



# 《模式识别》

**课程大作业**

学 院 自动化学院

专 业 自动化

年级班别 2016级（4）班

组长姓名、学号 王雅湘(3116000915)

指导教师 邢延老师

2018 年 6 月10日

**摘 要**

**关键词**：反射式，光纤，位移，测量

**目 录**

**1** 绪论 1

1.1 题目背景及目的 1

1.2 国内外研究状况 2

2.1 基础知识 5

2.1.1 有限元法 5

2.1.2 循环对称结构的分析方法 6

2.2 I级叶/盘转子振动特性的有限元分析 7

2.2.1 计算模型 7

2.2.2 有限元计算结果及分析 8

**3 I** 级叶/盘转子错频方案的对比分析 15

3.1 计算模型及主要分析思路 15

3.2 基本原理 17

3.2.1 多自由度系统的固有频率和振型 17

3.2.2 多自由度系统的振动响应 19

3.3 协调系统的模拟 19

3.4 错频方案的拟定 21

3.5 多自由度系统的强迫响应分析 23

3.5.1 动态响应的计算方法 23

3.5.2 强迫响应分析前的准备工作 25

结论 48

参考文献 49

附录A 51

附录B 52

# 绪论

**1.1 模式识别概论**

模式识别（英语：Pattern recognition），就是通过计算机用数学技术方法来研究模式的自动处理和判读。我们把环境与客体统称为“模式”。随着计算机技术的发展，人类有可能研究复杂的信息处理过程。计算机识别的显著特点是速度快、准确性高、效率高，在将来完全可以取代人工录入。信息处理过程的一个重要形式是生命体对环境及客体的识别。对人类来说，特别重要的是对光学信息（通过视觉器官来获得）和声学信息（通过听觉器官来获得）的识别。这是模式识别的两个重要方面。市场上可见到的代表性产品有光学字符识别、语音识别系统。

解决模式识别问题的方法主要可以归纳为基于知识的方法和基于数据的方法两大类。

1. 基于知识的方法，主要是指以专家系统为代表的方法，其基本思想是：根据人们已知的关于研究对象的知识，整理出若干描述特征与类别间关系的准则，建立一定的计算机推理系统，对未知样本通过这些知识推理决策其类别；
2. 基于数据的模式识别方法是在确定了描述样本所采用的特征之后，这些方法并不是依靠人们对所研究对象的认知来建立分类系统，而是收集一定数量的已知样本，用这些样本作为训练集来训练一定的模式识别机器，使之在训练后能够对未知样本进行分类。

基于数据的模式识别是模式识别最主要的方法，人们常说的模式识别通常就是指这一类方法。本文中讨论的人脸识别问题即是基于数据的模式识别。

从另一个角度，模式识别又分为监督模式识别问题 (Supervised Pattern Recognition) 与非监督学习问题 (Unsupervised Pattern Recognition)。监督模式识别问题中，我们已知要划分的类别，并且能够获得一定数量的类别已知的训练样本，这种情况下建立分类器的问题属于监督学习问题。本文讨论的问题是给定训练集及已知类别标签，建立分类器来判断测试集中未知样本的类别，属于监督模式识别问题。

**1.2 大作业任务概述**

给予实验数据为四个文本文件：

1. faceR: 训练数据，100个属性（其中第一个属性是编号，其余99个是本征脸(eigenface) 属性参数，共2000个人脸的数据。
2. faceS: 测试数据，100个属性（其中第一个属性是编号，其余99个是本征脸(eigenface) 属性参数，共2000个人脸的数据。
3. faceDR: 对faceR文件中的每一个人脸数据的说明。
4. faceDS: 对faceS文件中的每一个人脸数据的说明。

首先要删除缺失数据，在本征脸中表现为0值数据，在标签中为空值行或标注(\_missing descriptor)，用有监督学习机制设计并实现模式识别方法，用于进行人脸面部特征识别，如性别（男性、女性）、年龄（儿童、青少年、成年、老年）、佩戴眼镜（是、否）、戴帽子（是、否）、表情（微笑、严肃）等。

本人采用K-L变换 (Karhunen-Loeve Transform) 对数据进行降维处理，即特征提取，然后利用神经网络 (Nerual Networks)进行学习与分类“种族”这个特征。程序利用Python与numpy, pandas, matplotlib, sklearn科学计算库来实现以上功能。

1. **算法与程序说明**
   1. **人工神经网络 (Artificial Neural Network, ANN)**

神经网络是一种运算模型，由大量的节点（或称神经元）之间相互联接构成。每个节点代表一种特定的输出函数，称为激励函数（activation function）。每两个节点间的连接都代表一个对于通过该连接信号的加权值，称之为权重，这相当于人工神经网络的记忆。网络的输出则依网络的连接方式，权重值和激励函数的不同而不同。而网络自身通常都是对自然界某种算法或者函数的逼近，也可能是对一种逻辑策略的表达。

由于多层感知器神经网络具有通用非线性函数逼近器的性质，所以在模式识别问题，尤其是对于非线性模式识别问题，传统方法中需要设定特殊的非线性判别函数的形式才能设计分类器，或者需要设计分段线性分类器，而神经网络在此问题上显得非常方便，它具有“黑盒子”的特点，只要事先确定了神经网络结构，那么只需要用算法来训练神经网络，并不需要关心网络最后实现的分类器的具体形式。

* 1. **神经网络模型**

神经元由树突和轴突组成，一个神经元通常具有多个树突，主要用来接受传入信息；而轴突只有一条，轴突尾端有许多轴突末梢可以给其他多个神经元传递信息。轴突末梢跟其他神经元的树突产生连接，从而传递信号。

在单层人工神经网络模型中，传入信息为特征，输出信号为激励函数的结果。一般输入为偏置单元，总是等于1。激励函数为 sigmoid() 激励函数 ，其中参数常被称作为“权重”(weights)。所以一个简单的神经网络可以表示如下：

第一层为输入层，其输出进入第二层，第二层为输出层，输出激励函数的计算结果。输入层和输出层中间可以有多层作为隐藏层 (Hidden layer)。设中间层神经元为, 称为激活单元 (Activation Units)

表示第*j*层第*i*个单元的“激活”

表示从*j* 到*j+1*层的权重矩阵

如果加入隐藏层，神经网络可以表示为：

“激活”节点可以表示为下，输出结果是所有激活节点和的sigmoid函数计算结果：

* 1. **损失函数 (Cost Function) 与反向传播算法 (Backpropagation Algorithm)**

对于神经网络，其损失函数表示为：

其中：

L 为神经网络的总层数

为 *l* 层中的单元数量 (不算偏置单元)

K 为输出单元的数量

反向传播算法用于最小化损失函数，具体步骤如下：

1. 对于给定的训练集

对于所有的设

对于所有的训练集 t = 1 到 m

1. 令
2. 利用前向传播算法对所有*l*计算
3. 利用计算
4. 利用计算
5. 计算损失函数的偏导数

从此可以根据偏导数的性质，计算出损失函数最小的权重矩阵

* 1. **Python 实现代码、流程图及解释**

本人采用K-L变换 (Karhunen-Loeve Transform) 对数据进行降维处理，即特征提取，然后利用神经网络 (Nerual Networks)进行学习与分类“种族”这个特征。程序利用Python与numpy, pandas, matplotlib, sklearn科学计算库来实现以上功能。

**2.4.1 程序框图**

开始

将原始数据txt文件，转化为csv以逗号分隔的数据格式便于Python处理

将训练数据与测试数据及它们对应标签数据中的空行或错误数据行删除

利用K-L变换进行降维同时进行标准化处理，数据变为 (-1, 1) 范围内的值，且从99维降至赋予 n\_comonents 参数的任意维度

利用t-SNE进行数据可视化(降至2维或3维进行可视化)，利用seaborn库画出任意两维之间的分布图，或某一特征底下各类的分布。(可视化这一部分全部注释，取消注释即可进行可视化)

利用scikit-learn 库中的

Mluti-layer Perceptron 函数

MPLClassifier()对目标数据进行分类

**2.4.2 Python 代码及解释**

|  |  |
| --- | --- |
|  | #!/usr/bin/python |
|  | import os |
|  | import csv |
|  | import math |
|  | import time |
|  | import numpy as np |
|  | import pandas as pd |
|  | import seaborn as sns |
|  | sns.set\_palette('husl') |
|  | import matplotlib as mpl |
|  | from matplotlib import pyplot as plt |
|  |  |
|  | from sklearn import metrics |
|  | from sklearn.manifold import TSNE |
|  | from mpl\_toolkits.mplot3d import Axes3D |
|  | from sklearn.neural\_network import MLPClassifier |
|  | from sklearn.neighbors import KNeighborsClassifier |
|  | from sklearn.linear\_model import LogisticRegression |
|  | from sklearn.model\_selection import train\_test\_split |
|  |  |
|  |  |
|  |  |
|  | # reading eigenface data |
|  | training\_data = '/home/shawn/projects/pattern\_recognition\_class\_project/face/csv\_eigenfaces/faceR' |
|  | training\_data\_csv = '/home/shawn/projects/pattern\_recognition\_class\_project/face/csv\_eigenfaces/faceR\_csv' # This is a path in my Ubuntu Linux |
|  | testing\_data = '/home/shawn/projects/pattern\_recognition\_class\_project/face/csv\_eigenfaces/faceS' |
|  | testing\_data\_csv = '/home/shawn/projects/pattern\_recognition\_class\_project/face/csv\_eigenfaces/faceS\_csv' |
|  | label\_training = '/home/shawn/projects/pattern\_recognition\_class\_project/face/csv\_eigenfaces/faceDR' |
|  | label\_training\_csv = '/home/shawn/projects/pattern\_recognition\_class\_project/face/csv\_eigenfaces/faceDR\_csv' |
|  | label\_testing = '/home/shawn/projects/pattern\_recognition\_class\_project/face/csv\_eigenfaces/faceDS' |
|  | label\_testing\_csv = '/home/shawn/projects/pattern\_recognition\_class\_project/face/csv\_eigenfaces/faceDS\_csv' |
|  |  |
|  | train\_rename = '/home/shawn/projects/pattern\_recognition\_class\_project/face/csv\_eigenfaces/faceR\_csv' |
|  | train\_newname = '/home/shawn/projects/pattern\_recognition\_class\_project/face/csv\_eigenfaces/faceR\_csv.csv' |
|  | label\_train\_rename = '/home/shawn/projects/pattern\_recognition\_class\_project/face/csv\_eigenfaces/faceDR\_csv' |
|  | label\_train\_newname = '/home/shawn/projects/pattern\_recognition\_class\_project/face/csv\_eigenfaces/faceDR\_csv.csv' |
|  | test\_rename = '/home/shawn/projects/pattern\_recognition\_class\_project/face/csv\_eigenfaces/faceS\_csv' |
|  | test\_newname = '/home/shawn/projects/pattern\_recognition\_class\_project/face/csv\_eigenfaces/faceS\_csv.csv' |
|  | label\_test\_rename = '/home/shawn/projects/pattern\_recognition\_class\_project/face/csv\_eigenfaces/faceDS\_csv' |
|  | label\_test\_newname = '/home/shawn/projects/pattern\_recognition\_class\_project/face/csv\_eigenfaces/faceDS\_csv.csv' |
|  |  |
|  | # ---------------- Converting Original Data to CSV format ----------------- |
|  | def convert\_data\_to\_csv(original\_file\_path, csv\_file\_path, old\_file\_name, new\_file\_name): |
|  | header\_for\_data = [] # define header for csv, named 0 - 99 |
|  | header\_temp = [] |
|  | for i in range(0, 100): |
|  | header\_temp.append(i) |
|  | header\_for\_data = [str(i) for i in header\_temp] # convert numbers to string type as there are all strings in csv file |
|  | # print(header) |
|  |  |
|  | # converting faceR (training data) to csv format |
|  | if os.path.exists(new\_file\_name) == False: |
|  | os.mknod(csv\_file\_path) # create "faceR\_csv.csv" file for csv format data |
|  | with open(original\_file\_path, 'r+') as f: |
|  | lines = f.readlines() |
|  | for i in range(0, len(lines)): |
|  | lines[i] = lines[i].lstrip() # There are spaces as the first character in the oroginal data file, .lstrip will delete the spaces |
|  | lines[i] = lines[i].replace(' ', ',') # replacing "space" with "comma" makes the data in csv format |
|  | lines[i] = lines[i].replace(' ', ',') |
|  | with open(csv\_file\_path, 'r+') as training\_csv: # write changes |
|  | writer = csv.writer(training\_csv) |
|  | writer.writerow(header\_for\_data) # write header line in the first row |
|  | training\_csv.writelines(lines) # write other data lines |
|  | os.rename(old\_file\_name, new\_file\_name) # add filename extension '.csv' |
|  | # Here we have a complete csv format eigenface data file named 'faceR\_csv.csv' |
|  | else: |
|  | print("########### The csv format training data already exists! ############") |
|  | print('\n') |
|  |  |
|  |  |
|  | def convert\_label\_to\_csv(original\_label\_file\_path, csv\_label\_file\_path, old\_label\_file\_name, new\_label\_file\_name): |
|  | # Converting faceDR (label for training data) to csv format |
|  | header\_for\_label = ['#', 'sex', 'age', 'race', 'face', 'prop'] |
|  | if os.path.exists(new\_label\_file\_name) == False: |
|  | os.mknod(csv\_label\_file\_path) |
|  | with open(original\_label\_file\_path, 'r+') as f: |
|  | lines = f.readlines() |
|  | for i in range(0, len(lines)): |
|  | lines[i] = lines[i].lstrip() |
|  | lines[i] = lines[i].replace('\_sex ', '') |
|  | lines[i] = lines[i].replace('\_age ', '') |
|  | lines[i] = lines[i].replace('\_race ', '') |
|  | lines[i] = lines[i].replace('\_face ', '') |
|  | lines[i] = lines[i].replace("(\_prop '", "") # double quotes and single quotes in python can both be used to represent string. |
|  | lines[i] = lines[i].replace('hat ', 'hat\_') |
|  | lines[i] = lines[i].replace('moustache ', 'moustache\_') |
|  | lines[i] = lines[i].replace('glasses ', 'glasses\_') |
|  | lines[i] = lines[i].replace('bandana ', 'bandana\_') |
|  | lines[i] = lines[i].replace('beard ', 'beard\_') |
|  | lines[i] = lines[i].replace('(\_missing descriptor)', '(missing\_descriptor)') |
|  | lines[i] = lines[i].replace('\_)', '') |
|  | lines[i] = lines[i].replace('))', ')') |
|  | lines[i] = lines[i].replace('()', '(\)') |
|  | lines[i] = lines[i].replace(' ', ',') |
|  | lines[i] = lines[i].replace(' ', ',') |
|  | with open(csv\_label\_file\_path,'r+') as csv\_label\_file\_path: |
|  | writer = csv.writer(csv\_label\_file\_path) |
|  | writer.writerow(header\_for\_label) |
|  | csv\_label\_file\_path.writelines(lines) |
|  | os.rename(old\_label\_file\_name, new\_label\_file\_name) |
|  | else: |
|  | print("########### The csv format training label already exists! ###########") |
|  | print('\n') |
|  |  |
|  | convert\_data\_to\_csv(training\_data, training\_data\_csv, train\_rename, train\_newname) # training data to csv |
|  | convert\_data\_to\_csv(testing\_data, testing\_data\_csv, test\_rename, test\_newname) #testing data to csv |
|  |  |
|  | convert\_label\_to\_csv(label\_training, label\_training\_csv, label\_train\_rename, label\_train\_newname) # training label to csv |
|  | convert\_label\_to\_csv(label\_testing, label\_testing\_csv, label\_test\_rename, label\_test\_newname) # testing label to csv |
|  |  |
|  |  |
|  | # ----------------------- Append labels to training data -------------------- |
|  | train = pd.read\_csv(train\_newname, sep=',', header=0) # 'header' sets which row as index for columns/ 'index\_col' sets which column as index for rows |
|  | test = pd.read\_csv(test\_newname, sep=',', header=0) |
|  | train\_label = pd.read\_csv(label\_train\_newname, sep=',', header=0) |
|  | test\_label = pd.read\_csv(label\_test\_newname, sep=',', header=0) |
|  | train\_race = train\_label.iloc[:,3] |
|  | test\_race = test\_label.iloc[:,3] |
|  |  |
|  | training\_data\_with\_labels = pd.concat([train, train\_race], axis=1) |
|  | testing\_data\_with\_labels = pd.concat([test, test\_race], axis=1) |
|  | # training\_data\_with\_label = train.merge(race) # cannot use merge or join here |
|  | print('INFO: Labels of "RACE" have been appended to the last column of training amd testing data') |
|  | print('INFO: Now Training data is a Dataframe with %s rows x %s columns' %(training\_data\_with\_labels.shape[0], training\_data\_with\_labels.shape[1])) |
|  | print('INFO: Now Testing data is a Dataframe with %s rows x %s columns' %(testing\_data\_with\_labels.shape[0], testing\_data\_with\_labels.shape[1])) |
|  | # print(training\_data\_with\_labels) |
|  |  |
|  | # --------------- Deleting Missing Rows ------------- |
|  | def deleting\_missing\_rows(df, cleared\_data): |
|  | missing\_rows = [] |
|  | cleared\_data = df |
|  | for i in range(0, 1999): |
|  | sum = 0 |
|  | check\_empty = df.iloc[i,:] |
|  | if isinstance(check\_empty[100], float): |
|  | missing\_rows.append(i) |
|  | else: |
|  | for a in range(1, 99): |
|  | sum = sum + check\_empty[a] |
|  | if sum == 0 : |
|  | if a == 2: # 2 is random selected number, if will append 'a' 99 times otherwise |
|  | missing\_rows.append(i) |
|  | print('INFO: These rows have NaN labels ot zero data: %s' %(missing\_rows)) |
|  |  |
|  | # drop rows with missing data |
|  | a = 0 |
|  | for i in missing\_rows: |
|  | cleared\_data = cleared\_data.drop(cleared\_data.index[i-a]) |
|  | a = a + 1 # having a to solve index problems after deleting rows |
|  | print('INFO: Deleted Row %d' %(i,)) |
|  | cleared\_data = cleared\_data.reset\_index(drop=True) # reset row index that has become inconsecutive after deleting missing rows |
|  | return cleared\_data |
|  |  |
|  | cleared\_training\_data = training\_data\_with\_labels |
|  | cleared\_testing\_data = testing\_data\_with\_labels |
|  |  |
|  | cleared\_training\_data = deleting\_missing\_rows(training\_data\_with\_labels, cleared\_training\_data) |
|  | cleared\_testing\_data = deleting\_missing\_rows(testing\_data\_with\_labels, cleared\_testing\_data) |
|  |  |
|  | print('INFO: Now After clearing missing data, Training data is a Dataframe with [%s rows x %s columns]' %(cleared\_training\_data.shape[0], cleared\_training\_data.shape[1])) |
|  | print('INFO: Now After clearing missing data, Testing data is a Dataframe with [%s rows x %s columns]' %(cleared\_testing\_data.shape[0], cleared\_testing\_data.shape[1])) |
|  | print('\n') |
|  | print('----------------------------------------- Cleared Training Data -------------------------------------------') |
|  | print(cleared\_training\_data) |
|  | print('-----------------------------------------------------------------------------------------------------------') |
|  | print('\n') |
|  | print('----------------------------------------- Cleared Testing Data -------------------------------------------') |
|  | print(cleared\_testing\_data) |
|  | print('-----------------------------------------------------------------------------------------------------------') |
|  | print('\n') |
|  | print('--------- Training Dataset info ---------') |
|  | print(cleared\_training\_data.info()) |
|  | print('-----------------------------------------') |
|  | print('Number of samples in each label:') |
|  | print(cleared\_training\_data['race'].value\_counts()) |
|  | print('-----------------------------------------') |
|  | print('\n') |
|  | print('--------- Training Dataset info ---------') |
|  | print(cleared\_testing\_data.info()) |
|  | print('-----------------------------------------') |
|  | print('Number of samples in each label:') |
|  | print(cleared\_testing\_data['race'].value\_counts()) |
|  | print('-----------------------------------------') |
|  |  |
|  | # ------------------------ Plots ------------------------ |
|  | df\_train = cleared\_training\_data |
|  | df\_test = cleared\_testing\_data |
|  | # plot features in a pair plot, will plot 99\*99 figures |
|  | # tmp = df.drop('0', axis=1) |
|  | # g = sns.pairplot(tmp, hue='race', markers='+') |
|  | # plt.show() |
|  |  |
|  | g = sns.violinplot(y='race', x='50', data=df\_train, inner='quartile') # plot a violin plot with feature 27 |
|  |  |
|  | X\_train = df\_train.drop(['0', 'race'], axis=1) |
|  | y\_train = df\_train['race'] |
|  | X\_test = df\_test.drop(['0', 'race'], axis=1) |
|  | y\_test = df\_test['race'] |
|  |  |
|  | # ------------------ Plot 2D using TSNE --------------- |
|  | # tsne = TSNE(n\_components=2, random\_state=0) |
|  | # X\_2d = tsne.fit\_transform(X) |
|  | # label\_ids = ['(white)', '(black)', '(asian)', '(hispanic)', '(other)'] |
|  | # plt.figure(figsize=(10, 5)) |
|  | # colors = 'r', 'g', 'b', 'c', 'm' |
|  | # for i, c, label in zip(label\_ids, colors, label\_ids): |
|  | # plt.scatter(X\_2d[y == i, 0], X\_2d[y == i, 1], c=c, label=label, marker='+') |
|  |  |
|  | # ----------------- Plot 3D using TSNE -------------- |
|  | # tsne = TSNE(n\_components=3, random\_state=0, n\_iter=5000) |
|  | # X\_3d = tsne.fit\_transform(X) |
|  | # label\_ids = ['(white)', '(black)', '(asian)', '(hispanic)', '(other)'] |
|  | # colors = 'r', 'g', 'b', 'c', 'm' |
|  | # mid = int(len(X\_3d)/2) |
|  | # fig = plt.figure() |
|  | # print(mid) |
|  | # ax = fig.add\_subplot(111, projection='3d') |
|  | # for i, c, label in zip(label\_ids, colors, label\_ids): |
|  | # ax.scatter(X\_3d[y == i, 0][0: ], X\_3d[y==i, 1][0: ], X\_3d[y==i,2][0: ], c=c, s=100, label=label, marker='+') |
|  | plt.legend() |
|  | plt.show() |
|  |  |
|  |  |
|  | # ------------------ Applying Multi-layer Perceptron ------------------ |
|  | # X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0) |
|  | # print('There are {} samples in the training set and {} samples in the test set'.format( |
|  | # X\_train.shape[0], X\_test.shape[0])) |
|  |  |
|  | def print\_accuracy(f): |
|  | print('\n') |
|  | print('------------- Multi-layer Perceptron ------------') |
|  | print("Right Classification Samples: {0}".format(np.sum(f(X\_test)==y\_test))) |
|  | print("Wrong Classification Samples: {0}".format(np.sum(f(X\_test)!=y\_test))) |
|  | print("Accuracy for Multi-layer Perceptron = {0}%".format(100\*np.sum(f(X\_test) == y\_test)/len(y\_test))) |
|  | time.sleep(0.5) # to let the print get out before any progress bars |
|  | print('-------------------------------------------------') |
|  | print('\n') |
|  |  |
|  | nn = MLPClassifier(solver='lbfgs', alpha=1e-1, hidden\_layer\_sizes=(500,200), random\_state=0) |
|  | nn.fit(X\_train, y\_train) |
|  | print\_accuracy(nn.predict) |
|  |  |
|  |  |
|  | # ------------------ Applying K-Nearest Neighbors ------------------ |
|  | k\_range = list(range(1,26)) |
|  | scores = [] |
|  |  |
|  | k\_range = list(range(1,26)) |
|  | scores = [] |
|  | for k in k\_range: |
|  | knn = KNeighborsClassifier(n\_neighbors=k) |
|  | knn.fit(X\_train, y\_train) |
|  | y\_pred = knn.predict(X\_test) |
|  | scores.append(metrics.accuracy\_score(y\_test, y\_pred)) |
|  |  |
|  | plt.plot(k\_range, scores) |
|  | plt.xlabel('Value of k for KNN') |
|  | plt.ylabel('Accuracy Score') |
|  | plt.title('Accuracy Scores for Values of k of k-Nearest-Neighbors') |
|  | plt.show() |
|  | logreg = LogisticRegression() |
|  | logreg.fit(X\_train, y\_train) |
|  | y\_pred = logreg.predict(X\_test) |
|  |  |
|  | knn\_accuracy = 100\*metrics.accuracy\_score(y\_test, y\_pred) |
|  | knn\_accuracy = round(knn\_accuracy, 1) |
|  | print("Accuracy of KNN is {0}%".format(knn\_accuracy,)) |