机器学习-小作业

助教：吴志平、李姝萌、甄泰航

指导教师：高 阳，李文斌

zhipingwucn@163.com、lism@nju.edu.cn、3393938117@qq.com

（神经网络、K-means和决策树已经随堂布置）

剩下小作业提交截止时间：2023年5月1日23:59之前

发送到统一邮箱：nju_ml@163.com

神经网络作业

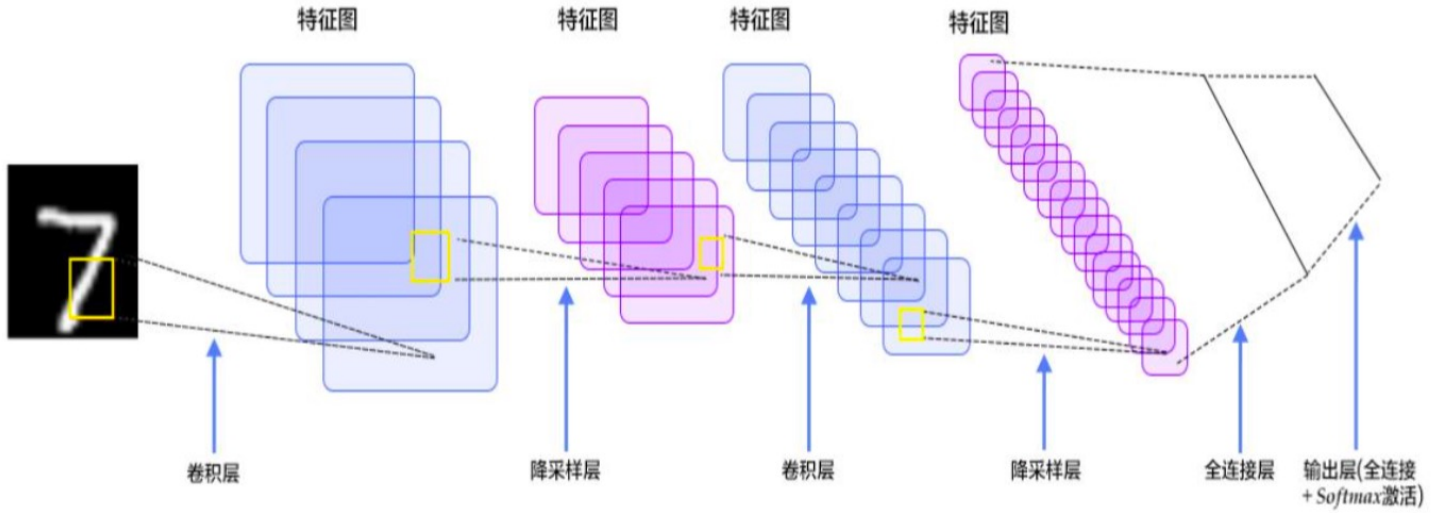
助教：甄泰航

指导教师：高 阳，李文斌

3393938117@qq.com

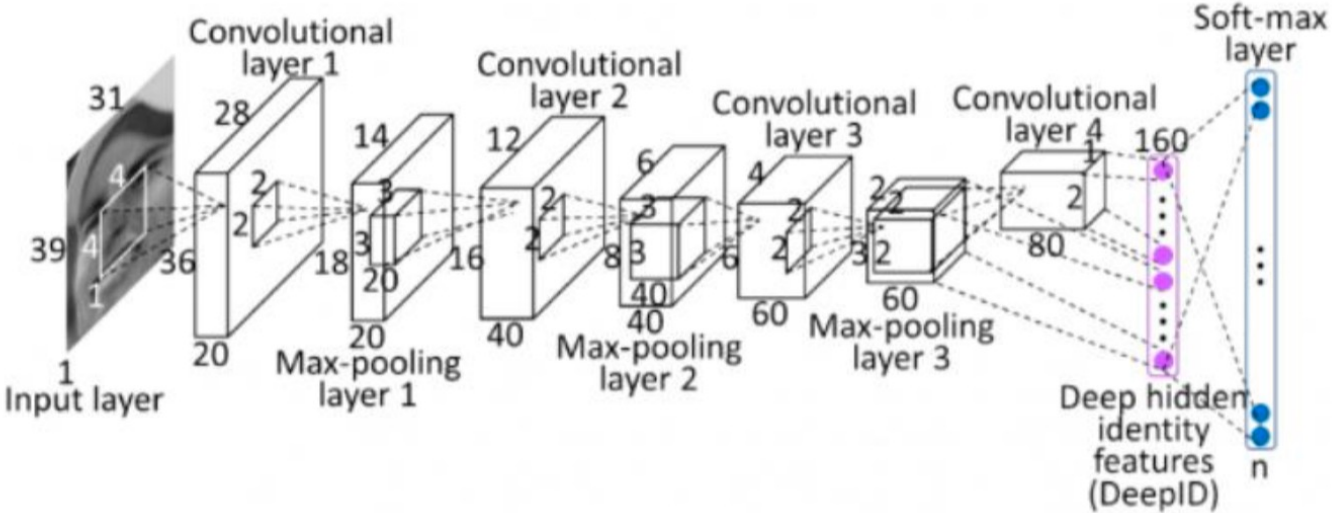
- 整体架构：

  - 输入层
  - 卷积层
  - 池化层
  - 全连接层



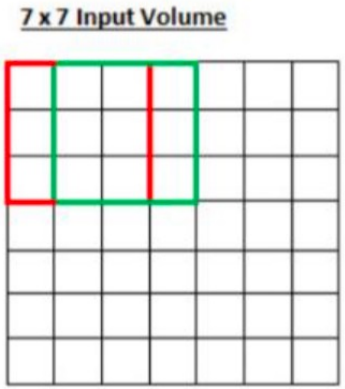
- 卷积层涉及参数：

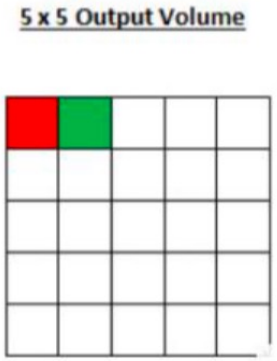
  - 滑动窗口步长
  - 卷积核尺寸
  - 边缘填充
  - 卷积核个数



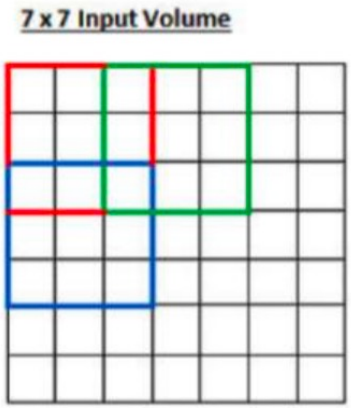
- 步长：

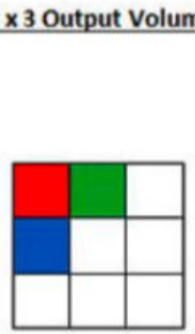
- 步长为1的卷积：



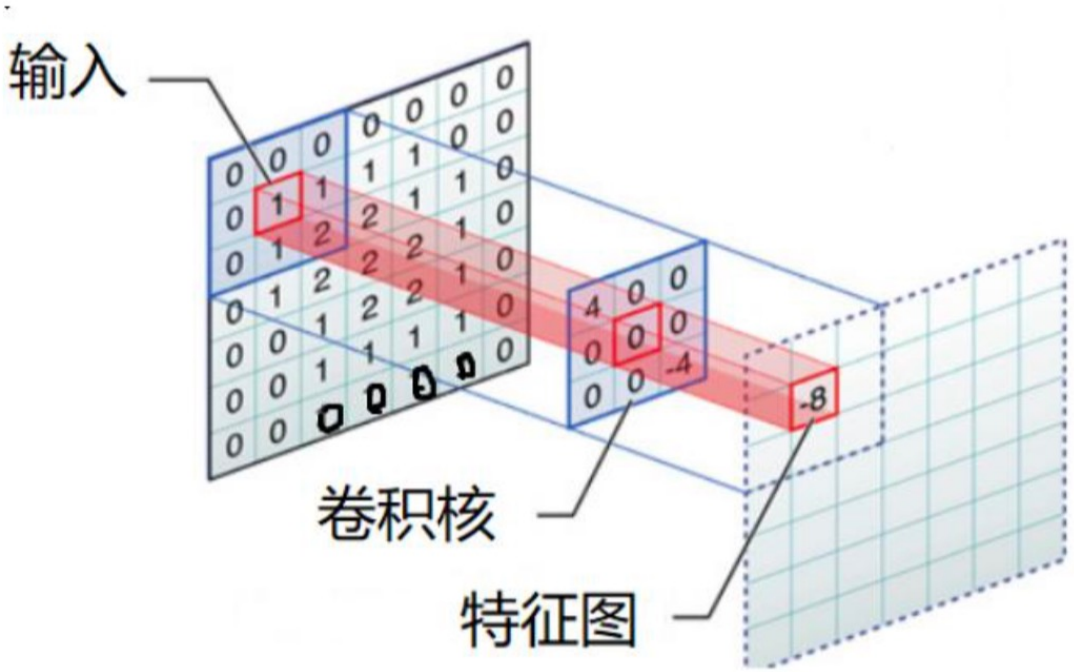


- 步长为2的卷积：





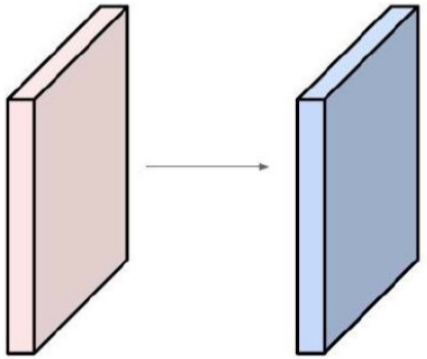
- 边界填充：



- 卷积结果计算公式：

  - 长度：$H_2 = \frac{H_1 - F_H + 2P}{S} + 1$

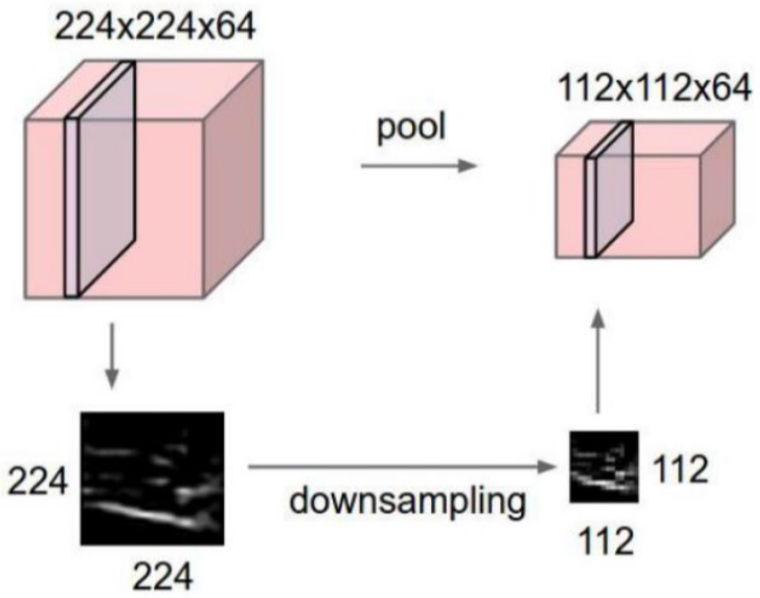
  - 宽度：$W_2 = \frac{W_1 - F_W + 2P}{S} + 1$



- 如果输入数据是32*32*3的图像，用10个5*5*3的filter来进行卷积操作，指定步长为1，边界填充为2，最终输入的规模为？

- (32-5+2*2)/1 + 1 = 32，所以输出规模为32*32*10，
  经过卷积操作后也可以保持特征图长度、宽度不变。

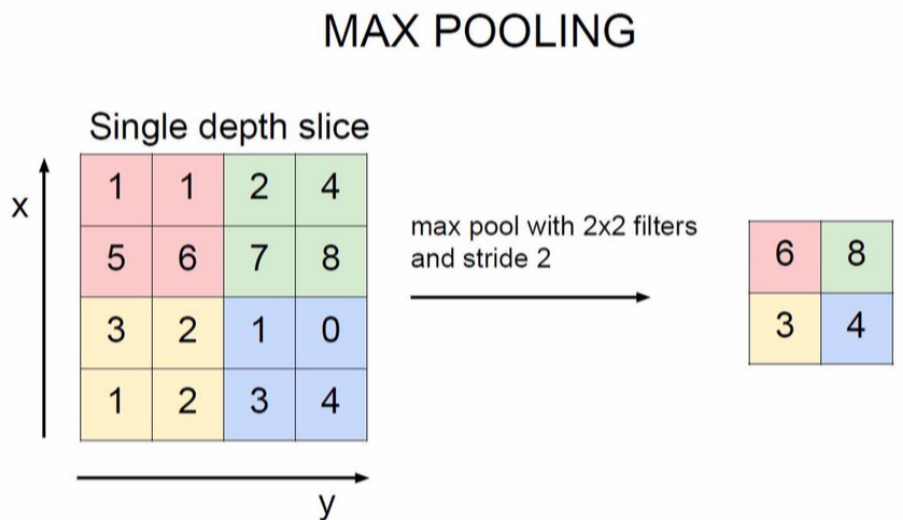
池化层：



1	3	2	9
7	4	1	5
8	5	2	3
4	2	1	4

7	9
8	

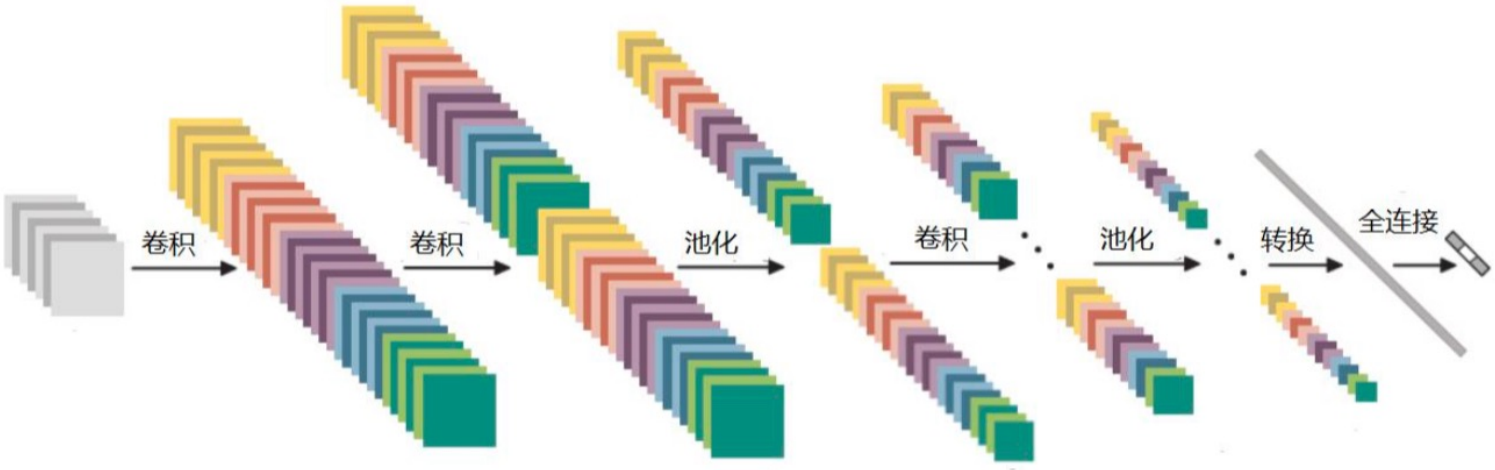
最大池化：



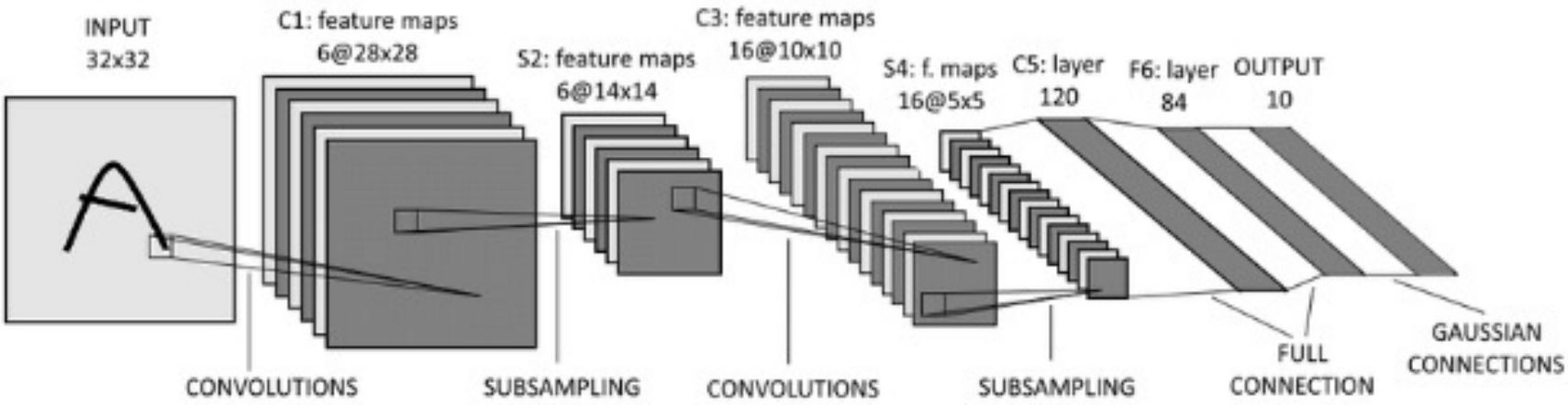
1	3	2	9
7	4	1	5
8	5	2	3
4	2	1	4

7	9
8	

- 特征图变化：



LeNet-5结构


INPUT
32x32
C1: feature maps
6@28x28
S2: feature maps
6@14x14
C3: feature maps
16@10x10
S4: f. maps
16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
CONVOLUTIONS
SUBSAMPLING
CONVOLUTIONS
SUBSAMPLING
FULL
CONNECTION
GAUSSIAN
CONNECTIONS

LeNet-5结构

0. Input image: 3x32x32
1. Conv layer:
    - kernel_size: 5x5
    - in_channels: 3
    - out_channels: 6
    - activation: ReLU
2. Max pooling:
    - kernel_size: 2x2
3. Conv layer:
    - kernel_size: 5x5
    - in_channels: 6
    - out_channels: 16
    - activation: ReLU
4. Max pooling:
    - kernel_size: 2x2
5. FC layer:
    - in_features: 16*5*5
    - out_features: 120
    - activation: ReLU
6. FC layer:
    - in_features: 120
    - out_features: 84
    - activation: ReLU
7. FC layer:
    - in_features: 84
    - out_features: 10 (number of classes)

作业要求

改进LeNet-5实现“手写体识别”。可以尝试使用一些图像预处理技术（去噪，归一化，分割等），再使用神经网络进行特征提取。

- **需要提交的内容包括但不限于：**
  - 可运行代码（.py），关键部分代码需要注释；
  - PDF报告，报告中要有明确的实验过程说明、精度截图等。

- **数据集说明：**
  - MNIST数据集：60000个训练样本和10000个测试样本组成，每个样本都是一张28 * 28像素的灰度手写数字0~9图片；

处理过程（样例）

- 搭配实验环境，导入包
- 给定输入输出分别构建训练集和测试集（验证集）
- **DataLoader来迭代取数据**

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import datasets,transforms
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

处理过程（样例）

- 搭配实验环境，导入包
- 给定输入输出分别构建训练集和测试集（验证集）
- DataLoader来迭代取数据

```
# 定义超参数
input_size = 28   #图像的总尺寸28*28
num_classes = 10   #标签的种类数
num_epochs = 3   #训练的总循环周期
batch_size = 64   #一个撮（批次）的大小，64张图片

# 训练集
train_dataset = datasets.MNIST(root='./data',
                               train=True,
                               transform=transforms.ToTensor(),
                               download=True)

# 测试集
test_dataset = datasets.MNIST(root='./data',
                              train=False,
                              transform=transforms.ToTensor())

# 构建batch数据
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)
```

- **卷积网络模块构建**

一般卷积层，relu层，池化层可以写成一个套餐

注意卷积最后结果还是一个特征图，需要把图转换成向量才能做分类或者回归任务。

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(         # 输入大小 (1, 28, 28)
            nn.Conv2d(
                in_channels=1,              # 灰度图
                out_channels=16,            # 要得到几多少个特征图
                kernel_size=5,              # 卷积核大小
                stride=1,                   # 步长
                padding=2,                  # 如果希望卷积后大小跟原来一样，需要设置padding
            ),                              # 输出的特征图为 (16, 28, 28)
            nn.ReLU(),                      # relu层
            nn.MaxPool2d(kernel_size=2),    # 进行池化操作（2x2 区域），输出结果为： (16, 14, 14)
        )
        self.conv2 = nn.Sequential(         # 下一个套餐的输入 (16, 14, 14)
            nn.Conv2d(16, 32, 5, 1, 2),     # 输出 (32, 14, 14)
            nn.ReLU(),                      # relu层
            nn.MaxPool2d(2),                # 输出 (32, 7, 7)
        )
        self.out = nn.Linear(32 * 7 * 7, 10)   # 全连接层得到的结果

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)           # flatten操作，结果为：(batch_size, 32 * 7 * 7)
        output = self.out(x)
        return output
```

- 准确率作为评估标准

```
def accuracy(predictions, labels):
    pred = torch.max(predictions.data, 1)[1]
    rights = pred.eq(labels.data.view_as(pred)).sum()
    return rights, len(labels)
```

- 训练网络模型

```
# 实例化
net = CNN()
#损失函数
criterion = nn.CrossEntropyLoss()
#优化器
optimizer = optim.Adam(net.parameters(), lr=0.001) #定义优化器，普通的随机梯度下降算法
```

```python
#开始训练循环
for epoch in range(num_epochs):
    #当前epoch的结果保存下来
    train_rights = []

    for batch_idx, (data, target) in enumerate(train_loader):  #针对容器中的每一个批进行循环
        net.train()
        output = net(data)
        loss = criterion(output, target)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        right = accuracy(output, target)
        train_rights.append(right)


        if batch_idx % 100 == 0:

            net.eval()
            val_rights = []

            for (data, target) in test_loader:
                output = net(data)
                right = accuracy(output, target)
                val_rights.append(right)

            #准确率计算
            train_r = (sum([tup[0] for tup in train_rights]), sum([tup[1] for tup in train_righ
            val_r = (sum([tup[0] for tup in val_rights]), sum([tup[1] for tup in val_rights]))

            print('当前epoch: {} [{}/{} ({:.0f}%)]\t损失: {:.6f}\t训练集准确率: {:.2f}%\t测试集正确率
```

评分标准

- 本次小实验设计希望同学们学习并掌握基本神经网络架构的同时进行代码实现，因此评分分数设计为：
  - 提交可运行代码（2`）
  - 按要求完成报告（2`）

备注：
- 所有分数总和若超过4分，一律按照4分处理；
- 在基本提交要求以外清晰的代码和报告书写，较高的精度作为加分项。

SVM小实验

助教：吴志平

指导教师：高阳 李文斌

zhipingwucn@163.com

作业要求

在自选数据集上用SVM实现一个分类器。数据集可自由选取，数据集下载。SVM算法可采用一些工具库（如scikit-learn等）完成。

- **需要提交的内容包括但不限于：**
  - 代码，关键部分代码需要注释；
  - 数据的可视化分析（不限制展现形式，条形，折线，点云等）
  - PDF文档，阐述SVM的基本原理，包含详细的步骤说明、运行结果和截图，给出精确率和召回率。
- **数据集说明：**
  - 数据说明（例：波士顿房价数据简略分析），可视化分析数据内部的分布偏差，清洗数据的过程；
  - 自己切分训练集和测试集设计完成实验；
  - 有兴趣的同学也可以尝试一下文本预处理的过程。
- **推荐阅读：**这里提供了一个比较基础的实验参考教程。

处理过程（邮件处理样例）

准备实验环境。推荐大家按照以下步骤（已经装好环境的同学可以跳过这一页）进行：

- 安装Anaconda。

  conda是一个Python环境管理工具，可以为你的项目建立一个纯净的Python环境，不受电脑里可能有多个Python版本的影响，链接。

- 在建立好的 conda 环境里安装 Jupyter Notebook（这里我的环境名字是PY3）。

  $(PY3) pip install jupyter

  同样也安装需要的其他库numpy、scikit-learn、pandas、matplotlib。

- 在项目目录运行jupyter。

  $(PY3) jupyter notebook

处理过程（邮件处理样例）

- Step 1：数据预处理

  去除无用字符，词干提取和词形还原（这些数据集里都已经完成）。

- Step2：读取文本内容

  拆分训练和测试数据集后，从文件夹中读取邮件的正文部分。

```
def extract_text(mail_dir):
    files = []
    for f in os.listdir(mail_dir):
        d = os.path.join(mail_dir, f)
        if os.path.isdir(d):
            files += [os.path.join(d, "ham", fi) for fi in os.listdir(os.path.join(d, "ham")) i

    for f in os.listdir(mail_dir):
        d = os.path.join(mail_dir, f)
        if os.path.isdir(d):
            files += [os.path.join(d, "spam", fi) for fi in os.listdir(os.path.join(d, "spam"))

    text_matrix = np.ndarray((len(files)), dtype=object)
    docID = 0
    for fil in files:
        with open(fil, 'r', errors="ignore") as fi:
            next(fi) # skip subject line
            data = fi.read().replace('\n', ' ')
            text_matrix[docID] = data
            docID += 1

    return text matrix
```

处理过程（邮件处理样例）

- Step 3：特征提取

  将文本内容转化成特征，这里我采用TF-IDF方法来计算词的权重，大家也可以尝试不同的方法。

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.svm import SVC, NuSVC, LinearSVC
import numpy as np
import os

train_dir = '/Users/dora/workspace/svm/train/'
train_matrix = extract_text(train_dir)

count_v1= CountVectorizer(stop_words = 'english', max_df = 0.5, decode_error = 'ignore', binary = True)
counts_train = count_v1.fit_transform(train_matrix)
tfidftransformer = TfidfTransformer()
tfidf_train = tfidftransformer.fit(counts_train).transform(counts_train)
```

- Step4：训练分类器

  使用scikit-learn可以很简单地构建一个SVM模型，这里使用了默认参数。

```
#训练
model = LinearSVC()
model.fit(tfidf_train, train_labels)
```

处理过程（邮件处理样例）

- Step5：测试和评估

  在测试集上验证我们的模型，测试数据提取特征时共用了训练集的词汇表，这里评估时计算了混淆矩阵、精确率和召回率。

```
from sklearn.metrics import confusion_matrix, precision_score, recall_score
import pandas as pd


test_dir = '/Users/dora/workspace/svm/test/'
test_matrix = extract_text(test_dir)
print(test_matrix.shape)

count_v2 = CountVectorizer(vocabulary=count_v1.vocabulary_,stop_words = 'english', max_df = 0.5, decode_error = 'ig
counts_test = count_v2.fit_transform(test_matrix)
tfidf_test = tfidftransformer.fit(counts_test).transform(counts_test)

test_labels = np.zeros(12718)
test_labels[6315:12718] = 1
result = model.predict(tfidf_test)

cm = pd.DataFrame(
    confusion_matrix(test_labels, result),
    index=['non-spam', 'spam'],
    columns=['non-spam', 'spam']
)
print(cm)

print("precision score:", precision_score(test_labels,result))
print("recall score:", recall_score(test_labels,result))
```

```
(12718,)
          non-spam  spam
non-spam      6010   305
spam            57  6346
precision score: 0.9541422342504886
recall score: 0.9910979228486647
```

评分标准

- 本次小实验设计希望同学们学习并掌握SVM原理的同时进行代码实现，因此评分分数设计为：
  - 完成作业要求，提交代码（3`）。
  - 按要求完成报告
    - 阐述SVM算法原理，对实验步骤进行阐述（1`）；
    - 展示实验结果，需要给出精确率和召回率，其他评估手段也可以加入到实验报告中。（2`）；
  - 注：所有对结果有帮助的步骤或尝试都可以作为加分项。

RL小作业

助教：吴志平（李超）

指导教师：高阳, 李文斌

zhipingwucn@163.com

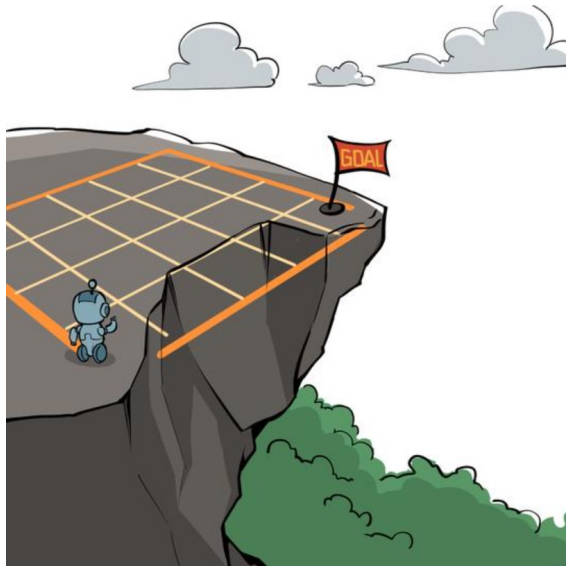
Demo with Cliff Walking

--悬崖漫步

强化学习（Reinforcement Learning，简称RL）是机器学习中的一个领域，强调如何基于环境而行动，以取得最大化的预期利益。

悬崖漫步（Cliff Walking）是一个非常经典的强化学习环境，它要求一个智能体从起点出发，避开悬崖行走，最终到达目标位置。

如图 1 所示，有一个 4×12 的网格世界，每一个网格表示一个状态。智能体的起点是左下角的状态，目标是右下角的状态，智能体在每一个状态都可以采取 4 种动作：上、下、左、右。

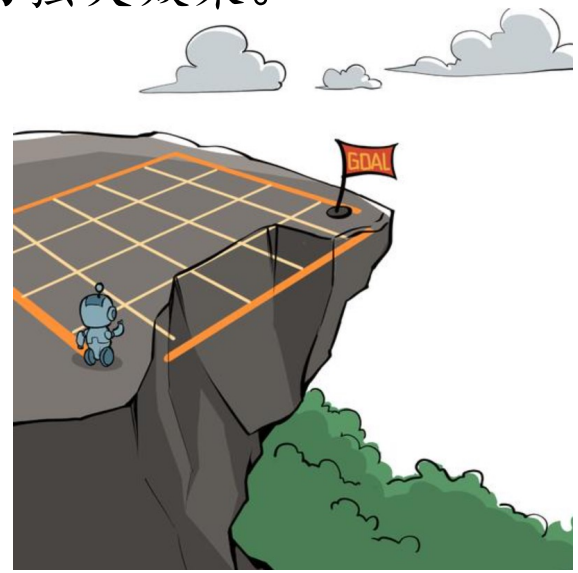


Demo with Cliff Walking

--悬崖漫步

如果智能体采取动作后触碰到边界墙壁则状态不发生改变，否则就会相应到达下一个状态。环境中有一段悬崖，智能体掉入悬崖或到达目标状态都会结束动作并回到起点，也就是说掉入悬崖或者达到目标状态是终止状态。

在一些需要执行动作，进行观察，积累经验，形成模型的现实任务场景中，强化学习算法展现出决策与博弈上的强大效果。



Demo with Cliff Walking （求解样例）

```
import copy
import gym
```

```
class CliffWalkingEnv:
    """ 悬崖漫步环境"""
    def __init__(self, ncol=12, nrow=4):
        self.ncol = ncol  # 定义网格世界的列
        self.nrow = nrow  # 定义网格世界的行
        # 转移矩阵P[state][action] = [(p, next_state, reward, done)]包含下一个状态和奖励
        self.P = self.createP()

    def createP(self):
        # 初始化
        P = [[[] for j in range(4)] for i in range(self.nrow * self.ncol)]
        # 4种动作, change[0]:上,change[1]:下, change[2]:左, change[3]:右。坐标系原点(0,0)
        # 定义在左上角
        change = [[0, -1], [0, 1], [-1, 0], [1, 0]]
        for i in range(self.nrow):
            for j in range(self.ncol):
                for a in range(4):
                    # 位置在悬崖或者目标状态,因为无法继续交互,任何动作奖励都为0
                    if i == self.nrow - 1 and j > 0:
                        P[i * self.ncol + j][a] = [(1, i * self.ncol + j, 0,
                                                    True)]
                        continue
                    # 其他位置
                    next_x = min(self.ncol - 1, max(0, j + change[a][0]))
                    next_y = min(self.nrow - 1, max(0, i + change[a][1]))
                    next_state = next_y * self.ncol + next_x
                    reward = -1
                    done = False
                    # 下一个位置在悬崖或者终点
                    if next_y == self.nrow - 1 and next_x > 0:
                        done = True
                        if next_x != self.ncol - 1:  # 下一个位置在悬崖
                            reward = -100
                    P[i * self.ncol + j][a] = [(1, next_state, reward, done)]
        return P
```

```python
def policy_improvement(self):  # 策略提升
    for s in range(self.env.nrow * self.env.ncol):
        qsa_list = []
        for a in range(4):
            qsa = 0
            for res in self.env.P[s][a]:
                p, next_state, r, done = res
                qsa += p * (r + self.gamma * self.v[next_state] * (1 - done))
            qsa_list.append(qsa)
        maxq = max(qsa_list)
        cntq = qsa_list.count(maxq)  # 计算有几个动作得到了最大的Q值
        # 让这些动作均分概率
        self.pi[s] = [1 / cntq if q == maxq else 0 for q in qsa_list]
    print("策略提升完成")
    return self.pi

def policy_iteration(self):  # 策略迭代
    while 1:
        self.policy_evaluation()
        old_pi = copy.deepcopy(self.pi)  # 将列表进行深拷贝,方便接下来进行比较
        new_pi = self.policy_improvement()
        if old_pi == new_pi: break
```

```python
class PolicyIteration:
    """ 策略迭代算法 """
    def __init__(self, env, theta, gamma):
        self.env = env
        self.v = [0] * self.env.ncol * self.env.nrow  # 初始化价值为0
        self.pi = [[0.25, 0.25, 0.25, 0.25]
                   for i in range(self.env.ncol * self.env.nrow)]  # 初始化为均匀随机策略
        self.theta = theta  # 策略评估收敛阈值
        self.gamma = gamma  # 折扣因子

    def policy_evaluation(self):  # 策略评估
        cnt = 1  # 计数器
        while 1:
            max_diff = 0
            new_v = [0] * self.env.ncol * self.env.nrow
            for s in range(self.env.ncol * self.env.nrow):
                qsa_list = []  # 开始计算状态s下的所有Q(s,a)价值
                for a in range(4):
                    qsa = 0
                    for res in self.env.P[s][a]:
                        p, next_state, r, done = res
                        qsa += p * (r + self.gamma * self.v[next_state] * (1 - done))
                        # 本章环境比较特殊,奖励和下一个状态有关,所以需要和状态转移概率相乘
                    qsa_list.append(self.pi[s][a] * qsa)
                new_v[s] = sum(qsa_list)  # 状态价值函数和动作价值函数之间的关系
                max_diff = max(max_diff, abs(new_v[s] - self.v[s]))
            self.v = new_v
            if max_diff < self.theta: break  # 满足收敛条件,退出评估迭代
            cnt += 1
        print("策略评估进行%d轮后完成" % cnt)
```

```python
def print_agent(agent, action_meaning, disaster=[], end=[]):
    print("状态价值：")
    for i in range(agent.env.nrow):
        for j in range(agent.env.ncol):
            # 为了输出美观,保持输出6个字符
            print('%6.6s' % ('%.3f' % agent.v[i * agent.env.ncol + j]), end=' ')
        print()

    print("策略：")
    for i in range(agent.env.nrow):
        for j in range(agent.env.ncol):
            # 一些特殊的状态,例如悬崖漫步中的悬崖
            if (i * agent.env.ncol + j) in disaster:
                print('****', end=' ')
            elif (i * agent.env.ncol + j) in end:  # 目标状态
                print('EEEE', end=' ')
            else:
                a = agent.pi[i * agent.env.ncol + j]
                pi_str = ''
                for k in range(len(action_meaning)):
                    pi_str += action_meaning[k] if a[k] > 0 else 'o'
                print(pi_str, end=' ')
        print()
```

```
env = CliffWalkingEnv()
action_meaning = ['^', 'v', '<', '>']
theta = 0.001
gamma = 0.9
agent = PolicyIteration(env, theta, gamma)
agent.policy_iteration()
print_agent(agent, action_meaning, list(range(37, 47)), [47])
```

```
策略评估进行60轮后完成
策略提升完成
策略评估进行72轮后完成
策略提升完成
策略评估进行44轮后完成
策略提升完成
策略评估进行12轮后完成
策略提升完成
策略评估进行1轮后完成
策略提升完成
状态价值:
-7.712 -7.458 -7.176 -6.862 -6.513 -6.126 -5.695 -5.217 -4.686 -4.095 -3.439 -2.710
-7.458 -7.176 -6.862 -6.513 -6.126 -5.695 -5.217 -4.686 -4.095 -3.439 -2.710 -1.900
-7.176 -6.862 -6.513 -6.126 -5.695 -5.217 -4.686 -4.095 -3.439 -2.710 -1.900 -1.000
-7.458  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
策略:
ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovoo
ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovoo
ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ovoo
^ooo **** **** **** **** **** **** **** **** **** **** EEEE
```

```python
class ValueIteration:
    """ 价值迭代算法 """
    def __init__(self, env, theta, gamma):
        self.env = env
        self.v = [0] * self.env.ncol * self.env.nrow  # 初始化价值为0
        self.theta = theta  # 价值收敛阈值
        self.gamma = gamma
        # 价值迭代结束后得到的策略
        self.pi = [None for i in range(self.env.ncol * self.env.nrow)]

    def value_iteration(self):
        cnt = 0
        while 1:
            max_diff = 0
            new_v = [0] * self.env.ncol * self.env.nrow
            for s in range(self.env.ncol * self.env.nrow):
                qsa_list = []  # 开始计算状态s下的所有Q(s,a)价值
                for a in range(4):
                    qsa = 0
                    for res in self.env.P[s][a]:
                        p, next_state, r, done = res
                        qsa += p * (r + self.gamma * self.v[next_state] * (1 - done))
                    qsa_list.append(qsa)  # 这一行和下一行代码是价值迭代和策略迭代的主要区别
                new_v[s] = max(qsa_list)
                max_diff = max(max_diff, abs(new_v[s] - self.v[s]))
            self.v = new_v
            if max_diff < self.theta: break  # 满足收敛条件,退出评估迭代
            cnt += 1
        print("价值迭代一共进行%d轮" % cnt)
        self.get_policy()

    def get_policy(self):  # 根据价值函数导出一个贪婪策略
        for s in range(self.env.nrow * self.env.ncol):
            qsa_list = []
            for a in range(4):
                qsa = 0
                for res in self.env.P[s][a]:
                    p, next_state, r, done = res
                    qsa += p * (r + self.gamma * self.v[next_state] * (1 - done))
                qsa_list.append(qsa)
            maxq = max(qsa_list)
            cntq = qsa_list.count(maxq)  # 计算有几个动作得到了最大的Q值
            # 让这些动作均分概率
            self.pi[s] = [1 / cntq if q == maxq else 0 for q in qsa_list]
```

```
env = CliffWalkingEnv()
action_meaning = ['^', 'v', '<', '>']
theta = 0.001
gamma = 0.9
agent = ValueIteration(env, theta, gamma)
agent.value_iteration()
print_agent(agent, action_meaning, list(range(37, 47)), [47])
```

```
价值迭代一共进行14轮
状态价值：
-7.712 -7.458 -7.176 -6.862 -6.513 -6.126 -5.695 -5.217 -4.686 -4.095 -3.439 -2.710
-7.458 -7.176 -6.862 -6.513 -6.126 -5.695 -5.217 -4.686 -4.095 -3.439 -2.710 -1.900
-7.176 -6.862 -6.513 -6.126 -5.695 -5.217 -4.686 -4.095 -3.439 -2.710 -1.900 -1.000
-7.458  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
策略：
ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovoo
ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovoo
ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ovoo
^ooo **** **** **** **** **** **** **** **** **** **** EEEE
```

```python
env = gym.make("FrozenLake-v1")  # 创建环境
env = env.unwrapped  # 解封装才能访问状态转移矩阵P
env.render()  # 环境渲染,通常是弹窗显示或打印出可视化的环境

holes = set()
ends = set()
for s in env.P:
    for a in env.P[s]:
        for s_ in env.P[s][a]:
            if s_[2] == 1.0:  # 获得奖励为1,代表是目标
                ends.add(s_[1])
            if s_[3] == True:
                holes.add(s_[1])
holes = holes - ends
print("冰洞的索引:", holes)
print("目标的索引:", ends)

for a in env.P[14]:  # 查看目标左边一格的状态转移信息
    print(env.P[14][a])
```

```
冰洞的索引: {11, 12, 5, 7}
目标的索引: {15}
[(0.3333333333333333, 10, 0.0, False), (0.3333333333333333, 13, 0.0, False), (0.3333333333333333, 14, 0.0, False)]
[(0.3333333333333333, 13, 0.0, False), (0.3333333333333333, 14, 0.0, False), (0.3333333333333333, 15, 1.0, True)]
[(0.3333333333333333, 14, 0.0, False), (0.3333333333333333, 15, 1.0, True), (0.3333333333333333, 10, 0.0, False)]
[(0.3333333333333333, 15, 1.0, True), (0.3333333333333333, 10, 0.0, False), (0.3333333333333333, 13, 0.0, False)]
```

```
# 这个动作意义是Gym库针对冰湖环境事先规定好的
action_meaning = ['<', 'v', '>', '^']
theta = 1e-5
gamma = 0.9
agent = PolicyIteration(env, theta, gamma)
agent.policy_iteration()
print_agent(agent, action_meaning, [5, 7, 11, 12], [15])
```

```
策略评估进行25轮后完成
策略提升完成
策略评估进行58轮后完成
策略提升完成
状态价值:
 0.069   0.061   0.074   0.056
 0.092   0.000   0.112   0.000
 0.145   0.247   0.300   0.000
 0.000   0.380   0.639   0.000
策略:
<ooo ooo^ <ooo ooo^
<ooo **** <o>o ****
ooo^ ovoo <ooo ****
**** oo>o ovoo EEEE
```

```
action_meaning = ['<', 'v', '>', '^']
theta = 1e-5
gamma = 0.9
agent = ValueIteration(env, theta, gamma)
agent.value_iteration()
print_agent(agent, action_meaning, [5, 7, 11, 12], [15])
```

```
价值迭代一共进行60轮
状态价值:
 0.069   0.061   0.074   0.056
 0.092   0.000   0.112   0.000
 0.145   0.247   0.300   0.000
 0.000   0.380   0.639   0.000
策略:
<ooo ooo^ <ooo ooo^
<ooo **** <o>o ****
ooo^ ovoo <ooo ****
**** oo>o ovoo EEEE
```

Demo with Cliff Walking

--悬崖漫步

- **实验要求：**

提交完成上述算法设计和代码，报告内容包括但不限于：

  - 伪代码描述悬崖漫步中策略迭代和价值迭代的过程；
  - 结合代码描述：
    - 悬崖漫步环境设计；
    - 描述策略迭代过程；
    - 描述价价值迭代过程；
  - 提交实验结果，输出策略、价值迭代过程。

Demo with Cliff Walking

--悬崖漫步

- **参考资料：**
  - 《机器学习算法视角第二版》
    - 11.2 状态和行动空间，奖赏函数，折扣，策略；
  - 莫烦RL
    - 强化学习方法汇总；
  - Sutton 《Reinforcement Learning: An Introduction》
    - 本书中文版；
    - 本书配套代码推荐: 代码。

Demo with Cliff Walking

--悬崖漫步

- **评分标准：**

本次RL小实验设计主要为了解动态规划过程，所以侧重于通过完成报告阐述对该算法了解，因此各分数设计为：

  - 实现环境，策略，价值编码，提交代码(1`)；
  - 报告中描述策略、价值迭代算法伪代码(1`)；
  - 报告中包含：悬崖漫步环境，策略迭代，价值迭代算法(3`)；
  - 报告中包含：分析策略迭代，价值迭代算法(1`)。

决策树作业

助教：甄泰航

指导教师：高 阳，李文斌

3393938117@qq.com

作业要求

给定药品数据集构造决策树，并用Micro-F1和Macro-F1分数进行验证集评估，预测测试集中的药品等级。

- **提交测试集文件，pdf格式报告以及可运行代码（.py），报告内容包括但不限于：**
  - 数据的分析与处理；
  - 决策树的设计原理和核心代码；
  - 验证集评估结果（ Micro-F1和Macro-F1 截图）。
- **数据集说明：**
  - csv数据集中包含有关药品的各种属性，文件中可能包含“脏”数据，药品等级分为5级（1~5）；
  - 训练集——6999条数据

    验证集——1199条数据

    测试集——1798条数据

处理过程

- 数据预处理;
- 构建决策树划分标准函数;
- 创建决策树;
- 验证集评估（ Micro-F1和Macro-F1 可以直接sk-learn库调用）;
- 测试集预测;

处理过程（样例）

1. 数据集清洗并去除无关列（属性）——自行设计此处仅举例！！：

	UsefulCount	SideEffects	Rating
1	22	Mild Side Effects	5
2	17	Severe Side Effects	4
3	3	No Side Effects	5
4	35	Mild Side Effects	5
5	4	Severe Side Effects	5

2. 验证集评估：

```
Micro F1: 0.3561301084236864
Macro F1: 0.18588773759518742
```

处理过程（样例）

3. 测试集预测：

recordId	drugName	condition	reviewCom	date	usefulCour	sideEffects	rating
219597	Microgesti	Birth Contr	"I was on M	#######	1	Severe Side	2
134044	Prednisone	Cluster Hea	"Have had	#######	0	Moderate S	1
68176	Plan B	Emergency	"On June 6	6-Jul-15	13	Mild Side E	2
200538	Varenicline	Smoking C	"I tried Cha	7-Jul-11	7	Mild Side E	5
46409	Modafinil	Narcolepsy	"I was diag	1-Oct-09	11	Moderate S	5
212846	Sofosbuvir	Hepatitis C	"I'v e had Hep c for 42 years, Unknown to me. Had an occular migraine and ended up in hospital to find I had low platelets. Which led to hep c diagnosis First month sick as can be nausea, headache, fatigue and insomnia. But after 4 weeks	5-Apr-17	21	No Side Ef	5
220008	Amitriptylir	Pain	"Have beer	#######	86	Mild Side E	5
210425	Linzess	Constipatic	"I just start	#######	76	Mild Side E	4
86573	Invokana	Diabetes, T	"I started 1	2-Mar-15	46	Mild Side E	1
117477	Nortriptylir	Irritable Bo	"This medi	#######	21	Severe Side	4

参考资料

- 教材《机器学习 算法视角 第二版》
  - 第十二章，决策树学习
- 机器学习之-常见决策树算法(ID3、C4.5、CART)

评分标准

- 本次小实验设计希望同学们学习并掌握基本决策树原理的同时进行代码实现，因此评分分数设计为：
  - 提交测试集文件（1`）
  - 提交可运行代码（1`）
  - 按要求完成报告（2`）

备注：

- 所有分数总和若超过4分，一律按照4分处理；
- 在基本提交要求以外清晰的代码和报告书写，相对较好的F1分数作为加分项。

Kmeans作业

助教: 李姝萌

指导教师：高 阳、李文斌

lism@nju.edu.cn

作业要求

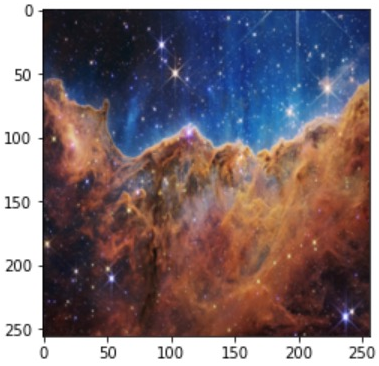
实现k-means聚类，并使用它进行无监督图像分割。挑选一张你喜欢的图片，如有需要可以进行预处理（缩放、去噪、归一化等），随机选取k个中心点作为聚类中心，进行迭代，得到不同k值下的分割预测结果。请自行设置停止迭代的条件。

- **需要提交的内容包括但不限于：**
  - 输入图像可视化、预处理可视化（如有）、分割结果可视化；
  - 可运行代码，关键部分代码需要注释；
  - PDF报告，其中阐述k-means算法的原理，包含详细的步骤说明、运行结果和截图。

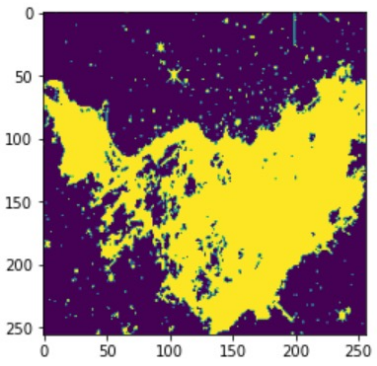
处理过程（样例）

- 如图是James Webb天文望远镜所拍摄到的一张船底座星云照片。
- 图中看起来很像“山脉”的区域是恒星形成区NGC3324的边缘，看起来从天体“山脉”升起的“蒸汽”实际上是由于强烈的紫外线辐射而从星云中流出的热电离气体和热尘埃。

- 图像可视化（重采样到256*256）



k-means无监督图像分割结果（k=2）



参考资料

- 教材《机器学习 算法视角 第二版》
  - 第十四章，无监督学习
- **KMEANS算法以及代码讲解**

评分标准

- 本次小实验设计希望同学们学习并掌握k-means聚类原理的同时进行代码实现，因此评分分数设计为：
  - 完成作业要求，提交代码（3`）
  - 按要求完成报告
    - 阐述k-means聚类算法原理（1`）；
    - 对实验步骤中的核心代码进行解释（1`）；
    - 图像可视化、展示实验结果（1`）。

备注：

- 在基本提交要求以外，所有对结果有帮助的步骤或尝试都可以作为加分项。