2023-2024学年第一学期本科生课程

《神经网络与深度学习》

课程实验作业(一)

主讲人：戴金晟(副教授，博士生导师)

daijincheng@bupt.edu.cn

神经网络与深度学习课程组

北京郵電大學

Beijing University of Posts and Telecommunications

内容导览

- 预备知识：特征工程
- 实验任务介绍
- 实验步骤说明

内容导览

- 预备知识：特征工程
- 实验任务介绍
- 实验步骤说明

特征工程

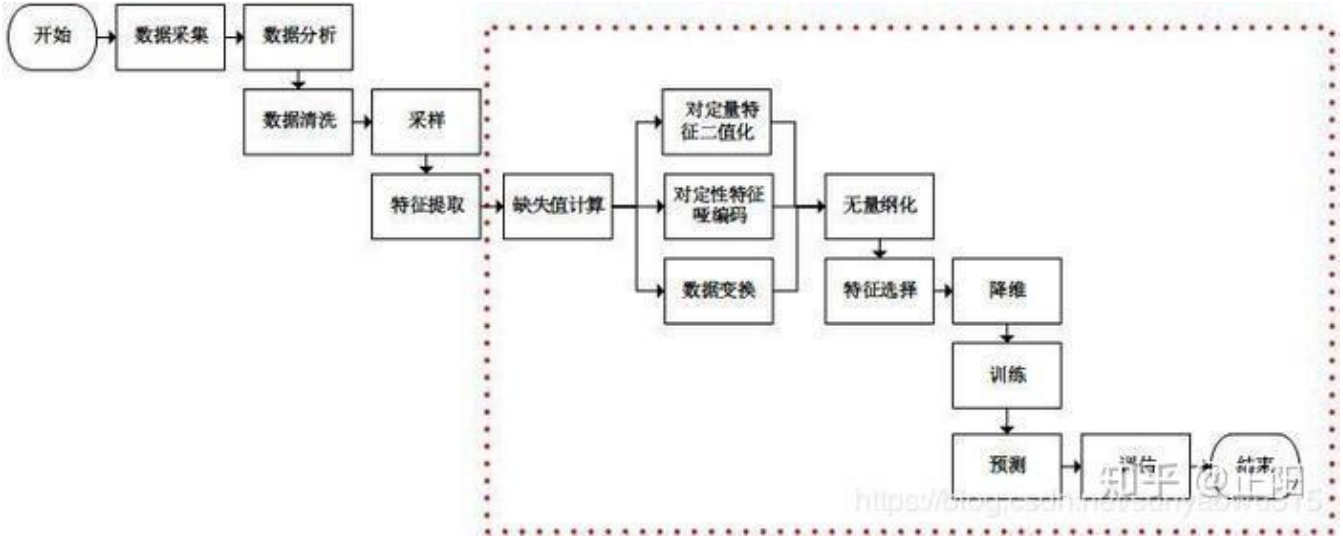
口特征工程的目的

- 将原始数据转化成更好的表达问题本质的特征的过程，使得将这些特征运用到预测模型中能提高对不可见数据的模型预测精度

机器学习算法的成功取决于如何呈现数据

口常用特征工程方法

- 数据清洗、采样、缺失值计算、数据变换、降维……



数据描述

- 特征信息
  - 一般包含**定量信息**（如体重）和**定性信息**（如吸烟史）
- 定性信息
  - 模型一般无法直接处理定性信息，需要将定性信息进行合适的编码（如独热编码），转变为定量信息
- 独热编码
  - 使用N位状态寄存器来对N个状态进行编码,每个状态都有它独立的寄存器位,并且在任意时候,其中只有一位有效。即,只有一位是1,其余都是零值。

$[0,\ 0,\ 1,\ 0,\ 0]$

五分类问题中的类别3

数据处理：缺失值处理

- 缺失值删除 (dropna)
  - 删除特征/删除实例
- 缺失值填充 (fillna)
  - 固定值填充：例如0，9999， -9999

    data['灰度分'] = data['灰度分'].fillna('-99')
  - 用均值填充：

    data['灰度分'] = data['灰度分'].fillna(data['灰度分'].mean()))
  - 用众数填充：

    data['灰度分'] = data['灰度分'].fillna(data['灰度分'].mode()))
  - 不填充：部分模型可将 NaN 作为数据一部分进行学习

数据处理：异常值处理

- 基于**统计**的异常点检测算法
  - 例如极差，四分位数间距，均差，标准差等，这种方法适合于挖掘单变量的数值型数据。
- 基于**距离**的异常点检测算法
  - 主要通过距离方法来检测异常点，将数据集中与大多数点之间距离大于某个阈值的点视为异常点，主要使用的距离度量方法有绝对距离和欧氏距离等方法。
- 基于**密度**的异常点检测算法
  - 考察当前点周围密度，可以发现局部异常点。
- 基于**相关知识**的检测方法
  - 基于一些预备知识进行判断，如人不可能活到200岁

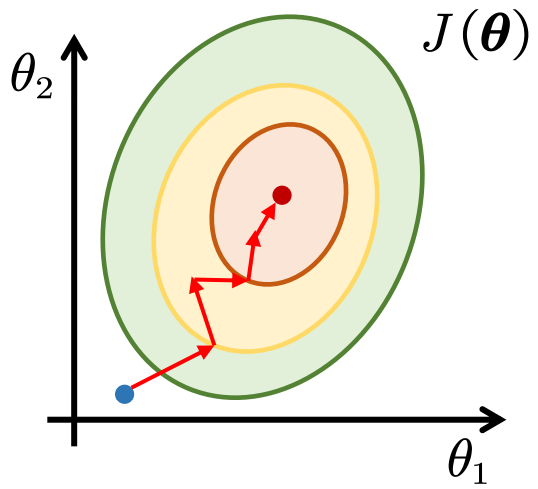
特征转换

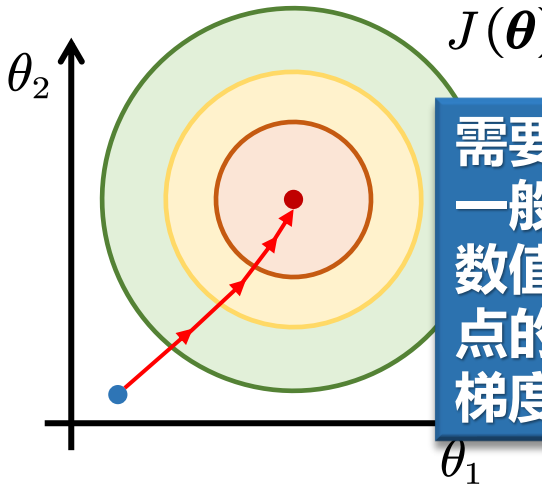
主要特征转换方式

类	功能	说明
StandardScaler	无量纲化	标准化，基于特征矩阵的列，将特征值转换至服从标准正态分布
MinMaxScaler	无量纲化	区间缩放，基于最大最小值，将特征值转换到[0, 1]区间上
Normalizer	归一化	基于特征矩阵的行，将样本向量转换为“单位向量”
Binarizer	二值化	基于给定阈值，将定量特征按阈值划分
OneHotEncoder	哑编码	将定性数据编码为定量数据
Imputer	缺失值计算	计算缺失值，缺失值可填充为均值等
PolynomialFeatures	多项式数据转换	多项式数据转换
FunctionTransformer	自定义单元数据转换	使用单变元的函数来转换数据

为什么要做特征归一化/标准化?

- **提升模型精度**
  - 在进行距离有关的计算时，尺度大的特征会起决定性作用，为了消除特征间单位和尺度差异的影响
- **提升收敛速度，避免神经网络饱和**
- **尺度差异导致损失函数的等高线图可能是椭圆形**
  - 梯度方向垂直于等高线，下降会走 zig-zag 路线，而不是指向局部最小值





需要注意的是，树形模型一般不需要归一化，因为数值的缩放并不影响分裂点的位置，而不需要通过梯度下降更新得到

常用的归一化方法

口标准化 (standardization)

- 对数据的特征属性减去均值，除以方差，将其转化为**均值为0，方差为1的标准正态分布。**
- 一般，我们会记录训练集上的均值和方差，对训练集进行标准化之后，在后续测试使用模型时，使用训练集上的方差和均值对测试集进行标准化。
- **优点**：对离群噪声点鲁棒性高

$$\hat{x} = \frac{x - \mu}{\sigma}$$

常用的归一化方法

Min-max 归一化 (rescaling)

- 对数据的特征属性减去其最小值，再除以数据范围，将其转化为取值范围为 $[a, b]$ 的分布，常用目标范围为 $[0, 1]$ 和 $[-1, 1]$ 。
- **优点**：可以保留稀疏特征中的0， 并且可以解决到特征的方差很小的情况时的数据
- **缺点**：有新点加入时，可能影响最大/最小值，进而改变归一化结果。所以，其对噪声点很敏感，这种方法只适用于**数据在一个范围内分布**的情况

$$\hat{x} = a + \frac{(x - \min(x))(b - a)}{\max(x) - \min(x)}$$

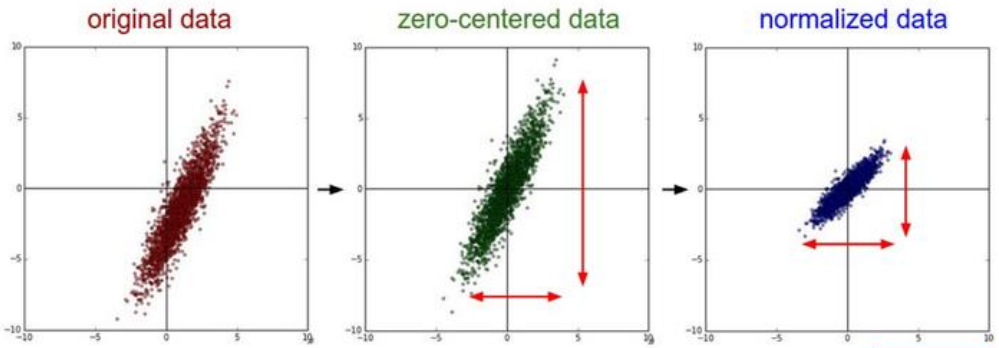
常用的归一化方法

口单位化 (scaling to unit length)

- 将每个样本的特征向量除以其长度，即对样本特征向量的长度进行归一化，长度的度量常使用的是 L_2 范数（欧氏距离），有时也会采用 L_1 范数
- 若选用 L_∞，此时除以特征的最大值，将特征缩放到 $[-1, 1]$ 之间。图像处理中，经常一个操作就是将int型图片矩阵，除以255转化为double型图片矩阵

$$\hat{x} = \frac{x}{\|x\|}$$

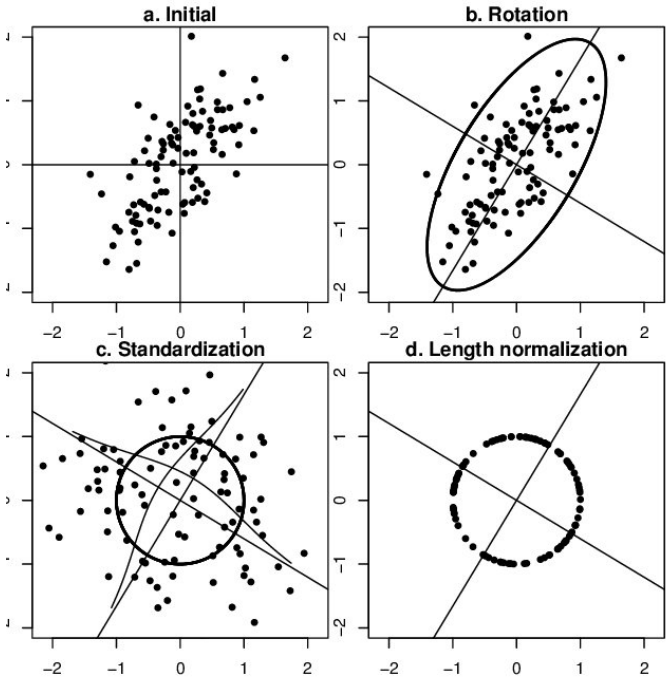
归一化方法分析



Common data preprocessing pipeline. **Left**: Original toy, 2-dimensional input data. **Middle**: The data is zero-centered by subtracting the mean in each dimension. The data cloud is now centered around the origin. **Right**: Each dimension is additionally scaled by its standard deviation. The red lines indicate the extent of the data - they are of unequal length in the middle, but of equal length on the right.

更直观地看到标准化过程的作用，会发现标准化也并不是万能的。使用标准化后，维度间也存在着相关性

可通过PCA方法解除特征间的线性相关性，即引入旋转，找到新的坐标轴方向，在新坐标轴方向上用“标准差”进行缩放，图右所示。图中同时描述了单位化的作用：将所有样本映射到单位球上。



主成分分析 (PCA)

- 主成分分析
  - 是一种线性降维算法，也是一种常用的数据预处理方法。它的目标是是用方差来衡量数据的差异性，并将差异性较大的高维数据投影到低维空间中进行表示。
- 使用 （本次实验暂不涉及）
  - 在sklearn中集成有该方法，该库的使用如下：

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
newX = pca.fit_transform(X)
```

内容导览

- 预备知识：特征工程
- 实验任务介绍
- 实验步骤说明

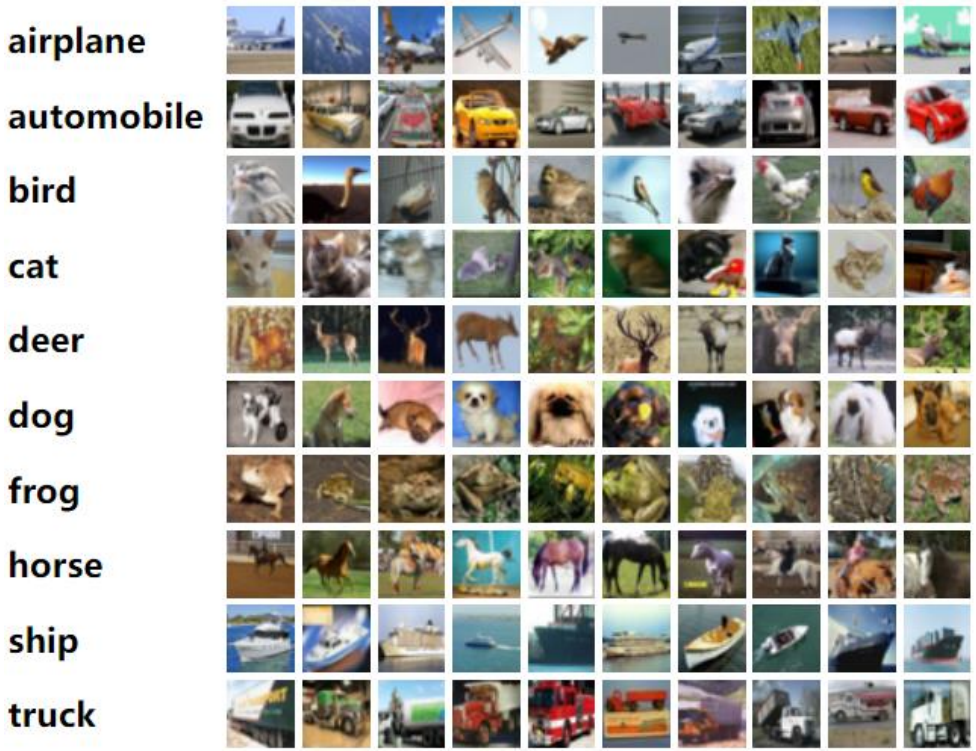
机器学习：回归问题

- 皮马印第安人糖尿病数据集由年龄大于等于21岁的皮马印第安女性的已有诊断信息组成，包含若干医学预测变量和一个目标变量Outcome，共九个字段。 预测变量包括患者的怀孕次数，BMI，胰岛素水平，年龄等。
- 请运用回归模型分析糖尿病数据集中自变量和因变量之间的关系，对某人是否患糖尿病进行预测。
- 数据集下载
  - Pima Indians Diabetes Database (kaggle.com)

序号	字段名	数据类型	字段描述
1	Pregnanci	Integer	怀孕次数
2	Glucose	Integer	口服葡萄糖耐量试验中血浆葡萄糖浓度为2小时
3	BloodPressure	Integer	舒张压（mm Hg）
4	SkinThickness	Integer	肱三头肌皮褶厚度（mm）
5	Insulin	Integer	2小时血清胰岛素（μU/ ml）
6	BMI	Integer	体重指数（体重kg /（身高m）^ 2）
7	DiabetesPedigreeFunction	Integer	糖尿病谱系功能
8	Age	Integer	年龄（岁）
9	Outcome	Integer	类变量（0或1）

多层感知机：分类问题

- 基于 CIFAR-10 数据集搭建包含三个及以上的全连接层的多层感知机网络，以解决10分类问题



包含10类目标，每类目标 6000 张图像，共60000张32×32彩色图像，其中训练图像50000张，测试图像10000张

多层感知机：分类问题

□CIFAR-10数据集的下载

- 官方下载

http://www.cs.toronto.edu/~kriz/cifar.html

- 腾讯微云下载（密码：nwdmtc）

https://link.zhihu.com/?target=https%3A//share.weiyun.com/56FKfYz

- 注意，请选择 python 版本的数据集进行下载

Download

If you're going to use this dataset, please cite the tech report at the bottom of this page.

Version	Size	md5sum
CIFAR-10 python version	163 MB	c58f30108f718f92721af3b95e74349a
CIFAR-10 Matlab version	175 MB	70270af85842c9e89bb428ec9976c926
CIFAR-10 binary version (suitable for C programs)	162 MB	c32a1d4ab5d03f1284b67883e8d87530

内容导览

- 预备知识：特征工程
- 实验任务介绍
- **实验步骤说明**

机器学习：回归问题

口数据集加载

- 直接使用 pandas.read_csv() 加载
- 使用 .head() 和 .describe() 查看数据集有关信息

In [2]:
```
import pandas as pd

diabetes = pd.read_csv('C:\\Users\\40412\\Desktop\\综合实验(一)实验材料\\数据集\\diabetes.csv')
diabetes.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [3]:
```
diabetes.describe()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

机器学习：回归问题

口模型加载

- 直接使用 sklearn.linear_model 加载
- 常用函数包含 fit, coef_, predict, score等

```
In [4]: from sklearn.linear_model import LogisticRegression, LinearRegression
        linear = LinearRegression()
        logic = LogisticRegression()
```

口数据集制作

- 使用 sklearn.model_selection 中的 train_test_split 划分训练集和测试集
- Train_size 控制训练集和测试集的比例，不同学号要求不同；random_state 控制采样结果，可以不设置

```
train_x, test_x, train_y, test_y = train_test_split(diabetes.data, diabetes.target, train_size=0.7, random_state=14)
```

机器学习：回归问题

口特征探索：挖掘特征间的关系

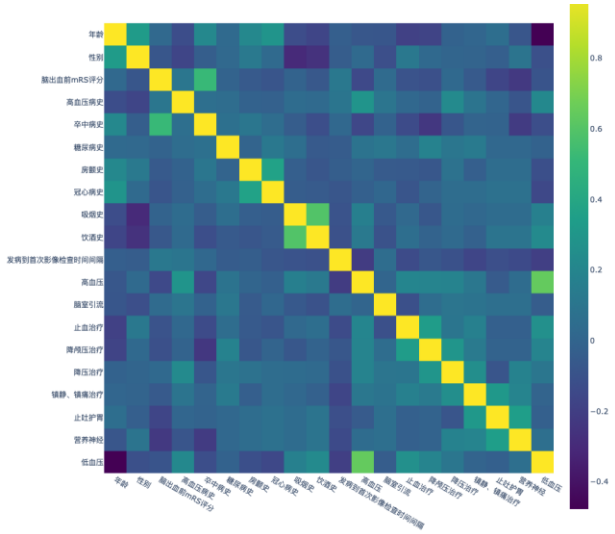
- 利用相关系数和相关性进行探索
- 需要注意，计算相关性时必须都是数值化特征

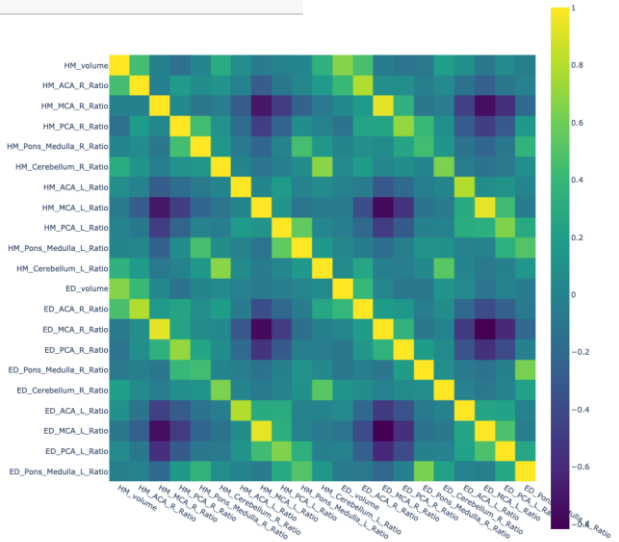
```
basic_info = iris_data_clean.columns[0:4]
```

```
## 可视化特征相关系数热力图
import plotly.express as px

## 计算数据的相关系数矩阵
corr = iris_data_clean[basic_info].corr()
fig = px.imshow(corr,width=1000,height=1000,
                ## 设置图像的填充颜色
                color_continuous_scale = px.colors.sequential.Viridis)
fig.update_layout(title={"x":0.5,"y":0.95})
fig.show()
```

这里的 basic_info 是由列名组成的序列

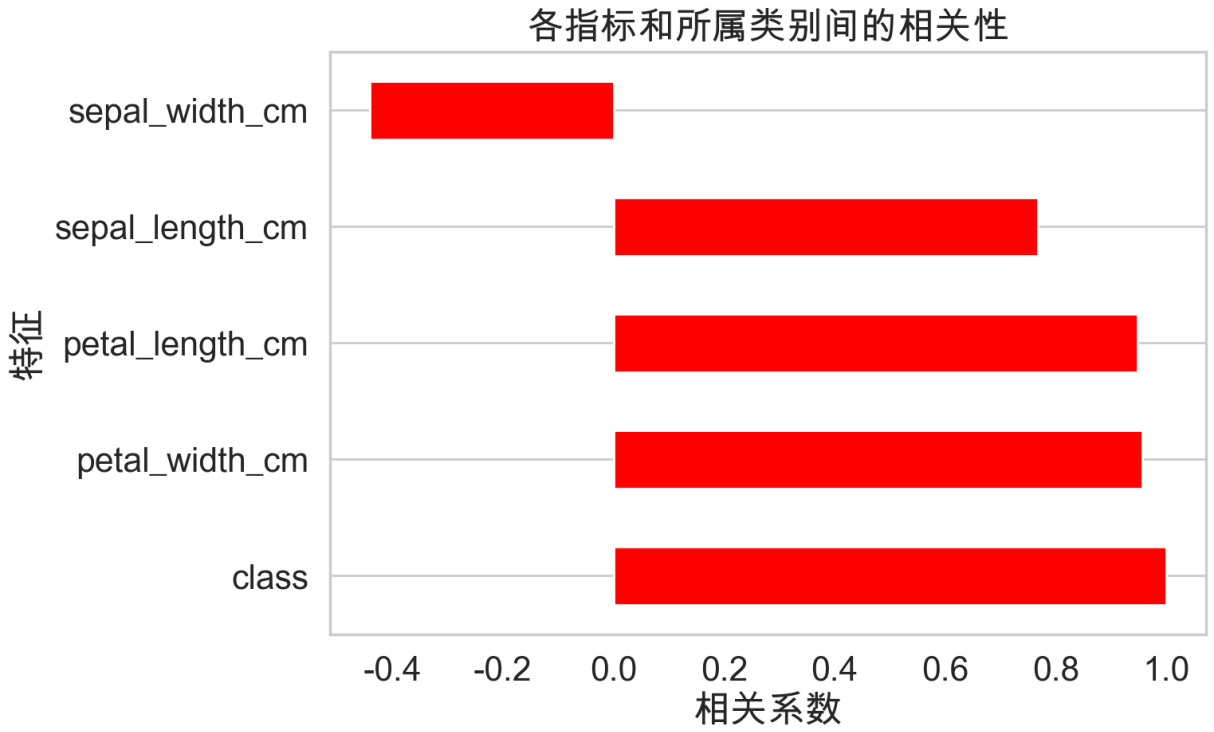




机器学习：回归问题

口特征探索：利用相关系数挖掘特征间的关系

```
correlations = iris_data_clean.corrwith(iris_data_clean['class']).dropna().sort_values(ascending=False)
plt.figure(figsize=(8, 5))
correlations.plot(kind='barh', color='red')
plt.title('各指标和所属类别间的相关性')
plt.xlabel('相关系数')
plt.ylabel('特征')
plt.grid(axis='x')
plt.tight_layout()
plt.show()
```



多层感知机：分类问题

口数据集加载

data_batch1~5是5个批次的训练数据，每个批次有10000张图像；test_batch是测试数据集，有10000张图像

名称	压缩后大小	原始大小
..		
batches.meta		158
data_batch_1		31,035,704
data_batch_2		31,035,320
data_batch_3		31,035,999
data_batch_4		31,035,696
data_batch_5		31,035,623
readme.html		88
test_batch		31,035,526

- 使用官方的解压代码处理数据集，解压后返回dict
- 也可以使用torchvision处理数据集

```python
#python2的解压代码
def unpickle(file):
    import cPickle
    with open(file, 'rb') as fo:
        dict = cPickle.load(fo)
    return dict

#python3的解压代码
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict
```

```python
#torchvision处理cifar10数据集代码
cifar10 = torchvision.datasets.CIFAR10(
    root='这里写存放cifar-10-batches-py的路径',
    train=True,
    download=False
)
cifar10_test = torchvision.datasets.CIFAR10(
    root='这里写存放cifar-10-batches-py的路径',
    train=False,
    download=False
)
```

多层感知机：分类问题

口模型构建

- 利用Pytorch框架搭建 MLP，num_i, num_h, num_o 分别为网络输入/隐藏层/输出层神经元个数

```
class MLP(torch.nn.Module):
    def __init__(self, num_i, num_h, num_o):
        super(MLP, self).__init__()

        self.linear1 = torch.nn.Linear(num_i, num_h) #输入层到第一层隐藏层的线性转换
        self.relu1 = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(num_h, num_h) #第一层隐藏层到第二层隐藏层的线性转换
        self.relu2 = torch.nn.ReLU()
        self.linear3 = torch.nn.Linear(num_h, num_o) #第二层隐藏层到输出层的线性转换

def forward(self, x):
        x = self.linear1(x)
        x = self.relu1(x)
        x = self.linear2(x)
        x = self.relu2(x)
        x = self.linear3(x)
        return x
```

如何将二维图像输入到 MLP中

多层感知机：分类问题

口模型训练

```
def train(model):
    loss_func = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters())
    epochs = 5
    for epoch in range(epochs):
        sum_loss = 0
        train_correct = 0
        for data in data_loader_train:
            inputs, labels = data
            inputs = torch.flatten(inputs, start_dim=1)
            outputs = model(inputs)
            optimizer.zero_grad()
            loss = loss_func(outputs, labels)
            loss.backward()
            optimizer.step()
            _, id = torch.max(outputs.data, 1)
            sum_loss += loss.data
            train_correct += torch.sum(id == labels.data)

        print('[%d/%d] loss:%.3f, correct:%.3f%%, time:%s' %
              (epoch + 1, epochs, sum_loss / len(data_loader_train),
               100 * train_correct / len(data_train),
               time.strftime('%Y-%m-%d %H:%M:%S', time.localtime())))
    model.eval()
```

- 设定优化器，采用Adam算法更新梯度
- 设定损失函数，分类问题可以通过交叉熵损失函数计算loss
- 设定超参数（迭代次数Epoch、批次大小batchsize、学习率lr）
- 获取数据和标签
- 将输入数据展平为一维向量
- 计算输出
- 梯度清零后计算损失函数
- 反向传播计算梯度
- 使用优化器更新模型参数
- 将每个批次的损失值累加用于计算平均损失
- 计算每个批次正确分类的图像数量

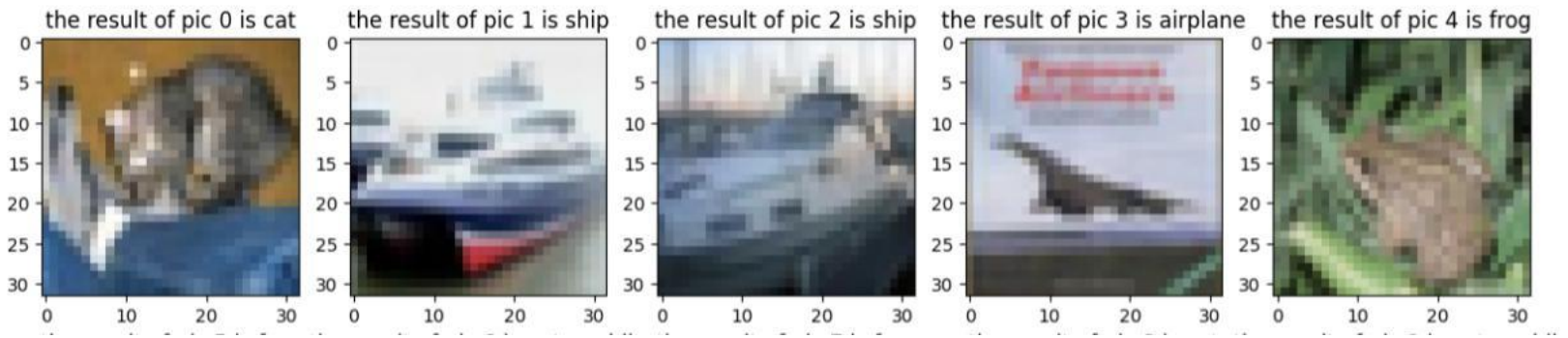
多层感知机：分类问题

口测试

```
def test(model, test_loader):
    test_correct = 0
    for data in test_loader:
        inputs, lables = data
        inputs, lables = Variable(inputs).cpu(), Variable(lables).cpu()
        inputs = torch.flatten(inputs, start_dim=1)
        outputs = model(inputs)
        _, id = torch.max(outputs.data, 1)
        test_correct += torch.sum(id == lables.data)
    print(f'Accuracy on test set: {100 * test_correct / len(data_test):.3f}%')
```

- 获取数据和标签
- 展平数据输入到网络
- 加载训练好的模型
- 获得输出

➢ 输出预测结果

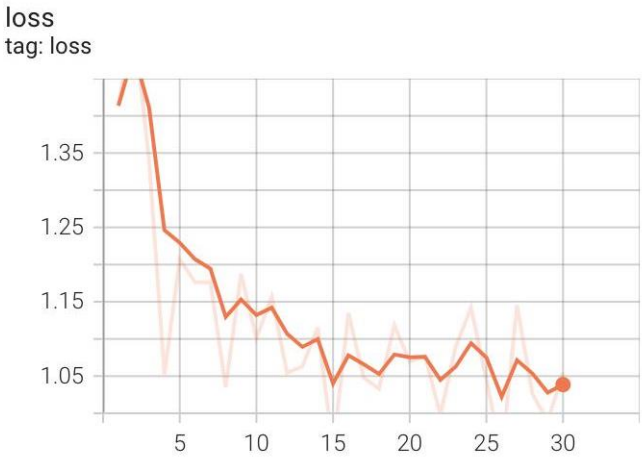


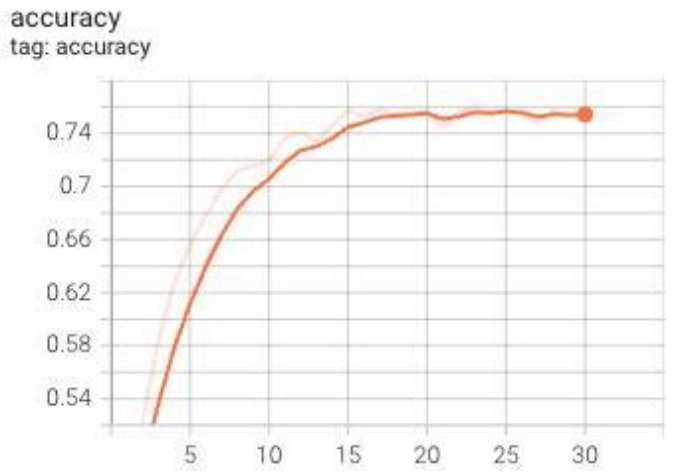
Tensorboard 的安装与使用

- 使用命令“pip install tensorboard”进行下载
- Tensorboard使用方法

```
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter('./path/to/log')
```

  - 导入tensorboard
  - 将代码运行过程中需要记录的数据保存在一个文件夹中，这一步由代码中的writer完成
  - “tensorboard –logdir=./path/to/the/folder --port 1002”读取这个文件夹中的数据，用浏览器访问localhost:1002/即可，这里的端口号是举例，也可以用其他未被占用的端口

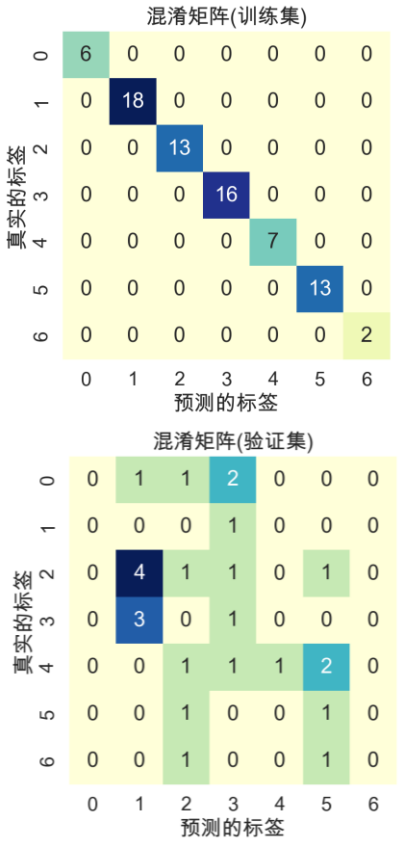




混淆矩阵的绘制

- 使用混淆矩阵可以评估分类模型的性能，能够直观的显示模型在哪一类样本里表现的分类效果不好
- 多分类任务的混淆矩阵如下图所示，其中矩阵的行表示真实标签，矩阵的列表示预测值

```
## 可视化在训练数据和验证数据上的混淆矩阵
train_confm = confusion_matrix(training_labels,model_pre_train)
val_confm = confusion_matrix(testing_labels,model_pre_test)
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
sns.heatmap(train_confm, square=True, annot=True, fmt='d',
            cbar=False,cmap="YlGnBu")
plt.xlabel("预测的标签")
plt.ylabel("真实的标签")
plt.title("混淆矩阵(训练集)")
plt.subplot(1,2,2)
sns.heatmap(val_confm, square=True, annot=True, fmt='d',
            cbar=False,cmap="YlGnBu")
plt.xlabel("预测的标签")
plt.ylabel("真实的标签")
plt.title("混淆矩阵(验证集)")
plt.tight_layout()
plt.show()
```



网络参数的调整

- 网络设计相关的参数
  - 网络深度、激活函数、隐藏层神经元的数量、损失函数的选择
- 训练过程相关的参数
  - 网络权重初始化方法、学习率、迭代次数、批次大小

```
class MLP(torch.nn.Module):
    def __init__(self, num_i, num_h, num_o):
        super(MLP, self).__init__()

        self.linear1 = torch.nn.Linear(num_i, num_h) #输入层到第一层隐藏层的线性转换
        self.relu1 = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(num_h, num_h) #第一层隐藏层到第二层隐藏层的线性转换
        self.relu2 = torch.nn.ReLU()
        self.linear3 = torch.nn.Linear(num_h, num_o) #第二层隐藏层到输出层的线性转换

def forward(self, x):
        x = self.linear1(x)
        x = self.relu1(x)
        x = self.linear2(x)
        x = self.relu2(x)
        x = self.linear3(x)
        return x
```

```
def train(model):
    loss_func = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters())
    epochs = 5
    for epoch in range(epochs):
        sum_loss = 0
        train_correct = 0
        for data in data_loader_train:
            inputs, labels = data
            inputs = torch.flatten(inputs, start_dim=1)
            outputs = model(inputs)
            optimizer.zero_grad()
            loss = loss_func(outputs, labels)
            loss.backward()
            optimizer.step()
            _, id = torch.max(outputs.data, 1)
            sum_loss += loss.data
            train_correct += torch.sum(id == labels.data)
```