



西安电子科技大学
XIDIAN UNIVERSITY

模式识别大作业五

支持向量机(SVM)

任俊杰

1702052

17170120015

模式识别 支持向量机(SVM)

一、支持向量机算法介绍

1.1 SVM 的理论基础

传统的统计模式识别方法只有在样本趋向无穷大时,其性能才有理论的保证。这就是经验风险最小化理论,比如人工神经网络。统计学习理论(STL)研究有限样本情况下的机器学习问题。SVM 的理论基础就是**统计学习理论**。

根据统计学习理论,学习机器的实际风险由**经验风险值**和**置信范围值**两部分组成。而基于经验风险最小化准则的学习方法只强调了训练样本的经验风险最小误差,没有最小化置信范围值,因此其推广能力较差。Vapnik 提出的支持向量机(Support Vector Machine, SVM)以训练误差(经验风险最小)作为优化问题的约束条件,以置信范围值最小化(推广能力)作为优化目标,即 SVM 是一种基于结构风险最小化准则的学习方法,其推广能力明显优于一些传统的学习方法。

由于 SVM 的求解最后转化成待约束的二次规划问题的求解,由于目标和约束是凸函数,因此 SVM 的解是**全局唯一的最优解**。

SVM 在解决**小样本**、**非线性及高维**模式识别问题中表现出许多特有的优势,它能够获取全局最优解,并能够推广应用到函数拟合等其他机器学习问题中。

1.2 线性判别函数和判别面

(1) 线性判别函数

一个**线性判别函数**(discriminant function)是指由 x 的各个分量的线性

组合而成的函数：

$$g(x) = w^T x + w_0$$

两类情况：对于两类问题的决策规则为：

如果 $g(x) > 0$ ，则判定 x 属于 $C1$

如果 $g(x) < 0$ ，则判定 x 属于 $C2$

如果 $g(x) = 0$ ，则可以将 x 任意分到某一类或者拒绝判定。

(2) 判别面

方程 $g(x) = 0$ 定义了一个判定面，它把归类于 $C1$ 的点与归类于 $C2$ 的点分开来，这个判定面称为**判别面**。当 $g(x)$ 是线性函数时，这个平面被称为“超平面” (hyperplane)。

当 x_1 和 x_2 都在判定面上时： $w^T x_1 + w_0 = w^T x_2 + w_0$ ，或者 $w^T (x_1 - x_2) = 0$ ，这表明 w 和超平面上任意向量正交，并称 w 为超平面的法向量。
 $x_1 - x_2$ 表示超平面上的一个向量。

可以求解出特征空间中某点 x 到超平面的距离为：

$$r = \frac{g(x)}{\|w\|}$$

总之：

线性判别函数利用一个超平面把特征空间分隔成两个区域。

超平面的方向由法向量 w 确定，它的位置由阈值 w_0 确定。

判别函数 $g(x)$ 正比于 x 点到超平面的代数距离（带正负号）。当 x 点在超平面的正侧时， $g(x) > 0$ ；当 x 点在超平面的负侧时， $g(x) < 0$

(3) 广义线性判别函数

二次判别函数 $g(x) = (x - a)(x - b)$ 可写成如下的一般形式:

$$g(x) = c_0 + c_1x + c_2x^2$$

如果选择 $x \rightarrow y$ 的映射, 则可以把二次判别函数化为关于 y 的线性函数:

$$g(x) = a^T y = \sum_{i=1}^3 a_i y_i$$

其中:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}, a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}$$

y 称为**广义线性判别函数**, a 叫做广义权向量。

一般地, 对于任意高次判别函数, 都可以通过适当的变换, 化为广义线性判别函数来处理, $a^T y$ 不是 x 的线性函数, 但却是 y 的线性函数。这样我们就可以利用线性判别函数的简单性来解决复杂问题。同时带来的问题是, 维数大大增加了, 这将使问题很快陷入所谓的“维数灾难”。

(4) 设计线性分类器

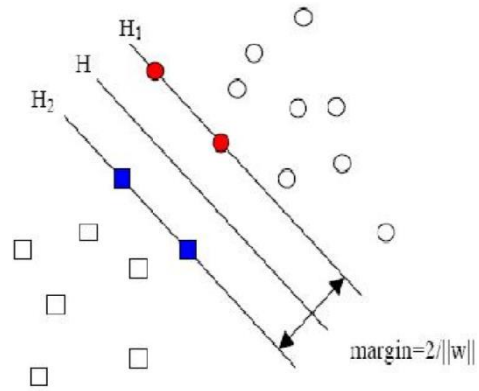
所谓设计线性分类器, 就是利用训练样本集建立线性判别函数 $g(x) = w^T x + w_0$ 或者广义线性判别函数 $g(x) = a^T y$ 。设计线性分类器的过程实质是寻找较好的 w 和 w_0 或 a 的过程, 最好的结果往往出现在准则函数的极值点上。这样, 设计线性分类器的问题就转化为利用训练集寻找准则函数的极值点 w^* 和 w_0^* 或 a^* 的问题了。

常见的准则函数有: Fisher 准则函数、感知准则函数、最小错分样本准则函数、最小平方误差准则函数、随机最小错误率线性判别准则函数。

1.3 最优分类超平面

SVM 是从线性可分情况下的最优分类面发展而来的, 基本思想可用图二维情况说明。图中, 方形点和圆形点代表两类样本, H 为分类线, H_1, H_2 分别

为过各类中离分类线最近的样本且平行于分类线的直线, 它们之间的距离叫做分类间隔(margin)。



最优分类超平面 (optimal separating hyperplane) 是指如果一个超平面能够将训练样本没有错误的分开, 而且两类训练样本中离超平面最近的样本与超平面之间的距离最大。

两类样本中离分类面最近的样本到分类面的距离称作**分类间隔** (margin)。

规范化的分类超平面: $y_1[w^T x_1 + b] \geq 1$ 。 $g(x) = 1$ 和 $g(x) = -1$ 就是过两类中各自离分类面最近的样本且与分类面平行的两个边界超平面。规划化, 使得离分类面最近的样本满足: $|g(x)| = 1$, 所以分类间隔为 $\frac{2}{||w||}$ 。**求解最优超平面问题就可以表示为:**

$$\begin{cases} \min_{w,b} \frac{1}{2} ||w||^2 \\ s.t. y_i[w^T x_i + b] - 1 \geq 0 \end{cases}$$

过两类样本中离分类面最近的点且平行于最优超平面的超平面 H1 和 H2 上的训练样本, 就是使上式等号成立的样本, 也叫**支持样本**。

这是一个不等式约束的优化问题, 可以通过拉格朗日法求解。可以转化为:

$$\min_{w,b} \max_{\alpha} L(w, b, \alpha) = \frac{1}{2} w^T w - \sum_{i=1}^N \alpha_i \{y_i[w^T x_i + b] - 1\}$$

由于这是一个二次凸规划问题, 由于目标函数和约束条件都是凸的, 根据最优化理论, 这一问题存在唯一全局最小解。这里, 它是满足 KKT 条件: $y_i[w^T x_i + b] - 1 = 0$ 。得到的最优解 (最优分类面的决策函数) 为:

$$f(x) = \text{sgn}\{g(x)\} = \text{sgn}\{(\mathbf{w}^* \cdot \mathbf{x}) + b^*\} = \text{sgn}\left\{\sum_{i=1}^N \alpha_i^* y_i (\mathbf{x}_i \cdot \mathbf{x}) + b^*\right\}$$

α_i 是支持向量系数，最优超平面的权重向量等于训练样本以一定的系数加权后进行线性组合。只有满足优化式等号成立的样本对应的才大于 0，其他样本都等于 0。求和只对少数支持向量进行。

当样本集不是线性可分时，引入松弛因子，使得不等式约束变为：

$y_i[w^T x_i + b] - 1 + \xi_i \geq 0$ ，线性不可分下的优化的目标函数转换为：

$\min_{w,b} \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^N \xi_i$ ，其中， $\sum_{i=1}^N \xi_i$ 越大，错分的样本越多。此时的目标变成一方面让分类间隔尽可能的大，另一方面让错分的样本尽可能的少且错误率尽可能的低。

1.4 支持向量机

线性情况下最优分类面的决策函数为：

$$f(x) = \text{sgn}\{g(x)\} = \text{sgn}\{(\mathbf{w}^* \cdot \mathbf{x}) + b^*\} = \text{sgn}\left\{\sum_{i=1}^N \alpha_i^* y_i (\mathbf{x}_i \cdot \mathbf{x}) + b^*\right\}$$

如果我们对 x 进行非线性变换，记作 $z = \varphi(x)$ ，则新的空间里构造的支撑向量机决策函数为：

$$f(x) = \text{sgn}\{g(x)\} = \text{sgn}\{(\mathbf{w}^\varphi \cdot \mathbf{z}) + b\} = \text{sgn}\left\{\sum_{i=1}^N \alpha_i y_i (\varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x})) + b\right\}$$

变换对支撑向量机的影响是把两个样本在原特征空间中的内积 $(\mathbf{x}_i \cdot \mathbf{x})$ 变成了新空间中的内积 $\varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x})$ 。新空间的内积可以记作：

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_j))$$

也称为核函数。这样，变换空间里的支持向量机就可以写作：

$$f(x) = \text{sgn}\left\{\sum_{i=1}^N \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b^*\right\}$$

从计算角度, 不论 $\varphi(x)$ 所生成的变换空间维数有多高, 这个空间里的线性支持向量机求解都可以在原空间通过核函数 $K(x_i, x_j)$ 进行, 这样就避免了高维空间里的计算, 而且计算核函数的复杂度与计算内积并没有实质性增加。

根据泛函的有关理论, 只要一种核函数满足 Mercer 条件, 它就对应某一变换空间中的内积。因此, 在最优分类面中采用适当的内积函数就可以实现某一非线性变换后的线性分类。

1.5 常用的核函数形式:

用不同核函数可以构造实现输入空间中不同类型的非线性决策面的学习机, 从而导致不同的支持向量算法。在实际问题中, 通常是直接给出核函数。

常用的核函数有:

1) 线性核函数

$$K(x, x_i) = (x, x_i)$$

2) 多项式核函数

$$K(x, x_i) = (s(x, x_i) + c)^d$$

3) 径向基核函数

$$K(x, x_i) = \exp\left(-\frac{\|x - x_i\|^2}{\sigma^2}\right)$$

4) Sigmoid 核函数

$$K(x, x_i) = \tanh(s(x, x_i) + c)$$

1.6 SVM 方法的特点

- 1. 非线性映射是 SVM 方法的理论基础,SVM 利用内积核函数代替向高维空间的非线性映射;
- 2. 对特征空间划分的最优超平面是 SVM 的目标,最大化 分类边际的思想是 SVM 方法的核心;
- 3. 支持向量是 SVM 的训练结果,在 SVM 分类决策中起决定作用的是支持向量.
- 4. 专门针对有限样本的情况，其目标是得到现有信息下的最优解而不仅仅是样本数趋于无穷大时的最优值；
- 5. 其解是全局最优解，解决了在神经网络方法中无法避免的局部极值问题；
- 6. 算法有较好的推广能力，同时它巧妙地避免了 “ 维数灾难” 。
- 7. 具有较强的非线性处理能力。

二、实验数据集介绍

| YaleB 人脸数据集 | | | |
|-------------|------|-------|-------|
| 类别 | 38 | | |
| 维度 | 1024 | | |
| 数据长度 | 2414 | 不定 | 1 |
| | | | |
| | | 不定 | 38 |

三、实验设置

本次实验是在 Python 环境下实现, 并使用了 Libsvm 库实现对 YaleB 人脸数据集的分类, LIBSVM 是台湾大学林智仁(Lin Chih-Jen)教授等开发设计的一个简单、易于使用和快速有效的 SVM 模式识别与回归的软件包, 他不但提供了编译好的可在 Windows 系列系统的执行文件, 还提供了源代码, 方便改进、修改以及在其它操作系统上应用; 该软件对 SVM 所涉及的参数调节相对比较少, 提供了很多的默认参数, 利用这些默认参数可以解决很多问题; 并提供了交互检验(Cross Validation)的功能。

由于仅使用 Libsvm 进行 SVM 分类的效果很不好 (可能是数据维度较高的原因), 又使用了 sklearn 库进行 PCA 降维再进行 SVM 分类, 达到了不错的效果。

下面阐述具体流程:

1、数据集预处理。由于拿到的数据集是 YaleB_32x32.mat, 是一 matlab 生成 mat 文件, 所以首先对数据集进行处理。使用 loadmat 将 mat 文件中数据读出, 按照一定比例分训练集和测试集, 并将其按照 LIBSVM 需要的数据格式进行保存。

LIBSVM 训练文件和测试文件的格式要求如下:

<label> <index1>:<value1> <index2>:<value2>

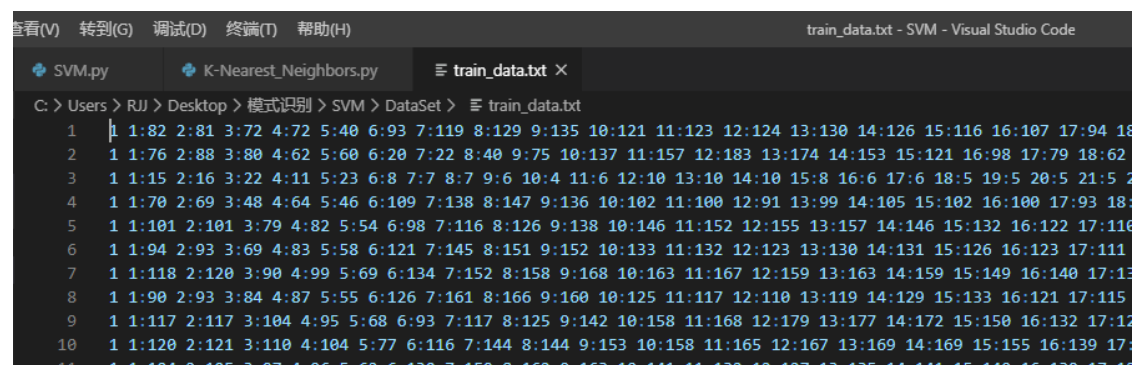
2、读取做好的训练集, 进行训练。利用 libsvm 提供的 svm_read_problem()和 svm_train()直接进行训练, 训练完成后将模型保存。

3、读取做好的测试集, 进行测试, 打印精确度。再 libsvm 中, 调用 svm_predict()即可自动完成测试, 并自动统计精准度。

4、分别使用径向基核、多项式核、线性核进行 20 次训练，比较三种核
对精度影响。由于 libsvm 得效果欠佳，故使用 sklearn 机器学习模块对数据
 进行归一化、PCA 降维，再次进行 SVM 分类。

四、实验结果展示与分析

将 YaleB_32x32.mat 文件转化为 LIBSVM 所要求的格式后，数据集文件
 如下所示：



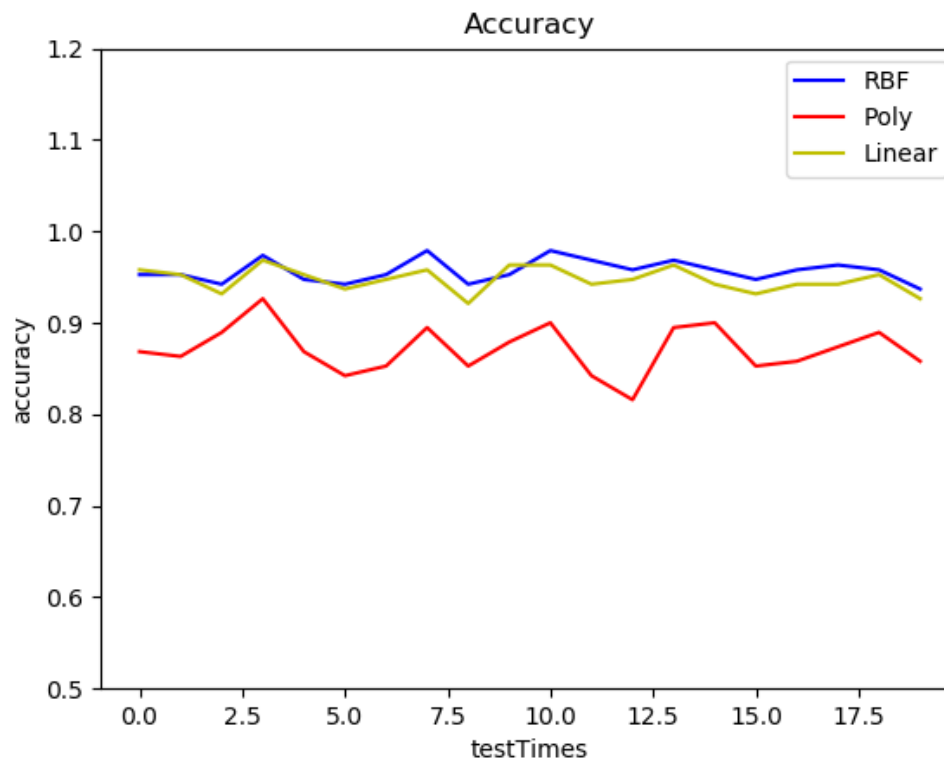
然后使用 Libsvm 直接对这 1024 维 38 类数据进行训练并分类，结果如
 下：

从上图可以看到，只有很小一部分分类正确，多次检查发现是在测试得时
 候，所有的测试样本均被预测到了一类，故仅有一类分类正确，由于没有找
 到具体原因，无奈之下只好放弃使用 libsvm，又使用 sklearn 重新进行相关
 处理，最后使用 sklearn，对三种核函数训练 30 次得到如下结果：

三种核函数得训练 30 次的平均准确率：

```
径向基核,Accuracy_AVE: 0.9565789473684211
多项式核,Accuracy_AVE: 0.8710526315789473
线性核,Accuracy_AVE: 0.9471052631578948
```

变化趋势：



从上图可以看出，使用多项式核函数的准确率最差，在 90%以下，使用径向基核函数的准确率最高，达 95%以上，可见，核函数的选择对 SVM 的分类影响很大的。

五、Python 代码

5.1 使用 Libsvm

```
from scipy.io import loadmat
from svmutil import *
import random

# 读取数据，并随机分为测试集和训练集
def process_dataset(test_rate):

    m = loadmat("DataSet/YaleB_32x32.mat")
    feature = m['fea']
    label = m['gnd']
    print(m.keys())

    test_Labels = random.sample(range(0,2414-1),int(2414*test_rate))
```

```
f_train = open("DataSet/train_data.txt",'w')
f_test = open("DataSet/test_data.txt",'w')
for i in range(len(feature)):
    if i in test_Labels:
        f = f_test
    else:
        f = f_train

    f.write(str(label[i][0]) + ' ')
    for j in range(len(feature[i])):
        f.write("%d:%d " %(j+1,int(feature[i][j])))
    f.write('\n')

# 输出所有数据
f = open("DataSet/YaleB.data",'w')
for i in range(len(feature)):
    f.write(str(label[i][0]) + ' ')
    for j in range(len(feature[i])):
        f.write("%5d:%d" %(j+1,int(feature[i][j])))
    f.write('\n')
f.close()

if __name__ == "__main__":
    # process_dataset(1/5)

    y_train,x_train = svm_read_problem("DataSet/train_data.txt")
    y_test,x_test = svm_read_problem("DataSet/test_data.txt")
    model = svm_train(y_train[:,x_train[:,])
    # model = svm_load_model("SVM_trained_model")

    svm_save_model("SVM_trained_model",model)

    p_label, p_acc, p_val = svm_predict(y_test[:, x_test[:, model)
    print(p_label)
```

5.1 使用 sklearn

```
# -*- coding: utf-8 -*-

import numpy as np
from sklearn import svm, datasets
import matplotlib.pyplot as plt
import skimage.io as io
from sklearn import datasets, decomposition, manifold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.decomposition import PCA

# 设定训练样本的数量, m<64
m = 45
# 类别个数
w = 10

# 将一组 64 张的图像转换为 64*32256 的矩阵
def read_pgm(string):
    image_64 = np.zeros(192*168)

    # 读取路径为 string 下全部的 pgm 图片
    coll = io.ImageCollection(string)
    for i in range(len(coll)):
        a = coll[i]
        a = a.astype(float)
        a = a.reshape(1, 192*168)
        image_64 = np.row_stack((image_64, a))
    image_64 = np.delete(image_64, 0, axis=0)
    return image_64

# 给样本矩阵加标签
def add_label(matrix, n):
    a = np.full(64, 1)
    for i in range(2, n+1):
        a = np.hstack((a, np.full(64, i)))
    matrix = np.hstack((matrix, a.reshape(64*w, 1)))
    return matrix

# 对样本作 L2 归一化
def maxminnorm(array):
```

```
maxcols=array.max(axis=0)
mincols=array.min(axis=0)
data_shape = array.shape
data_rows = data_shape[0]
data_cols = data_shape[1]
t=np.empty((data_rows,data_cols))
for i in range(data_cols):
    t[:,i]=(array[:,i]-mincols[i])/(maxcols[i]-mincols[i])
return t

# 创建样本矩阵,64 个样本为第一类,有 32556 个特征
def read_image(w):
    string = []
    for i in range(1,w+1):
        if i<10:
            string.append('CroppedYale/yaleB' + '0' + str(i) + '/*.pgm'
        )
        else:
            string.append('CroppedYale/yaleB' + str(i) + '/*.pgm')
    sample = np.zeros(32256)
    for i in string:
        if i!='CroppedYale/yaleB14/*.pgm':
            sample = np.vstack((sample,read_pgm(i)))
    sample = np.delete(sample,0,axis=0)

    return sample

testTime = 20
acc_rbf,acc_poly,acc_linear = [],[],[]
for i in range(testTime):
    # 创建样本矩阵 image_matrix
    image_matrix = read_image(w)

    # L2 归一化
    image_matrix = maxminnorm(image_matrix)

    # 用 PCA 降维
    pca = PCA(n_components=150)
    image_matrix = pca.fit_transform(image_matrix)
    # print(image_matrix)

    # 给样本矩阵加标签
```

```
image_matrix = add_label(image_matrix,w)

# 将样本矩阵按行随机打乱
image_matrix = np.random.permutation(image_matrix)

# 将前 2*m 个样本作为训练集
train = image_matrix[0:w*m,:]

# 将后 64*w-2*m 个样本作为测试集
test = image_matrix[w*m:64*w,:]

# 进行 SVM 模型训练
svc_rbf = svm.SVC(kernel='rbf',C=500,gamma=1e-5)           # 径向基核
svc_poly = svm.SVC(kernel='poly')                         # 多项式核
svc_linear = svm.SVC(kernel='linear')                     # 线性核
svc_rbf.fit(train[:, :train.shape[1]-1],train[:,train.shape[1]-1:].ravel())
svc_poly.fit(train[:, :train.shape[1]-1],train[:,train.shape[1]-1:].ravel())
svc_linear.fit(train[:, :train.shape[1]-1],train[:,train.shape[1]-1:].ravel())

# 对测试集进行测试
result_rbf = svc_rbf.predict(test[:, :train.shape[1]-1])
result_poly = svc_poly.predict(test[:, :train.shape[1]-1])
result_linear = svc_linear.predict(test[:, :train.shape[1]-1])

# 对结果进行准确率计算
count1,count2,count3 = 0,0,0
for i in range(64*w-w*m):
    if test[i,train.shape[1]-1]==result_rbf[i]:
        count1 += 1
    if test[i,train.shape[1]-1]==result_poly[i]:
        count2 += 1
    if test[i,train.shape[1]-1]==result_linear[i]:
        count3 += 1

accuracy_rbf = count1 / (64*w-w*m)
accuracy_poly = count2 / (64*w-w*m)
accuracy_linear = count3 / (64*w-w*m)

acc_linear.append(accuracy_linear)
acc_poly.append(accuracy_poly)
acc_rbf.append(accuracy_rbf)
```

```
print('\n 径向基（RBF）核函数的 accuracy:',accuracy_rbf)
print('多项式（Poly）核函数的 accuracy:',accuracy_poly)
print('线性（Linear）核函数的 accuracy:',accuracy_linear)

print("径向基核,Accuracy_AVE:",np.mean(np.array(acc_rbf)))
print("多项式核,Accuracy_AVE:",np.mean(np.array(acc_poly)))
print("线性核,Accuracy_AVE:",np.mean(np.array(acc_linear)))
plt.title("Accuracy")
plt.xlabel("testTimes")
plt.ylabel("accuracy")
x=[i for i in range(testTime)]
plt.plot(x,acc_rbf,color = 'b',label='RBF')
plt.plot(x,acc_poly,color = 'r',label='Poly')
plt.plot(x,acc_linear,color = 'y',label='Linear')
plt.ylim(0.5,1.2)

plt.legend() # 显示图例

plt.show()
```