电子科技大学

计算机科学与工程学院

实 验 指 导 书

（实验）课程名称 人工智能综合实验 I

电子科技大学教务处制表

实验四-II　支持向量机实现手写文字识别

一、 实验目的和任务

学习并掌握支持向量机（SVM）算法，理解基于最大间隔数据分类原理，以及学习如何寻找最大间隔及分类器求解的优化问题。掌握非线性支持向量机的核函数理解和应用。

二、 实验原理

2.1 支持向量机简介

支持向量机是一类按监督学习方式对数据进行二元分类的广义线性分类器，其决策边界是对学习样本求解的最大间隔超平面，而所谓的间隔就是指样本到超平面的距离。我们按照间隔尽可能大的形式来设置超平面，这样可以在犯错或者是在有限数据上训练分类器时，训练得到的分类器能够具有更好的鲁棒性。支持向量就是距离分隔超平面最近的那些点。由此问题转化为求解最大化支持向量到分割面距离的问题，需要找到这个问题的优化求解方法。一般最常用的算法是序列最小优化算法（SMO 算法）。以上都是针对于线性可分的数据，那么当数据是非线性可分时，就需要将数据转换成分类器易于理解的表达，这种处理可以引入核函数来完成。

核函数可以实现将数据从一个空间映射到另一个空间，在此空间内获得新的特征表示来表示数据，该新特征更易于分类器理解，在此基础上，根据支持向量机原理实现数据分类。如果核函数是非线性的，那么就是非线性数据分类。一般这种映射都是将数据从低维空间映射

到高维空间中，在高维空间中，提高数据可分性。核函数不仅仅应用于支持向量机，很多其他机器学习方法也会用到核函数，进行非线性分类。常用的核函数有线性核、高斯核（径向基核函数）、多项式核、拉普拉斯核以及 Sigmoid 核等。

2.2 算法 demo

Demo 1: 实现 SMO 算法

```
from numpy import *
from time import sleep
import matplotlib.pyplot as plt

# 读取数据集
def loadDataSet(fileName):
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr = line.strip().split('\t')
        dataMat.append([float(lineArr[0]), float(lineArr[1])])
        labelMat.append(float(lineArr[2]))
    return dataMat,labelMat

# selectJrand 函数根据 i，选择与 i 不同的角标作为另一个 alpha 的角标.
def selectJrand(i,m):
    j=i #希望 alpha 的标号不同
    while (j==i):
        j = int(random.uniform(0,m))
    return j
#根据不同的 yi，调整 alphaj 的范围
def clipAlpha(aj,H,L):
    if aj > H:
        aj = H
    if L > aj:
        aj = L
    return aj

def kernelTrans(X, A, kTup): #通过数据计算转换后的核函数
    m,n = shape(X)
    K = mat(zeros((m,1)))
    if kTup[0]=='lin': #线性核函数
        K = X * A.T
```

```
    elif kTup[0]=='rbf':#高斯核
        for j in range(m):
            deltaRow = X[j,:] - A
            K[j] = deltaRow*deltaRow.T
        K = exp(K/(-1*kTup[1]**2))

    elif kTup[0] == 'laplace':#拉普拉斯核
        for j in range(m):
            deltaRow = X[j,:] - A
            K[j] = deltaRow*deltaRow.T
            K[j] = sqrt(K[j])
        K = exp(-K/kTup[1])

    elif kTup[0] == 'poly':#多项式核
        K = X * A.T
        for j in range(m):
            K[j] = K[j]**kTup[1]

    elif kTup[0] == 'sigmoid':#Sigmoid 核
        K = X * A.T
        for j in range(m):
            K[j] = tanh(kTup[1]*K[j]+kTup[2])

    else: raise NameError('执行过程出现问题 -- \
    核函数无法识别')
    return K
# 统一数据结构，方便之后的函数调用
class optStruct:
    def __init__(self,dataMatIn, classLabels, C, toler, kTup):
    # Initialize the structure with the parameters
        self.X = dataMatIn
        self.labelMat = classLabels
        self.C = C
        self.tol = toler
        self.m = shape(dataMatIn)[0]
        self.alphas = mat(zeros((self.m,1)))
        self.b = 0
        self.eCache = mat(zeros((self.m,2))) #first column is valid flag
        self.K = mat(zeros((self.m,self.m)))
        for i in range(self.m):
            self.K[:,i] = kernelTrans(self.X, self.X[i,:], kTup)
#这里根据公式 E = f(xi) - yi ， 负责计算误差 E
def calcEk(oS, k):
```

```
        fXk = float(multiply(oS.alphas,oS.labelMat).T*oS.K[:,k] + oS.b)
        Ek = fXk - float(oS.labelMat[k])
        return Ek
# 选取第二个 alpha 或是内循环的 alpha 值
def selectJ(i, oS, Ei):
        maxK = -1; maxDeltaE = 0; Ej = 0
        oS.eCache[i] = [1,Ei]
        validEcacheList = nonzero(oS.eCache[:,0].A)[0]
        # print(validEcacheList)
        if (len(validEcacheList)) > 1:
                for k in validEcacheList:
                        if k == i: continue
                        Ek = calcEk(oS, k)
                        deltaE = abs(Ei - Ek)
                        if (deltaE > maxDeltaE):
                                maxK = k; maxDeltaE = deltaE; Ej = Ek
                return maxK, Ej
        else:     #in this case (first time around) we don't have any valid eCache values
                j = selectJrand(i, oS.m)
                Ej = calcEk(oS, j)
        return j, Ej
# 更新误差
def updateEk(oS, k):
        Ek = calcEk(oS, k)
        oS.eCache[k] = [1,Ek]

# SMO 算法的核心部分，请同学们自行完成
def innerL(i, oS):
        pass
# SMO 算法主体部分，输入包括数据集，标签，松弛因子，容忍度，最大迭代数与核参数。
#请同学们自行完成此部分
def smoP(dataMatIn, classLabels, C, toler, maxIter,kTup=('rbf', 1.3)):
    return oS.b,oS.alphas       # 返回值为 alpha 和截距 b

# 根据计算的 alpha 推导出 W 的值
def calculateW(alphas,dataArr,labelArr):
        alphas, dataMat, labelMat = array(alphas), array(dataArr), array(labelArr)
        sum = 0
        for i in range(shape(dataMat)[0]):
                sum += alphas[i]*labelMat[i]*dataMat[i].T
        print(sum)
        return sum
# 测试，可以选取不同核函数查看不同核函数下支持向量机的性能。
def testRbf(k1=1.3):
```

```
    dataArr,labelArr = loadDataSet('testSetRBF.txt')
    b,alphas = smoP(dataArr, labelArr, 200, 0.0001, 10000, ('rbf', k1))

    datMat=mat(dataArr); labelMat = mat(labelArr).transpose()
    svInd=nonzero(alphas.A>0)[0]
    sVs=datMat[svInd]
    labelSV = labelMat[svInd];
    print ("there are %d Support Vectors" % shape(sVs)[0])
    m,n = shape(datMat)
    errorCount = 0
    for i in range(m):
        kernelEval = kernelTrans(sVs,datMat[i,:],('rbf', k1))
        predict=kernelEval.T * multiply(labelSV,alphas[svInd]) + b


        if sign(predict)!=sign(labelArr[i]): errorCount += 1
    print ("the training error rate is: %f" % (float(errorCount)/m))

    dataArr,labelArr = loadDataSet('testSetRBF2.txt')
    errorCount = 0
    datMat=mat(dataArr); labelMat = mat(labelArr).transpose()
    m,n = shape(datMat)
    for i in range(m):
        kernelEval = kernelTrans(sVs,datMat[i,:],('rbf', k1))
        predict=kernelEval.T * multiply(labelSV,alphas[svInd]) + b
        if sign(predict)!=sign(labelArr[i]): errorCount += 1
    print ("the test error rate is: %f" % (float(errorCount)/m))
# 可视化支持向量，此部分可参考选做
def plot_point(filename,alphas,dataMat):
      filename = filename
      fr = open(filename)
      X1 = [];y1 = []
      X2 = [];y2 = []
      for line in fr.readlines():
            lineArr = line.strip().split('\t')
            if float(lineArr[-1]) == 1:
                  X1.append(lineArr[0])
                  y1.append(lineArr[1])
            elif float(lineArr[-1]) == -1:
                  X2.append(lineArr[0])
                  y2.append(lineArr[1])
      plt.scatter(X1[:],y1[:],c='y',s=50)
      plt.scatter(X2[:],y2[:],c='b',s=50)
      for i, alpha in enumerate(alphas):
```

```
            if abs(alpha) > 0:
                x, y = dataMat[i]
                plt.scatter(x, y, s=100, c = '', alpha=0.5, linewidth=1.5, edgecolor='red')
    plt.show()
# 总体运行
if __name__ == "__main__":
    dataArr,labelArr = loadDataSet('testSetRBF.txt')
    b,alphas = smoP(dataArr,labelArr,0.6,0.001,40)
    testRbf()
    plot_point('testSetRBF.txt',alphas,dataArr)
    calculateW(alphas,dataArr,labelArr)
```

demo2：上面的方法是通过自己实现的方式来构建 SVM，同学们可自行选做此部分，此部分使用 sklearn 数据包中自带的 svm 函数完成，需要同学们自行学习 sklearn 中相关函数的使用。这里仅提供部分参考代码。

```
from sklearn import svm
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import colors
from sklearn.model_selection import train_test_split
def iris_type(s):
    it = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
    return it[s]
def show_accuracy(y_hat, y_test, param):
    pass

path = '../iris.data' # 数据文件路径
data = np.loadtxt(path, dtype=float, delimiter=',', converters={4: iris_type})

x, y = np.split(data, (4,), axis=1)
x = x[:, :2]
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=1, train_size=0.6)
# 可以通过修改 kernel 参数来实现不同核函数的验证
# clf = svm.SVC(C=0.1, kernel='linear', decision_function_shape='ovr')
clf = svm.SVC(C=0.8, kernel='rbf', gamma=20, decision_function_shape='ovr')
clf.fit(x_train, y_train.ravel())

print clf.score(x_train, y_train)  # 精度
y_hat = clf.predict(x_train)
show_accuracy(y_hat, y_train, '训练集')
print clf.score(x_test, y_test)
y_hat = clf.predict(x_test)
show_accuracy(y_hat, y_test, '测试集')
```

```
print 'decision_function:\n', clf.decision_function(x_train)
print '\npredict:\n', clf.predict(x_train)

x1_min, x1_max = x[:, 0].min(), x[:, 0].max()  # 第 0 列的范围
x2_min, x2_max = x[:, 1].min(), x[:, 1].max()  # 第 1 列的范围
x1, x2 = np.mgrid[x1_min:x1_max:200j, x2_min:x2_max:200j]  # 生成网格采样点
grid_test = np.stack((x1.flat, x2.flat), axis=1)  # 测试点

mpl.rcParams['font.sans-serif'] = [u'SimHei']
mpl.rcParams['axes.unicode_minus'] = False

cm_light = mpl.colors.ListedColormap(['#A0FFA0', '#FFA0A0', '#A0A0FF'])
cm_dark = mpl.colors.ListedColormap(['g', 'r', 'b'])

# print 'grid_test = \n', grid_test
grid_hat = clf.predict(grid_test)          # 预测分类值
grid_hat = grid_hat.reshape(x1.shape)  # 使之与输入的形状相同

alpha = 0.5
plt.pcolormesh(x1, x2, grid_hat, cmap=cm_light)      # 预测值的显示
# plt.scatter(x[:, 0], x[:, 1], c=y, edgecolors='k', s=50, cmap=cm_dark)  # 样本
plt.plot(x[:, 0], x[:, 1], 'o', alpha=alpha, color='blue', markeredgecolor='k')
plt.scatter(x_test[:, 0], x_test[:, 1], s=120, facecolors='none', zorder=10)  # 圈中测试集样本
plt.xlabel(u'花萼长度', fontsize=13)
plt.ylabel(u'花萼宽度', fontsize=13)
plt.xlim(x1_min, x1_max)
plt.ylim(x2_min, x2_max)
plt.title(u'鸢尾花 SVM 二特征分类', fontsize=15)
plt.show()
```

三、 实验内容

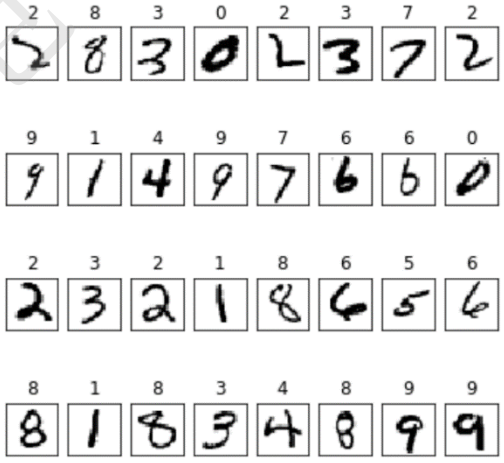
采用 demo1 或者 demo2 的方法，实现基于核函数的 SVM 算法，对手写数据实现分类。手写数据集这里提供两种，MNIST 数据集和 trainingDigits 数据集，其中 trainingDigits 数据集数量适中，建议在手提电脑和普通台式电脑上运行 SVM 算法时采用该数据集。

1） MNIST 手写数据集

MNIST 数据集来自美国国家标准与技术研究所，National Institute of Standards and Technology (NIST). 训练集 (training set) 由来自 250 个不同人手写的数字构成，包含数字图片和标签。MNIST 数据集可在 http://yann.lecun.com/exdb/mnist/ 获取，它包含了四个部分:

- Training set images: train-images-idx3-ubyte.gz (9.9 MB，解压后 47 MB，包含 60,000 个样本)
- Training set labels: train-labels-idx1-ubyte.gz (29 KB，解压后 60 KB，包含 60,000 个标签)
- Test set images: t10k-images-idx3-ubyte.gz (1.6 MB，解压后 7.8 MB，包含 10,000 个样本)
- Test set labels: t10k-labels-idx1-ubyte.gz (5KB，解压后 10 KB，包含 10,000 个标签)

数据如下图所示：



2） trainingDigits 手写数据集

trainingDigits 手写数据集包括 0-9 数字字符，每个字符有 200 左

右样本组成，每个样本由 0，1 像素值表示的数字图片和标签组成。

四、 实验报告要求

根据实验内容按步骤完成实验，并给出最终实验的实验结果，在选取不同核函数下的分类准确率展示，以及可选做完成分类结果的可视化。分析实验结果，总结实验收获和经验。

五、 实验仪器设备

机房电脑一台，编程平台为 Anaconda 下 Spyder 编辑器。