人工智能课程报告

——基于python图像识别机器学习实现“斗牛”及红绿灯算法

胡景文

2015201993

组长：赵元培

组员：杨文清

1. 在本门课堂中学习到的知识

**第一章**

我们大概学习了解了机器学习的相关内容，了解到了机器学习的一些基本的内容，机器学习分为有监督的学习和无监督的学习，区别在于有无人工标注。有监督的学习算法有k邻近算法，线性回归，logistic回归等，无监督的学习算法有聚类，可视化及降维等。另外比较重要的东西就是交叉验证，即将训练集划分为子集，每个模型针对不同的子集进行训练，并对剩余部分进行验证。

**第二章**

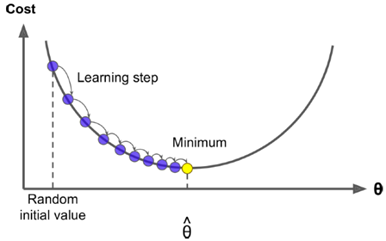
主要以一个实例来帮助我们学习如何选择模型，如何调整超参数的值以及如何通过交叉验证的方法来评价模型的好坏。

**第三章**

以MNIST为例，训练二分类器，训练数据要求保证交叉验证集相似，算法有可能要求训练实例是有一定顺序的。在训练二分类器的时候，用SGD分类器可能更优。在评价二分类器的好坏方面，可以通过K-fold cross-validation，Confusion Matrix，precision/recall tradeoff或ROC Curve**。**在区分两个以上的类时，可以使用随机森林分类器和朴素贝叶斯分类器，有OVO策略和OVA策略。

**第四章**

这章讲了线性回归，线性回归有许多参数，包括系数和偏差，我们可以通过梯度下降的方法来计算出最优的一组系数，并用它来进行结果的预测。梯度下降又有多种梯度下降算法，如随机梯度下降法，小批量梯度下降法，前者在每个步骤中在训练集中选择一个随机实例，仅基于单个实例计算渐变，比批处理梯度下降少得多，后者是在小批量的实例的随机集上的梯度。



**第五章**

SVM也是一种分类器，SVM考虑寻找一个满足分类要求的超平面,并且使训练集中的点距离分类面尽可能的远,也就是寻找一个分类面使它两侧的空白区域最大，通过sklearn的包可以完成svm分类。

**第六章**

主要介绍了决策树，决策树又称为判定树，是运用于分类的一种树结构，其中的每个内部节点代表对某一属性的一次测试，每条边代表一个测试结果，叶节点代表某个类或类的分布。决策树的决策过程需要从决策树的根节点开始，待测数据与决策树中的特征节点进行比较，并按照比较结果选择选择下一比较分支，直到叶子节点作为最终的决策结果。在 sklearn.tree这个包里可以实现。

**第七章**

随机森林顾名思义，是用随机的方式建立一个森林，森林里面有很多的决策树组成，随机森林的每一棵决策树之间是没有关联的。在得到森林之后，当有一个新的输入样本进入的时候，就让森林中的每一棵决策树分别进行一下判断，看看这个样本应该属于哪一类（对于分类算法），然后看看哪一类被选择最多，就预测这个样本为那一类。

**第八章**

关于降低维数，有两种典型的方法，PCA和LLE，前者识别与数据最接近的平面并将数据投射到它上，后者测量每个训练实例如何与其最近的邻居线性相关，寻找训练集的低维表示，使这些局部关系得到最好的保存。

**第九章**

介绍了tensorflow是什么以及如何安装，建工程，运行，存储及可视化等基本功能，初步了解了tensor的回归和存储。

**第十章**

本章介绍了人工智能神经网络，一层或多层LTU称为隐层，最后一层为LTU，反向传播算法通过每个层测量每个连接错误，微调连接权重以减少错误。

**第十一章**

对于深度学习，神经网络的激活函数有多种，有sigmoid函数，ReLU函数和leaky ReLU函数。不同于sigmoid函数，后面两个在神经网络反向传递时会减少参数的相互依存关系，缓解了过拟合的发生。

**第十三章**

CNN相较于传统的图像处理算法的优点之一在于，避免了对图像复杂的前期预处理过程（提取人工特征等），可以直接输入原始图像。

1. 在三个阶段承担的工作及完成情况
2. **组装小车，并调试程序**

**目标：**

完成小车组装，能烧程序，能实现各部件间的配合

**承担任务：**

在本阶段，我们购买了组装小车的必备零件，并进行了组装，我和杨文清同学利用电烙铁，面包板，面包线，超声波传感器，arduino板及小车相关零件组装了我们的小车。在本阶段，我们还和组长一起调试了组长写的代码，达到了本阶段的目标——能够实现arduino板和小车的配合，实现了小车的避障功能。小车的工作原理大概如下：

超声波传感器探测到障碍物在30cm范围内——>左轮速度降低，右轮速度增加——>当小车探测到障碍物不在30cm以内时两轮速度调为一致



**代码：**

#include "AFMotor.h"

//设置两个motor的接口和频率

AF\_DCMotor motor\_left**(**1**,**MOTOR12\_1KHZ**);**

AF\_DCMotor motor\_right**(**4**,**MOTOR34\_1KHZ**);**

//设置两个舵机接口

const int TrigPin **=** 9**;**

const int EchoPin **=** 10**;**

//距离变量

float cm**;**

void setup**()**

**{**

Serial**.**begin**(**9600**);**

pinMode**(**TrigPin**,**OUTPUT**);**

pinMode**(**EchoPin**,**INPUT**);**

//设置左右motor速度

motor\_left**.**setSpeed**(**180**);**

motor\_right**.**setSpeed**(**210**);**

**}**

void loop**()**

**{**

digitalWrite**(**TrigPin**,**LOW**);** //低高低电平发一个短时间脉冲去TrigPin

delayMicroseconds**(**2**);**

digitalWrite**(**TrigPin**,**HIGH**);**

delayMicroseconds**(**10**);**

digitalWrite**(**TrigPin**,**LOW**);**

cm **=** pulseIn**(**EchoPin**,**HIGH**)** **/** 58.0**;** //将回波时间换算成cm

cm **=** **(**int**(**cm **\*** 100.0**))** **/** 100.0**;** //保留两位小数

**if(**cm **<** 15**){**

motor\_left**.**setSpeed**(**180**);**

motor\_left**.**run**(**FORWARD**);**

motor\_right**.**run**(**FORWARD**);**

delay**(**300**);**

**}**

**else** **if(**cm **<** 40**){**

motor\_left**.**setSpeed**(**180**);**

motor\_left**.**run**(**BACKWARD**);**

motor\_right**.**run**(**BACKWARD**);**

**}**

delay**(**100**);**

**}**

1. **完善蓝牙串口等硬件组装，opencv等的环境搭建，查阅图片相似度的算法**

**目标：**

蓝牙模块和其他设备的通信，利用软件的视频传输及python的视频处理和图像分类的初步探索

**承担任务：**

在本阶段，我和杨文清同学一起研究了蓝牙串口等硬件的组装，查阅处理图像相似度的算法，并自己完成了openCV等的环境搭建。蓝牙的BT06串口模块VCC：接Arduino的5V，GND：接Arduino的GND，TXD：一般表示为自己的发送端，接Arduino的RX，RXD：一般表示为自己的接收端，接Arduino的TX。在硬件安装完成后，我们在android手机端下载了蓝牙串口app，实现了小车蓝牙模块与其他蓝牙设备连接这一目标。蓝牙小车app提供了源代码方便我们学习。在python处理方面，install了关于图像处理及可视化方面的包。



图像识别方面，首先我们使在pc端和android端同时安装的droidcam通过USB，蓝牙连接来将手机摄像头拍到的视频同步到电脑上，之后通过python程序实现每隔一段时间将视频中的图像截取并存储下来，之后机器学习算法通过分析新存储的图片来对小车看到的图片进行分析并将分析得到的结果发送给小车，指导小车的行动。

在图像识别算法调研方面，我们决定使用pyssim这个包来计算两张图片的相结构相似性。该方法基于亮度，对比度和结构能大致区分两张图片的相似度，但由于其效果不好，所以仅作为一个补充。

我们判断两个样本之间的相似度一般是通过计算两者之间的距离，在图片分析中常用的是夹角余弦距离，由于图像是RGB像素点值的组合，故可以通过将两张图片缩到相同规模，向量每一维的值为对应点RGB值的平均，这之后计算图片的平均，但计算量过大，速度过慢，不合适。

色彩分布直方图通过判断RGB图片对应的颜色来判断是否接近。如果每种原色都可以取 256 个值，那么整个颜色空间共有 1600 万种颜色 （256 的三次方）。针对这 1600 万种颜色比较直方图，计算量实在太大了，因此 需要采用简化方法。可以将 0～255 分成四个区：0～63 为第 0 区，64～127 为第 1 区，128～191 为第 2 区，192～255 为第 3 区。这意味着红绿蓝分别有 4 个区， 总共可以构成 64 种组合（4 的 3 次方）。任何一种颜色必然属于这 64 种组合中的一种，这样就可以统计每一种组合 包含的像素数量。

小车对图片的分析，因为行进道路为红色，所以对于道路的判断可以根据RGB分布进行判断，但RGB组合过多，计算依旧很复杂，又需要压缩，引入了灰度直方图算法。

**判断代码：**

**if(**ch **==** 'F'**){**//前进

Serial**.**println**(**"up"**);**

motor\_left**.**setSpeed**(**210**);**

motor\_right**.**setSpeed**(**180**);**

motor\_left**.**run**(**BACKWARD**);**

motor\_right**.**run**(**BACKWARD**);**

**}**

**else** **if(**ch **==** 'L'**){**

//左转

Serial**.**println**(**"left"**);**

motor\_left**.**setSpeed**(**0**);**

motor\_right**.**setSpeed**(**180**);**

motor\_left**.**run**(**BACKWARD**);**

motor\_right**.**run**(**BACKWARD**);**

**}**

视频截取代码：

# coding=utf-8

# using python2.7

**import** time

**from** PIL **import** ImageGrab

start\_job **=** **True**

i**=**0

**while** start\_job**:**

# do something here.

img **=** ImageGrab**.**grab**()**

img**.**save**(**str**(**i**)+**".png"**)**

i**=**i**+**1

# print 1 # 这里改成你想做的任务

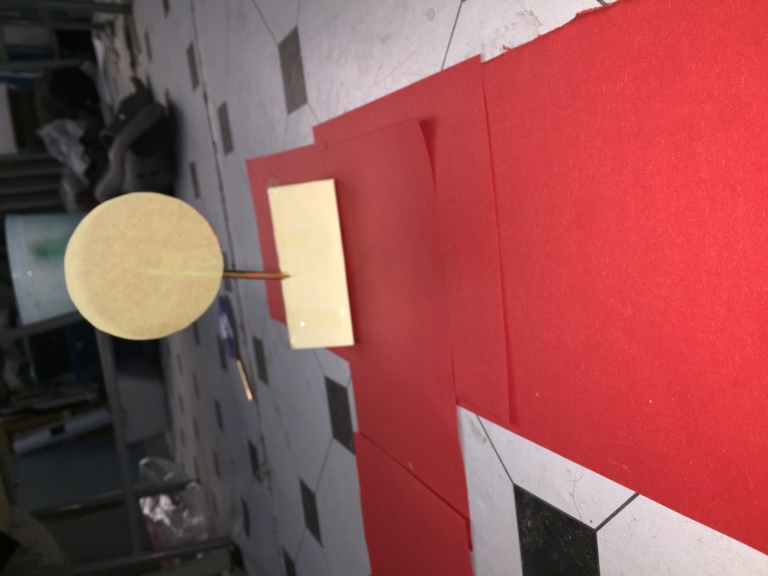
time**.**sleep**(**0.13**)** # 每隔一分钟执行一次print任务, sleep里的参数按秒算

1. **研究CNN神经网络算法，并测试CNN神经网络代码的运行效果，获取小车图像识别的数据集**

**目标：**

探究不同深度学习算法对于图像分类的效果差别，建立图像分类模型，利用软件处理上传到电脑端的视频，再利用蓝牙将电路板和模型进行连接，指导小车运行，最终结果测试

**承担任务：**





在本阶段，我和杨文清同学负责准备图片训练集，除此之外，我还负责研究CNN 神经网络算法，不过经过查询资料，我们暂时决定使用RNN算法来对图像进行分析来识别小车的行进方向。

图片训练集的准备方面，我们使用红色卡纸制作了道路，红绿灯，标志牌以及“红布”，模拟小车前进和斗牛算法并用手机拍摄视频，利用电脑上的python程序，得到训练需要的图片训练集。

关于CNN算法，在卷积神经网络（Convolutional Neural Network,CNN）中，卷积层的神经元只与前一层的部分神经元节点相连，即它的神经元间的连接是非全连接的，且同一层中某些神经元之间的连接的权重 w 和偏移 b 是共享的（即相同的），这样大量地减少了需要训练参数的数量。

设移动的步长为1：从左到右扫描，每次移动 1 格，扫描完之后，再向下移动一格，再次从左到右扫描。

**代码：**

**import** tensorflow **as** tf

**from** sklearn**.**datasets **import** load\_digits

**import** numpy **as** np

digits **=** load\_digits**()**

X\_data **=** digits**.**data**.**astype**(**np**.**float32**)**

Y\_data **=** digits**.**target**.**astype**(**np**.**float32**).**reshape**(-**1**,**1**)**

**print** X\_data**.**shape

**print** Y\_data**.**shape

**from** sklearn**.**preprocessing **import** MinMaxScaler

scaler **=** MinMaxScaler**()**

X\_data **=** scaler**.**fit\_transform**(**X\_data**)**

**from** sklearn**.**preprocessing **import** OneHotEncoder

Y **=** OneHotEncoder**().**fit\_transform**(**Y\_data**).**todense**()** #one-hot编码

matrix**([[** 1.**,** 0.**,** 0.**,** **...,** 0.**,** 0.**,** 0.**],**

**[** 0.**,** 1.**,** 0.**,** **...,** 0.**,** 0.**,** 0.**],**

**[** 0.**,** 0.**,** 1.**,** **...,** 0.**,** 0.**,** 0.**],**

**...,**

**[** 0.**,** 0.**,** 0.**,** **...,** 0.**,** 1.**,** 0.**],**

**[** 0.**,** 0.**,** 0.**,** **...,** 0.**,** 0.**,** 1.**],**

**[** 0.**,** 0.**,** 0.**,** **...,** 0.**,** 1.**,** 0.**]])**

X **=** X\_data**.**reshape**(-**1**,**8**,**8**,**1**)**

batch\_size **=** 8 # 使用MBGD算法，设定batch\_size为8

**def** generatebatch**(**X**,**Y**,**n\_examples**,** batch\_size**):**

**for** batch\_i **in** range**(**n\_examples **//** batch\_size**):**

start **=** batch\_i**\***batch\_size

end **=** start **+** batch\_size

batch\_xs **=** X**[**start**:**end**]**

batch\_ys **=** Y**[**start**:**end**]**

**yield** batch\_xs**,** batch\_ys # 生成每一个batch

tf**.**reset\_default\_graph**()**

# 输入层

tf\_X **=** tf**.**placeholder**(**tf**.**float32**,[None,**8**,**8**,**1**])**

tf\_Y **=** tf**.**placeholder**(**tf**.**float32**,[None,**10**])**

# 卷积层+激活层

conv\_filter\_w1 **=** tf**.**Variable**(**tf**.**random\_normal**([**3**,** 3**,** 1**,** 10**]))**

conv\_filter\_b1 **=** tf**.**Variable**(**tf**.**random\_normal**([**10**]))**

relu\_feature\_maps1 **=** tf**.**nn**.**relu**(**\

tf**.**nn**.**conv2d**(**tf\_X**,** conv\_filter\_w1**,**strides**=[**1**,** 1**,** 1**,** 1**],** padding**=**'SAME'**)** **+** conv\_filter\_b1**)**

# 池化层

max\_pool1 **=** tf**.**nn**.**max\_pool**(**relu\_feature\_maps1**,**ksize**=[**1**,**3**,**3**,**1**],**strides**=[**1**,**2**,**2**,**1**],**padding**=**'SAME'**)**

**print** max\_pool1

# 卷积层

conv\_filter\_w2 **=** tf**.**Variable**(**tf**.**random\_normal**([**3**,** 3**,** 10**,** 5**]))**

conv\_filter\_b2 **=** tf**.**Variable**(**tf**.**random\_normal**([**5**]))**

conv\_out2 **=** tf**.**nn**.**conv2d**(**relu\_feature\_maps1**,** conv\_filter\_w2**,**strides**=[**1**,** 2**,** 2**,** 1**],** padding**=**'SAME'**)** **+** conv\_filter\_b2

**print** conv\_out2

Tensor**(**"add\_4:0"**,** shape**=(?,** 4**,** 4**,** 5**),** dtype**=**float32**)**

# BN归一化层+激活层

batch\_mean**,** batch\_var **=** tf**.**nn**.**moments**(**conv\_out2**,** **[**0**,** 1**,** 2**],** keep\_dims**=True)**

shift **=** tf**.**Variable**(**tf**.**zeros**([**5**]))**

scale **=** tf**.**Variable**(**tf**.**ones**([**5**]))**

epsilon **=** 1e-3

BN\_out **=** tf**.**nn**.**batch\_normalization**(**conv\_out2**,** batch\_mean**,** batch\_var**,** shift**,** scale**,** epsilon**)**

**print** BN\_out

relu\_BN\_maps2 **=** tf**.**nn**.**relu**(**BN\_out**)**

Tensor**(**"batchnorm/add\_1:0"**,** shape**=(?,** 4**,** 4**,** 5**),** dtype**=**float32**)**

max\_pool2 **=** tf**.**nn**.**max\_pool**(**relu\_BN\_maps2**,**ksize**=[**1**,**3**,**3**,**1**],**strides**=[**1**,**2**,**2**,**1**],**padding**=**'SAME'**)**

**print** max\_pool2

Tensor**(**"MaxPool\_1:0"**,** shape**=(?,** 2**,** 2**,** 5**),** dtype**=**float32**)**

max\_pool2\_flat **=** tf**.**reshape**(**max\_pool2**,** **[-**1**,** 2**\***2**\***5**])**

fc\_w1 **=** tf**.**Variable**(**tf**.**random\_normal**([**2**\***2**\***5**,**50**]))**

fc\_b1 **=** tf**.**Variable**(**tf**.**random\_normal**([**50**]))**

fc\_out1 **=** tf**.**nn**.**relu**(**tf**.**matmul**(**max\_pool2\_flat**,** fc\_w1**)** **+** fc\_b1**)**

out\_w1 **=** tf**.**Variable**(**tf**.**random\_normal**([**50**,**10**]))**

out\_b1 **=** tf**.**Variable**(**tf**.**random\_normal**([**10**]))**

pred **=** tf**.**nn**.**softmax**(**tf**.**matmul**(**fc\_out1**,**out\_w1**)+**out\_b1**)**

loss **=** **-**tf**.**reduce\_mean**(**tf\_Y**\***tf**.**log**(**tf**.**clip\_by\_value**(**pred**,**1e-11**,**1.0**)))**

train\_step **=** tf**.**train**.**AdamOptimizer**(**1e-3**).**minimize**(**loss**)**

y\_pred **=** tf**.**arg\_max**(**pred**,**1**)**

bool\_pred **=** tf**.**equal**(**tf**.**arg\_max**(**tf\_Y**,**1**),**y\_pred**)**

accuracy **=** tf**.**reduce\_mean**(**tf**.**cast**(**bool\_pred**,**tf**.**float32**))** # 准确率

**with** tf**.**Session**()** **as** sess**:**

sess**.**run**(**tf**.**global\_variables\_initializer**())**

**for** epoch **in** range**(**1000**):** # 迭代1000个周期

**for** batch\_xs**,**batch\_ys **in** generatebatch**(**X**,**Y**,**Y**.**shape**[**0**],**batch\_size**):** # 每个周期进行MBGD算法

sess**.**run**(**train\_step**,**feed\_dict**={**tf\_X**:**batch\_xs**,**tf\_Y**:**batch\_ys**})**

**if(**epoch**%**100**==**0**):**

res **=** sess**.**run**(**accuracy**,**feed\_dict**={**tf\_X**:**X**,**tf\_Y**:**Y**})**

**print** **(**epoch**,**res**)**

res\_ypred **=** y\_pred**.**eval**(**feed\_dict**={**tf\_X**:**X**,**tf\_Y**:**Y**}).**flatten**()** # 只能预测一批样本，不能预测一个样本

**print** res\_ypred

结果：

(0, 0.36338341)

(100, 0.96828049)

(200, 0.99666113)

(300, 0.99554813)

(400, 0.99888706)

(500, 0.99777406)

(600, 0.9961046)

(700, 0.99666113)

(800, 0.99499166)

(900, 0.99888706)

[0 1 2 ..., 8 9 8]

在第100次batch size 迭代时，准确率就快速接近收敛了，这得归功于Batch Normalization 的作用！需要注意的是，这个模型还不能用来预测单个样本，因为在进行BN层计算时，单个样本的均值和方差都为0，会得到相反的预测效果。

1. 总结

在本学期的课中学到了很多，更是把知识运用到了小车中，亲眼见证了自己的知识应用到了实际中，通过代码一步步实现并完善小车的动作，中期的时候做出了“斗牛”这一动作（即小车识别到红色的板时加速），后期最终实现了小车红绿灯系统的识路功能。与队友的合作学习十分有效，在这门课中锻炼了自己，提高了合作学习和写代码的能力。