))[0]
152.             for i in nonBoundIs:
153.                 alphaPairedChanged += innerL(i,oS)
154.             iter += 1
155.
156.         if entireSet:
157.             entireSet = False
158.         elif (alphaPairedChanged == 0):
159.             entireSet = True
160.     return oS.b,oS.alphas
161. #计算 W
162. def calcWs(alpha,dataArr,classLabels):
163.     X = mat(dataArr)
164.     labelMat = mat(classLabels).T #变成列向量
165.     m,n = shape(X)
166.     w =zeros((n,1)) #w 的个数与 数据的维数一样
167.     for i in range(m):
168.         w += multiply(alpha[i] * labelMat[i],X[i,:].T) #alpha[i] * label-
Mat[i]就是一个常熟 X[i,:]每（行）个数据，因为 w 为列向量，所以需要转置

```

## 十二、对本实验过程及方法、手段的改进建议

本实验在 SMO 算法理解上难度比较大，学生需要花费较多的课下时间来学习相关的知识。本实验可以着重考虑对任意手写数字图片识别的实现。

报告评分：

指导教师签字：