远程科研项目报告

谭镇涛

2019/9/15

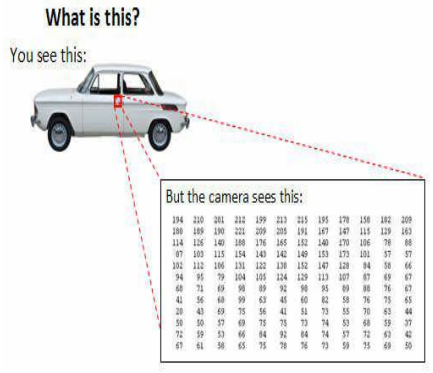
目录：

1、学习背景：

1.1 学习背景：

数据分析是指用适当的统计分析方法对收集来的大量数据进行分析，提取有用信息和形成结论而对数据加以详细研究和概括总结的过程。这一过程也是质量管理体系的支持过程。在实用中，数据分析可帮助人们作出判断，以便采取适当行动。

数据分析的数学基础在 20 世纪早期就已确立，但直到计算机的出现才使得实际操作成为可能，并使得数据分析得以推广。数据分析是数学与计算机科学相结合的产物。



数据分析在图像处理方面可谓是独占鳌头，在现实生活中，我们可以快速识别一张照片里面的所有物体，但如何教机器来识别一辆汽车呢？比如说左边这张图片，我们一眼便可以看出来这是一辆汽车，在机器眼中，则是这种二维矩阵，里面记载着像素点的数值。一般而言,我们肉眼可以很快的提取这样的特征。我们人可以从图像本身找到这个图像的规律，所以说要让机器从事物的特征中找到规律，其实是一个如何在数字中找规律的问题。

现实生活中我们往往要根据多个特征(多串数字)来分析一件事情，每个原始特征我们都看作是一个维度。例如一个学生的学习成绩好坏要根据语文、数学、英语等多门课程的分数来综合判断，这里每门课程都是一个维度。当使用二次曲线和多变量(多维)拟合的情况下，特征的数量会剧增，特征数=维度^2/2 这个公式可以大概计算出特征增加的情况，例如一个 100 维的数据，二次多项式拟合后，特征会增加到 100*100/2=5000 个。

一张 50*50 像素的灰度图片，如果用二次多项式拟合的话，它有多少个特征呢？——大约有 3 百万！

所以这么大的数据量，我们该如何去处理就成了很大的问题。

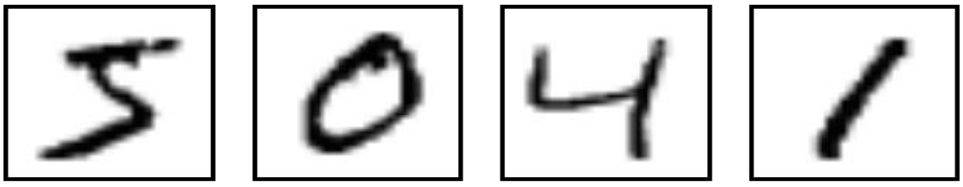
本次学习周期，我们主要围绕着手写体识别算法（MNIST），运用 anaconda 编译环境集成器，我使用的是 spyder 编译环境，进行对数字 0-9 的识别。为何使用 spyder 来进行代码的编辑，其原因主要是：

1） Spyder 是 Python 的作者为它开发的一个简单的集成开发环境。和其他的 Python 开发环境相比，它最大的优点就是模仿 MATLAB 的“工作空间”的功能，可以很方便地观察和修改数组的值。

2） Spyder 的户可以根据自己的喜好调整它们的位置和大小。比如说：“Editor”、“Object inspector”、“Variable explorer”、“File explorer”、“Console”、“History log”以及两个显示图像的窗格。看起来一目了然，清晰易懂。

1.2MINST 算法简介：

MNIST 是一个入门级的计算机视觉数据集，它包含各种手写数字图片：

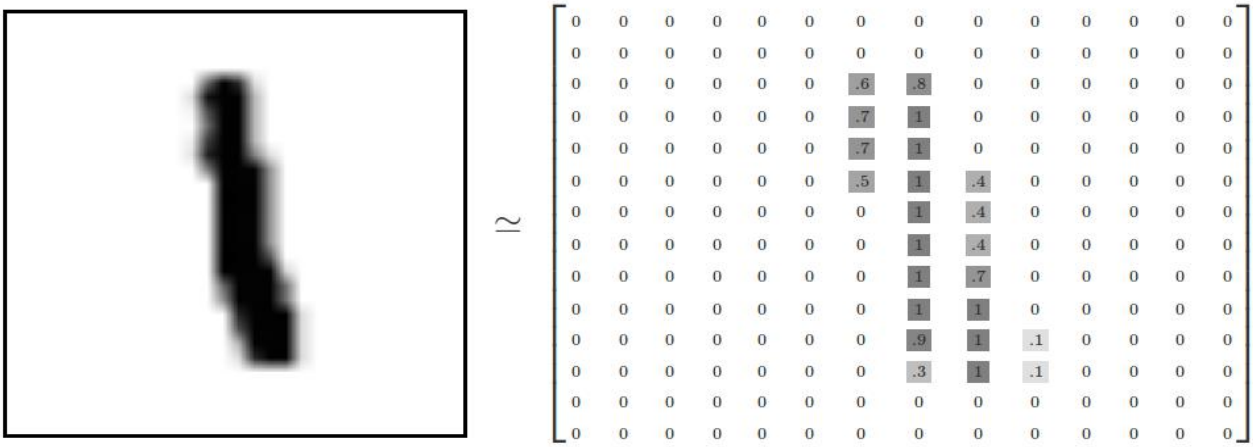


它也包含每一张图片对应的标签，告诉我们这个是数字几。比如，上面这四张图片的标签分别是 5，0，4，1。

2、MINST 的原理及其实现：

2.1 算法原理：

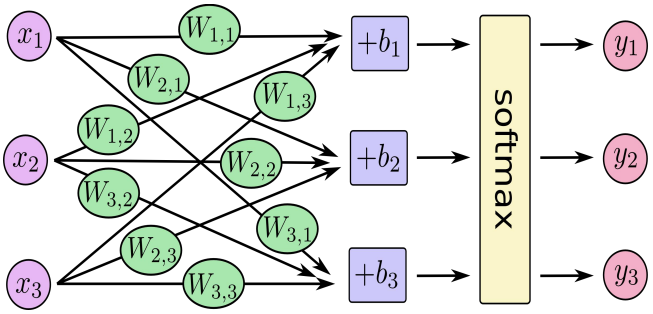
Minst 算法识别的图像是 28*28 像素的图片，且只能识别 0-9 这几个数字。这就代表着构成这些图片的实际是一个 28*28 的矩阵，每个矩阵有着 0-1 间的任意一个数，比如下图：



将其展开成一个 28*28 = 784 的向量，然后保证 mnist 的测试集的图片都以这种方式展开成向量。然后我们将这些输入值带入一个以 softmax 回归模型为激活函数的神经网络中进行训练然后得到一组输出值。

关于基于 softmax 回归模型的神经网络，我们作以下解释：

1、对于输入的 xs 加权求和，再分别加上一个偏置量，最后再输入到 softmax 函数中：



这个神经网络实际上可以写成如下的等式：

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{matrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{matrix} \right)$$

2、将输出值与样本值进行对比，然后计算出准确率与残差，依据梯度下降和神经网络反向传播的方法，不断更新权重值 w 和偏移量 b，使得准确率达到最高。

其中关于残差的计算，我们采用经典的成本函数：“交叉熵”，其表达式定义如下：

$$H_{y'}(y) = -\sum_i y'_i \log(y_i)$$

其中 y'是实际的分布,y 使我们的预测分布。

3、最后返回一个训练完毕后的权重 w 和偏移量 b，将其存在本地。

2.2 代码实现：

此代码实现，我们采用 spyder 进行实现：

一、训练模型：

首先，我们读取 mnist 库当中的用于训练的图片：

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
```

然后我们给自变量 x 一个 784 的占位符和因变量 y 一个 10 位占位符：

```
x = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])
```

接着我们定义神经网络层：

```
def weight_variable(shape):
    initial = tf.truncated_normal(shape,stddev = 0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1,shape = shape)
    return tf.Variable(initial)

def conv2d(x,W):
    return tf.nn.conv2d(x, W, strides = [1,1,1,1], padding = 'SAME')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
```

然后我们定义各种权值：

```
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])

x_image = tf.reshape(x,[-1,28,28,1])

h_conv1 = tf.nn.relu(conv2d(x_image,W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

keep_prob = tf.placeholder("float")
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
```

最后我们给出 softmax 回归模型的函数：

```
y_conv=tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
```

然后我们定义“交叉熵”函数：

```
cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))
```

最后我们进行模型训练：

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(20000):
        batch = mnist.train.next_batch(50)
        if i % 100 == 0:
            train_accuracy = accuracy.eval(feed_dict={
                x: batch[0], y_: batch[1], keep_prob: 1.0})
            print('step %d, training accuracy %g' % (i, train_accuracy))
        train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})
    saver.save(sess ,'C:/Users/admin/Desktop/Python文件/MIT科研/model.ckpt') #

    print('test accuracy %g' % accuracy.eval(feed_dict={
        x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
```

代码的输出是以每 100 张图位一个 epoch，来进行准确度预测的，如下图所示：

```
step 0, training accuracy 0.18
step 100, training accuracy 0.86
step 200, training accuracy 0.84
step 300, training accuracy 0.94
step 400, training accuracy 0.96
```

由于 20000 个点的计算量太大，代码运行起来时间成本较高，所以我们仅仅才用了 2000 个点进行训练，最后得到一个最终准确度，能够发现即使训练点集没有 20000 个点那么多，但是最终的准确率还是十分可人的：

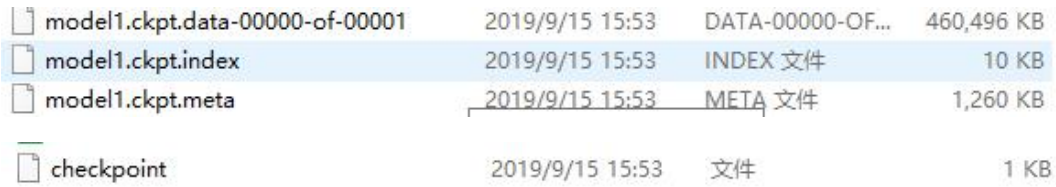
```
test accuracy 0.9475
An exception has occurred, use %tb to see the full traceback.
```

二、保存模型：

保存模型我们采用标准的 TensorFlow 的 save（）函数进行保存，代码如下：

```
saver = tf.train.Saver()

saver.save(sess ,'C:/Users/admin/Desktop/Python文件/MIT科研/model.ckpt')
```

我们来看一下保存在这一路径下的文件长什么样子：



一个 checkpoint 文件存储了计算图的位置，三个 model.ckpt 文件储存了模型的数据，之后我们就直接调入我们保存的模型，来进行图片的预测。

三、处理图片：

正如之前我们对 mnist 算法的分析，我们知道 mnist 算法只能处理像素为 28*28 的图像，所以并不是任何一张图片，这个算法都能识别的，为了验证我们模型的准确度，并将我们的模型运用于现实环境，我们需要得到一系列像素点为 28*28 的图形。为了得到这样的图形，我们采用 PS 软件来获得，操作步骤如下：

首先创建一个画板，然后设定其像素值，然后就可以在画板上写你想测试的数字了，我下面给出我所画的几张图片，以供参考：

四、读取模型：

读取模型，我们用的是 TensorFlow 的 restore 函数，具体代码如下：

首先我们要将我们的图片输入到模型当中，这里，我们定义了一个输入图片函数：

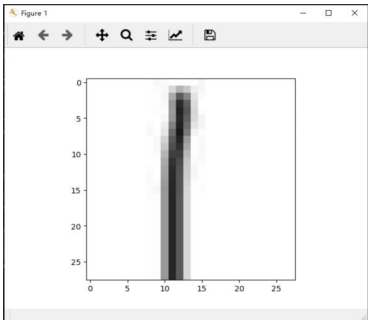
```
def imageprepare():
    im = Image.open('C:/Users/admin/Desktop/photo1.jpg')
    plt.imshow(im)  #显示需要识别的图片
    plt.show()
    im = im.convert('L')
    tv = list(im.getdata())
    tva = [(255-x)*1.0/255.0 for x in tv]
    return tva

result=imageprepare()
```

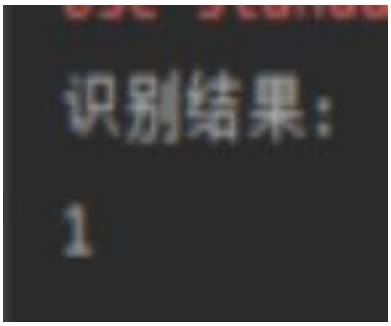
然后我们开始输入我们的模型：

```
saver = tf.train.Saver()

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    saver.restore(sess, 'C:/Users/admin/Desktop/Python文件/MIT科研/model1.ckpt')

    prediction=tf.argmax(y_conv,1)
    predint=prediction.eval(feed_dict={x: [result],keep_prob: 1.0}, session=sess

    print('识别结果:')
    print(predint[0])
```

中间省略了一些激活这个计算图所需要定义的参数，我在此只展示比较关键的代码部分。

最后展示一下结果，上面代码可以看出我们预测的是放在桌面的‘photo，jpg’文件，它是数字为 1 的图片（由于 spyder 出了一些问题，我们测试函数在 pycharm 上实现的）





3. Docker 介绍与应用：

3.1docker 介绍：

Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化，容器是完全使用沙箱机制，相互之间不会有任何接口。

一个完整的 Docker 有以下几个部分组成：

1、dockerClient 客户端

2、Docker Daemon 守护进程

3、Docker Image 镜像

4、DockerContainer 容器

起源：

Docker 是 PaaS 提供商 dotCloud 开源的一个基于 LXC 的高级容器引擎，源代码托管在 Github 上，基于 go 语言并遵从 Apache2.0 协议开源。

Docker 自 2013 年以来非常火热，无论是从 github 上的代码活跃度，还是 Redhat 在 RHEL6.5 中集成对 Docker 的支持，就连 Google 的 Compute Engine 也支持 docker 在其之

上运行。

一款开源软件能否在商业上成功，很大程度上依赖三件事 - 成功的 user case(用例), 活跃的社区和一个好故事。 dotCloud 自家的 PaaS 产品建立在 docker 之上，长期维护且有大量的用户，社区也十分活跃，接下来我们看看 docker 的故事。

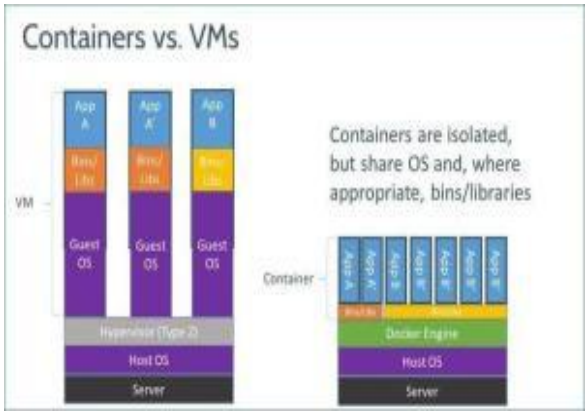
环境管理复杂 - 从各种 OS 到各种中间件到各种 app, 一款产品能够成功作为开发者需要关心的东西太多，且难于管理，这个问题几乎在所有现代 IT 相关行业都需要面对。

云计算时代的到来 - AWS 的成功, 引导开发者将应用转移到 cloud 上, 解决了硬件管理的问题,然而中间件相关的问题依然存在 (所以 openstack HEAT 和 AWS cloudformation 都着力解决这个问题)。开发者思路变化提供了可能性。

虚拟化手段的变化 - cloud 时代采用标配硬件来降低成本，采用虚拟化手段来满足用户按需使用的需求以及保证可用性和隔离性。然而无论是 KVM 还是 Xen 在 docker 看来,都在浪费资源，因为用户需要的是高效运行环境而非 OS, GuestOS 既浪费资源又难于管理, 更加轻量级的 LXC 更加灵活和快速

LXC 的移动性 - LXC 在 linux 2.6 的 kernel 里就已经存在了，但是其设计之初并非为云计算考虑的，缺少标准化的描述手段和容器的可迁移性，决定其构建出的环境难于迁移和标准化管理(相对于 KVM 之类 image 和 snapshot 的概念)。docker 就在这个问题上做出实质性的革新。这是 docker 最独特的地方。

面对上述几个问题，docker 设想是交付运行环境如同海运，OS 如同一个货轮，每一个在 OS 基础上的软件都如同一个集装箱，用户可以通过标准化手段自由组装运行环境，同时集装箱的内容可以由用户自定义，也可以由专业人员制造。这样，交付一个软件，就是一系列标准化组件的集合的交付，如同乐高积木，用户只需要选择合适的积木组合，并且在最顶端署上自己的名字(最后一个标准化组件是用户的 app)。这也就是基于 docker 的 PaaS 产品的原型。



此图是为了对比容器相对于虚拟器的优势，可以看出，容器省去了 os 操作系统这一层次，从而大大增大了软件的可用性

3.2docker 应用：

首先使用 docker version 来看 docker 的版本：

```
(base) tanzhentaodeMacBook-Pro:~ tanzhentao$ docker version
Client: Docker Engine - Community
 Version:           19.03.1
 API version:       1.40
 Go version:        go1.12.5
 Git commit:        74b1e89
 Built:             Thu Jul 25 21:18:17 2019
 OS/Arch:           darwin/amd64
 Experimental:      false
```

然后使用 docker info 看一些更多的信息，比如说你的 docker 自身的网络端口：

```
(base) tanzhentaodeMacBook-Pro:~ tanzhentao$ docker info
Client:
 Debug Mode: false

Server:
 Containers: 19
  Running: 2
  Paused: 0
  Stopped: 17
 Images: 16
 Server Version: 19.03.1
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
```

然后我们使用 docker run hello-world 运行容器里的一个程序：

```
(base) tanzhentaodeMacBook-Pro:~ tanzhentao$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

4. Flask 实现与用户互动：

4.1flask 简介：

Falsk 是由 python 开发的轻量的 web 框架，小巧，灵活，一个脚本就可以启动一个 web 项目。

4.2flask 代码实现：

我们期望用 flask 实现在 web 上与用户进行交互，HTTP 操作有四种：put，get，post，delete，根据我们项目的需要我们需要写出 post 方法，下面我们给出 post 方法：

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 23 16:25:15 2019

@author: tanzhentao
"""
import os
from flask import Flask, flash, request, redirect, url_for,send_from_directory
from werkzeug.utils import secure_filename

UPLOAD_FOLDER = '/path/to/the/uploads'
ALLOWED_EXTENSIONS = {'txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif','py'}

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        # check if the post request has the file part
        if 'file' not in request.files:
            flash('No file part')
            return redirect(request.url)
        file = request.files['file']
        # if user does not select file, browser also
        # submit an empty part without filename
        if file.filename == '':
            flash('No selected file')
            return redirect(request.url)
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            return redirect(url_for('uploaded_file',
                                    filename=filename))
    return '''
    <!doctype html>
    <title>Upload new File</title>
    <h1>Upload new File</h1>
    <form method=post enctype=multipart/form-data>
      <input type=file name=file>
      <input type=submit value=Upload>
    </form>
    '''
@app.route('/uploads/<filename>')
def uploaded_file(filename):
    return send_from_directory(app.config['UPLOAD_FOLDER'],filename)
```

运行结果如下：


127.0.0.1
Upload new File
选取文件
未选择文件
Upload

5、学习感想：

本次暑期科研，张帆老师向我们展示了大数据的整片森林，让我们大开了眼界，也帮我确定了我学习的目标。

第一周老师介绍了 python 的基础内容，并向我们提出了一个有意思的问题，字典中的一个特殊现象，课下我经过探索的思考，得到了答案，并获得了老师的肯定，这算是本次科研第一次探索与获得的经历。之后老师给我们介绍了 github 的各种操作方法，如何上传自己的文件，等等，让我第一次感受到用终端来输入命令行的精彩过程。最后老师发给我们了两段 mnist，使用 python 中的 tensorflow 库写出来的，由于对于此库不太熟悉，我在课下查阅大量文献，总算能够明白其大致运行原理。

第二周老师向我们介绍了大数据里的大杀器：Docker，它是一种集成部署的软件，老师向我们简述了其原理，并将其与一般的虚拟器进行比较，向我们解释了为何 docker 可以优于虚拟器那么多，是由于其舍去了下载 OS 操作系统的步骤，从而省下了大量存储空间。课后老师很详细的给我们布置了任务，我在课下学习和探索老师关于 docker 的文件，在实际操作中收获了成功的喜悦。

第三周老师给我讲述了 Cassandra 和一些老师博士阶段的研究内容，虽然内容比较有难度，但是老师会时不时关心同学，生怕我们听不懂。在老师的谆谆善诱下，我最终还是学会了 Cassandra 在 docker 上的使用。

第四周老师向我展示了正片大数据的森林，让我们耳目一新，眼前一亮，更让我坚定了一定要脚踏实地，不断努力学习的意志。

在短短的四周科研学习中，老师课下积极回答我提出的问题，为我答疑解难，也让我进步很多，一个月说长不长，说短不短，但是却让我的目标，让我的未来更加明晰，让我更加愿意付出自己的学习在数据分析，大数据这么一个未来充满着机会和机遇的方向！