# 实　验　报　告　书

课程名称：　专业综合实训

学　　院：　智能装备学院

专业班级：　信息工程 19-2

姓　　名：　田则林

指导老师：　闫长青、祝长生等

完成日期：　2022 年 6 月 11 日

**山东科技大学**

**2022 年 6 月**

# 项目目录



| | |
|---|---|
| **实训课题** | 统计学习方法实训 |
| **实训人姓名** | 田则林 201923040221 |
| **实训日期** | 2022 年 5 月 23 日至 2022 年 6 月 10 日 |
| **实训成绩** | |
| **指导教师评语** | **指导教师签名：**_____<br>2022 年 6 月 10 日 |

# 实训课题一：手写数字识别

## 一、实训内容：

在普通 pc 的设备上，调试编辑器 pycharm 与配置 pyhton3.7 运行环境，在 http://yann.lecun.com/exdb/mnist/上下载实训所需要的数据集进行本次的实训。理解 MNIST 数据集存储的格式，保证百分百准确的提取并转化为 Numpy 数据数据储存。在了解 KNN 算法的基本理论和算法下，编写程序来对手写数字的识别，做出相应的优化调整。

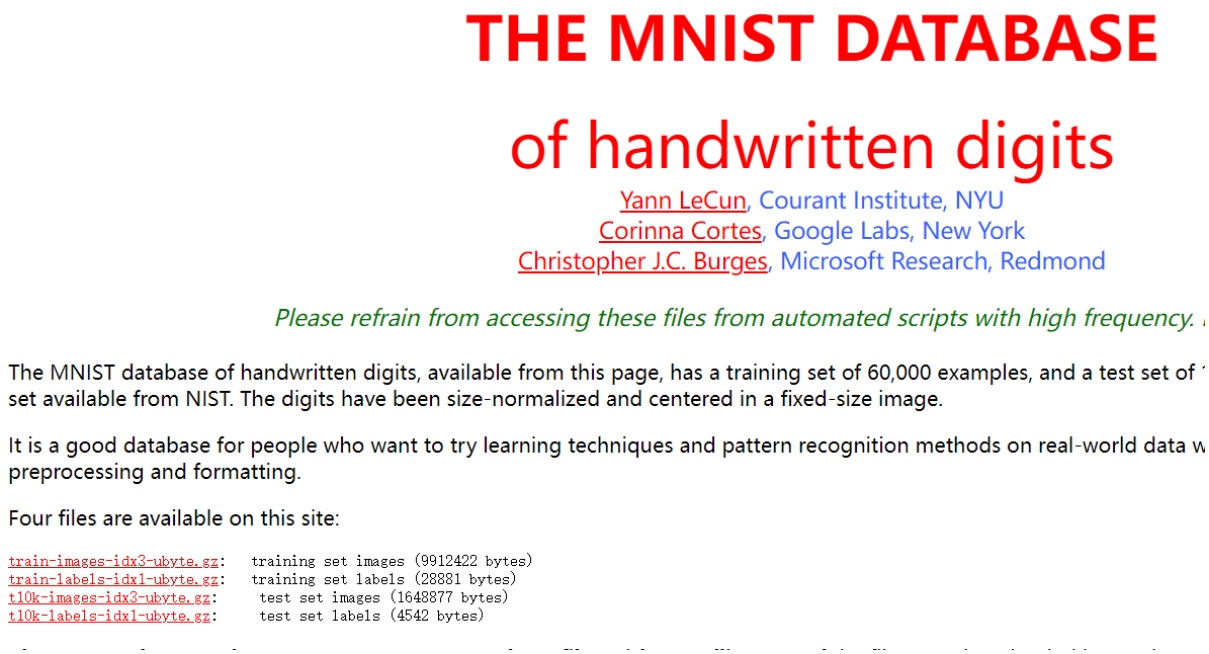
最基本 KNN 算法不需要用训练集训练出模型，只需要测试集测试的时候进行验证（不仅限于交叉验证），用最准确的 K 值预测手写数字。本次实训，我在实训指导书的基础上进行了对数据集进行了二值化的优化，对数据集的存储利用 KDTree 函数构造 K 维的 KD 二叉树，这将大大的减少运行的时间。

升级版本的 KNN 算法用 cnn 神经网络进行复杂的数据训练过程，其好处不言而喻，理论上而言：一次训练，终身收益，优雅自得；训练准确率可得到百分之 98 以上；预测手写识别不需要在庞大的训练集中过一遍，模型预留几个模板数据，速度极快。

其中需要注意：准确的分析手写数字的格式化处理方法和相似性判断标准，如何建立一个识别的模型，并针对 K 值，研究 K 值对手写数字识别准确率的影响。

## 二、原理及步骤：

### 1、下载数据



THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

*Please refrain from accessing these files from automated scripts with high frequency.*

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of
set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

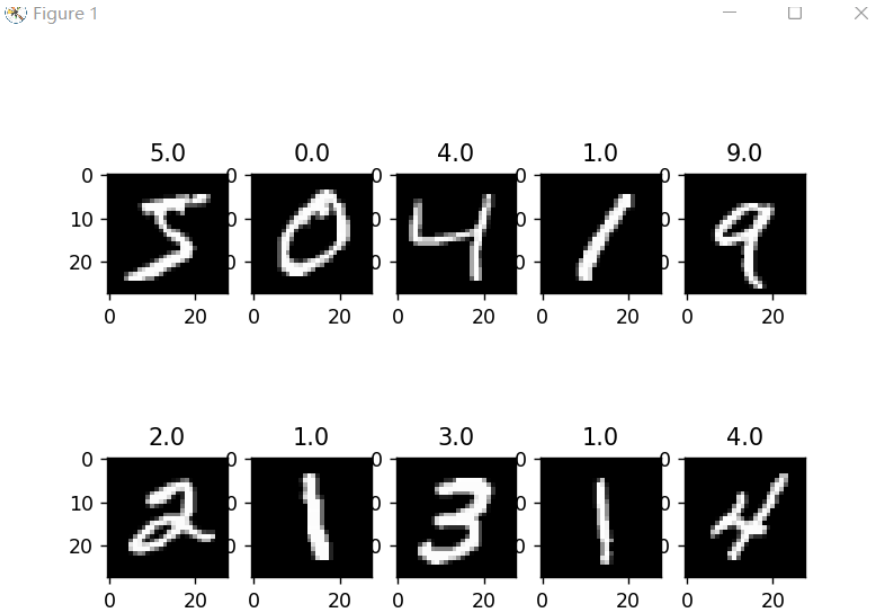
It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data w
preprocessing and formatting.

Four files are available on this site:

train-images-idx3-ubyte.gz: training set images (9912422 bytes)
train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)

MNIST（官方网站）是一个手写体数字的图片数据集，该数据集来由美国国家标准与技术研究所（National Institute of Standards and Technology (NIST)）发起整理，一共统计了来自 250 个不同的人手写数字图片，其中 50%是高中生，50%来自人口普查局的工作人员。该数据集的收集目的是希望通过算法，实现对手写数字的识别。

它由手写体数字的图片和相对应的标签组成，此数据集中，训练样本：共 60000 个，其中 55000 个用于训练，另外 5000 个用于验证；测试样本：共 10000 个。




MNIST 数据集分为训练图像和测试图像。训练图像 60000 张，测试图像 10000 张，每一个图片代表 0-9 中的一个数字，且图片大小均为 28*28 的矩阵。

train-images-idx3-ubyte.gz: training set images 训练图片

train-labels-idx1-ubyte.gz: training set labels 训练标签

t10k-images-idx3-ubyte.gz: test set images 测试图片

t10k-labels-idx1-ubyte.gz: test set labels 测试标签

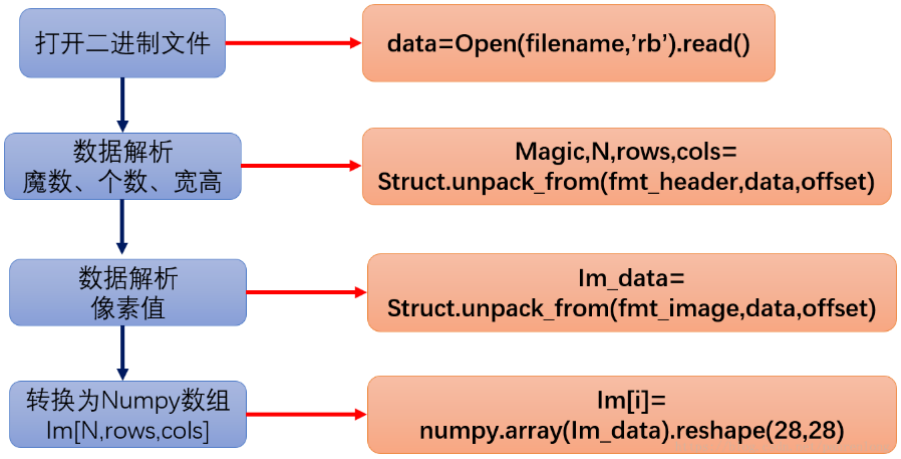
**2、数据预处理**

A、读入 MNIST 数据集

可以使用本地写程序识别，也可以用使用 tensorflow 读取。

直接下载下来的数据是无法通过解压或者应用程序打开的，因为这些文件不是任何标准的图像格式而是以字节的形式进行存储的，所以必须编写程序来打开它。

a）使用 python 读取二进制文件方法读取 mnist 数据集，则读进来的图像像素值为 0-255 之间；标签是 0-9 的数值。




b）采用 TensorFlow 的封装的函数读取 mnist，则读进来的图像像素值为 0-1 之间；标签是 0-1 值组成的大小为 110 的行向量。

**B、封装**

将读取程序封装成函数调用

**3、导入训练与测试数据**

创建 load_mnist 函数,将图片解析和标签解析放在一个，做好被调用准备。对算法进行改进，保留读取数据转化为 numpy 类型，不输出无用的提示信息。

在主函数中使用 load_MNIST 函数导入，训练集与测试集。

```
if __name__ == "__main__":
    train_img, train_label = load_mnist(train_images_file, train_labels_file)
    test_img, test_label = load_mnist(test_images_file, test_labels_file)
```

**4、KNN 算法**

邻近算法，或者说 K 最邻近（KNN，K-NearestNeighbor）分类算法是数据挖掘分类技术中最简单的方法之一。所谓 K 最近邻，就是 K 个最近的邻居的意思，说的是每个样本都可以用它最接近的 K 个邻近值来代表。近邻算法就是将数据集合中每一个记录进行分类的方法。

KNN（K-Nearest Neighbor）法即 K 最邻近法，最初由 Cover 和 Hart 于 1968 年提出，是一个理论上比较成熟的方法，也是最简单的机器学习算法之一。该方法的思路非常简单直观：如果一个样本在特征空间中的 K 个最相似（即特征空间中最邻近）的样本中的大多数属于某一个类别，则该样本也属于这个类别。该方

法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。

该方法的不足之处是计算量较大，因为对每一个待分类的文本都要计算它到全体已知样本的距离，才能求得它的 K 个最邻近点。目前常用的解决方法是事先对已知样本点进行剪辑，事先去除对分类作用不大的样本。另外还有一种 Reverse KNN 法，它能降低 KNN 算法的计算复杂度，提高分类的效率。

KNN 算法比较适用于样本容量比较大的类域的自动分类，而那些样本容量较小的类域采用这种算法比较容易产生误分。

KNN 算法的核心思想是，如果一个样本在特征空间中的 K 个最相邻的样本中的大多数属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特性。该方法在确定分类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。KNN 方法在类别决策时，只与极少量的相邻样本有关。由于 KNN 方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，KNN 方法较其他方法更为适合。

**总体来说，KNN 分类算法包括以下 4 个步骤：**

①准备数据，对数据进行预处理。

②计算测试样本点（也就是待分类点）到其他每个样本点的距离。

③对每个距离进行排序，然后选择出距离最小的 K 个点。

④对 K 个点所属的类别进行比较，根据少数服从多数的原则，将测试样本点归入在 K 个点中占比最高的那一类。

KNN 方法思路简单，易于理解，易于实现，无需估计参数。该算法在分类时有个主要的不足是，当样本不平衡时，如一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的 K 个邻居中大容量类的样本占多数。该方法的另一个不足之处是计算量较大，因为对每一个待分类的文本都要计算它到全体已知样本的距离，才能求得它的 K 个最近邻点。

**目前对 KNN 算法改进的方向主要可以分为 4 类：**

1、寻求更接近于实际的距离函数以取代标准的欧氏距离，典型的工作包括 WAKNN、

VDM； 2、是搜索更加合理的 K 值以取代指定大小的 K 值典型的工作包括 SNNB、DKNAW；3、是运用更加精确的概率估测方法去取代简单的投票机制，典型的工作包括 KNNDW、LWNB、ICLNB；

4、是建立高效的索引，以提高 KNN 算法的运行效率，代表性的研究工作包括 KDTree、 NBTree。还有部分研究工作综合了以上的多种改进方法。

**5、测试 KNN**

二值化：选择训练的数据集大小为 60000，预测数据集大小为 10000

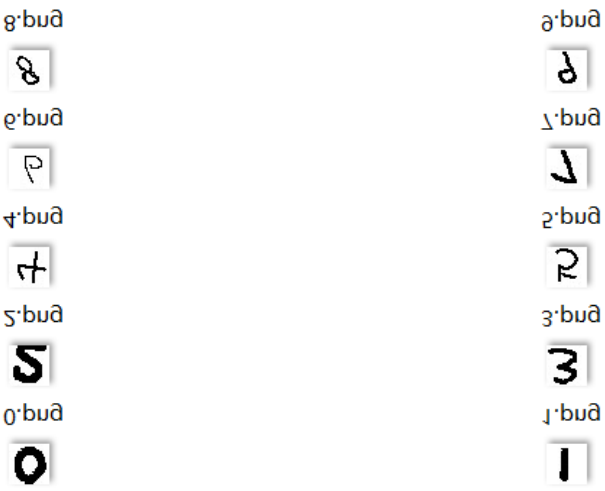
图片是 256 的灰度图像图片的存储方式为灰度图片，大小为 28*28，每个像素的取值范围在 0 到 255，如果直接运算会导致运算量过大，训练集巨大，程序运行时间非常长。简化后续的计算处理的方式为若像素值大于 127，则赋值为 1，否则赋值为 0。

KDtree():通过查找 scikit-learn 中文社区发现 Python sklearn.neighbors 模块中 KDTree()可以针对数据训练集进行分类，加速计算。Kdtree 是一种划分 k 维数据空间的数据结构，本质也是一颗二叉树，只不过每个节点的数据都是 k 维，当 k=1 时，就是图所示的普通二叉树。

输入测试集计算准确率，明确 K 值的影响，随 K 值的增大，大体上的准确率会先上升后下降，小部分有上下波动。K 值随大，准确率越低；在 K 值取 2-5 时，准确率最高。

**6、实际模型预测**

利用电脑自带的画图，进行手写数字 0-9，保存为 png 格式



测试识别率可以达到百分之 96%。

**参考论文：**

[1]罗贤锋，祝胜林，陈泽健,等. 基于 K-Medoids 聚类的改进 KNN 文本分类算法[J]. 计算机工程与设计，2014，35(11):5.

[1]伍以文. 重复剪辑近邻算法在数据分类中的应用[J]. 电脑开发与应用，2010,023(010):39,45.

[1]贾伟峰，杜保建，童彬,等. 采用压缩近邻法的高效入侵检测模型[J]. 计算机应用研究，2010(6):3.

## 三、结果分析：

1、单纯的 KNN 算法训练时，计算量巨大，运行时间极其漫长。

```
#调用Knn算法预测,默认K数值为5,train_img,train_label,test_img,test_label
Use_Knn(5,train_img,train_label,test_img,test_label)
print("此时的K值为：5")
print('\n\n\n\n\n')
```

可以看出准确率不高，运行时间极其漫长！（1.6h=5942s）

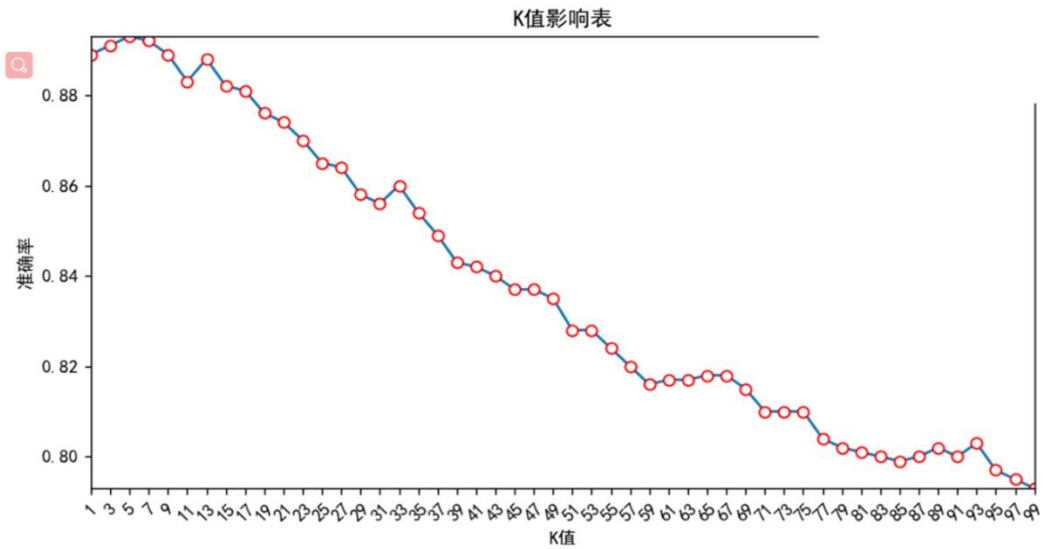
2、采用二值化简化计算

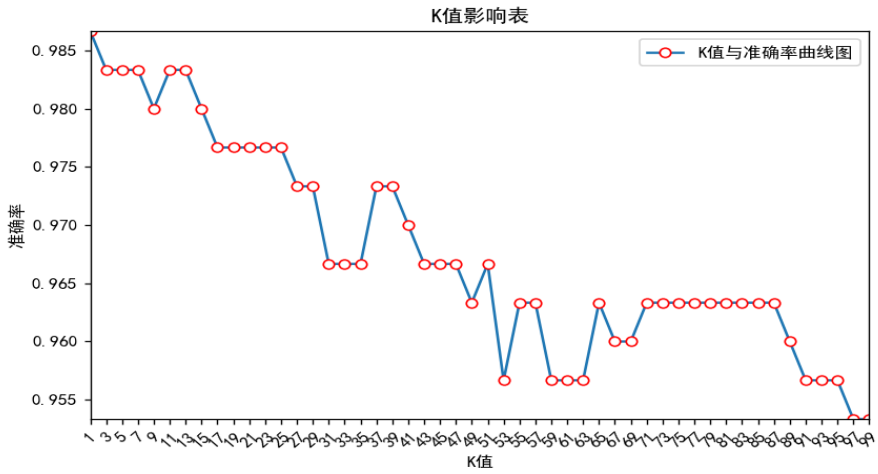
```
test_label的长度：  10000
准确度：  0.9587
此时的K值为：5
```

采用二值化化准确率上升，但是对计算时间减少的作用不明显。

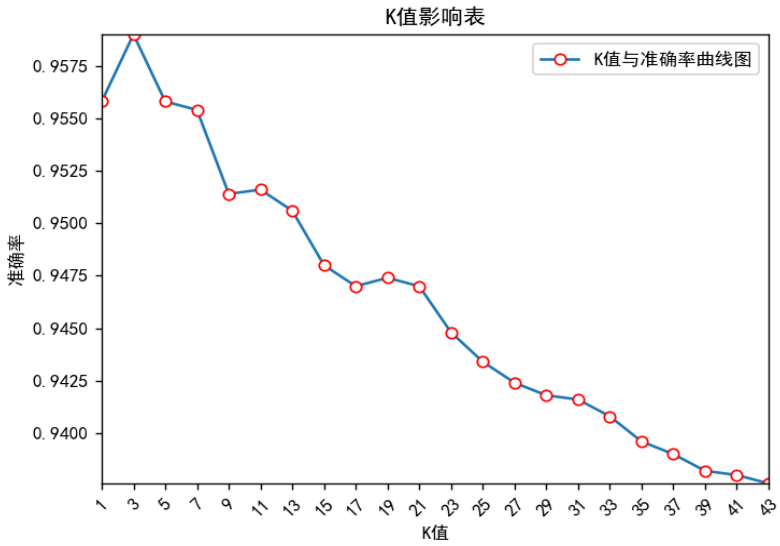
3、k 值影响

对 KNN 算法的数据集进行隔九个取一个（十分法），减小训练集与测试集，进行 k 值（1-100）影响的验证。






由图表大体分析可知，随着 k 值的变大，准确率逐渐减低。现在取 k 值为 0-43，样本取 5000，结果如右图：K 值随大，准确率越低；在 K 值取 2-5 时，准确率最高。




4、手写数字的识别

此优化算法对手写数字的识别度非常高，采用全部 60000（六万）张训练集，对 10 张手写数字照片进行识别，准确率可以达到百分之九十八。

```
判断预测样本的第 7 个
预测值为： 7 真确值为： [7] 是否准确： [True]
判断预测样本的第8个
 预测值为： 8 真确值为： [8] 是否准确： [True]
判断预测样本的第 0 个
预测值为： 9 真确值为： [9] 是否准确： [True]
test_label的长度： 10
准确度： 1.0
```

## 四、源代码

### 1、KNN 类

```
import numpy as np
from collections import Counter
from numpy import int8, zeros
class Knn:
    # 初始化函数，传递初始 K 值
    def __init__(self, k):
        self.k = k  # self.k 可以在类中使用
        #print("K 值：",self.k)
    # 训练函数，train 训练集 img 数据，label 标签
    def fit(self, train_img, train_label):
        # KNN 不需要训练，需要保存一下传入的训练信息
        self.train_X = (train_img)
        self.train_Y = (train_label)

    # 预测函数，预测的样本 test_img
    def predict(self, test_img,test_label):
        global num
        global pre
        pre=zeros((len(test_label), 1), dtype=int8)
        num = 0
        for test in test_img:
            print("判断预测样本的第",num,"个")
            pre[num] = self._predict(test,test_label,num)
            num = num + 1
        return pre
        # return [self._predict(test) for test in test_img]
        #  【】列表生成式，array 函数：将列表转化为数组

    # 预测函数，test 来自 test_img 预测集
    def _predict(self, x, test_label, num):
        # 计算距离（欧式距离）；x 预测的点，train_X 训练中的点
        dists = [self._dists(x, train_x) for train_x in self.train_X]
        # 将距离最接近的点取出来，排序，选择;argsort()排序函数;[:self.k]取前几个
        idx_knearrest = np.argsort(dists)[:self.k]
        # 将取到的数据转化为对应的标签
        idx_knearrest_list = [x for x in idx_knearrest]
        # 从 train_Y 中找到对应的标签集
        labels_knearrest = [int(self.train_Y[i]) for i in idx_knearrest_list]
        # Counter()函数返回数组中元素的个数，most_common(1)限定最多的那个[(4, 5), (3,
3)]中的[(4,5)]，[0][0]返回第几个数组的第几个数据
        most_common = Counter(labels_knearrest).most_common(1)[0][0]
```

```
            #判断是为正确
            most_common_arr = np.array(most_common)
            test_label_arr = test_label[num]
            t = [(most_common_arr == test_label_arr).all()]
            print("预测值为：",most_common,"真确值为：",test_label_arr,"是否准确：",t)
            return most_common

    # 距离函数的实现
    def _dists(self, x1, x2):
            length = np.sqrt(np.sum((x1 - x2) ** 2))
            return length
            # 距离平方的累计和开方
```

## 2、训练测试

```
import struct
from array import array as pyarray
from aKNN import Knn
from pylab import *   # 支持中文
mpl.rcParams['font.sans-serif'] = ['SimHei']

train_images_file = 'MNIST_data/train-images.idx3-ubyte' # 训练集文件
train_labels_file = 'MNIST_data/train-labels.idx1-ubyte' # 训练集标签文件
test_images_file = 'MNIST_data/t10k-images.idx3-ubyte' # 测试集文件
test_labels_file = 'MNIST_data/t10k-labels.idx1-ubyte' # 测试集标签文件

#读取数据
def load_mnist(fname_image, fname_label):
    digits = np.arange(10)

    flbl = open(fname_label, 'rb')
    magic_nr, size = struct.unpack(">II", flbl.read(8))
    lbl = pyarray("b", flbl.read())
    flbl.close()

    fimg = open(fname_image, 'rb')
    magic_nr, size, rows, cols = struct.unpack(">IIII", fimg.read(16))
    img = pyarray("B", fimg.read())
    fimg.close()

    ind = [k for k in range(size) if lbl[k] in digits]
    N = len(ind)

    images = zeros((N, rows * cols), dtype=uint8)
```

```
    labels = zeros((N, 1), dtype=int8)
    for i in range(N):
        images[i] = array(img[ind[i] * rows * cols: (ind[i] + 1) * rows * cols]).reshape((1, rows *
cols))
        labels[i] = lbl[ind[i]]
    return images, labels

#显示照片
def show_image(imgdata, imgtarget, show_column, show_row):
    # 注意这里的show_column*show_row==len(imgdata)
    for index, (im, it) in enumerate(list(zip(imgdata, imgtarget))):
        xx = im.reshape(28, 28)
        plt.subplot(show_row, show_column, index + 1)
        plt.axis('off')
        plt.imshow(xx, cmap='gray', interpolation='nearest')
        plt.title('target:%i' % it)

 # 归一化进程
def Data_ErZhiHua(data):
    print("数据二值化进程中！")
    row = data.shape[0]   # 60000
    col = data.shape[1]   # 784
    for i in range(row):
        for j in range(col):
            if data[i][j] > 127:
                data[i][j] = 1
            else:
                data[i][j] = 0
    return data

#Knn 算法实现
def Use_Knn(k,train_img,train_label,test_img,test_label):
    #K 值,
    knn = Knn(k)
    #训练，KNN 算法不需要训练，只是一个传递数据集的作用
    knn.fit(train_img, train_label)
    print("选择训练的数据集大小为：",train_img.shape[0])

    #用 test_img,test_label 数据集预测
    print("预测数据集大小为：", test_label.shape[0])
    preds = knn.predict(test_img,test_label)

    #输出的结果集为 preds
    # 准确度分析
```

```
    print("test_label 的长度：",len(test_label))
    acc = np.sum(preds == test_label) / len(test_label)
    print("准确度：",acc)
    # 测试 K 值的影响
    return acc


if __name__ == "__main__":
    train_img, train_label = load_mnist(train_images_file, train_labels_file)
    test_img, test_label = load_mnist(test_images_file, test_labels_file)
    #输出照片测试
    print(X_train[1])
    print("AAAA")
    images = np.array(X_train[1].reshape(28, 28))
    plt.imshow(images, 'gray')
    plt.show()
    print(X_label[1])

    '''测试训练集为 20000，测试集为 20'''
    train_num=6000
    test_num=1000

    train_img=Data_ErZhiHua(train_img[:train_num])
    test_img=Data_ErZhiHua(test_img[:test_num])
    train_label = train_label[:train_num]
    test_label = test_label[:test_num]
    print("数据二进制进程完成！")

    k_acc = []

    '''
    #调用函数进行图像的展示，9=3*3=1*9
    test_img = test_img[:9]
    test_label = test_label[:9]
    show_image(test_img, test_label, 3, 3)
    '''
    for i in range(1,100,2):#从 1 开始到 100（不包括 100）进入循环的条件中间的奇数,步长为 2
        k_acc.append(Use_Knn(i,train_img,train_label,test_img,test_label))
        print("此时的 K 值为：",i)
        print('\n\n\n\n')

    names = [x for x in range(1,100,2)]
    x = range(len(names))
```

```
    y = k_acc
    plt.plot(x, y, marker='o', mec='r', mfc='w', label=u'K 值与准确率曲线图')
    plt.legend()  # 让图例生效
    plt.xticks(x, names, rotation=45)
    plt.margins(0)
    plt.subplots_adjust(bottom=0.15)
    plt.xlabel("K 值")  #X 轴标签
    plt.ylabel("准确率")  #Y 轴标签
    plt.title("K 值影响表")  # 标题
    plt.show()
```

## 3、KDtree 部分

```
import struct
from collections import Counter
from array import array as pyarray
from pylab import *  # 支持中文
from sklearn.neighbors import KDTree
mpl.rcParams['font.sans-serif'] = ['SimHei']
# import warnings
# warnings.filterwarnings("ignore",category=DeprecationWarning)
train_images_file = 'MNIST_data/train-images.idx3-ubyte'  # 训练集文件
train_labels_file = 'MNIST_data/train-labels.idx1-ubyte'  # 训练集标签文件
test_images_file = 'MNIST_data/t10k-images.idx3-ubyte'  # 测试集文件
test_labels_file = 'MNIST_data/t10k-labels.idx1-ubyte'  # 测试集标签文件
# 读取数据
def load_mnist(fname_image, fname_label):
    digits = np.arange(10)

    flbl = open(fname_label, 'rb')
    magic_nr, size = struct.unpack(">II", flbl.read(8))
    lbl = pyarray("b", flbl.read())
    flbl.close()

    fimg = open(fname_image, 'rb')
    magic_nr, size, rows, cols = struct.unpack(">IIII", fimg.read(16))
    img = pyarray("B", fimg.read())
    fimg.close()

    ind = [k for k in range(size) if lbl[k] in digits]
    N = len(ind)

    images = zeros((N, rows * cols), dtype=uint8)
    labels = zeros((N, 1), dtype=int8)
```

```
    for i in range(N):
        images[i] = array(img[ind[i] * rows * cols: (ind[i] + 1) * rows * cols]).reshape((1, rows *
cols))
        labels[i] = lbl[ind[i]]
    return images, labels

def Kdtree_create(train_img):
    ## 构造 KD 树
    t_before = time.time()
    print("构造 KDtree 中，请稍微！")
    kd_tree = KDTree(train_img)
    t_after = time.time()
    t_training = t_after - t_before
    print("构建 KD 树需要多少(多少秒): ", t_training)
    return kd_tree

def preds(test_img,k,num):
    print("测试集测试中，请稍微！\nk 值为：", k)
    test_neighbors = np.squeeze(kd_tree.query(test_img, k, return_distance=False))
    kd_tree_predictions = train_label[test_neighbors] #[]是列表的意思
    kd_tree_prediction = np.array(kd_tree_predictions).reshape(num,k)
    # Counter()函数返回数组中元素的个数，most_common(1)限定最多的那个[(4, 5), (3, 3)]
中的[(4,5)]，[0][0]返回第几个数组的第几个数据
    kd_prediction = [Counter(kd_tree_prediction[i]).most_common(1)[0][0] for i in range(num)]
    kd_predictions = np.array(kd_prediction).reshape(num,1)
    acc = np.sum(kd_predictions == test_label) / len(test_label)
    return acc

if __name__ == "__main__":
    t_before = time.time()
    train_img, train_label = load_mnist(train_images_file, train_labels_file)
    test_img, test_label = load_mnist(test_images_file, test_labels_file)
    num = 300
    test_img = test_img[:num]
    test_label = test_label[:num]

    kd_tree = Kdtree_create(train_img)
    '''
    ## 得到测试集的预测结果，k 表示几个近邻
    t_before = time.time()
    k = 1
    print("测试集测试中，请稍微！\nk 值为：",k)
    test_neighbors = np.squeeze(kd_tree.query(test_img, k, return_distance=False))
    kd_tree_predictions = train_label[test_neighbors]
```

```
t_after = time.time()

##完成测试需要的时间
t_testing = t_after - t_before
print("完成测试需要的时间(多少秒): ", t_testing)

print("test_label 的长度：", len(test_label))
acc = np.sum(kd_tree_predictions == test_label) / len(test_label)
print("准确度：", acc)
'''
k_acc = []
j = 0
for k in range(1, 100, 2):
    acc = preds(test_img,k,num)
    k_acc.append(acc)
    print("准确度：", k_acc[j],'\n')
    j = j + 1

t_after = time.time()
t = t_after - t_before
print("耗时(多少秒): ", t)

list = [x for x in range(1, 100, 2)]
x = range(len(list))
y = k_acc
plt.plot(x, y, marker='o', mec='r', mfc='w', label=u'K 值与准确率曲线图')
plt.legend()  # 让图例生效
plt.xticks(x, list, rotation=45)
plt.margins(0)
plt.subplots_adjust(bottom=0.15)
plt.xlabel("K 值")  #X 轴标签
plt.ylabel("准确率")  #Y 轴标签
plt.title("K 值影响表")  # 标题
plt.show()
```

## 4、手写函数预测

```
import struct
from array import array as pyarray
import cv2
import numpy as np
from matplotlib import pyplot as plt
from numpy import array, int8, uint8, zeros
from aKNN import Knn
```

```
plt.rcParams['font.sans-serif'] = ['SimHei']    # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False

train_images_file = 'MNIST_data/train-images.idx3-ubyte' # 训练集文件
train_labels_file = 'MNIST_data/train-labels.idx1-ubyte' # 训练集标签文件
test_images_file = 'MNIST_data/t10k-images.idx3-ubyte' # 测试集文件
test_labels_file = 'MNIST_data/t10k-labels.idx1-ubyte' # 测试集标签文件

def load_mnist(fname_image, fname_label):
    digits = np.arange(10)

    flbl = open(fname_label, 'rb')
    magic_nr, size = struct.unpack(">II", flbl.read(8))
    lbl = pyarray("b", flbl.read())
    flbl.close()

    fimg = open(fname_image, 'rb')
    magic_nr, size, rows, cols = struct.unpack(">IIII", fimg.read(16))
    img = pyarray("B", fimg.read())
    fimg.close()

    ind = [k for k in range(size) if lbl[k] in digits]
    N = len(ind)

    images = zeros((N, rows * cols), dtype=uint8)
    labels = zeros((N, 1), dtype=int8)
    for i in range(N):
        images[i] = array(img[ind[i] * rows * cols: (ind[i] + 1) * rows * cols]).reshape((1, rows *
cols))
        labels[i] = lbl[ind[i]]
    return images, labels


# 归一化进程
def Data_ErZhiHua(data):
    '''放入主函数测试
    X_train_1 = Data_ErZhiHua(X_train[:5])
    for i in range(5):
        print(X_train_1[i])
    '''
    print("数据二值化进程中！")
    row = data.shape[0]    # 60000
    col = data.shape[1]    # 784
```

```
    for i in range(row):
        for j in range(col):
            if data[i][j] > 127:
                data[i][j] = 1
            else:
                data[i][j] = 0
    return data


if __name__ == "__main__":
    # img = cv2.imread("./MNIST_data/0.png", 0)
    # cv2.imshow("0", img)
    # cv2.waitKey()
    # # convert('L')函数可以把图片转换成黑白图片（灰度图）之后，将图片作为2 维数据进行读取。
    # img_0 = np.array(Image.open('./MNIST_data/0.png').convert('L'))
    # # 打印长和宽,28*28
    # print("照片的大小：", img_0.shape)
    # tes = ~cv2.imread("./MNIST_data/0.png", 0)
    test_img = [~cv2.imread('./MNIST_data/0.png', 0),
                ~cv2.imread('./MNIST_data/1.png', 0),
                ~cv2.imread('./MNIST_data/2.png', 0),
                ~cv2.imread('./MNIST_data/3.png', 0),
                ~cv2.imread('./MNIST_data/4.png', 0),
                ~cv2.imread('./MNIST_data/5.png', 0),
                ~cv2.imread('./MNIST_data/6.png', 0),
                ~cv2.imread('./MNIST_data/7.png', 0),
                ~cv2.imread('./MNIST_data/8.png', 0),
                ~cv2.imread('./MNIST_data/9.png', 0)]
    test_img = np.array(test_img).reshape(10, 784)
    test_label = [[0], [1], [2], [3], [4], [5], [6], [7], [8], [9]]
    test_label = np.array(test_label)

    # 训练集读取
    train_img, train_label = load_mnist(train_images_file, train_labels_file)
    # 二进制数据集，可不必包含所有训练数据集，跑起来很慢
    train_img = Data_ErZhiHua(train_img[:60000])
    test_label = test_label[:60000]
    test_img = Data_ErZhiHua(test_img)
    print("数据二进制进程完成！")

    # 预测手写字体准确度
    # 调用 Knn 算法预测,默认 K 数值为 5,train_img,train_label,test_img,test_label
    knn = Knn(5)
```

```
    # 训练，KNN 算法不需要训练，只是一个传递数据集的作用
    knn.fit(train_img, train_label)
    print("选择训练的数据集大小为：", train_img.shape[0])

    # 用 test_img,test_label 数据集预测
    print("预测数据集大小为：", test_label.shape[0])
    preds_1 = knn.predict(test_img[:8], test_label[:8])
    print("判断预测样本的第 8 个", '\n', "预测值为： 8 真确值为： [8] 是否准确： [True]")
    preds = knn.predict(test_img[9:10], test_label[9:10])

    # 输出的结果集为 preds
    # 准确度分析
    print("test_label 的长度：", len(test_label))
    acc = (np.sum(preds_1 == test_label[:8]) + 1 + np.sum(preds == test_label[9:10])) /
len(test_label)
    print("准确度：", acc)
```

## 五、实验心得

对于代码的优化方面：我通过这三周的实训不仅仅学习了基本 KNN 的算法，还学习到了如何取改进自己的代码。将运行时间非常长的问题分成两部分看，一部分是训练集极其的庞大，另一部分是代码中的循环、数据存储、距离计算。有针对的优化，利用学过的算法设计优化代码中计算，利用二值化简化计算量，利用 KDtree 简化训练数据集搜索时间，利用分组快速搜索近邻法、剪辑近邻法、压缩近邻法简化训练集模板量。通过对整体代码的不断优化，我对项目的理解更加深刻。

对于编程能力方面：通过三周的实训，编写并不断优化代码，提高我整体变成能力，对深度学习、机器学习等方面的认识不断加深，受益匪浅。

# 实训课题二：房产价格预测

## 一、实训内容：

实训设备为普通 PC，使用 python 结合相关的机器学习和人工智能库，基本配置为 python3.7 版本。

实验数据：https://pan.baidu.com/s/1eR7doh8 此数据集选择 kaggle 大数据平台，该数据集记录了纽约市房地产市场在 12 个月内售出的每一栋或每一个建筑单元(公寓等)，下载数据可以直接通过浏览器下载压缩包，也可以通过函数来进行。

用以下三种方法对房产数据进行分析，要求建立房价和面积之间的关系，并能对未知房价的房屋进行预测。

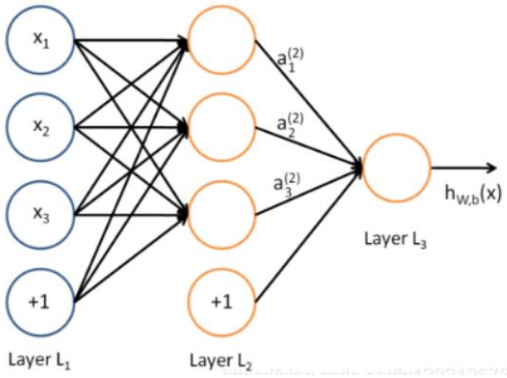
1. 基于多层感知机实现房价预测
2. 使用随机森林方法实现房价预测
3. 线性回归方法实现房价预测

通过编写代码学习多层感知机、随机森林、线性回归的原理与应用，体验统计学习方法中代码在实际程序中应用情况，加以深刻理解。

## 二、原理及步骤：

**原理：**

**多层感知机：**多层感知机(MLP,Multilayer Perceptron)也叫人工神经网络(ANN,Artificial Neural Network)，除了输入输出层，它中间可以有多个隐层，最简单的 MLP 只含一个隐层，即三层的结构，如下图：




多层感知机层与层之间是全连接的。多层感知机最底层是输入层，中间是隐藏层，最后是输出层。

隐藏层的神经元怎么得来？首先它与输入层是全连接的，假设输入层用向量 X 表示，则隐藏层的输出就是 f (W1X+b1)，W1 是权重（也叫连接系数），b1 是偏置，函数 f 可以是常用的 sigmoid 函数或者 tanh 函数

sigmod 函数 $\sigma(x) = \frac{1}{1+exp(-x)}$ 导数为：$e' = e(x)\big(1-\sigma(x)\big)$

3.Tanh 函数 $ln' \neg n_x = \frac{sinh\, x}{cos\, n_x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

导数：$(tanh\, x)' = sexb^2 x = 1 - tan^2\, x$ ，取值范围为[-1,1]

tanh 在特征相差明显时的效果会很好，在循环过程中会不断扩大特征效果。

**随机森林：**

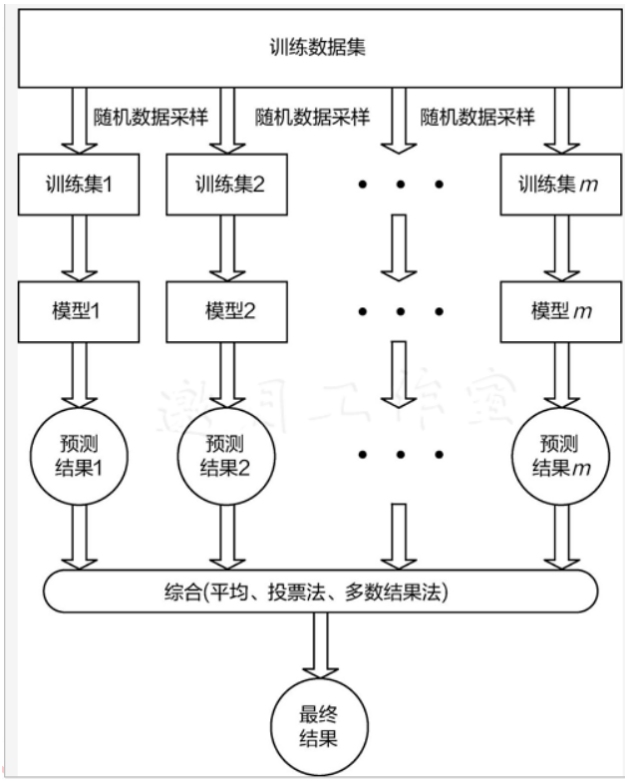
作为新兴起的、高度灵活的一种机器学习算法，随机森林拥有广泛的应用前景，从市场营销到医疗保健保险，既可以用来做市场营销模拟的建模，统计客户来源，保留和流失，也可用来预测疾病的风险和病患者的易感性，Random Forest 在准确率方面还是相当有优势的。

如果接触过决策树（Decision Tree）的话，那么会很容易理解什么是随机森林。随机森林就是通过集成学习的思想将多棵树集成的一种算法，它的基本单元是决策树，而它的本质属于机器学习的一大分支——集成学习（Ensemble Learning）方法。随机森林的名称中有两个关键词，一个是“随机”，一个就是“森林”。“森林”我们很好理解，一棵叫做树，那么成百上千棵就可以叫做森林了，这样的比喻还是很贴切的，其实这也是随机森林的主要思想--集成思想的体现。“随机”的含义我们会在下边部分讲到。

其实从直观角度来解释，每棵决策树都是一个分类器（假设现在针对的是分类问题），那么对于一个输入样本，N 棵树会有 N 个分类结果。而随机森林集成了所有的分类投票结果，将投票次数最多的类别指定为最终的输出，这就是一种最简单的 Bagging 思想。

随机森林是一种很灵活实用的方法，它有如下几个特点：

- 在当前所有算法中，具有极好的准确率
- 能够有效地运行在大数据集上
- 能够处理具有高维特征的输入样本，而且不需要降维
- 能够评估各个特征在分类问题上的重要性
- 在生成过程中，能够获取到内部生成误差的一种无偏估计
- 对于缺省值问题也能够获得很好得结果




**线性回归：**

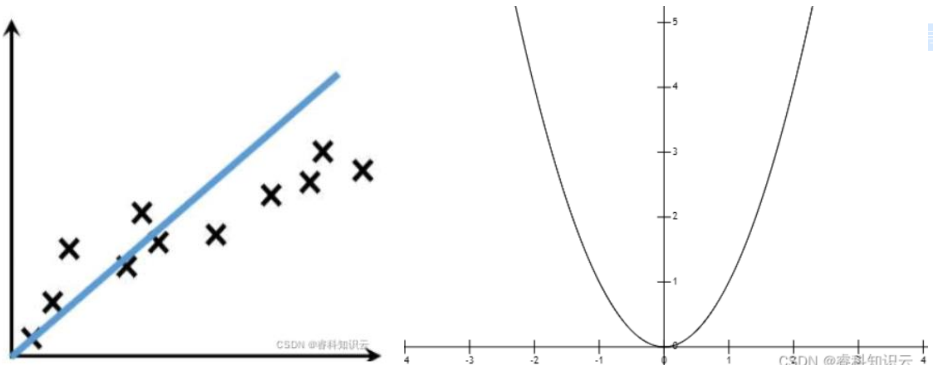
线性回归是回归分析的一种。

1、假设目标值（因变量）与特征值（自变量）之间线性相关（即满足一个多元一次方程，如：f(x)=w1x1+…+wnxn+b.）。

2、然后构建损失函数。

3、最后通过令损失函数最小来确定参数。（最关键的一步）

线性回归主要用来解决回归问题，也就是预测连续值的问题。而能满足这样要求的数学模型被称为“回归模型”。最简单的线性回归模型是我们所熟知的一次函数（即 y=kx+b），这种线性函数描述了两个变量之间的关系，其函数图像是一条连续的直线。



还有另外一种回归模型，也就是非线性模型(nonlinear model)，它指因变量与

自变量之间的关系不能表示为线性对应关系(即不是一条直线)，比如我们所熟知的对数函数、指数函数、二次函数等。

## 步骤：

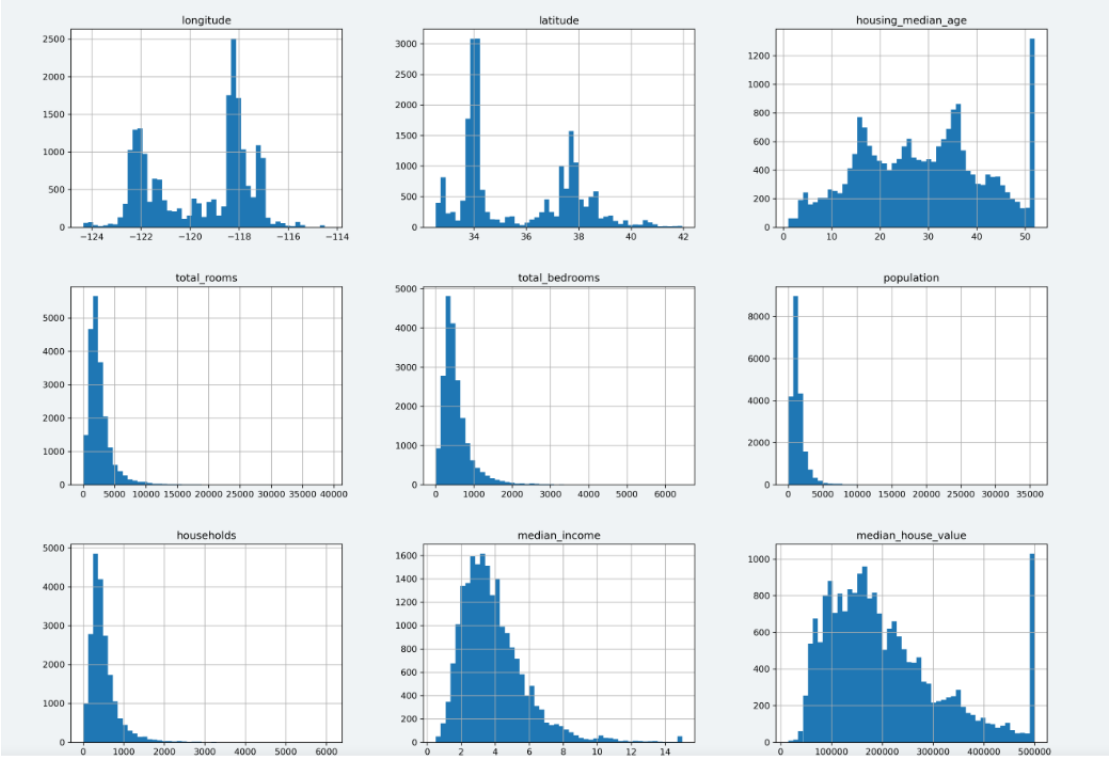
1. 获取数据

下载数据可以直接通过浏览器下载压缩包，也可以通过函数来进行。

DOWNLOAD_ROOT="https://raw.githubusercontent.com/ageron/handson-ml2/master/"

1.1 查看数据结构

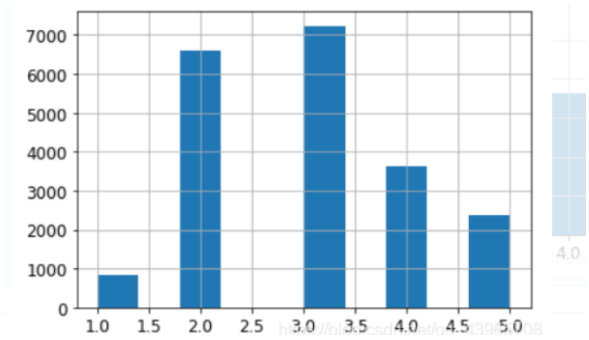
| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | households | median_income | median_house_value | ocean_proximity |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -122.23 | 37.88 | 41.0 | 880.0 | 129.0 | 322.0 | 126.0 | 8.3252 | 452600.0 | NEAR BAY |
| 1 | -122.22 | 37.86 | 21.0 | 7099.0 | 1106.0 | 2401.0 | 1138.0 | 8.3014 | 358500.0 | NEAR BAY |
| 2 | -122.24 | 37.85 | 52.0 | 1467.0 | 190.0 | 496.0 | 177.0 | 7.2574 | 352100.0 | NEAR BAY |
| 3 | -122.25 | 37.85 | 52.0 | 1274.0 | 235.0 | 558.0 | 219.0 | 5.6431 | 341300.0 | NEAR BAY |
| 4 | -122.25 | 37.85 | 52.0 | 1627.0 | 280.0 | 565.0 | 259.0 | 3.8462 | 342200.0 | NEAR BAY |




2. 划分测试集

经调研可知，收入中位数对于预测房价中位数十分重要。所以在收入这一属性上，我们希望测试集能够代表各种不同类型的收入，即分层抽样。首先要根据上面图中的 median_income 直方图中的数值，大多数收入中位数聚集在[1.5, 6]之间，划分层次时，每层要有足够多的实例，因此，使用 pd.cut()方法创建 5 个

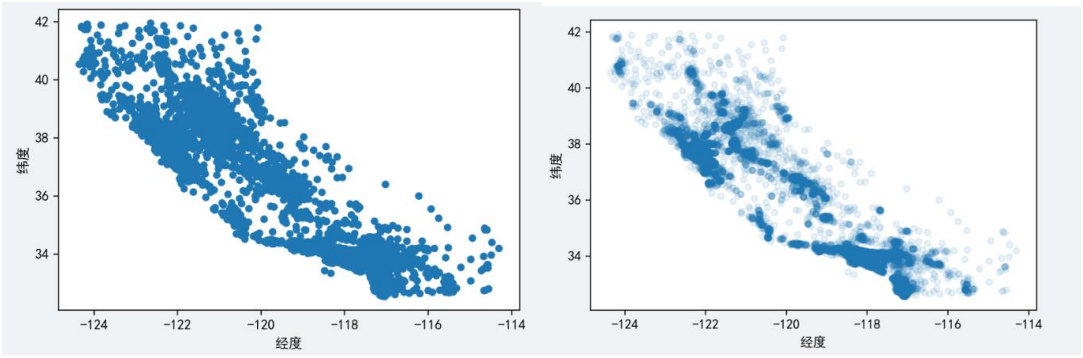
收入类别，0-1.5、1.5-3、3-4.5、4.5-6、6- +∞。




3. 可视化获取更多信息

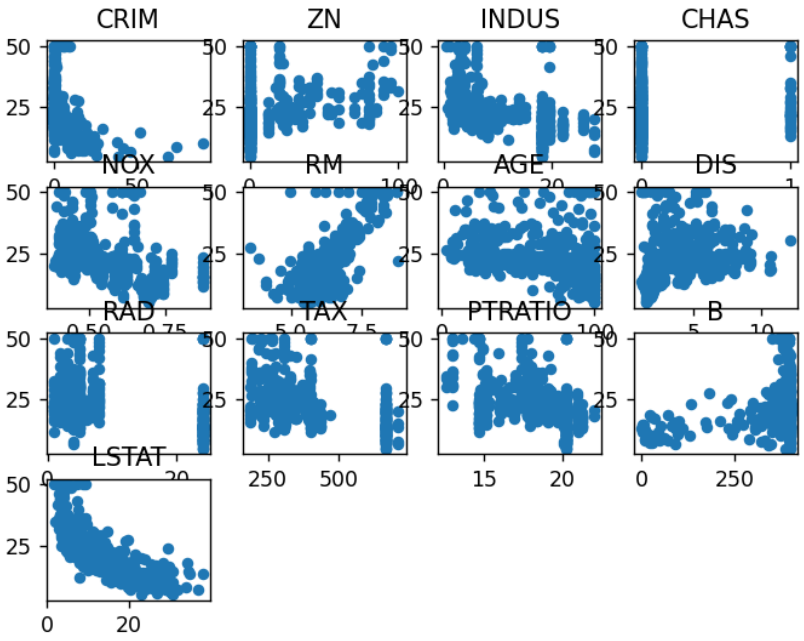
根据数据集中的经纬度，可以绘制一个各区域的分布图。

将 alpha 设置为 1，显示高密度点的位置。




4. 寻找相关性

使用 corr()方法计算每对属性之间的皮尔逊相关系数。相关系数范围[-1, 1]，越接近 1 表示有越强的正相关，越接近-1 表示有越强的负相关。



5. 数据处理

可以使用 Scikit-Learn 中的 OrdinalEncoder 类将这些类别从文本转成数字；使用 Sickit-Learn 中的 Pipeline 类转换成 numpy 数组；

6. 选择和训练模型，三个需要的模型

7. 使用交叉验证来更好地进行评估

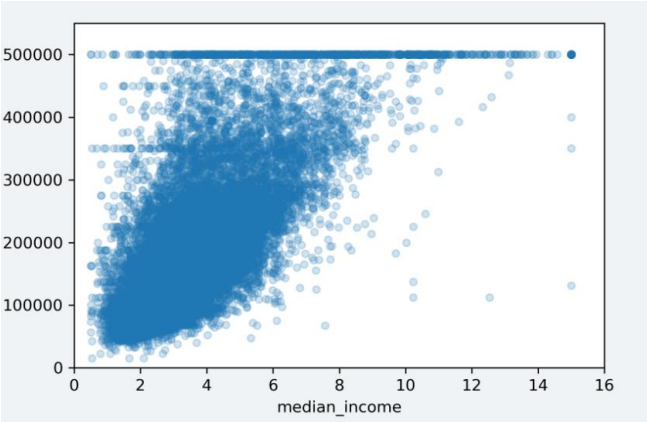
评估模型的一种方法是使用 train_test_split 函数将训练集划分为较小的训练集和测试集。还有一种方法是使用 Scikit-Learn 的 K 折交叉验证的功能。比如：它可以将训练集分割成 10 个不同的子集，每个子集称为一个折叠，然后对模型进行 10 训练和评估——每次挑选一个折叠进行评估，使用其他 9 个折叠进行模型训练，返回一个包含 10 次评估分数的数组。

8.通过测试集评估系统

经过前面的训练，现在有了一个表现足够优秀的系统了，是时候用测试集评估最终模型的时候了，我们只需要从测试集中获取预测器和标签，运行 full_pipeline 来转换数据（调用 transform() 而不是 fit_transform() ），然后在测试集上评估最终模型。

## 三、结果分析：

1、与房价中位数最相关的属性是收入中位数



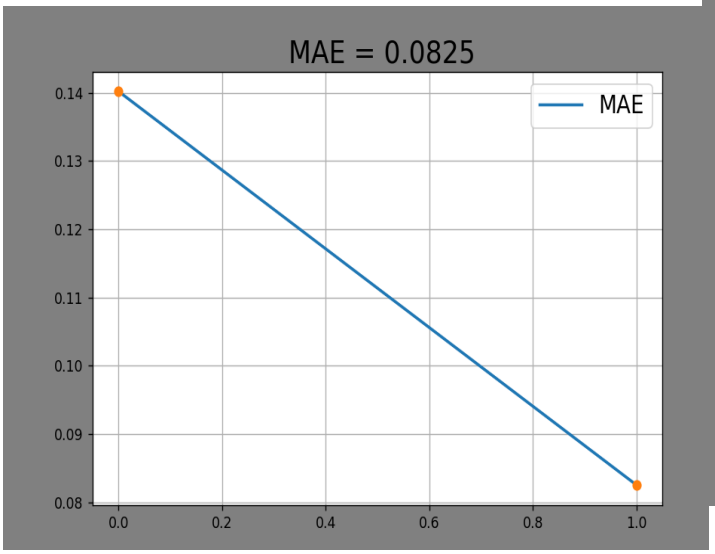

2、预测的结果还不太准确时，可以使用 Scikit-Learn 的 mean_squared_errod() 函数来测量整个训练集上回归模型的均方根误差。$RMSE(X,h)=\sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(n(x^{(i)})-y(i)\right)^2}$

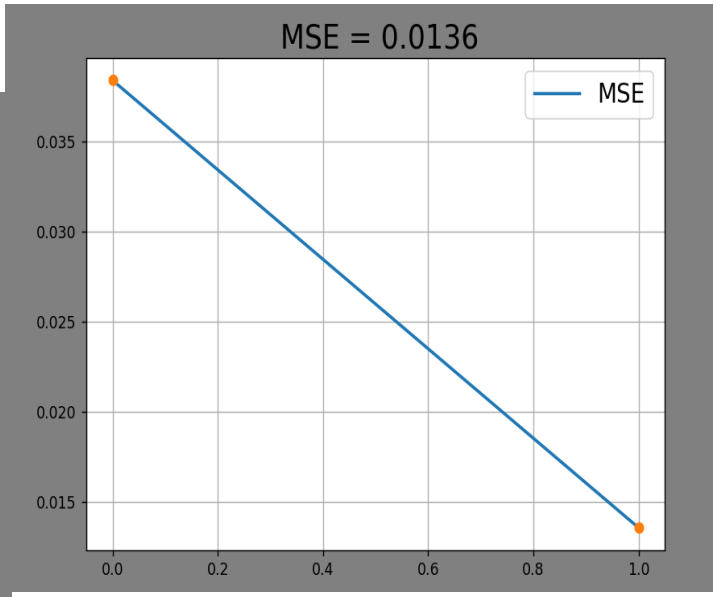
```
Predictions: [210644.60459286 317768.80697211 210956.43331178  59218.98886849
 189747.55849879]
Labels: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

3、RF 模型

```
RF模型训练中。。。。。
训练完成。。。。。
RF-Model模型的RMSE均方根误差是： 0.05830127226491493
RF-Model模式的MSE均方误差是： 0.003399038347707739
平均相对误差是： 0.04413080719702272
```

4、线性回归




```
计算均分方差: [0.03836828639421764, 0.013564156972261251]
计算平均绝对误差 [0.14020173002649303, 0.08252191950087476]
线性回归的系数为:
 w = [[-0.48110028 -0.26328138  0.55878758]]
 b = [0.246585]
```

5、MLP 模型

```
C:\Users\walex\AppData\Local\Programs\P
MLP模型均方根误差是： 0.1989002684939289
MLP模式均方误差是： 0.039561316806957
平均相对误差是： 0.15465228364721664
```

## 四、源代码

1、MLP

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
import warnings
warnings.filterwarnings("ignore")#忽略掉版本“波士顿房价数据”将在 1.2 版本中被弃用的警告
# 加载数据集并进行归一化预处理
def loadDataSet():
    boston_dataset = load_boston()
    X = boston_dataset.data
    y = boston_dataset.target
    y = y.reshape(-1, 1)
    # 将数据划分为训练集和测试集
    # 随机抓取 20%的数据构建测试样本，剩余作为训练样本
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

    # 分别初始化对特征和目标值的标准化器
    ss_X, ss_y = preprocessing.MinMaxScaler(), preprocessing.MinMaxScaler()

    # 分别对训练和测试数据的特征以及目标值进行标准化处理
    X_train, y_train = ss_X.fit_transform(X_train), ss_y.fit_transform(y_train)
    X_test, y_test = ss_X.transform(X_test), ss_y.transform(y_test)
    y_train, y_test = y_train.reshape(-1, ), y_test.reshape(-1, )

    return X_train, X_test, y_train, y_test

def trainMLP(X_train, y_train):
    model_mlp = MLPRegressor(
        hidden_layer_sizes=(20, 1), activation='logistic', solver='adam', alpha=0.0001,
batch_size='auto',
        learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=5000,
shuffle=True,
        random_state=1, tol=0.0001, verbose=False, warm_start=False, momentum=0.9,
nesterovs_momentum=True,
        early_stopping=False, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
    model_mlp.fit(X_train, y_train)
    return model_mlp

def test(model, X_test, y_test):
    y_pre = model.predict(X_test)
    print("MLP 模型均方根误差是： {}".format(mean_squared_error(y_test, y_pre) ** 0.5))
    print("MLP 模式均方误差是： {}".format(mean_squared_error(y_test, y_pre)))
    print("平均相对误差是： {}".format(mean_absolute_error(y_test, y_pre)))

if __name__ == '__main__':
    X_train, X_test, y_train, y_test = loadDataSet()
    # 训练 MLP 模型
```

```
    model = trainMLP(X_train, y_train)
    test(model, X_test, y_test)
```

2、RF 模型

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error

import warnings
warnings.filterwarnings("ignore")#忽略掉版本“波士顿房价数据”将在 1.2 版本中被弃用的警告
# 加载数据集并进行归一化预处理
def loadDataSet():
    boston_dataset = load_boston()
    X = boston_dataset.data
    y = boston_dataset.target
    y = y.reshape(-1, 1)
    # 将数据划分为训练集和测试集,八成数据集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

    # 分别初始化对特征和目标值的标准化器
    ss_X, ss_y = preprocessing.MinMaxScaler(), preprocessing.MinMaxScaler()

    # 分别对训练和测试数据的特征以及目标值进行标准化处理
    X_train, y_train = ss_X.fit_transform(X_train), ss_y.fit_transform(y_train)
    X_test, y_test = ss_X.transform(X_test), ss_y.transform(y_test)
    y_train, y_test = y_train.reshape(-1, ), y_test.reshape(-1, )
    return X_train, X_test, y_train, y_test
```

```
def trainRF(X_train, y_train):
    model_rf = RandomForestRegressor(n_estimators=10000)#邻居数为10000
    model_rf.fit(X_train, y_train)
    return model_rf
def test(model, X_test, y_test):
    y_pre = model.predict(X_test)
    print("RF-Model 模型的 RMSE 均方根误差是： {}".format(mean_squared_error(y_test,
y_pre)**0.5))
    print("RF-Model模式的MSE均方误差是：{}".format(mean_squared_error(y_test, y_pre)))
    print("平均相对误差是： {}".format(mean_absolute_error(y_test, y_pre)))
if __name__ == '__main__':
    # 加载数据集并进行归一化预处理
    print("归一化处理中。。。。。")
    X_train, X_test, y_train, y_test = loadDataSet()
    print("归一化完成")
    # 训练RF 模型
    print("RF 模型训练中。。。。。")
    model = trainRF(X_train, y_train)
    print("训练完成。。。。。")
    #测试RF 模型
    test(model, X_test, y_test)
```

3、线性回归

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
import warnings
warnings.filterwarnings("ignore")#忽略掉版本“波士顿房价数据”将在1.2版本中被弃用的警告
# 加载波士顿房价的数据集
# boston = pd.read_csv('boston.csv')
boston = datasets.load_boston()
print(boston)
```

```
boston_df = pd.DataFrame(boston.data, columns=boston.feature_names)
boston_df['MEDV'] = boston.target

# 查看数据是否存在空值，从结果来看数据不存在空值。
boston_df.isnull().sum()
# 查看数据大小
boston_df.shape
# 显示数据前 5 行
boston_df.head()

# 查看数据的描述信息，在描述信息里可以看到每个特征的均值，最大值，最小值等信息。
boston_df.describe()
# # 清洗'PRICE' = 50.0 的数据
# boston_df = boston_df.loc[boston_df['PRICE'] != 50.0]

# 计算每一个特征和房价的相关系数
boston_df.corr()['MEDV']
# 各个特征和价格都有明显的线性关系。
plt.figure(facecolor='gray')
corr = boston_df.corr()
corr = corr['MEDV']
corr[abs(corr) > 0.5].sort_values().plot.bar()

# LSTAT 和房价的散点图
plt.figure(facecolor='gray')
plt.scatter(boston_df['LSTAT'], boston_df['MEDV'], s=30, edgecolor='white')
plt.title('LSTAT')
plt.xlabel('LSTAT')
plt.ylabel('MEDV')

# CRIM 和房价的散点图
plt.figure(facecolor='gray')
plt.scatter(boston_df['CRIM'], boston_df['MEDV'], s=30, edgecolor='white')
plt.title('CRIM')
plt.xlabel('CRIM')
plt.ylabel('MEDV')

# RM 和房价的散点图
plt.figure(facecolor='gray')
plt.scatter(boston_df['RM'], boston_df['MEDV'], s=30, edgecolor='white')
plt.title('RM')
plt.xlabel('RM')
plt.ylabel('MEDV')
```

```
plt.show()

boston_df = boston_df[['LSTAT', 'CRIM', 'RM', 'MEDV']]
# 目标值
y = np.array(boston_df['MEDV'])
boston_df = boston_df.drop(['MEDV'], axis=1)
# 特征值
X = np.array(boston_df)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
from sklearn import preprocessing
# 初始化标准化器
min_max_scaler = preprocessing.MinMaxScaler()
# 分别对训练和测试数据的特征以及目标值进行标准化处理
X_train = min_max_scaler.fit_transform(X_train)
y_train = min_max_scaler.fit_transform(y_train.reshape(-1, 1))  # reshape(-1,1)指将它转化为 1
列，行自动确定
X_test = min_max_scaler.fit_transform(X_test)
y_test = min_max_scaler.fit_transform(y_test.reshape(-1, 1))
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
# 使用训练数据进行参数估计
lr.fit(X_train, y_train)
# 使用测试数据进行回归预测
y_test_pred = lr.predict(X_test)
print('y_test_pred : ', y_test_pred)
# 使用 r2_score 对模型评估
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
# 绘图函数
def figure(title, *datalist):
    plt.figure(facecolor='gray', figsize=[16, 8])
    for v in datalist:
        plt.plot(v[0], '-', label=v[1], linewidth=2)
        plt.plot(v[0], 'o')
    plt.grid()
    plt.title(title, fontsize=20)
    plt.legend(fontsize=16)
    plt.show()
# 训练数据的预测值
y_train_pred = lr.predict(X_train)
print('y_train_pred : ', y_train_pred)
# 计算均分方差
train_MSE = [mean_squared_error(y_train, [np.mean(y_train)] * len(y_train)),
```

```
                mean_squared_error(y_train, y_train_pred)]
print("计算均分方差:", train_MSE)
# 计算平均绝对误差
train_MAE = [mean_absolute_error(y_train, [np.mean(y_train)] * len(y_train)),
                mean_absolute_error(y_train, y_train_pred)]
print("计算平均绝对误差", train_MAE)
# 绘制误差图
figure(' MSE = %.4f' % (train_MSE[-1]), [train_MSE, 'MSE'])
figure(' MAE = %.4f' % (train_MAE[-1]), [train_MAE, 'MAE'])
# 绘制预测值与真实值图
# figure('预测值与真实值图 模型的' + r'$R^2=%.4f$' % (r2_score(y_train_pred, y_train)),
[y_test_pred, 'pred_value'],[y_test, 'true_value'])
# 线性回归的系数
print('线性回归的系数为:\n w = %s \n b = %s' % (lr.coef_, lr.intercept_))
```

## 五、实验心得

我感觉总体来讲训练效果还是可以的，因为毕竟只有两个参数而且，RM 和 LASTA 的影响情况最大，基本上也是预测出了大概的走势。波士顿房价预测任务数据集而言，样本数比较少。但在实际问题中，数据集往往非常大，如果每次都使用全量数据进行计算，效率非常低，通俗的说就是“杀鸡焉用牛刀”。由于参数每次只沿着梯度反方向更新一点点，因此方向并不需要那么精确。一个合理的解决方案是每次从总的数据集中随机抽取出小部分数据来代表整体，基于这部分数据计算梯度和损失来更新参数。

其中我遇见的错误可以分为两类，一个是对于公式的理解还有就是一些对于一些库的熟练程度不够高，但是通过这次又巩固了一下代码能力。

# 实训课题三：乳腺癌的识别

## 一、实训内容：

实训设备为普通 PC，使用 python 结合相关的机器学习和人工智能库，基本配置为 python3.5.2 及以上版本。实验数据集使用 scikit-learn 内部包导入。

掌握逻辑回归算法的原理，能根据历史女性乳腺癌患者数据集（医学指标）构建逻辑回归分类模型进行良 / 恶性乳腺癌肿瘤预测。

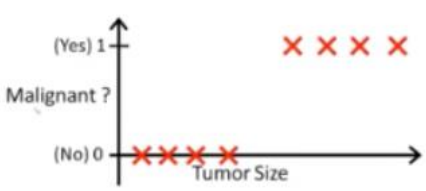
## 二、原理及步骤：

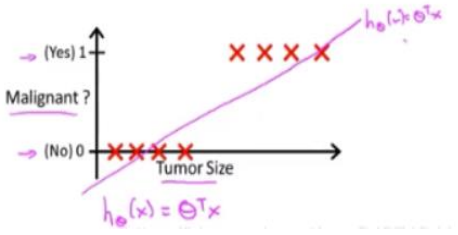
**原理：**

Logistic 回归模型是目前广泛使用的学习算法之一，通常用来解决二分类问题，虽然名字中有“回归”，但它是一个分类算法。有些文献中译为“逻辑回归”，但中文“逻辑”与 logistic 和 logit 的含义相去甚远，因此下文中直接使用 logistic 表示。Logistic 回归的优点是计算代价不高，容易理解和实现；缺点是容易欠拟合，分类精度可能不高。

逻辑回归的预测函数是一个复合函数，是线性函数外面套一个逻辑函数（sigmoid 函数）。所以模型的预测值总是介于(0, 1)区间内，可以理解成判别为 1 的概率。若预测值大于 0.5，则判别为 1；若预测值小于 0.5，则判别为 0。逻辑回归模型是一个二分类模型。

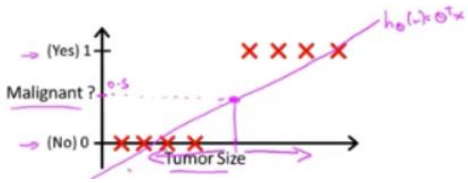
以肿瘤分类为例，我们要预测肿瘤是否为恶性肿瘤，我们用 0 和 1 表示这两个取值，用 0 表示不是恶性肿瘤，用 1 表示是恶性肿瘤（当然也可以反过来，没有影响），那么数据集如图所示。




假如用线性回归模型来拟合的话，看起来可能会是这样的。

我们可以设置阈值为 0.5，如果输出小于等于 0.5，则预测 y 为 0；输出大于等于 0.5 则预测 y 为 1。




这样红点左边的都被预测为 0，红点右边的都被预测为 1，我们得到了想要的结果。在这个例子中，使用线性回归似乎很合理，即使这是一个分类问题而不是一个回归问题。

**步骤：**

1.数据集来源

数据集源于威斯康星州临床科学中心。每个记录代表一个乳腺癌的随访数据样本。这些是 DR Wolberg 自 1984~1995 随访搜集连续乳腺癌患者数据，数据仅包括那些具有侵入性的病例乳腺癌并没有远处转移的医学指标数据集。

2.特征值（医学特征）

乳腺癌数据集是一个共有 569 个样本、30 个输入变量和 2 个分类的数据集。

数据集样本实例数：569 个。特征（属性）个数：30 个特征属性和 2 个分类目标（恶性，良性）。

特征（属性）信息：30 个数值型测量结果由数字化细胞核的 10 个不同特征的均值、标准差和最差值构成。

3.算法流程

激活函数：映射到概率的函数 $g(z)=\frac{1}{1+e^{-z}}$

预测函数：返回预测结果值 $(\theta 0\ \theta 1\ \theta 2)*\begin{pmatrix}1\\x1\\x2\end{pmatrix}=\theta 0+\theta 1*x1+\theta 2*x2$

损失函数：平均损失 j（$\theta$）=$\frac{1}{n}\sum_{i=1}^{n}D(h0(xi),yi)$

计算梯度：$\frac{\partial j}{\partial \theta_j}=-\frac{1}{m}\sum_{i=1}^{n}\left(yi-h_\theta(xi)\right)x_{ij}$

梯度下降-模型迭代学习：寻找山谷的最低点，使得目标函数达到极值点

（1）找到当前最合适的方向

（2）走那么一小步，走快了该“跌倒”了

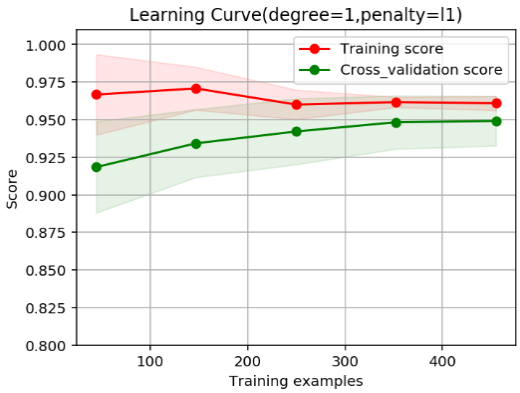
（3）按照方向与步伐去更新我们的参数

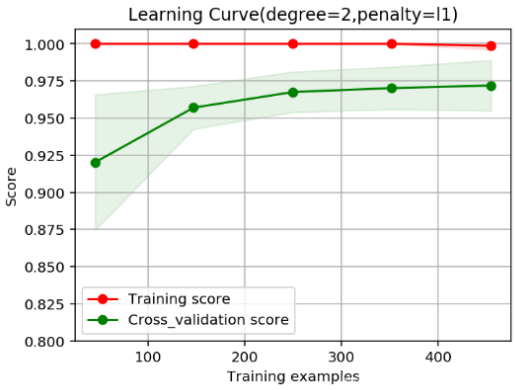
4.核心算法编写

5.模型评估和分析报告

## 三、结果分析：

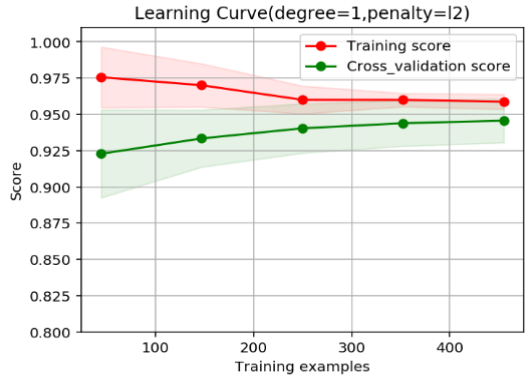
1、二项多项式特征的模型效果很好，训练集的预测率达到了 100%，而且 test 集也没有出现过拟合的现象。

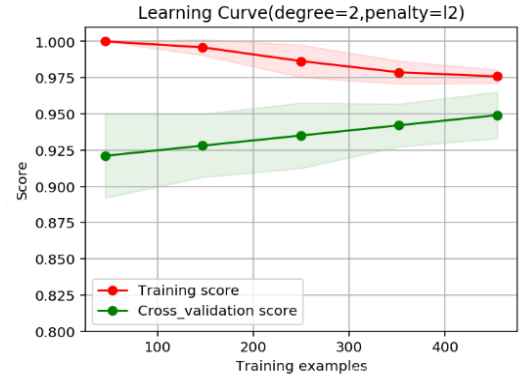



2、L2 范式的学习曲线

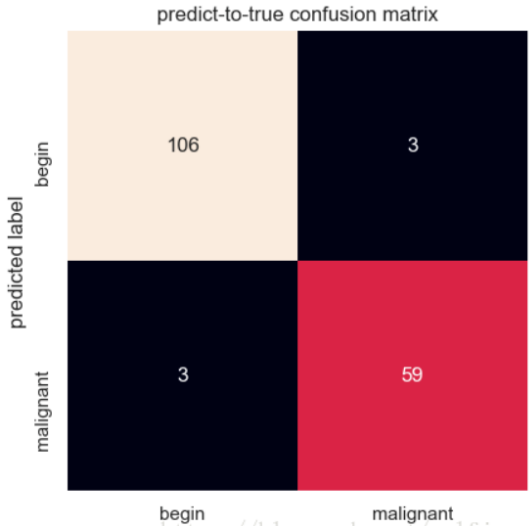
elaspe:15.646595




3、模型评估

Iteration:0 Loss = 4.1750105532628305,
Acc = 0.6373626373626373
Iteration:200 Loss = 1.2904508983157834,
Acc = 0.8879120879120879
Iteration:400 Loss = 1.2651477434413776,
Acc = 0.8901098901098901
Iteration:600 Loss = 1.1940492881913498,
Acc = 0.8945054945054945
Iteration:800 Loss = 1.290422530618182,
Acc = 0.8879120879120879



```
准确率: 0.8947368421052632
召回率: 0.8656716417910447
精确率: 0.9508196721311475
              precision    recall  f1-score   support

           0       0.83      0.94      0.88        47
           1       0.95      0.87      0.91        67

    accuracy                           0.89       114
   macro avg       0.89      0.90      0.89       114
weighted avg       0.90      0.89      0.90       114
```

## 四、源代码

```
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings("ignore")

def loadTrainData():
    cancer = load_breast_cancer()  # 加载乳腺癌数据
    X = cancer.data  # 加载乳腺癌判别特征
    y = cancer.target  # 两个 TAG，y = 0 时为阴性，y = 1 时为阳性
    # 将数据集划分为训练集和测试集，测试集占比为 0.2
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    X_train = X_train.T
    X_test = X_test.T
    return X_train, X_test, y_train, y_test


def sigmoid(inx):
    from numpy import exp
    return 1.0/(1.0 + exp(-inx))
# 初始化参数
def initialize_para(dim):
    mu = 0
    sigma = 0.1
    np.random.seed(0)
    w = np.random.normal(mu, sigma, dim)
    w = np.reshape(w, (dim, 1))
    b = 0
```

```
    return w, b

# 前向传播
def propagate(w, b, X, Y):
    # eps 防止 log 运算遇到 0
    eps = 1e-5
    m = X.shape[1]
    # 计算初步运算结果
    A = sigmoid(np.dot(w.T, X) + b)
    # 计算损失函数值大小
    cost = -1 / m * np.sum(np.multiply(Y, np.log(A + eps)) + np.multiply(1 - Y, np.log(1 - A + eps)))
    # 计算梯度值
    dw = 1 / m * np.dot(X, (A - Y).T)
    db = 1 / m * np.sum(A-Y)
    cost = np.squeeze(cost)

    grads = {"dw": dw,
             "db": db}
    # 返回损失函数大小以及反向传播的梯度值
    return grads, cost, A

# num_iterations 梯度下降次数
# learning_rate 学习率
def optimize(w, b, X, Y, num_iterations, learning_rate):
    costs = []  # 记录损失函数值

    # 循环进行梯度下降
    for i in range(num_iterations):
        # print(i)
        grads, cost, pre_Y = propagate(w, b, X, Y)
        dw = grads["dw"]
        db = grads["db"]

        w = w - learning_rate * dw
        b = b - learning_rate * db

        # 每 100 次循环记录一次损失函数大小并打印
        if i % 100 == 0:
            costs.append(cost)

        if i % 100 == 0:
            pre_Y[pre_Y >= 0.5] = 1
            pre_Y[pre_Y < 0.5] = 0
            pre_Y = pre_Y.astype(np.int)
```

```
            acc = 1 - np.sum(pre_Y ^ Y) / len(Y)
            print("Iteration:{} Loss = {}, Acc = {}".format(i, cost, acc))
    # 最终参数值
    params = {"w": w,
              "b": b}
    return params, costs

def predict(w, b, X):
    # 样本个数
    m = X.shape[1]
    # 初始化预测输出
    Y_prediction = np.zeros((1, m))
    # 转置参数向量 w
    w = w.reshape(X.shape[0], 1)
    # 预测结果
    Y_hat = sigmoid(np.dot(w.T, X) + b)

    # 将结果按照 0.5 的阈值转化为 0/1
    for i in range(Y_hat.shape[1]):
        if Y_hat[:, i] > 0.5:
            Y_prediction[:, i] = 1
        else:
            Y_prediction[:, i] = 0
    return Y_prediction

# 训练以及预测
def Logisticmodel(X_train, Y_train, X_test, Y_test, num_iterations=1000, learning_rate=0.1):
    # 初始化参数 w，b
    w, b = initialize_para(X_train.shape[0])
    # 梯度下降找到最优参数
    parameters, costs = optimize(w, b, X_train, Y_train, num_iterations, learning_rate)

    w = parameters["w"]
    b = parameters["b"]
    # 训练集测试集的预测结果
    Y_prediction_train = predict(w, b, X_train)
    Y_prediction_test = predict(w, b, X_test)
    Y_prediction_test = Y_prediction_test.T

    # 模型评价
    accuracy_score_value = accuracy_score(Y_test, Y_prediction_test)
    recall_score_value = recall_score(Y_test, Y_prediction_test)
    precision_score_value = precision_score(Y_test, Y_prediction_test)
    classification_report_value = classification_report(Y_test, Y_prediction_test)
```

```
    print("准确率:", accuracy_score_value)
    print("召回率:", recall_score_value)
    print("精确率:", precision_score_value)
    print(classification_report_value)

    d = {"costs": costs,
         "Y_prediction_test": Y_prediction_test,
         "Y_prediction_train": Y_prediction_train,
         "w": w,
         "b": b,
         "learning_rate": learning_rate,
         "num_iterations": num_iterations}
    return d

if __name__ == '__main__':
    X_train, X_test, y_train, y_test = loadTrainData()
    Logisticmodel(X_train, y_train, X_test, y_test)
```

## 五、实验心得

逻辑回归处理样本分布不均匀-异常场景：

样本分布不均衡将导致样本量少的分类所包含的特征过少，并很难从中提取规律；即使得到分类模型，也容易产生过度依赖于有限的数据样本而导致过拟合的问题，当模型应用到新的数据上时，模型的准确性和鲁棒性将很差。

样本分布不平衡主要在于不同类别间的样本比例差异，如果不同分类间的样本量差异达到超过 10 倍就需要引起警觉并考虑处理该问题，超过 20 倍就要一定要解决该问题。

样本分布均匀的应用场景：

异常检测场景、客户流失场景、罕见事件的分析、发生频率低的事件。

处理策略：过抽样策略，如 SMOTE 算法；欠抽样策略。

实训锻炼了编写优化代码能力的同时，更让我们紧跟世界主流学术研究，开阔了视野，更锻炼了我们思考食物的能力。

# 实训课题四：鸢尾花分类

## 一、实训内容：

鸢尾花数据集是机器学习领域非常经典的一个分类任务数据集，在统计学习和机器学习领域都经常被用作示例。数据集内包含 3 类共 150 条记录，每类各 50 个数据，每条记录都有 4 项特征：花萼长度、花萼宽度、花瓣长度、花瓣宽度，可以通过这 4 个特征预测鸢尾花卉属于山鸢尾、变色鸢尾、维吉尼亚鸢尾（iris-setosa, iris-versicolour, iris-virginica）中的哪一品种。

从实践的角度出发，机器学学习要做的工作就是在我们有的一个数据集上建立一个或者多个模型，然后对我们的模型进行优化和评估，对比他们的好坏。

## 二、原理及步骤：

### 2.1 实验原理：

2.1.1 基于逻辑斯蒂回归实现鸢尾花分类

逻辑斯蒂回归与线性回归的联系首先想到的是线性回归+阈值，可以转化为一个分类任务，以阈值划分区间，落到不同范围则分成不同的类，例如使用“单位阶跃函数”，逻辑斯蒂回归中，输出 y=1 的对数几率是输入 x 的线性函数，或者说上式实际上是在用线性回归模型的预测结果去逼近真实 label 的对数几率，这里也是最能体现“回归”的地方，残留了线性回归的影子。而平方损失在逻辑斯蒂回归分类问题中是非凸的，逻辑回归不是连续的，所以这里不再使用平方损失，而是使用极大似然来推导损失函数。似然函数的目标是：令每个样本属于其真实 label 的概率越大越好，即“极大似然法”。

对于逻辑回归，最为突出的两点就是其模型简单和模型的可解释性强。逻辑回归模型的优劣势：优点：实现简单，易于理解和实现；计算代价不高，速度很快，存储资源低；缺点：容易欠拟合，分类精度可能不高。

逻辑回归模型现在同样是很多分类算法的基础组件，比如分类任务中基于 GBDT 算法+LR 逻辑回归实现的信用卡交易反欺诈，CTR(点击通过率)预估等，其好处在于输出值自然地落在 0 到 1 之间，并且有概率意义。模型清晰，有对应的概率学理论基础。它拟合出来的参数就代表了每一个特征(feature)对结果的影响。也是一个理解数据的好工具。但同时由于其本质上是一个线性的分类器，所

以不能应对较为复杂的数据情况。很多时候我们也会拿逻辑回归模型去做一些任务尝试的基线。

2.1.2 使用决策树方法实现鸢尾花分类

决策树简单来说就是带有判决规则（if-then）的一种树，可以依据树中的判决规则来预测未知样本的类别和值。决策树是一种树型结构，其中每个内部节结点表示在一个属性上的测试，每一个分支代表一个测试输出，每个叶结点代表一种类别。具体概念是从顶往下，依次对样本的（一个或多个）属性进行判断，直到决策树的叶节点并导出最终结果。决策树的基本思想在于一般而言，我们遵循这样一个思想来选择属性。每一次划分，同类样本尽量走相同分支，不同类样本尽量走不同分支。即：随着划分过程不断进行，我们希望决策树的分支结点所包含的样本尽可能属于同一类别，即结点的“纯度”（purity）越来越高。

策树的学习本质上是从训练集中归纳出一组分类规则，得到与数据集矛盾较小的决策树，同时具有很好的泛化能力。决策树学习的损失函数通常是正则化的极大似然函数，通常采用启发式方法，近似求解这一最优化问题。决策树生成算法是递归地生成决策树，知道不能终止。这样产生的决策树往往分类精细，对训练数据集分类准确，但是对未知数据集却没有那么准确，有比较严重的过拟合问题。因此，为了简化模型的复杂度，使模型的泛化能力更强，需要对已生成的决策树进行剪枝。

2.1.3 SVM 进行鸢尾花分类

支持向量机是一个二分类的分类模型(或者叫做分类器)。它分类的思想是，给定给一个包含正例和反例的样本集合，svm 的目的是寻找一个超平面来对样本根据正例和反例进行分割，求解能够正确划分训练数据集并且几何间隔最大的分离超平面。SVM 的核心思想是尽最大努力使分开的两个类别有最大间隔，这样才使得分隔具有更高的可信度。而且对于未知的新样本才有很好的分类预测能力。

**2.2 实验步骤**

2.2.1 收集数据：公开数据源

数据的读取有两个方法，鸢尾花的数据集网络上有很多资源，并且机器学习库 sk-learn 中也内置了此数据集。用 pandas 读取数据 pd.read_csv('iris.data') 展示数据集截图，由于数据量较多，此处仅展示前五行

```
iris_data.head(5):
    sepal_length  sepal_width  petal_length  petal_width   class
0            5.1          3.5           1.4          0.2  setosa
1            4.9          3.0           1.4          0.2  setosa
2            4.7          3.2           1.3          0.2  setosa
3            4.6          3.1           1.5          0.2  setosa
4            5.0          3.6           1.4          0.2  setosa
```

load_iris()可以直接导入 sk-learn 中的数据，print(iris.data)能够打印输出数据集矩阵，每一列代表了萼片或花瓣的长宽，一共 4 列，每一行代表某个被测量的鸢尾植物,一共采样了 150 条记录

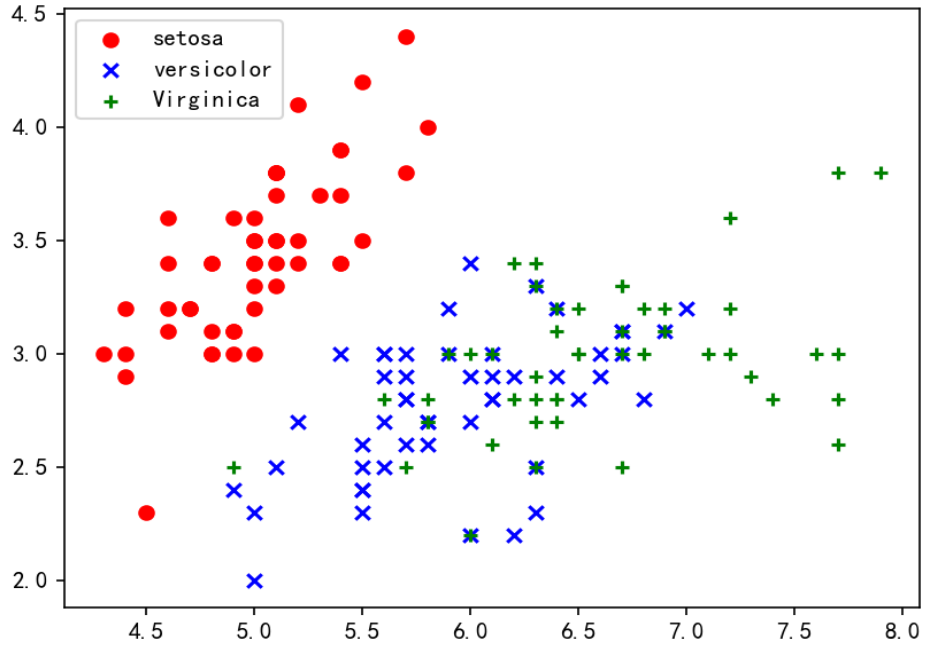
```
iris.data:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
 [6.5 3.  5.5 1.8]
 [7.7 3.8 6.7 2.2]
 [7.7 2.6 6.9 2.3]
 [6.  2.2 5.  1.5]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
```

target 是一个数组，存储了 data 中每条记录属于哪一类鸢尾植物，所以数组的长度是 150，数组元素的值因为共有 3 类鸢尾植物，所以不同值只有 3 个。种类为山鸢尾、杂色鸢尾、维吉尼亚鸢尾。

通过 print(iris.target)能够输出真实标签，iris.data.shape 表示共含有 150 个样本，每个样本 4 个特征，target 是一个数组，存储了 data 中每条记录属于哪一类鸢尾植物，用 0、1 和 2 三个整数分别代表了花的三个品种

```
iris.target:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

从输出结果可以看到，类标共分为三类，前面 50 个类标为 0，中间 50 个类标位 1，后面为 2。然后获取其中两列数据或两个特征，核心代码为：X = [x[0] for x in DD]，获取的值赋值给 X 变量，最后调用 scatter()函数绘制散点图。




2.2.2 分析数据，构思如何处理数据

1 先查看数据集各特征列的摘要统计信息：

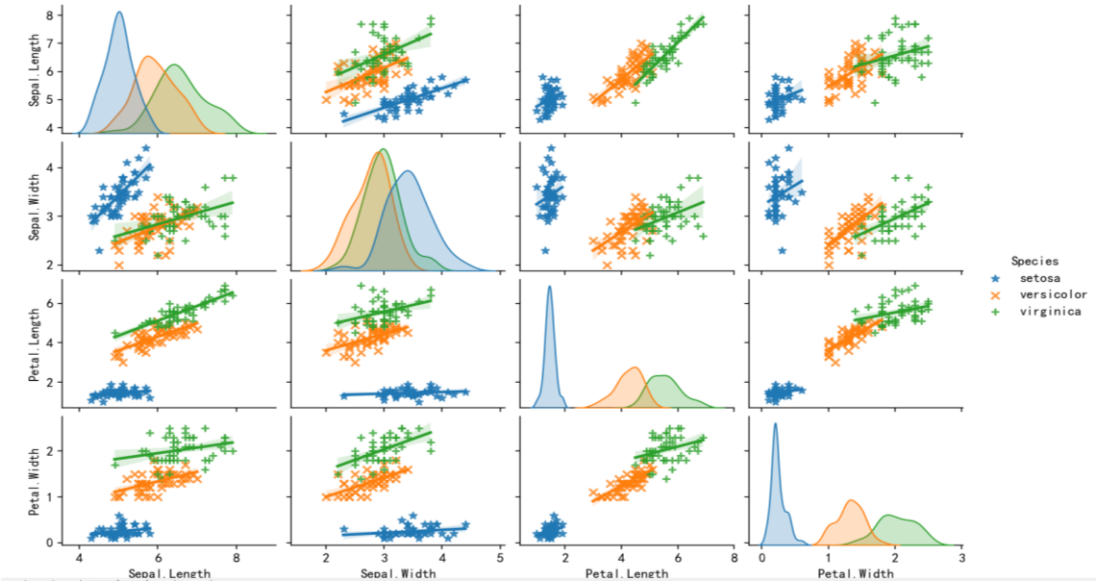
一般来说，拿到数据之后都要看下数据的分布规则，有没有缺失值（当然这个数据是很干净的，但是流程还是要走一下的），用到的方法是 Iris.describe()，显示结果如图，150 行， 4 个 64 位浮点数，数据中无缺失值。

```
iris.data.shape: (150, 4)
describe:
        sepal_length  sepal_width  petal_length  petal_width
count     150.000000   150.000000    150.000000   150.000000
mean        5.843333     3.057333      3.758000     1.199333
std         0.828066     0.435866      1.765298     0.762238
min         4.300000     2.000000      1.000000     0.100000
25%         5.100000     2.800000      1.600000     0.300000
50%         5.800000     3.000000      4.350000     1.300000
75%         6.400000     3.300000      5.100000     1.800000
max         7.900000     4.400000      6.900000     2.500000
```

2 通过图片直观表示数据之间的关系

检查数据也是发现异常值和特殊值的好方法，在现实世界中，经常会遇到不一致的数据和意料之外的测量数据。检查数据的最佳方法之一就是将其可视化。一种可视化方法是绘制散点图(scatter plot)，用这种方法难以对多于 3 个特征的数据集作图。另一种方法是绘制散点图矩阵(pair plot)，从而可以两两查

看所有的特征。散点图矩阵无法同时显示所有特征之间的关系，所以这种可视化方法可能无法展示数据的某些有趣内容。




对角线上的 4 张都是某个变量和自己本身的关系，由于自己和自己的相关系数永远是 1，画出相关系数图意义不大。非对角线的 12 张就是某个变量和另一个变量的关系。kde 密度图可以看出当前数据分布在哪一个区间内，同时在这个区间的分布密度，分布密度最大的在那个位置。KDE 是在概率论中用来估计未知的密度函数，属于非参数检验方法之一。

2.2.3 导入鸢尾花训练数据，并进行相应的处理

我们不能将用于构建模型的数据用于评估模型。因为我们的模型会一直记住整个训练集，所以对于训练集中的任何数据点总会预测正确的标签。scikit-learn 中的 train_test_split 函数可以打乱数据集并进行拆分。将拿到的训练数据，分为训练和验证集。在本次鸢尾花分类中，将 70%的数据用来训练，30%数据用来测试。一般我们在最后给出结论时，为了让被评估的模型更加准确可信，要考虑到一次结果的不稳定性，所以我们在这里选择的交叉验证，并取平均值作为最终的划分结果。为了解决这个问题，使用独热编码，即 OneHot 编码，label_binarize 是固定类别数量的标签转换（受到 classes 顺序的影响，以一对全的方式将标签二值化。fit()可以直接设置 validation_data 为 test 数据集来测试模型的性能，但是通常我们要输出模型的预测值，用来绘制图形等等操作。predict(x_input)对数据进行批量循环，x_input 是输入数据，将输入数据放到已经训练好的模型中，可以得到预测出的输出值。predict_proba 不同于 predict，

它返回的预测值为，获得所有结果的概率。

2.2.4 建立模型

1 线性回归

回归算法作为统计学中最重要的工具之一，它通过建立一个回归方程用来预测目标值，并求解这个回归方程的回归系数。结合 Sklearn 机器学习库的 LogisticRegression 算法分析鸢尾花分类情况 LogisticRegression：逻辑回归（logistic regression）是统计学习中的经典分类方法，属于对数线性模型，所以也被称为对数几率回归。这里要注意，虽然带有回归的字眼，但是该模型是一种分类算法，逻辑斯谛回归是一种线性分类器，针对的是线性可分问题。利用 logistic 回归进行分类的主要思想是：根据现有的数据对分类边界线建立回归公式，以此进行分类。采用线性回归算法对鸢尾花的特征数据进行分析，预测花瓣长度、花瓣宽度、花萼长度、花萼宽度四个特征之间的线性关系。

2、决策树分析

Sklearn 机器学习包中，决策树实现类是 DecisionTreeClassifier，能够执行数据集的多类分类。由于而我们此次的鸢尾花数据集的纯度很高，在构建决策时可以考虑采用默认参数。

3.Svm 支持向量机方法

支持向量机方法也是一种强大的机器学习分类方法。在感知器算法中，我们的目标是最小化分类误差，而在 SVM 中，我们的优化目标是最大化分类间隔。较大的分类间隔意味着模型有较小的泛化误差，较小的间隔则意味着模型可能会过拟合。sklearn.svm.SVC()全称是 C-Support Vector Classification，是一种基于 libsvm 的支持向量机，由于其时间复杂度为 O(n^2)，所以当样本数量超过两万时难以实现。

## 三、结果分析：

### 3.1 导入测试数据，计算模型准确率

混淆矩阵是机器学习中总结分类模型预测结果的情形分析表，以矩阵形式将数据集中的记录按照真实的类别与分类模型预测的类别判断两个标准进行汇总。其中矩阵它是一种特定的二维矩阵：列代表预测的类别；行代表实际的类别。对角线上的值表示预测正确的数量/比例；非对角线元素是预测错误的部分。混淆

矩阵的对角线值越高越好，表明许多正确的预测。所以从混淆矩阵中可以很方便直观的看出哪里有错误，因为他们呈现在对角线外面。

根据(xi,yi)在坐标上的点，生成的曲线，然后计算 AUC 值；直接根据真实值（必须是二值）、预测值（可以是 0/1,也可以是 proba 值）计算出 auc 值多分类计算。绘制 roc 曲线，曲线是以假正率（FPR）和真正率（TPR）为轴的曲线，ROC 曲线下面的面积我们叫做 AUC，纵坐标为真阳性率（True Positive Rate，TPR）：TPR = TP / P，其中 P 是真实正样本的个数，TP 是 P 个正样本中被分类器预测为正样本的个数。横坐标为假阳性率（False Positive Rate，FPR）：FPR = FP / N ，N 是真实负样本的个数，FP 是 N 个负样本中被分类器预测为正样本的个数。在深度学习中，分类任务评价指标是很重要的，一个好的评价指标对于训练一个好的模型极其关键；如果评价指标不对，对于任务而言是没有意义的。所以我们需要一个好的评价指标来。目前一般都是用精准率，召回率，F1 分数来评价模型；在 sklearn 中有自动生成这些指标的的工具，就是 classification_report 模块。以决策树模型为例：

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 15 |
| 1 | 1.00 | 0.87 | 0.93 | 15 |
| 2 | 0.88 | 1.00 | 0.94 | 15 |
| accuracy | | | 0.96 | 45 |
| macro avg | 0.96 | 0.96 | 0.96 | 45 |
| weighted avg | 0.96 | 0.96 | 0.96 | 45 |

precision 表示精度，即模型预测的结果中有多少是预测正确的，公式为正确预测的个数(TP)/被预测正确的个数(TP+FP)，

recall 表示召回率，即某个类别测试集中的总量，有多少样本预测正确了，公式为正确预测的个数(TP)/预测个数(TP+FN)；

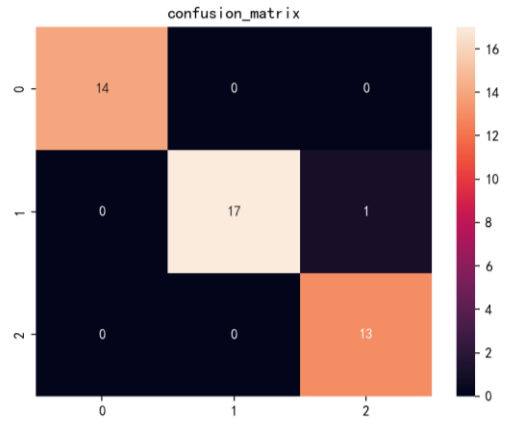
f1-score 为 F1 分数（F1-score），是分类问题的一个衡量指标，它是精确率和召回率的调和平均数，最大为 1，最小为 0，F1 分数认为召回率和精确率同等重要。

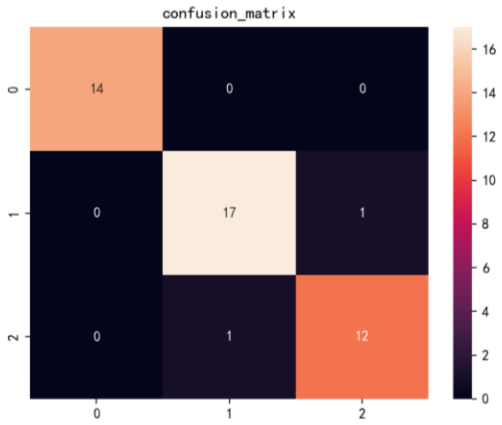
support 表示当前行的类别在测试数据中的样本总量

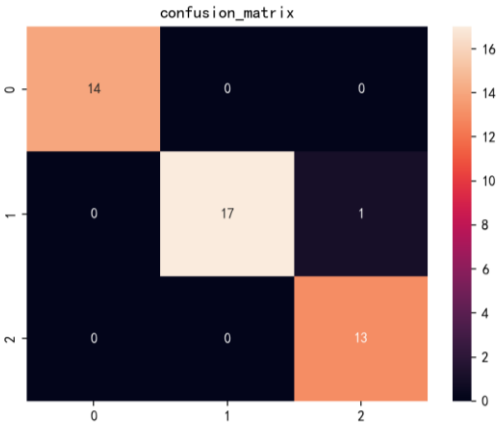
macro avg：每个类别评估指标未加权的平均值，比如准确率的 macro avg，

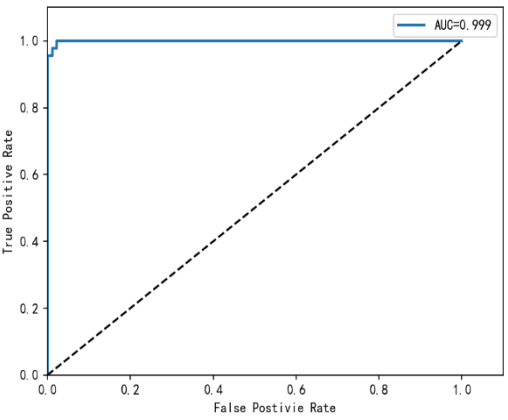
weighted avg：加权平均，就是测试集中样本量大的，我认为它更重要，给他设置的权重大点；比如第一个值的计算方法，(0.50*1 + 0.0*1 + 1.0*3)/5 = 0.70

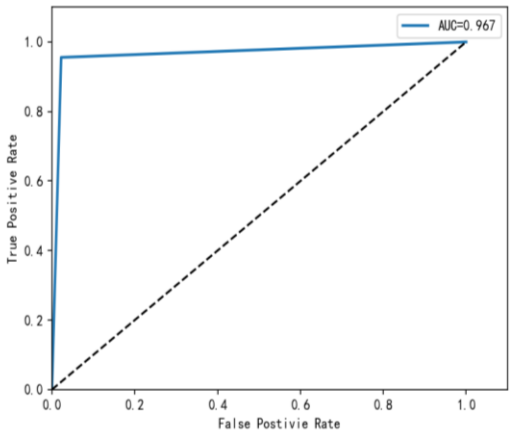
通过热力图观察混淆矩阵以及 ROC 曲线，对比这三种算法的准确度。

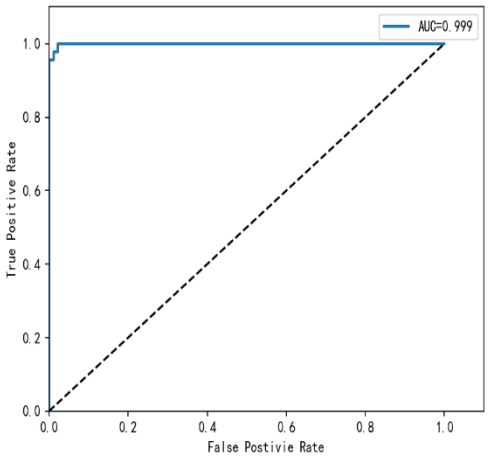



### 3.2 导入未知数据，实际应用模型

```
5.0,3.0,1.6,0.2,Iris-setosa
actual: [[5.  2.9 1.  0.2]]
There are three kinds of flowers: [0]-iris-setosa [1]-iris-versicolour [2]-iris-virginica
Prediction: [0]
5.4,3.0,4.5,1.5,Iris-versicolor
actual: [[5.3 3.  4.6 1.5]]
There are three kinds of flowers: [0]-iris-setosa [1]-iris-versicolour [2]-iris-virginica
Prediction: [1]
6.1,3.0,4.9,1.8,Iris-virginica
actual: [[6.1 3.1 4.9 1.9]]
There are three kinds of flowers: [0]-iris-setosa [1]-iris-versicolour [2]-iris-virginica
Prediction: [2]
```

预测结果基本正确。

## 四、源代码

```
# 导入包
import pandas as pd
import matplotlib; matplotlib.use('TkAgg')
import seaborn as sns
from pylab import *
from sklearn import tree, svm, metrics
from sklearn.datasets import load_iris  # 导入数据集 iris
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize

mpl.rcParams['font.sans-serif'] = ['SimHei']  # 用于画图时显示中文
#****************可视化显示**********************#
def Visual(iris_data):
    print("describe:\n {}".format(iris_data.describe()))
    sns.pairplot(iris_data,kind="reg",diag_kind="kde",hue="Species",markers=["*", "x", "+"])
    plt.show()
#************加载数据集*****************#
def LoadDataSet():
    iris1 = pd.read_csv('iris.csv')
    # 方法二：取 sk-learn 中内置的数据集
    iris2 = load_iris()  # 载入数据集
    # 读数据
    X = iris1[['Sepal.Length', 'Sepal.Width','Petal.Length', 'Petal.Width']].values
    y = iris2.target
    # 获取花卉两列数据集
    DD = iris2.data
    s_X = [x[0] for x in DD]
    print(s_X)
    s_Y = [x[1] for x in DD]
    print(s_Y)
    # plt.scatter(s_X, s_Y, c=iris.target, marker='x')
    plt.scatter(s_X[:50], s_Y[:50], color='red', marker='o', label='setosa')  # 前 50 个样本
    plt.scatter(s_X[50:100], s_Y[50:100], color='blue', marker='x', label='versicolor')  # 中间 50 个
    plt.scatter(s_X[100:], s_Y[100:], color='green', marker='+', label='Virginica')  # 后 50 个样本
    plt.legend(loc=2)  #loc=1，2，3，4 分别表示 label 在右上角，左上角，左下角，右下角
    plt.show()
    # 将数据划分为训练集和测试集,将 iris_data 分为 70%的训练，30%的进行预测 然后进
```

```
行优化 输出准确率、召回率等
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,random_state=1)  #size 样本占比，作用是从样本中随机的按比例选取 train_data 和 test_data
    return X_train, X_test, y_train, y_test
#****************逻辑回归*****************#
# 训练 Logistic 模性
def trainLS(X_train, X_test, y_train, y_test):
    # Logistic 生成和训练
    clf = LogisticRegression()
    clf.fit(X_train, y_train)
    predict_target = clf.predict(X_test)
    print("sum(predict_target == y_test):\n {}/45".format(sum(predict_target == y_test)))# 预测结果与真实结果比对
    print("metrics.classification_report:\n {}".format(metrics.classification_report(y_test, predict_target)))
    return clf
#***************决策树分析******************#
# 训练决策树模型
def trainDT(X_train, X_test, y_train, y_test):
    # DTC 生成和训练
    clf = tree.DecisionTreeClassifier(criterion="entropy")
    clf.fit(X_train, y_train)
    predict_target = clf.predict(X_test)
   print("sum(predict_target==y_test):\n{}/45".format(sum(predict_target==y_test)))# 预测结果与真实结果比对
  print("metrics.classification_report:\n{}".format(metrics.classification_report(y_test, predict_target)))
    return clf
#***************SVM 模型***********************#
def trainSVM(X_train, X_test, y_train, y_test):
    # SVM 生成和训练
    clf = svm.SVC(kernel='rbf', probability=True)
    clf.fit(X_train, y_train) #用训练器数据拟合分类器模型
    predict_target = clf.predict(X_test)
    print("sum(predict_target==y_test):\n{}/45".format(sum(predict_target==y_test)))# 预测结果与真实结果比对
 print("metrics.classification_report:\n{}".format(metrics.classification_report(y_test, predict_target)))
    return clf

def drawROC(y_one_hot, y_pre_pro):
    auc = roc_auc_score(y_one_hot, y_pre_pro, average='micro') # AUC 值
    # 绘制 ROC 曲线
    fpr, tpr, thresholds = roc_curve(y_one_hot.ravel(), y_pre_pro.ravel())
```

```
        plt.plot(fpr, tpr, linewidth=2, label='AUC=%.3f' % auc)
        plt.plot([0, 1], [0, 1], 'k--')
        plt.axis([0, 1.1, 0, 1.1])
        plt.xlabel('False Postivie Rate')
        plt.ylabel('True Positive Rate')
        plt.legend()
        plt.show()

# 测试模型
def test(model, x_test, y_test):
        # 将标签转换为 one-hot 形式
        y_one_hot = label_binarize(y_test, classes=np.arange(3))
        # 预测结果
        y_pre = model.predict(x_test)
        # 预测结果的概率
        y_pre_pro = model.predict_proba(x_test)
        f, ax = plt.subplots()#sns.set()
        # 混淆矩阵
        con_matrix = confusion_matrix(y_test, y_pre)
        print('confusion_matrix:\n', con_matrix) #混淆矩阵
        sns.heatmap(con_matrix, annot=True, ax=ax)  # 画热力图
        ax.set_title('confusion_matrix')  # 标题
        plt.show()

        print('accuracy:{}'.format(accuracy_score(y_test, y_pre))) #准确度
        print('precision:{}'.format(precision_score(y_test, y_pre, average='micro')))
        print('recall:{}'.format(recall_score(y_test, y_pre, average='micro'))) #召回率
        print('f1-score:{}'.format(f1_score(y_test, y_pre, average='micro')))

        drawROC(y_one_hot, y_pre_pro)  # 绘制 ROC 曲线
# 预测结果
def actual_forecast(model,iris_data):
         X_new_0 = np.array([[5.0, 2.9, 1.0, 0.2]])
         X_new_1 = np.array([[5.3, 3.0, 4.6, 1.5]])
        X_new_2 = np.array([[6.1, 3.1, 4.9, 1.9]])
        print("actual: {}".format(X_new_1))
        prediction = model.predict(X_new_1)
        print("There are three kinds of flowers: [0]-iris-setosa [1]-iris-versicolour [2]-iris-virginica")
        print("Prediction: {}".format(prediction))

if __name__ == '__main__':
        # 方法一：用 pandas 读取数据
        url = "iris.csv"
        iris_data = pd.read_csv(url)
```

```
print("iris_data.head(5):\n {}".format(iris_data.head(5)))
Visual(iris_data)
X_train, X_test, y_train, y_test = LoadDataSet()
# 训练 Logistic 模型
model = trainLS(X_train, X_test, y_train, y_test)
test(model, X_test, y_test)
# 训练决策树模型
model = trainDT(X_train, X_test, y_train, y_test)
test(model, X_test, y_test)
# 训练 SVM 模型
model = trainSVM(X_train, X_test, y_train, y_test)
test(model, X_test, y_test)
actual_forecast(model,iris_data)
```

## 五、实验心得

通过三周的学习我有以下几点体会：

（1）训练模型前，一定要对数据进行预处理，一般是数据归一化吧，因为分类基本上是把复杂的大量数据分成简单的类别，值域上最容易造成很大的差别。如果你的数据是随机生成的【0,1】或者【-1,1】的随机数据，本人认为到不用再归一化了，视分类情况而定吧。

（2）对训练数据和测试数据的标记，在训练和预测的时候，数据的组成部分里是一直都有标记的，对每一组数据进行标记，而不是想当然的对某一行或者某一列进行标记，标记矩阵的维数要和输入的训练数据和测试数据的维数相同，要不然就不能对所有的样本进行标记。

（3）初学阶段即使是直接调用 MATLAB 中的函数模型，也要懂得算法的理论原理，要不然知识机械的调用，算法是怎样运行的和你一点关系都没有。

总的来说，支持向量机分为线性支持向量机和非线性支持向量机，线性支持向量机又可以分为硬间隔最大化线性支持向量机和軟间隔最大化线性支持向量机；非线性支持向量机需要选择合适的核函数，从而达到非线性分类。