实验报告 4

SPI 总线与支持向量机

范皓年 1900012739　信息科学技术学院

2020 年 11 月 6 日

目录

1 实验目的

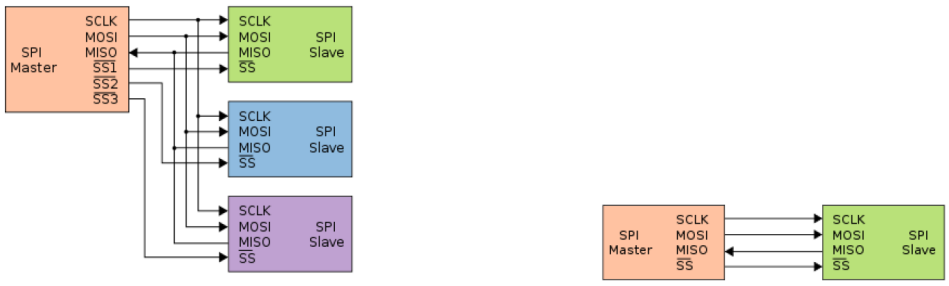
1. 了解 SPI 总线传输的和 OLED 设备基本原理。
2. 编写 Python 程序，访问 SPI 连接的 OLED 设备。
3. 掌握支持向量机的基本原理。使用并初步认识 sklearn 中提供的图像学习的多种组件。

2 实验原理

2.1 SPI 总线及其通信原理

SPI 是串行外设接口（Serial Peripheral Interface）的缩写，是一种高速的，全双工，同步的通信总线，并且在芯片的管脚上只占用四根线，节约了芯片的管脚，同时为 PCB 的布局上节省空间，提供方便。由于这种便利性，SPI 接口标准广泛用于 MCU 与外部设备直接的数据交互，如一些传感器芯片、控制芯片、LCD 等。最早由 Motorola 公司提出，目前由 Freescale 公司进行标准的维护。

SPI 的通信结构比较简明，以主从方式进行工作。一个主设备支持多个 SPI 从设备连接，主要以两种方式：并行连接和链式连接。并行连接方式 SPI 需要多个片选信号线区别多个从设备；链式链接仅需要一根片选线，从设备的输入连接到下一个从设备的输入。



(a) (b)

图 1: 两种连接方式

(a) CPHA=0 SPI 传输时序图 (b) CPHA=1 SPI 传输时序图

图 2: 两种传输方式

当有数据传输时，SPI 主设备产生串行时钟，通过移位寄存器将数据逐 bit 进行传输，根据所配置的传输模式，从设备进行数据采样，完成数据接收。时钟极性（CPOL）和时钟相位（CPHA）用于设定从

设备何时采样数据。两种传输模式中，SCK 与数据的相对位置不同，因此，在两种模式下对于数据采样位置有不同要求，CPHA=1 模式下，采样位置比 SCK 晚半个时钟周期。

SPI 波特率时钟由波特率控制寄存器决定，波特率由如下公式决定：

$$BaudRate = \frac{BusClock}{BaudRateDivisor}$$

其中 BaudRateDivisor 由波特率控制寄存器中的 6 个 bit 决定：

$$BaudRateDivisor = (SPPR + 1) \times 2^{SPR+1}$$

在 BusClock=25MHz，波特速率从 12.21kbps 12.5Mbps，波特率应有如下设置：

SPPR2	SPPR1	SPPR0	SPR2	SPR1	SPR0	BaudRateDivisor	BuadRate
0	0	0	0	0	0	2	12.5MHz

2.2 OLED 设备及其访问

实验中采用 128*64 点阵的 OLED 显示屏，使用 SSD1306 驱动芯片驱动 OLED 屏幕。SSD1306 支持多种数据接口类型如 8bit 68xx/80xx 并行数据接口，SPI 数据接口、I2C 数据接口。具体支持接口类型由相关寄存器配置确定，在实验中我们采用 SPI 数据接口类型。

树莓派与驱动芯片之间采用了 SPI 接口，同时需要对树莓派的相关管脚属性进行配置。在之前的环境配置试验中已经完成了 SPI-dev 的安装。

以下给出了一系列的 OLED 的使用示例：

初始化：

```
import time
import spidev as SPI
import SSD1306
from PIL import Image # 调用相关库文件
RST = 19
DC = 16
bus = 0
device = 0 # 树莓派管脚配置
disp = SSD1306.SSD1306(rst=RST,dc=DC,spi=SPI.SpiDev(bus,device))
```

清屏：

```
disp.begin()
disp.clear()
disp.display() # 初始化屏幕相关参数及清屏
```

显示图片：

```
image = Image.open('happycat.png').resize((disp.width, disp.height),
Image.ANTIALIAS).convert('1')
disp.image(image)
disp.display() # 显示图片
```

显示字符：

```
from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont
import time
import spidev as SPI
import SSD1306
RST = 19
DC = 16
bus = 0
device = 0 # 树莓派管脚配置
disp = SSD1306.SSD1306(rst=RST,dc=DC,spi=SPI.SpiDev(bus,device))

font = ImageFont.load_default() # 字体完成初始化
image = Image.new('RGB',(disp.width,disp.height),'black').convert('1')
draw = ImageDraw.Draw(image)
draw.bitmap((0,0), logo, fill=1)
x = 0; top = 0
draw.text((x,top), 'Hello, Pi!', font=font, fill=255)
disp.image(image)
disp.display() # 显示
```

构建图文：

```
image = Image.new('RGB',(disp.width,disp.height),'black').convert('1')
draw = ImageDraw.Draw(image)
logo = Image.open('/home/pi/pku.png').resize((40,40), \
                    Image.ANTIALIAS).convert('1')
draw.bitmap((0,20), logo, fill = 1) # 在图版上添加绘图
draw.text((30, 0), 'TEST', font=font, fill=255) # 在图版上添加文字
disp.image(image)
disp.display()
```

2.3 Python 的对象封装

Python 语言是一种面向对象的编程语言，和 C++ 等 OO 语言类似，面向对象可以使得代码更模块化，更易于维护，在编写较大的程序当中是非常重要的设计思想。上面代码中所使用的 SSD1306 模块就通过“类”的方式对相关的程序模块进行了封装。在 Python 中定义“类”的关键字是 class，其构造函数用 ___init___ 来定义。例如 SSD1306 中的代码：

```
class SSD1306(object):
"""class for SSD1306 128*64 0.96inch OLED displays."""
    def __init__(self,rst,dc,spi):
```

```
        self.width = 128
        self.height = 64
        ....
```

2.4 机器学习简介：支持向量机与 scikit-learn

对于给定的样本集，分类算法的目的就是要找到一个划分超平面，将不同类别的样本分开，但能实现这个目的的超平面可能有很多，要如何确定最好的那一个呢？支持向量机（Support Vector Machine）就是这样一种寻找最优超平面的算法，它可以保证距离超平面最近的几个训练样本（所谓支持向量）到这个超平面的距离之和最大，也就是获得最鲁棒的解。

对于有些样本集，并没有一个超平面可以完成类别的划分，这时候可以通过所谓核函数（Kernel）将数据集变换的高维空间，再来寻找合适的划分平面。常用的核函数有 linear、poly、rbf、sigmoid 等。不同的核函数对于不同的问题具有不同的效果，在实验中我们将对这一点进行验证比较。利用如下代码可以更换模型的核函数：

```
classifier = svm.SVC(kernel = 'rbf', gamma=0.001)
```

scikit-learn 是一套包含许多机器学习算法的 Python 库，其中集成了大量的机器学习算法组件。包括前面介绍 KMeans 还有这节课介绍的 SVM。并通常和 matplotlib、numpy、pandas、scipy 等库进行协同。以下将给出一个 SVM 与相关 Python 库协同的示例代码，用于实现手写数字的图像识别：

```
import matplotlib.pyplot as plt
from sklearn import datasets, svm, metrics
digits = datasets.load_digits() # 导入手写数字的数据
# 将标签和图像一一对应生成数组
images_and_labels = list(zip(digits.images, digits.target))
# 选取前四个进行灰度绘图
for index, (image, label) in enumerate(images_and_labels[:4]):
    plt.subplot(2, 4, index + 1)
    plt.axis('off')
    # 图谱选择灰度反色，图片密排
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Training: %i' % label)
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))
# 利用SVM进行机器学习
classifier = svm.SVC(gamma=0.001)
classifier.fit(data[:n_samples // 2], digits.target[:n_samples // 2])
expected = digits.target[n_samples // 2:]
predicted = classifier.predict(data[n_samples // 2:])
print("Classification report for classifier %s:\n%s\n"
    % (classifier, metrics.classification_report(expected, predicted)))
print("Confusion matrix:\n%s"
```

```
    % metrics.confusion_matrix(expected, predicted))
images_and_predictions =
    list(zip(digits.images[n_samples // 2:], predicted))
for index, (image, prediction) in enumerate(images_and_predictions[:4]):
    plt.subplot(2, 4, index + 5)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Prediction: %i' % prediction)
plt.show()
```

这段代码的功能是将手写数字的图像进行识别。dataset 类中包含手写数字的图像和标签，程序使用前一半数据进行训练，然后用后一半对训练结果进行验证，并使用 matplotlib 显示了部分训练数据和训练数据。

3 实验内容

3.1 OLED 设备图文显示

利用实验原理中给出的示例代码，完成初始化、清屏。首先利用图文操作，在图版上排布文字组成的欢迎界面。从而完成第一项。

倒计时的实现利用屏幕刷新完成。每一次都重新构建图版，图版上添加图像和文字。文字利用 datetime 组件实现，将当前时间和未来设定时间的 datetime 对象相减得到 datetime_delta 对象，随后转化为两行字符，并添加到图版上。随着当前秒数的变化，反复刷新屏幕获得倒计时显示。

```
import time
from datetime import datetime
import spidev as SPI
import SSD1306
from PIL import Image # 调用相关库文件
from PIL import ImageDraw
from PIL import ImageFont
RST = 19
DC = 16
bus = 0
device = 0 # 树莓派管脚配置

if __name__ == "__main__":
    disp = SSD1306.SSD1306(rst=RST,dc=DC,spi=SPI.SpiDev(bus,device))
    disp.begin()
    disp.clear()
    disp.display() # 初始化屏幕相关参数及清屏

    # welcome interface
```

```
    font = ImageFont.load_default()
    image = Image.new('RGB',(disp.width,disp.height),'black').convert('1')
    draw = ImageDraw.Draw(image)
    x = 30; y = 30;
    draw.text((x,y), 'Hello , Pi!!', font=font, fill=255)
    disp.image(image)
    disp.display()
    time.sleep(1)
    del draw

    # counting down
    double11 = datetime(2020, 11, 11)
    try:
        while True:
            image = Image.new('RGB',(disp.width,disp.height),\
                'black').convert('1')
            draw = ImageDraw.Draw(image)

            logo = Image.open('/home/pi/pku.png').resize((40,40), \
                Image.ANTIALIAS).convert('1')
            draw.bitmap((0,20), logo, fill = 1)
            draw.text((30, 0), str(double11), font=font, fill=255)
            delta = double11 - datetime.now()
            delta_str = str(delta)
            line1 = delta_str[:delta_str.find(',')]
            line2 = delta_str[delta_str.find(',')+2: delta_str.find('.')]
            draw.text((70, 20), line1, font=font, fill=255)
            draw.text((50, 40), line2 + '  left', font=font, fill=255)
            disp.image(image)
            disp.display()
            del draw
    except KeyboardInterrupt:
        pass
```

3.2 SVM 示例代码验证

如图 3，给出了利用不同核函数在 Gamma 值为 0.001 时的四个评估结果。可以看到，精确度的三个量标较为接近，从实际的识别结果上看，其值越大，识别越准确。在拟合极佳时，confusion matrix 应为对角矩阵，比如其中典型的 sigmoid 作核函数时，除对角线外，其他各处都有较多的非零元素，表明其拟合效果不佳；与之相比的是 rbf 作核函数，对角线外仅有极少的非零元素，且较多为个位数，偏离极小。

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        88
           1       0.99      0.97      0.98        91
           2       0.99      0.99      0.99        86
           3       0.98      0.87      0.92        91
           4       0.99      0.96      0.97        92
           5       0.95      0.97      0.96        91
           6       0.99      0.99      0.99        91
           7       0.96      0.99      0.97        89
           8       0.94      1.00      0.97        88
           9       0.93      0.98      0.95        92

    accuracy                           0.97       899
   macro avg       0.97      0.97      0.97       899
weighted avg       0.97      0.97      0.97       899


Confusion matrix:
[[87  0  0  0  1  0  0  0  0  0]
 [ 0 88  1  0  0  0  0  0  1  1]
 [ 0  0 85  1  0  0  0  0  0  0]
 [ 0  0  0 79  0  3  0  4  5  0]
 [ 0  0  0  0 88  0  0  0  0  4]
 [ 0  0  0  0  0 88  1  0  0  2]
 [ 0  1  0  0  0  0 90  0  0  0]
 [ 0  0  0  0  0  1  0 88  0  0]
 [ 0  0  0  0  0  0  0  0 88  0]
 [ 0  0  0  1  0  1  0  0  0 90]]
```

(a) kernel = 'rbf'

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98        88
           1       0.94      0.90      0.92        91
           2       1.00      0.99      0.99        86
           3       0.97      0.86      0.91        91
           4       0.99      0.95      0.97        92
           5       0.90      0.97      0.93        91
           6       0.98      0.99      0.98        91
           7       0.97      0.96      0.96        89
           8       0.88      0.92      0.90        88
           9       0.87      0.93      0.90        92

    accuracy                           0.94       899
   macro avg       0.95      0.94      0.94       899
weighted avg       0.95      0.94      0.94       899


Confusion matrix:
[[87  0  0  0  0  0  1  0  0  0]
 [ 0 82  0  0  0  0  0  0  3  6]
 [ 1  0 85  0  0  0  0  0  0  0]
 [ 0  0  0 78  0  4  0  1  8  0]
 [ 1  0  0  0 87  0  0  0  0  4]
 [ 0  0  0  0  0 88  1  0  0  2]
 [ 0  1  0  0  0  0 90  0  0  0]
 [ 0  1  0  0  1  2  0 85  0  0]
 [ 0  3  0  1  0  1  0  1 81  1]
 [ 1  0  0  1  0  3  0  1  0 86]]
```

(b) kernel = 'linear'

```
              precision    recall  f1-score   support

           0       0.82      0.97      0.89        88
           1       0.67      0.33      0.44        91
           2       0.53      0.85      0.65        86
           3       0.37      0.88      0.52        91
           4       0.89      0.86      0.87        92
           5       0.80      0.62      0.70        91
           6       0.98      0.95      0.96        91
           7       0.65      0.96      0.78        89
           8       0.00      0.00      0.00        88
           9       0.39      0.08      0.13        92

    accuracy                           0.65       899
   macro avg       0.61      0.65      0.59       899
weighted avg       0.61      0.65      0.59       899


Confusion matrix:
[[85  0  0  0  3  0  0  0  0  0]
 [ 0 30 36  4  1  1  1 11  0  7]
 [ 3  1 73  9  0  0  0  0  0  0]
 [ 0  1  2 80  0  0  0  8  0  0]
 [ 3  1  1  0 79  1  0  7  0  0]
 [ 1  0  0 27  2 56  1  1  0  3]
 [ 0  2  0  0  3  0 86  0  0  0]
 [ 0  1  2  0  1  0  0 85  0  0]
 [ 0  8 25 30  0 10  0 14  0  1]
 [12  1  0 66  0  2  0  4  0  7]]
```

(c) kernel = 'sigmoid'

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.98        88
           1       0.99      0.92      0.95        91
           2       1.00      0.97      0.98        86
           3       0.94      0.89      0.92        91
           4       0.99      0.96      0.97        92
           5       0.93      0.97      0.95        91
           6       0.98      0.99      0.98        91
           7       0.97      0.98      0.97        89
           8       0.91      0.97      0.94        88
           9       0.88      0.95      0.91        92

    accuracy                           0.96       899
   macro avg       0.96      0.96      0.96       899
weighted avg       0.96      0.96      0.96       899


Confusion matrix:
[[86  0  0  0  1  0  1  0  0  0]
 [ 0 84  0  0  0  0  0  0  2  5]
 [ 1  0 83  2  0  0  0  0  0  0]
 [ 0  0  0 81  0  3  0  1  5  1]
 [ 0  0  0  0 88  0  0  0  0  4]
 [ 0  0  0  0  0 88  1  0  0  2]
 [ 0  1  0  0  0  0 90  0  0  0]
 [ 0  0  0  0  0  1  0 87  1  0]
 [ 0  0  0  1  0  1  0  1 85  0]
 [ 0  0  0  2  0  2  0  1  0 87]]
```

(d) kernel = 'poly'

图 3: 不同 *kernel* 函数在 *gamma=0.001* 时的对比

3.3 使用 OLED 显示结果

在实验 1 和 2 的基础上，这一问是容易完成的。利用如下代码可以图片的单色化：

```
# kk 中保存了图像的原始数据
digit = Image.fromarray((kk*8).astype(np.uint8),\
     mode='L').resize((48,48)).convert('1')
img = Image.new('1',(disp.width,disp.height),'black')
img.paste(digit, (0, 16, digit.size[0], digit.size[1]+16))
disp.clear()
disp.image(img)
disp.display()
```

随后利用图文排列的方式进行组织即可。

代码如下。利用实验二中的 GPIO 接口与按钮关联，为了实现每按一次按钮查看一幅新的预测，在回调函数当中我们进行一次换幅，即更换当前显示的待预测数据的灰度图。将对应的标签也打印在 OLED 屏幕上。

```
import matplotlib.pyplot as plt
from sklearn import datasets, svm, metrics
import numpy as np
import spidev as SPI
import SSD1306
from PIL import Image # 调用相关库文件
from PIL import ImageDraw
from PIL import ImageFont

import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
channel = 20
GPIO.setmode(GPIO.BCM)
GPIO.setup(channel,GPIO.IN, GPIO.PUD_UP)


digits = datasets.load_digits()
images_and_labels = list(zip(digits.images, digits.target))
for index, (image, label) in enumerate(images_and_labels[:4]):
    plt.subplot(2, 4, index + 1)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Training: %i' % label)
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))
```

```
classifier = svm.SVC(kernel = 'rbf', gamma=0.001)
classifier.fit(data[:n_samples // 2], digits.target[:n_samples // 2])
expected = digits.target[n_samples // 2:]
predicted = classifier.predict(data[n_samples // 2:])
print("Classification report for classifier %s:\n%s\n"
    % (classifier, metrics.classification_report(expected, predicted)))
print("Confusion matrix:\n%s"
    % metrics.confusion_matrix(expected, predicted))
images_and_predictions = list(zip(digits.images[n_samples // 2:], predicted))
for index, (image, prediction) in enumerate(images_and_predictions[:4]):
    plt.subplot(2, 4, index + 5)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Prediction: %i' % prediction)
plt.show()
print('plt has shown')

RST = 19
DC = 16
bus = 0
device = 0 # 树莓派管脚配置
disp = SSD1306.SSD1306(rst=RST,dc=DC,spi=SPI.SpiDev(bus,device))
disp.begin()
disp.clear()
disp.display() # 初始化屏幕相关参数及清屏

cur = 0

def cb(ch):
    global cur
    cur = cur + 1
    if cur >= 4:
        cur = 0
    kk = digits.images
    digit = Image.fromarray((kk[cur+4]*8).astype(np.uint8)\
        , mode='L').resize((48,48)).convert('1')
    img = Image.new('1',(disp.width,disp.height),'black')
    img.paste(digit, (0, 16, digit.size[0], digit.size[1]+16))
    disp.clear()
    font = ImageFont.load_default()
    image = Image.new('RGB',(disp.width,disp.height),'black').convert('1')
```

```
68      draw = ImageDraw.Draw(image)
69      draw.bitmap((10,0), img, fill = 1)
70      draw.text((70,0), "Prediction: ", font=font, fill=255)
71      draw.text((90,30), str(predicted[cur]), font=font, fill=255)
72      disp.image(image)
73      disp.display()
74
75  if __name__ == "__main__":
76      font = ImageFont.load_default()
77      image = Image.new('RGB',(disp.width,disp.height),'black').convert('1')
78      draw = ImageDraw.Draw(image)
79      x = 30; y = 30;
80      draw.text((x,y), 'Hello , Pi!!', font=font, fill=255)
81      disp.image(image)
82      disp.display()
83      time.sleep(1)
84      del draw
85      try:
86          GPIO.add_event_detect(channel, GPIO.RISING, callback=cb,\
87              bouncetime=200)
88      except KeyboardInterrupt:
89          pwm.stop()
90          GPIO.cleanup()
91          GPIO.remove_event_detect(channel)
```

3.4 思考题

3.4.1 训练集与测试集

这八个图片数据对应着机器学习中的两类数据集。前四个数据就对应训练集，用于降低机器学习模型的泛化误差，后四个数据对应测试集，用于反映模型的拟合程度。

测试集是在正式部署模型之前进行的考核的所用材料，像考试一样，在正式完成模型训练之前，要对模型的拟合程度给予一个合理的度量。正因作用和考试类似，所以考题在考试之前是不能泄露的。如果将测试集提前进行了标注，那么就相当于考试做原题一样，无法达成考核的目的。我们可能无法评判利用哪种 kernel 函数更优。

3.4.2 树莓派 OLED 灰度显示原理

类似 PWM，不同灰度的显示本质上是数字的，对于每个特定的发光点由 4 个亮度呈 2 倍等比级数的发光子节点构成，从而实现 16 级亮度的变化。

A 附录代码

这里的代码不带有行号，易于复制并试运行。

实验一代码：

```
import time
from datetime import datetime
import spidev as SPI
import SSD1306
from PIL import Image # 调用相关库文件
from PIL import ImageDraw
from PIL import ImageFont
RST = 19
DC = 16
bus = 0
device = 0 # 树莓派管脚配置

if __name__ == "__main__":
    disp = SSD1306.SSD1306(rst=RST,dc=DC,spi=SPI.SpiDev(bus,device))
    disp.begin()
    disp.clear()
    disp.display() # 初始化屏幕相关参数及清屏

    # welcome interface
    font = ImageFont.load_default()
    image = Image.new('RGB',(disp.width,disp.height),'black').convert('1')
    draw = ImageDraw.Draw(image)
    x = 30; y = 30;
    draw.text((x,y), 'Hello , Pi!!', font=font, fill=255)
    disp.image(image)
    disp.display()
    time.sleep(1)
    del draw

    # counting down
    double11 = datetime(2020, 11, 11)
    try:
        while True:
            image = Image.new('RGB',(disp.width,disp.height),'black').convert('1')
            draw = ImageDraw.Draw(image)

            logo = Image.open('/home/pi/pku.png').resize((40,40), \
```

```
                Image.ANTIALIAS).convert('1')
            draw.bitmap((0,20), logo, fill = 1)
            draw.text((30, 0), str(double11), font=font, fill=255)
            delta = double11 - datetime.now()
            delta_str = str(delta)
            line1 = delta_str[:delta_str.find(',')]
            line2 = delta_str[delta_str.find(',')+2: delta_str.find('.')]
            draw.text((70, 20), line1, font=font, fill=255)
            draw.text((50, 40), line2 + '  left', font=font, fill=255)
            disp.image(image)
            disp.display()
            del draw
    except KeyboardInterrupt:
        pass
```

实验 3 代码：

```
import matplotlib.pyplot as plt
from sklearn import datasets, svm, metrics
import numpy as np
import spidev as SPI
import SSD1306
from PIL import Image # 调用相关库文件
from PIL import ImageDraw
from PIL import ImageFont

import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
channel = 20
GPIO.setmode(GPIO.BCM)
GPIO.setup(channel,GPIO.IN, GPIO.PUD_UP)


digits = datasets.load_digits()
images_and_labels = list(zip(digits.images, digits.target))
for index, (image, label) in enumerate(images_and_labels[:4]):
    plt.subplot(2, 4, index + 1)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Training: %i' % label)
n_samples = len(digits.images)
```

```
data = digits.images.reshape((n_samples, -1))
classifier = svm.SVC(kernel = 'rbf', gamma=0.001)
classifier.fit(data[:n_samples // 2], digits.target[:n_samples // 2])
expected = digits.target[n_samples // 2:]
predicted = classifier.predict(data[n_samples // 2:])
print("Classification report for classifier %s:\n%s\n"% (classifier, metrics.classifi
print("Confusion matrix:\n%s" % metrics.confusion_matrix(expected, predicted))
images_and_predictions = list(zip(digits.images[n_samples // 2:], predicted))
for index, (image, prediction) in enumerate(images_and_predictions[:4]):
    plt.subplot(2, 4, index + 5)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Prediction: %i' % prediction)
plt.show()
print('plt has shown')

RST = 19
DC = 16
bus = 0
device = 0 # 树莓派管脚配置
disp = SSD1306.SSD1306(rst=RST,dc=DC,spi=SPI.SpiDev(bus,device))
disp.begin()
disp.clear()
disp.display() # 初始化屏幕相关参数及清屏

cur = 0

def cb(ch):
    global cur
    cur = cur + 1
    if cur >= 4:
        cur = 0
    kk = digits.images
    digit = Image.fromarray((kk[cur+4]*8).astype(np.uint8), mode='L').resize((48,48))
    img = Image.new('1',(disp.width,disp.height),'black')
    img.paste(digit, (0, 16, digit.size[0], digit.size[1]+16))
    disp.clear()
    font = ImageFont.load_default()
    image = Image.new('RGB',(disp.width,disp.height),'black').convert('1')
    draw = ImageDraw.Draw(image)
    draw.bitmap((10,0), img, fill = 1)
```

```
    draw.text((70,0), "Prediction: ", font=font, fill=255)
    draw.text((90,30), str(predicted[cur]), font=font, fill=255)
    disp.image(image)
    disp.display()

if __name__ == "__main__":
    font = ImageFont.load_default()
    image = Image.new('RGB',(disp.width,disp.height),'black').convert('1')
    draw = ImageDraw.Draw(image)
    x = 30; y = 30;
    draw.text((x,y), 'Hello, Pi!!', font=font, fill=255)
    disp.image(image)
    disp.display()
    time.sleep(1)
    del draw
    try:
        GPIO.add_event_detect(channel, GPIO.RISING, callback=cb, bouncetime=200)
    except KeyboardInterrupt:
        pwm.stop()
        GPIO.cleanup()
        GPIO.remove_event_detect(channel)
```