基于MNIST数据集的图像模式识别实践

20338051 卢旖旎

一、理论基础与相关知识预备

本次图像模式识别实践先后采用四种方法：KNN，Naive Bayes, SVM, Netural network，涉及编程语言有Matlab, R, Python，为了逻辑畅通，写作顺序为：Naive Bayes, KNN, SVM, Netural network，和当时实现顺序略有不同。

本部分内容将阐述不同算法中一些共通的部分，如特征提取(feature selection) 的思路推导，k折交叉验证的理论部分，以及模型解释的相关判别准则。至于不同方法的具体实现过程，将留在下文解释。

1. 特征提取方法

PCA算法

(1) Introduction

高维数据会增加数据分析的复杂性，这是因为变量之间有可能存在极强的相关性，于是采集过多的存在相关性的数据不仅会带来“信息冗余”，其多重共线性也会使模型估计失真，同时处理数据时过大的数据量会增加时间成本。因此我们会很自然地想到：能否将高维的数据通过某种线性（或者非线性）变换映射到低维空间，并且不会损失过多的重要信息，也就是说映射后的低维数据依旧拥有原数据的特征，并且低维数据向量之间是两两不相关的。

而PCA(Principal Component Analysis) 算法是目前较为成熟有效的通过线性变换进行数据降维的方法。PCA 算法是利用线性变换，将数据从r 维线性空间映射到k 维线性空间，也就是说从原始的空间中按照变量方差降序的方式找一组相互正交的坐标轴，其中，第一个新坐标轴选择是原始数据中方差最大的方向（因为方差是一种衡量变量信息的数字特征），第二个新坐标轴选取是与第一个坐标轴正交的平面中使得方差最大的，第三个轴是与第1,2个轴正交的平面中方差最大的。

(2) 数学推导

假设现在有p×n维随机矩阵$\boldsymbol{X}=(X_1,X_2,\ldots,X_n)$,p 代表样本个数，n 代表特征个数 $E(X)=\bar{X}$ covariance matrix Σ , 通过线性变换

$$\xi_j=b_j^T X^T=b_{j1}X_1+\ldots+b_{jn}X_n, j=1,2,\ldots t(t<n) \qquad (1)$$

将$\boldsymbol{X}$从n维空间映射到t维空间，$\boldsymbol{\xi}=(\xi_1,\ldots\xi_t):=\boldsymbol{BX}$，$\boldsymbol{B}=(\boldsymbol{b_1},\boldsymbol{b_2},\ldots\boldsymbol{b_t})^T$为$t\times n$维权重矩阵。系数$b_{ji}$的确定原则：a) $\xi_i\perp\xi_j$ b) ξ_1在$X_1,\ldots X_n$的所有线性组合中方差最大 c) ξ_m是与$\xi_1,\ldots\xi_{m-1}$不相关的，且在与$X_1..X_n$的所有组合中方差最大

下面阐述基于矩阵$\boldsymbol{SVD}$(singular value decomposition，centered and possibly scaled)的主成分分析过程

① 标准化：$\boldsymbol{X_1}=\boldsymbol{X}-\bar{\boldsymbol{X}}$（有时也可做方差标准化，此时有 center=0,scale=1）

② 计算 X_1 的协方差阵$\boldsymbol{A}$

③ 利用$\boldsymbol{SVD}$分解：$\boldsymbol{A}=\boldsymbol{UDV^T}$ ($\boldsymbol{UU^T}=\boldsymbol{VV^T}$=I), D 为对角矩阵

④ 取D的前t列$\boldsymbol{D_t}$，则降维后的数据矩阵 $\boldsymbol{Y_t}=\boldsymbol{UD_t}$

关于 t 的选取：一般取累计贡献率超过 80% 的特征值所对应的第一、第二、…、第 m（m ≤ n）个主成分。

2. 交叉验证

1. The Validation Set Approach

实际上就是我们平常说的训练集(training dataset)和测试集(testing dataset) 把整个数据集分成两部分，一部分用于训练，一部分用于验证。 但弊端在于划分依据是人为主观判断，容易导致偏差，比如不同的划分方式可能导致不同的模型误差。最终模型与参数的选取将极大程度依赖于你对训练集和测试集的划分方法。

200　　5. Resampling Methods

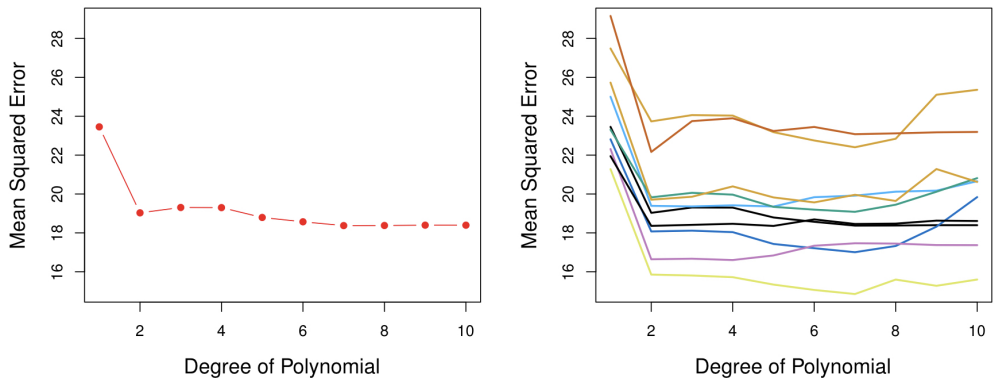


FIGURE 5.2. *The validation set approach was used on the* `Auto` *data set in order to estimate the test error that results from predicting* `mpg` *using polynomial functions of* `horsepower`. Left: *Validation error estimates for a single split into training and validation data sets.* Right: *The validation method was repeated ten times, each time using a different random split of the observations into a training and a validation set. This illustrates the variability in the estimated test MSE that results from this approach.*

右边是十种不同的训练集和测试集划分方法得到的test MSE，可以看到，在不同的划分方法下，test MSE的变动是很大的，而且对应的最优degree也不一样。所以如果我们的训练集和测试集的划分方法不够好，很有可能无法选择到最好的模型与参数。

因为对数据集进行划分分别进行数据的训练和预测的话，训练只是用到了部分的数据，而通常模型的效果与数据量成正相关，划分训练集测试集会让我们无法充分利用已有的数据，所以需要用交叉验证的方法。

2. k-Fold Cross-Validation (k折交叉验证)

a) 将所有数据集分成k份

b) 不重复地每次取其中一份做测试集，用其他 k-1份做训练集训练模型，之后计算该模型在测试集上的 MSE_i

c) 计算k次的test error

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} Err_i \qquad (2)$$

k 的选取基于 $Bias$ 和 $Variance$ 之间的一个平衡，k越大，训练集的数据越多，模型的Bias越小。但是K越大，又意味着每一次选取的训练集之前的相关性越大。而这种大相关性会导致最终的test error具有更大的Variance。

根据经验我们一般选择k=5或10。

3. 判断模型优劣的相关指标

(1) 混淆矩阵

	Fact:1	**Fact:0**
$pred:1$	TP (True Positive)	FP (False Positive)
$pred:0$	FN (False Negative)	TN (True Negative)

Accuracy=$\frac{TP+TN}{TP+FP+FN+TN}$, 预测准确率 =$P(X=1|Y=1)$

Sensitivity=$\frac{TP}{TP+FN}$,灵敏度（真阳率 TPR），也称召回率(recall) =$P(X=0|Y=0)$

Specificity=$\frac{TN}{TN+FP}$, 特异度，一般关注的是(1-specificity) 即假阳率

二、数据下载与预处理

本次实践的流程如下：1.Matlab上加载数据，完成KNN算法，用时半小时。然后求KNN算法对应的混淆矩阵和准确率，由于加载时间过长所以后续算法转用R语言。 2.在R语言上完成主成分降维后的KNN算法，朴素贝叶斯算法和部分SVM模型的搭建。 3.在Python上进行SVM分类和深度学习结构搭建和处理。

由于不同软件上加载数据方式不同，所以下面将描述自己处理数据的方式

(1) Matlab 上的数据加载

利用附件的*LoadMNISTImages* 和*LoadMNISTlabels* 文件下载数据：

```
%%图像读取,因为所使用的库函数要求行为图像，列为特征，于是作转置处理
images_train=(LoadMNISTImages('train-images.idx3-ubyte'))';
images_test=(LoadMNISTImages('t10k-images.idx3-ubyte'))';
labels_test=LoadMNISTLabels('t10k-labels.idx1-ubyte');
labels_train=LoadMNISTLabels('train-labels.idx1-ubyte');
```

(2) R语言上加载数据

在Matlab的mat文件基础上将其转化为.txt或者.csv,存到本地后再读入R中

```
setwd("C:/Users/Lucille/Desktop/major courses/image-processing/MNIST")
test=read.csv('test.csv', sep = ',', header = FALSE)
train=read.csv('train.csv',sep = ',',header = FALSE)#行为图像，列为特征
label_train=read.table('labeltrain.txt')
## data 文件中含有label_test 和 在matlab运行KNN得到的predict_index,将其读入R是为了画混淆矩阵
data<-read.table("hunxiaojuzhen.txt",header = TRUE,sep = '\t')
label_test=data$label_test;
lebel_test=as.data.frame(label_test) ##转化为数据框形式
```

(3) Python 上加载数据

因为个人电脑Python环境还没有完全配置好，即用不了tensorflow, 于是所有python代码在 google colab 上运行，google colab 有自带的mnist 数据集，直接下载即可

```
#import necessary libraries
import tensorflow as tf

#load training data and split into train and test sets
mnist = tf.keras.datasets.mnist

(x_train_mnist,y_train_mnist), (x_test_mnist,y_test_mnist) = mnist.load_data()
x_train_mnist, x_test_mnist = x_train_mnist / 255.0, x_test_mnist / 255.0
```

此外，还尝试了自己导入数据：

```
from google.colab import drive
import os

# Mount Google Drive
drive.mount('/gdrive', force_remount=True)

# Location of Zip File
```

```
drive_path = '/gdrive/MyDrive/imagedata.zip'
local_path = '/content'

# Copy the zip file and move it up one level (AKA out of the drive folder)
!cp '{drive_path}' .

# Navigate to the copied file and unzip it quietly
os.chdir(local_path)
!unzip -q 'imagedata.zip'
```

以上是文件解压缩的方法，下面读入矩阵

```
import pandas as pd
train=pd.read_csv('train.csv',sep=',',header=None)
test=pd.read_csv('test.CSV',sep=',',header=None)
label_train=pd.read_table('label_train.txt',header=None)
label_test=pd.read_table('label_test.txt',header=None)
#转化为dataframe类型，利于后续用concat 函数进行矩阵拼接，进行交叉验证
train=pd.DataFrame(train)
test=pd.DataFrame(test)
label_train=pd.DataFrame(label_train)
label_test=pd.DataFrame(label_test)
```

三、算法实现

1.朴素贝叶斯分类

(1) 理论推导

根据贝叶斯公式：

$$P(C|F_1,\dots F_n)=\frac{P(F_1,\dots F_n|C)P(C)}{P(F_1,\dots,F_n)} \tag{3}$$

其中，F_i为类别，C为特征，且C有若干类别。

对于给出的待分类项，求解在此项出现的条件下各个类别出现的概率，哪个最大，就认为此待分类项属于哪个类别。

我们假设$\boldsymbol{x}=\{x_1,\dots x_n\}$为一个待分类的样本，而$x_i$ 为$\boldsymbol{x}$ 的一个特征； 类别集合$C=\{y_1,\dots y_m\}$，并且**特征之间条件独立**，即 $P(\boldsymbol{x}|y_i)=P(x_1|y_i)P(x_2|y_i)\dots P(x_n|y_i)$

算法流程如下：

1. 在训练集上统计得到在各类别下各个特征属性的条件概率估计，即

$$P(x_1|y_1),\dots P(x_n|y_1),P(x_1|y_2),\dots P(x_n|y_2),\dots\dots,P(x_1|y_m),\dots P(x_n|y_m)$$

2. 根据贝叶斯公式计算

$$P(y_i|x_1,\dots x_n)=\frac{P(x_1,\dots x_n|y_i)P(y_i)}{P(x_1,\dots x_n)} \tag{4}$$

在特征x_i条件独立的条件下，$P(x_1,\dots x_n|y_i)=P(x_1|y_i)P(x_2|y_i)\dots P(x_n|y_i)$，故(4)式的分子部分可写作:

$$P(y_i)\prod_{j=1}^{n}P(x_j|y_i) \tag{5}$$

3. 因为分母对于所有类别为常数，因为我们只要将分子最大化皆可，若 $P(y_k|x_1,\dots x_n)=max\{P(y_1|x_1,\dots x_n),\dots P(y_m|x_1,\dots x_n)\}$，则认为$\boldsymbol{x}\in y_k$

(2) R实现

首先加载常用的与机器学习及绘图有关的包

```
## caret,e1071,class 包用于naive Bayes,SVM and KNN
library(caret)
library(e1071)
library(class)
## gmodels, ggpolt2 用于画图
library(gmodels)
library(ggplot2)
library("factoextra")##PCA
```

1.数据未降维时的算法实现

```
model_Bayes<-naiveBayes(train,label_train) ## model
pre_Bayes=predict(model_Bayes,test) ## predict test data
cm=confusionMatrix(factor(pre_Bayes),factor(label_test))
cm
```

结果如下:

（1）混淆矩阵:对角线表示分类正确的数据

```
             Reference
Prediction    0    1    2    3    4    5    6    7    8    9
         0  887    0   47   19   11   71   17    1   12    6
         1    1 1095   32   43    8   37   18   19   91   13
         2    3    2  435   11    5    5    6    1    5    4
         3    2    3   83  599    1   35    2   11   12    7
         4    3    0    5    2  321   10    1   12    6    8
         5    6    1    4    7    8  105    7    3   12    0
         6   28    7  202   35   52   34  881    5   12    1
         7    2    0    7   10    8    4    0  385    7   14
         8   34   25  203  199  103  508   25   34  671   20
         9   14    2   14   85  465   83    1  557  146  936
```

(2) 准确率：63.15%

```
Overall Statistics

               Accuracy : 0.6315
                 95% CI : (0.622, 0.641)
    No Information Rate : 0.1135
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5901

 Mcnemar's Test P-Value : < 2.2e-16
```

朴素贝叶斯模型与其他分类方法相比具有最小的误差率，但实际表明我们的模型准确率只有63.15%，并不是一个很理想的数字，究其原因大概是数据量过大，朴素贝叶斯模型假设特征属性之间相互独立在本情况下并不成立，在属性个数比较多或者属性之间相关性较大时，分类效果不好。

于是，我们可以尝试进行一下改进：对数据进行降维，取前面若干个主成分，消除变量之间的相关性。

2. PCA+Naive Bayes

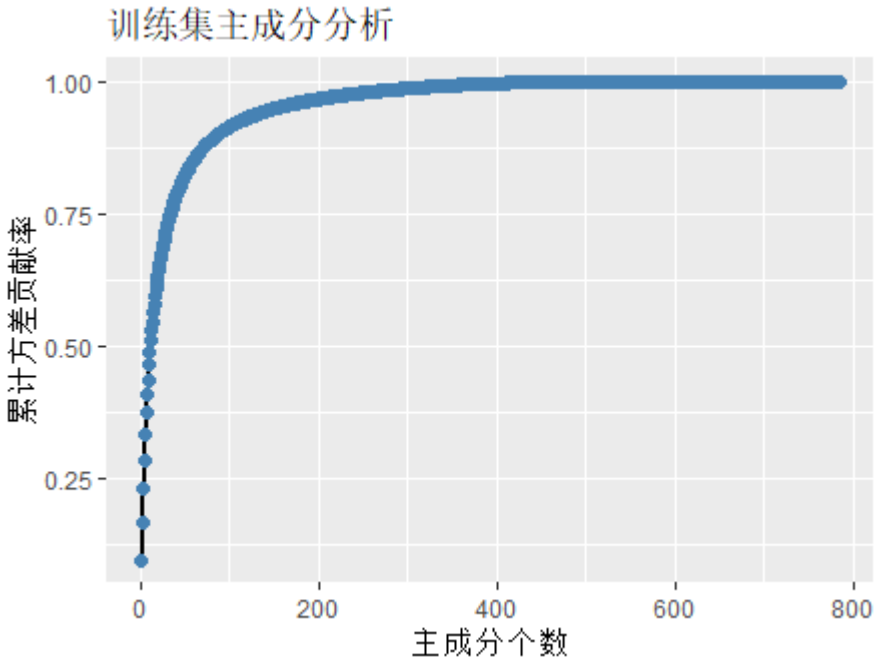
(1) PCA

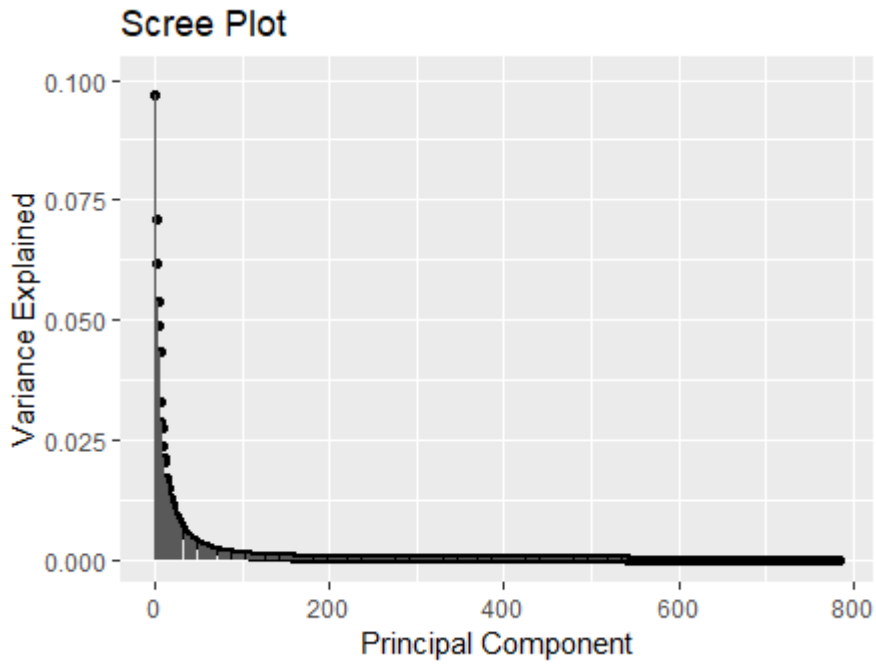
值得注意的是，由于图像矩阵中有部分特征列全为0，无法对方差标准化。本来试着删除全部为0的相关列然后进行主成分，但是训练集和测试集中全为0的列数不同，因此用在训练集得到的roatation matrix无法应用于测试集（不满足矩阵乘法条件）。

所以最后在进行主成分分析时只进行了中心化处理

```
##images_test: Naive-Bayes

#主成分分析
com1 <- prcomp(train, center = TRUE,scale. = FALSE)#不统一方差，因为有特征全部为0
PC1<-summ1$importance ##SD,Proportion of Variance,Cumulative Proportion

## 绘制累积方差贡献率图
x=as.vector(1:ncol(PC1))
y=PC1[3,]
plot1=data.frame(x,y)
p = ggplot(plot1, aes(x = x, y = y) )+
  geom_line(size = 0.8) +
  geom_point(color = "steelblue", size = 2)
p + xlab("主成分个数") + ylab("累计方差贡献率") + ggtitle("训练集主成分分析")

##绘制碎石图
variance = com1 $sdev^2 / sum(com1 $sdev^2)
qplot(c(1:784), variance) +

  geom_col()+

  xlab("Principal Component") +

  ylab("Variance Explained") +

  ggtitle("Scree Plot") +

```

```
28    ylim(0, 0.1)
```





判断大致的拐点，在主成分个数为50左右，取主成分个数为50

将特征向量按对应特征值大小从上到下按行排列成矩阵，取前50行组成矩阵 m_df,相关代码为：

```r
m_df <- as.data.frame(com1$x) #经过变换后的训练集
pre_train<-m_df[,1:50] # 降维成功的训练集，60000*50维
S<-as.matrix(com1$rotation) ##主成分与原特征之间的线性变换
test2<-as.matrix(test)
##转化后的测试集数据矩阵，取前50列特征值
 pre_test=test2%*%S ;pre_test=pre_test[,1:50]

model_Bayes<-naiveBayes(pre_train,label_train) ## model
pre_Bayes=predict(model_Bayes,pre_test) ## predict test data
cm=confusionMatrix(factor(pre_Bayes),factor(label_test))
cm
```

结果如下：

(1) 混淆矩阵

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2   3   4   5   6   7   8   9
         0 950   1  46  70  21 119 121  31  46  35
         1   0 290   0   0   0   0   0   0   0   0
         2  11  12 859  27 108  25  55  75  20  81
         3   2 286  23 808  19 123   9  40  15  21
         4   0   0   4   0 539   4   1  11   0   7
         5   2   0   0   0   0 283   6   1   0   0
         6   2   0   2   1  13   6 722   2   0   2
         7   0   2   0   2   0   0   0 667   0   1
         8  13 541  97  98 185 324  44 127 893 174
         9   0   3   1   4  97   8   0  74   0 688
```

(2) 准确率

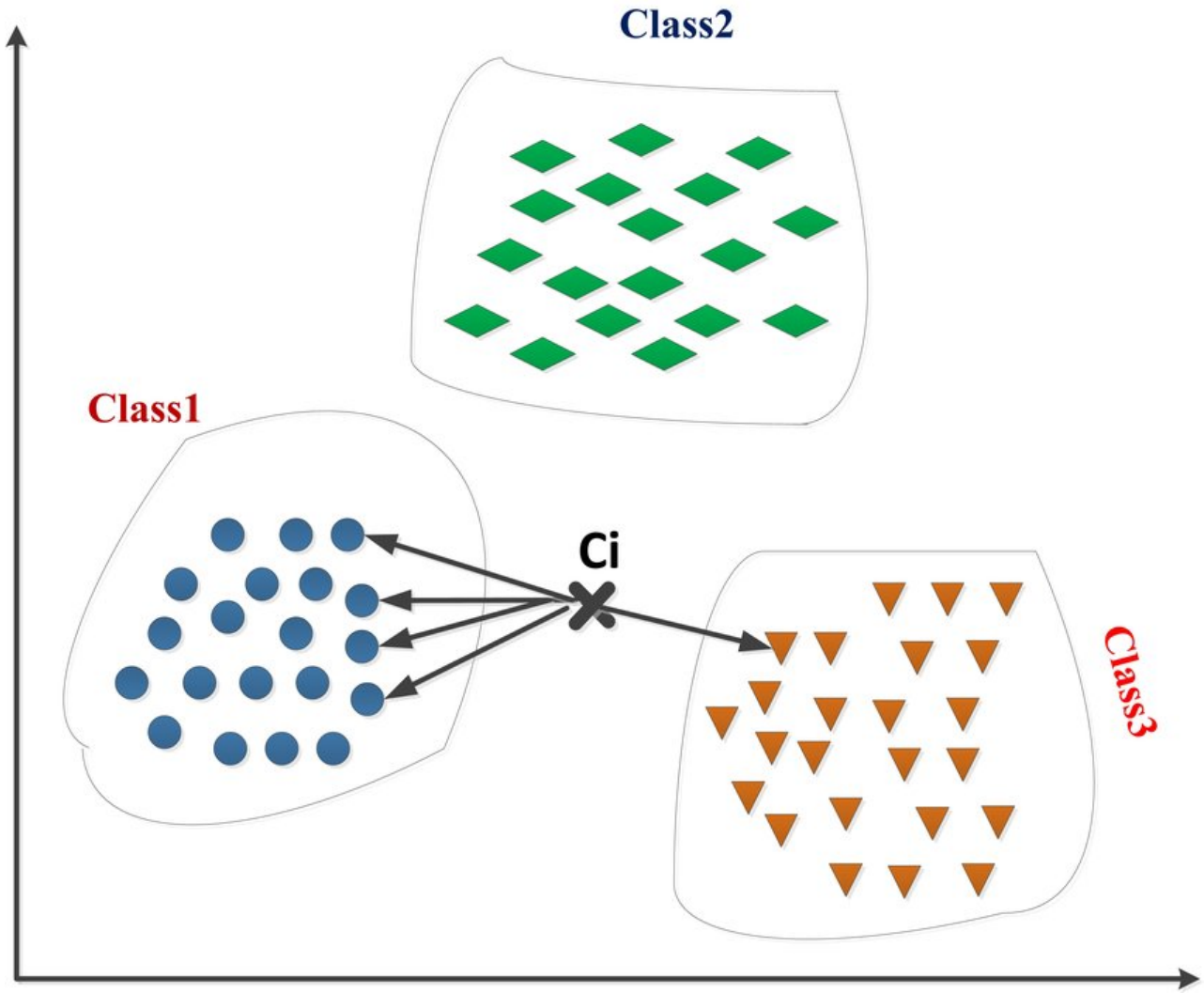
```
Overall Statistics

               Accuracy : 0.6699
                 95% CI : (0.6606, 0.6791)
    No Information Rate : 0.1135
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6334
```

可以看到准确率为66.99%，有所提升，但是还不是很高，归根结底就是朴素贝叶斯适用于小规模的数据，在数据量变大的时候容易因为变量之间的相关性而无法训练出一个较好的分类器。

2. KNN 分类

(1) 理论推导

KNN分类算法的分类预测过程思想是：对于一个需要预测的输入向量$\boldsymbol{x}$，我们只需要在训练数据集中寻找k个与向量$\boldsymbol{x}$ 最近的向量的集合，然后把$\boldsymbol{x}$ 的类别预测为这k个样本中类别数最多的那一类。



算法流程如下：

1. 输入训练集：$Train = \{(x_1, y_1), \dots (x_N, y_N)\}$, 其中$x_i \in \boldsymbol{X} \subset R^n$ 为n 维向量 ,$y_i \in \boldsymbol{Y} = \{c_1, \dots c_k\}$为类别
2. 根据给定的距离量度方式找到与样本X最相近的k个样本，并将这k个样本点记作$N_K(X)$

   这里采用欧氏距离:

$$L_2(x_i, x_j) = (\sum_{l=1}^{n} |x_i^{(l)} - x_j^{(l)}|^2)^{\frac{1}{2}} \tag{6}$$

3. 根据多数投票原则确定X所属的类别Y

$$Y = argmax \sum_{x_i \in N_K(X)} I(y_i, c_j) i = 1, 2, \dots N; j = 1, 2, \dots K \tag{7}$$

(2) 算法实现

1.未降维情况下的knn分类(matlab+R)

```
%首先在matlab尝试用KNN方法
%建立KNN_Classify.m
function [predictIndex,accuracy] =
KNN_Classify(trainFeatures,trainLabels,testFeatures,testLabels)
%K-近邻分类器
    classifer = fitcknn(trainFeatures,trainLabels,'NumNeighbors',5);#k=5
    [predictIndex,~] = predict(classifer,testFeatures);
    accuracy=length(find(predictIndex==testLabels))/length(testLabels);
end
```

```
[predictIndex,accuracy] =
KNN_Classify(images_train,labels_train,images_test,labels_test);%96.7%
```

虽然准确率较高，但是问题在于k值是提前确定的，我们不知道是否存在更优的k。此外，在数据未降维的情况下knn算法计算速度非常慢，在matlab环境运行大概半小时，效率较低。后来计算混淆矩阵时也卡顿了很久，所以决定使用R语言完成接下来的优化。一方面是R语言有比较多现成的包，方便调参；二来属于轻量级，运行方便。

于是接下来在R下进行knn算法模型的评估

```
data<-read.table("hunxiaojuzhen.txt",header = TRUE,sep = '\t')
label_train=read.table('labeltrain.txt')
label_test=data$label_test;
predict_index=data$predict_index##KNN未进行数据降维分类预测出来的结果，由matlab得到

cmat=confusionMatrix(factor(predict_index),factor(label_test)) #confusionMatrix 将预测类别的因子作为第一个参数，将用作真实结果的类别因子作为第二个参数
cmat

Confusion Matrix and Statistics

          Reference
Prediction    0    1    2    3    4    5    6    7    8    9
         0  972    0   13    0    1    4    5    0    6    5
         1    1 1132   12    3   11    0    4   27    4    6
         2    1    2  982    3    0    0    0    3    5    3
         3    0    0    2  971    0    6    0    0   11    6
         4    0    0    1    1  941    1    3    2    7   10
         5    2    0    0   13    0  869    2    0    9    4
         6    3    1    2    1    4    6  944    0    4    1
         7    1    0   17    9    1    1    0  984    7   11
         8    0    0    3    6    1    1    0    0  914    2
         9    0    0    0    3   23    4    0   12    7  961

Overall Statistics

               Accuracy : 0.967
                 95% CI : (0.9633, 0.9704)
    No Information Rate : 0.1135
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9633
```

2. PCA降维后的KNN分类

重点是k值的确定方式，参数的选取可以通过模型的反复试验得到。这里主要采用R语言Caret包进行KNN分类模型评估和选择，Caret包中trainControl函数可以用于设置交叉验证参数，train函数可以训练和评估模型，函数的操作流程如下：首先是选择一系列需要评估的参数和参数值的组合，然后设置重采样评估方式，循环训练模型评估结果、计算模型的平均性能，根据设定的度量值选择最好的模型参数组合，使用全部训练集和最优参数组合完成模型的最终训练。

这里所采用的模型评估指标为Kappa系数和Accuracy

```
Define sets of model parameter values to evaluate
for each parameter set do
    for each resampling iteration do
        Hold–out specific samples
        [Optional] Pre–process the data
        Fit the model on the remainder
        Predict the hold–out samples
    end
    Calculate the average performance across hold–out predictions
end
Determine the optimal parameter set
Fit the final model to all the training data using the optimal parameter set
```

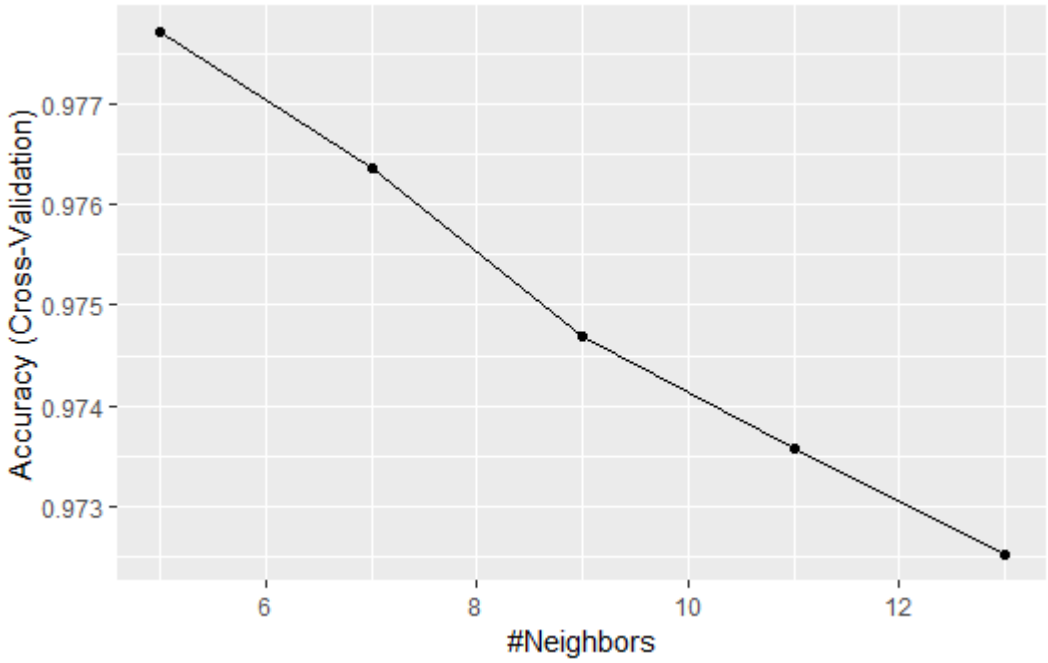
```r
##进行降维后的KNN训练
control <- trainControl(method = 'cv',number = 10) ## 10折交叉验证

label_train=unlist(label_train)
label_train=as.factor(label_train)
model <- train(pre_train,label_train,
                                method = 'knn',
                                trControl = control,
                                tuneLength = 5) ##tuneLength : 表示调整参数的组合数
```

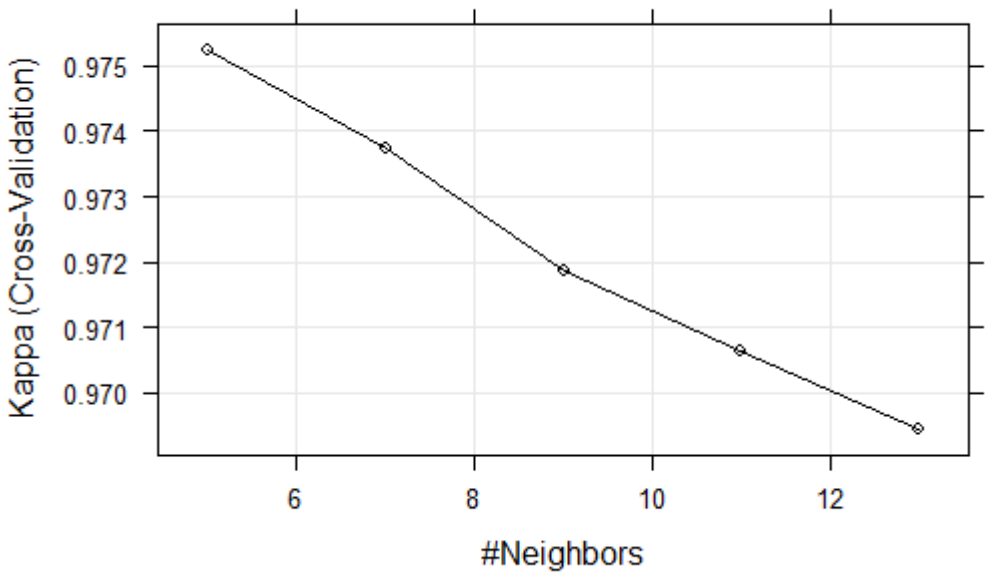
model 的结果为：

```
model
k-Nearest Neighbors

60000 samples
   50 predictor
   10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 54001, 53999, 53998, 54000, 54001, 54002, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   5  0.9777165  0.9752319
   7  0.9763665  0.9737312
   9  0.9746832  0.9718600
  11  0.9735831  0.9706371
  13  0.9725166  0.9694515

Accuracy was used to select the optimal model using the
 largest value.
The final value used for the model was k = 5.
```

Accuracy的这一列是通过交叉验证计算出的准确率。Kappa这一列是通过重抽样结果计算的Cohen Kappa统计量（非加权）。

```
library(lattice)
trellis.par.set(caretTheme())
plot(model,metric = "Kappa")
ggplot(model)
```





由kappa系数和accuracy的趋势可知，最优模型在k=5时取得。如果k值较大，虽然距离较远的训练样本也能够对实例预测结果产生影响，但算法的近邻误差会偏大，距离较远的点（与预测实例不相似）也会同样对预测结果产生影响，使得预测结果产生较大偏差，此时模型容易发生欠拟合。

下面利用上面得到的model在测试集上进行测试

```
predict_index_knn_pca=predict(model,pre_test)
tab<-table(predict_index_knn_pca,label_test)
confusionMatrix(tab)

Confusion Matrix and Statistics

                     label_test
predict_index_knn_pca    0    1    2    3    4    5    6    7    8    9
                    0  971    1   20    6   10   13   15    5    4   11
                    1    0 1118    0    0    2    0    4    3    1    4
                    2    0    1  947    2    0    0    0   13    1    0
                    3    1    4    5  926    0   38    0    7    5    3
                    4    0    0    3    0  781    0    2    3    1    1
                    5    0    0    0    8    0  719    3    0    1    2
                    6    2    3    8    0   15    9  923    1    1    2
                    7    1    0    4    3    1    0    0  864    1    0
                    8    5    8   45   61   25  101   11   27  958   42
                    9    0    0    0    4  148   12    0  105    1  944
```

```
Overall Statistics

               Accuracy : 0.9151
                 95% CI : (0.9095, 0.9205)
    No Information Rate : 0.1135
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9056
```

准确率为91.51%，虽然效果没有前面的好，但是运行时间为12mins,效率高了很多，而且准确率也在可接受的范围内。下降5%的原因可能在于PCA降维的时候损失了部分信息，但是不多，所以综上可以认为PCA降维+交叉验证的KNN分类法更好。

3. SVM

(1) 理论推导

SVM基于统计学习理论中的结构风险最小化原则，通过寻找最优超平面将数据集划分为不同类别，使得不同类别之间的边界最大化，同时对于非线性支持向量机，可以使用核函数将数据映射到高维空间，解决非线性分类问题。而SVM求解最优超平面（最大间隔超平面）的问题可以化为约束优化问题，具体想法如下：

给定训练集$T=\{(\boldsymbol{x_1},y_1),(\boldsymbol{x_2},y_2),\dots(\boldsymbol{x_N},y_N)\}$，$\boldsymbol{x_i}$ 为n维特征向量，y_i 为类标记。假设超平面为$\boldsymbol{\omega x}+\boldsymbol{b}=\boldsymbol{0}$，定义超平面到样本点$(\boldsymbol{x_i},y_i)$ 的间隔为：

$$\gamma_i=y_i(\frac{\boldsymbol{\omega}}{||\boldsymbol{\omega}||}\cdot\boldsymbol{x_i}+\frac{b}{||\boldsymbol{\omega}||}) \tag{8}$$

关于所有样本点的最小间隔为：

$$\gamma = \min_{i=1,2,...N} \gamma_i \tag{9}$$

而γ就是支持向量到超平面的距离，根据以上定义，SVM模型的求解最大分割超平面问题可以表示为以下约束最优化问题：

$$\begin{aligned}
&\max_{\omega,b} \gamma \\
&s.t.\, y_i(\frac{\omega}{||\omega||}.x_i + \frac{b}{||\omega||}) \geq \gamma \\
&\iff y_i(\frac{\omega}{||\omega||\gamma}.x_i + \frac{b}{||\omega||\gamma}) \geq 1 \\
&令\ \omega = \frac{\omega}{||\omega||\gamma}, b = \frac{b}{||\omega||\gamma} \\
&\iff y_i(\omega.x_i + b) \geq 1
\end{aligned}$$

又由(8)式和(9)式可知，最大化γ等价于最小化$||\omega||$, 即最小化$\frac{1}{2}||\omega^2||$, 原式为含有不等式约束的凸二次规划问题，同时引入松弛变量ξ_i，采用Hinge损失，原问题可改写为：

$$\begin{aligned}
&\min_{\omega,b,\xi_i} \frac{1}{2}||\omega||^2 + C\sum_{i=1}^{m}\xi_i \\
&s.t.\, y_i(\omega.x_i + b) \geq 1 - \xi_i \\
&\quad \xi_i \geq 0, i = 1,2,...N \\
&\quad where\ \xi_i = max(0, 1 - y_i(\omega.x_i + b))
\end{aligned}$$

C称为惩罚参数，C越大，对分类的惩罚越大。

$$f(x) = sign\left(\sum_{i=1}^{N} \alpha_i^* y_i K(x, x_i) + b^*\right)$$

$f(x)$为分类函数，$K(x, x_i)$为核函数。

(2) 算法实践

1.PCA降维后的SVM多分类问题

```
model_svm1=svm(pre_train,label_train,kernel="radial",type='C-classification',scale=FALSE,gamma = 1/ncol(pre_train)) #scale= FALSE是因为要避免方差被标准化为1，原因在前面已阐述

##模型运行信息
model_svm1
Call:
svm.default(x = pre_train, y = label_train, scale = FALSE,
    type = "C-classification", kernel = "radial", gamma = 1/ncol(pre_train))

Parameters:
SVM-Type:  C-classification
SVM-Kernel:  radial
cost:  1
Number of Support Vectors:  1027
```

```
 #predict

 pre_svm1<-predict(model_svm1,pre_test)

 confusionMatrix(pre_svm1,as.factor(label_test))
```

```
          Reference
Prediction   0   1   2   3   4   5   6   7   8   9
         0 958   0  18   2  21   6  59   4  36  20
         1   0 669   0   0   0   0   1   1   1   7
         2   1  11 890   1   4   0   3  28   2   1
         3   0   2  23 933   0  16   5  14 212   9
         4   0   0   5   0 768   0   5   7   0  17
         5   2   2   2   9   0 633  62   0  80   1
         6   0   0   2   0   2   3 752   0   0   0
         7   0   1   3   0   2   0   0 819   4  41
         8  17 450  89  65  89 233  71 101 632 102
         9   2   0   0   0  96   1   0  54   7 811
```

Overall Statistics

```
               Accuracy : 0.7865
                 95% CI : (0.7783, 0.7945)
    No Information Rate : 0.1135
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7629
```

总准确率为78.65%，感觉可能是自己对e1071 package的用法不是很熟悉，不知道function svm() 参数可选的范围较少，导致了这个问题。所以下面经朋友建议，使用了可调函数较多的python。

2.Python 下的SVM多分类问题

```
 # data pre-processing
from google.colab import drive
import os
```

```
4
5  # Mount Google Drive
6  drive.mount('/gdrive', force_remount=True)
7
8  # Location of Zip File
9  drive_path = '/gdrive/MyDrive/imagedata.zip'
10 local_path = '/content'
11
12 # Copy the zip file and move it up one level (AKA out of the drive folder)
13 !cp '{drive_path}' .
14
15 # Navigate to the copied file and unzip it quietly
16 os.chdir(local_path)
17 !unzip -q 'imagedata.zip'
18
19 train=pd.read_csv('train.csv',sep=',',header=None)
20 test=pd.read_csv('test.CSV',sep=',',header=None)
21 label_train=pd.read_table('label_train.txt',header=None)
22 label_test=pd.read_table('label_test.txt',header=None)
23 train=pd.DataFrame(train)
24 test=pd.DataFrame(test)
25 label_train=pd.DataFrame(label_train)
26 label_test=pd.DataFrame(label_test)
27
28 import numpy as np
29 from sklearn.model_selection import train_test_split
30 ##堆叠行拼接，因为引入数据时一行代表一张图像，列代表特征
31 x=pd.concat([train,test],axis=0) ##70000张图像的特征
32 y=pd.concat([label_train,label_test],axis=0)## 70000张图像的标签
33
34 ##交叉验证,test_size=0.2的意思是测试集样本量占总样本量的20%，即14000.
35 X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
36
37 ##模型训练
38 from sklearn import svm
39 from scipy import stats
40 from sklearn.metrics import accuracy_score
41 import matplotlib as mpl
42 import matplotlib.pyplot as plt
43 from sklearn import metrics
44 model
   =svm.SVC(kernel='rbf',gamma=1/64,probability=True,random_state=0,class_weight='balanced',dec
   ision_function_shape='ovo')
45
46 y_test_pred = clt.predict(X_test)
47 ov_acc = metrics.accuracy_score(y_test_pred,y_test)
48 print(ov_acc) #0.9780714285714286
49
50 #可视化实现
51 from sklearn.metrics import confusion_matrix
52 print("Confusion matrix:")
53 print(confusion_matrix(y_test,y_test_pred))
54
55 Confusion matrix:
56
57 [[1375    0    2    0    1    2    5    0    2    0]
58
```

```
[   0 1569    4    1    2    0    0    2    1    1]

[   2    2 1415    1    3    0    2    8    8    2]

[   0    0   15 1384    0   16    0    5   13    2]

[   1    1    3    0 1324    1    2    4    0   14]

[   1    1    3    8    1 1198   10    0    8    1]

[   5    1    1    0    6    4 1368    0    2    0]

[   0    6   13    0   11    1    0 1414    1   12]

[   1    9    3    4    1    6    4    0 1332    8]

[   2    0    1   11   15    5    1    8    4 1314]]
```

准确率为97.8%，相比上述采用PCA降维后再进行SVM分类的方法，未降维进行SVM分类的准确率明显提高。可能是因为上述采用PCA降维后主成分选取个数不太对，所以特征损失部分后对准确率的影响比较大。

另外，与课件的准确率98.46%比较略低，比较参数设置时发现应该是惩罚系数的原因，在默认为cost=1的情况下可能准确率确实会比cost=4的时候低。然后因为在python中利用train_test_split函数进行测试集和训练集的选取，样本量的选取方式也会对准确率造成影响。

可以优化的点：如何选取最优参数的设置(cost+gamma) 2. 还有没有其他提取特征的方法?

此部分可稍后稍作讨论。

4. 神经网络

(1) 理论介绍

主要使用python中的tensorflow进行深度学习，在Keras中可以通过组合层来构建模型。

过程如下：

(a) 创建 Sequential模型

(b) 添加所需要的神经层

(c) 使用.compile方法确定模型训练结构,配置学习流程；通俗来讲就是配置各种评估指标来保证学习效率比

(d) 使用.fit方法，使模型与训练集拟合

(e) predict方法进行预测

相关概念如下：

神经网络的结构

(a) tf.keras.layers.Flatten():拉直层，**变换张量的尺寸**，把输入特征拉直为一维数组，是不含计算参数的层

(b) tf.keras.layers.Dense():全连接层，**起到“特征提取器”的作用**，将学到的特征表示映射到样本的标记空间，

(c) tf.keras.layers.Dropout() : 防止过拟合，提高模型的泛化能力。

rate：0~1之间的小数。让神经元以一定的概率rate停止工作，提高模型的泛化能力。

(d) tf.keras.layers.Conv2D():卷积层

神经网络的参数解释

(a) Unit(神经元)：接受输入数据（input data),然后有一个输出，其中每个输入值都有一个**权重(weight)**，表明其重要性。

(b) Activation(激活函数):非线性函数，对输入到神经元的数据进行“激活”，从而输出是非线性的数据

常用的激活函数有Sigmoid,tanh,ReLU 等

(c) loss function: 用于衡量真实值与预测值的不一致程度。Cross-Entropy Loss 是非常重要的损失函数，也是应用最多的损失函数之一，用于计算多分类问题的交叉熵。其余常见损失函数还有Logistics Loss，Softmax Loss等等。

(d) Optimizer:用于更新模型中可学习的参数。常见的有BGD,SGD等，这里采用的是Adam(**Adaptive Moment Estimation**) 可以看做使用动量的滑动平均，适用于大数据集和高维空间。

下面分别使用NN和CNN进行尝试

(2) 模型搭建

1.Neural network model

对于模型搭建的理解附在注释里

```
#import necessary libraries
import tensorflow as tf

#load training data and split into train and test sets
mnist = tf.keras.datasets.mnist

(x_train_mnist,y_train_mnist), (x_test_mnist,y_test_mnist) = mnist.load_data()
x_train_mnist, x_test_mnist = x_train_mnist / 255.0, x_test_mnist / 255.0

#define model

model = tf.keras.models.Sequential([
    # flatten the input image (which is a 28x28 pixel grayscale image in this case) into a 1-dimensional array of 784 values. This is required because the subsequent layers in the model expect a 1-dimensional input.
                    tf.keras.layers.Flatten(input_shape=(28,28)),

    # Adds a densely-connected layer with 128 units and ReLu Activation to the model
    #The dense layer performs a matrix multiplication of the input with a set of weights and
```

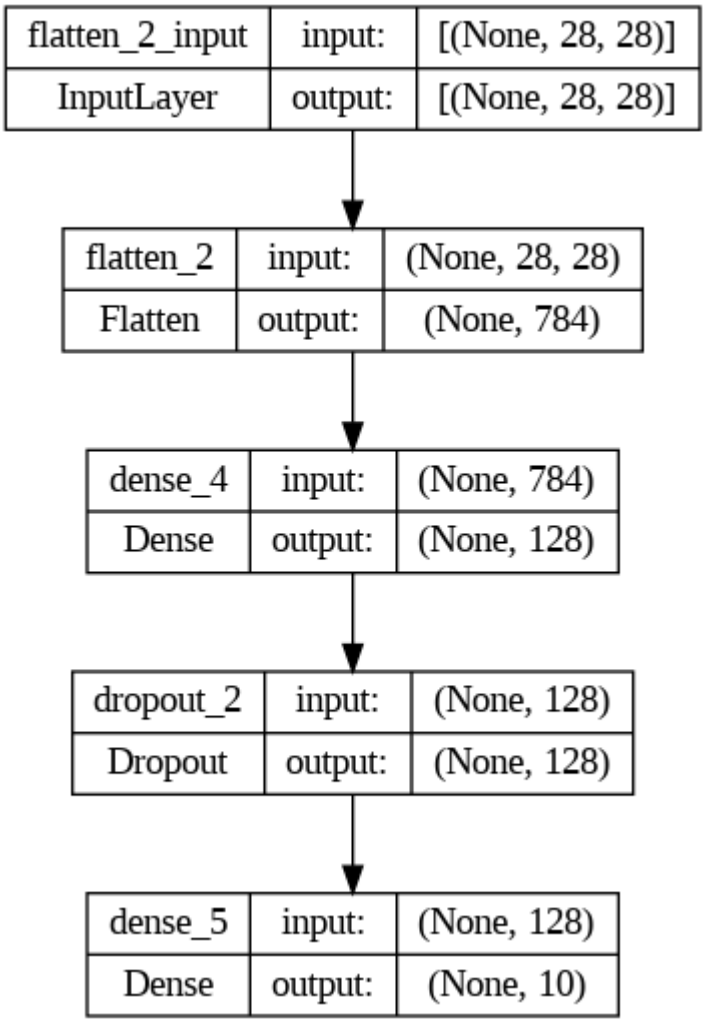
```
    biases, and the activation function applies a non-linear transformation to the output
                    tf.keras.layers.Dense(128,activation='relu'),

                    tf.keras.layers.Dropout(0.2),

    # Add a layer with 10 output units without activation.This layer corresponds to the output layer of the model, and since no activation function is specified, it will output raw scores for each of the 10 possible classes.
                    tf.keras.layers.Dense(10)
])

#define loss function
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

#define optimizer,loss function and evaluation metric
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])

#train the model
model.fit(x_train_mnist,y_train_mnist,epochs=5)
#test the model
#The verbose parameter controls the verbosity of the evaluation output (0 = silent, 1 = progress bar, 2 = one line per epoch).
model.evaluate(x_test_mnist,y_test_mnist,verbose=2)
#extend the base model to predict softmax output
##The resulting probability_model can be used to make predictions on new input data, and then applying the softmax function to obtain the predicted class probabilities.
probability_model = tf.keras.Sequential([model,tf.keras.layers.Softmax()])
```

模型训练结果如下：

```
Epoch 1/5
1875/1875 [==============================] - 6s 2ms/step - loss: 0.2997 - accuracy: 0.9124
Epoch 2/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1464 - accuracy: 0.9563
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1114 - accuracy: 0.9660
Epoch 4/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0906 - accuracy: 0.9718
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0786 - accuracy: 0.9764
<keras.callbacks.History at 0x7f4d47db6530>
```

```
313/313 - 1s - loss: 0.0738 - accuracy: 0.9765 - 656ms/epoch - 2ms/step
```

测试集上的准确率为97.65%

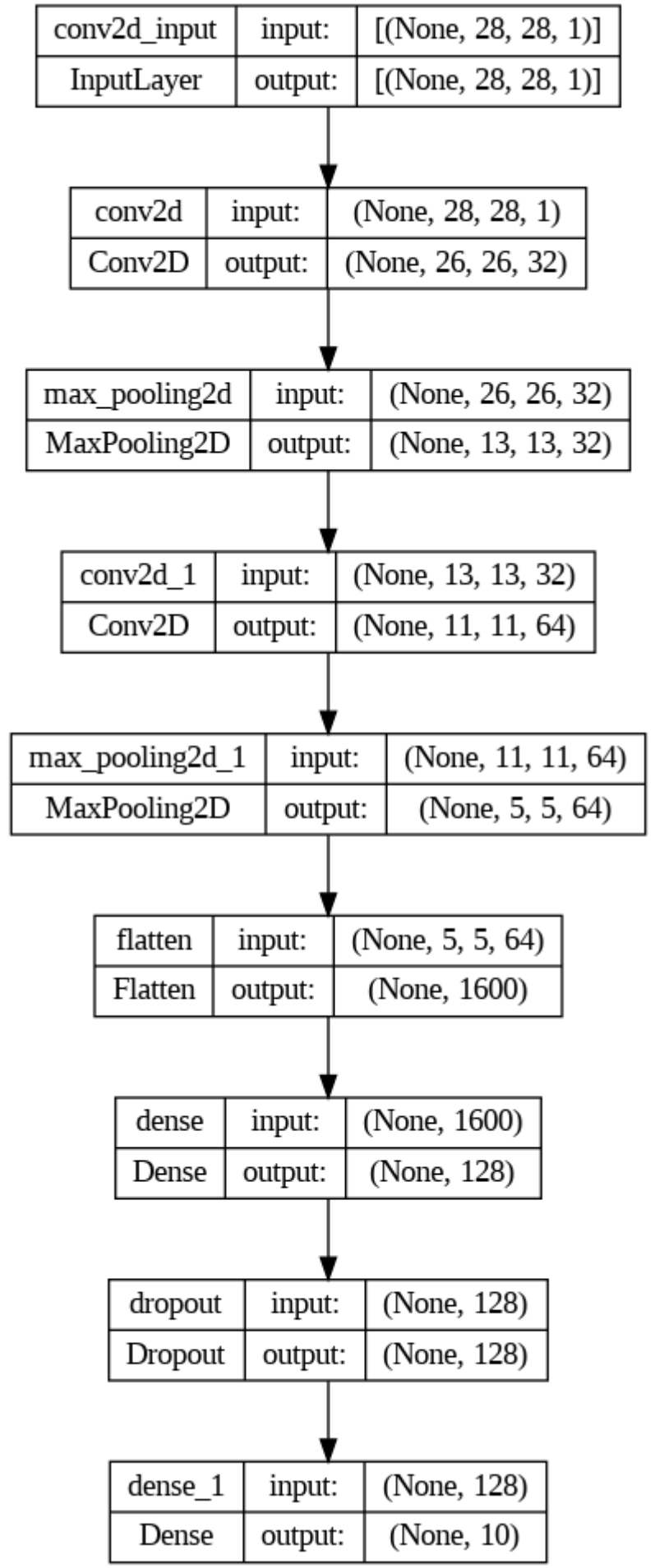


2. Convolutional Neural Network

```
import tensorflow as tf

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

# Reshape input data to 4D tensor with shape (batch_size, height, width, channels)
# to be compatible with the convolutional layers
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

# Define the CNN model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(28,28,1)),
    #Maxpooling layers:downsamples the output of the convolutional layer by taking the maximum value over a local region.
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),

    tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
```

```
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    #The output of the second max pooling layer is then flattened into a 1D vector using the tf.keras.layers.Flatten() function.
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    #The final dense layer has 10 neurons (since there are 10 possible classes in the MNIST dataset) and uses the softmax activation function to output class probabilities.
    tf.keras.layers.Dense(units=10, activation='softmax')
])

# Compile the model with categorical cross-entropy loss and Adam optimizer
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model for 5 epochs
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
# plot the progress of the model
plot_model(model, to_file='cnn_model.png', show_shapes=True)
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

代码共设置了两层卷积层和两层池化层，主要是为了提高模型的适用性，可以学习更多的数据特征；同时增加特征提取量，也就是说第二层的卷积层可以学习来自前面一层的更为抽象和复杂的特征，增加准确率。


conv2d_input | input: | [(None, 28, 28, 1)]
InputLayer | output: | [(None, 28, 28, 1)]
conv2d | input: | (None, 28, 28, 1)
Conv2D | output: | (None, 26, 26, 32)
max_pooling2d | input: | (None, 26, 26, 32)
MaxPooling2D | output: | (None, 13, 13, 32)
conv2d_1 | input: | (None, 13, 13, 32)
Conv2D | output: | (None, 11, 11, 64)
max_pooling2d_1 | input: | (None, 11, 11, 64)
MaxPooling2D | output: | (None, 5, 5, 64)
flatten | input: | (None, 5, 5, 64)
Flatten | output: | (None, 1600)
dense | input: | (None, 1600)
Dense | output: | (None, 128)
dropout | input: | (None, 128)
Dropout | output: | (None, 128)
dense_1 | input: | (None, 128)
Dense | output: | (None, 10)

输出结果如下：

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
Epoch 1/5
1875/1875 [==============================] - 29s 7ms/step - loss: 0.2083 - accuracy: 0.9366 - val_loss: 0.0478 - val_accuracy: 0.9840
Epoch 2/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0788 - accuracy: 0.9768 - val_loss: 0.0333 - val_accuracy: 0.9883
Epoch 3/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0578 - accuracy: 0.9822 - val_loss: 0.0250 - val_accuracy: 0.9917
Epoch 4/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0483 - accuracy: 0.9860 - val_loss: 0.0281 - val_accuracy: 0.9898
Epoch 5/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0376 - accuracy: 0.9883 - val_loss: 0.0243 - val_accuracy: 0.9919
313/313 [==============================] - 1s 2ms/step - loss: 0.0243 - accuracy: 0.9919
Test accuracy: 0.9919000267982483
```

准确率为99.19%

四、模型优化调整思路

1. 特征提取优化：SIFT算法

该优化方案由一位在港科大读计算数学的朋友提供，她尝试过没有特征提取过程，PCA降维，SIFT算法提取特征三种处理方式进行SVM多分类训练。其中PCA方法因为运行时间过长被舍弃，最后只比较了没有特征提取和用了SIFT算法提取的SVM分类问题。可以从下图看到，SIFT准确率确实最佳。(dataset: https://www.kaggle.com/datasets/imonbilk/industry-biscuit-cookie-dataset)

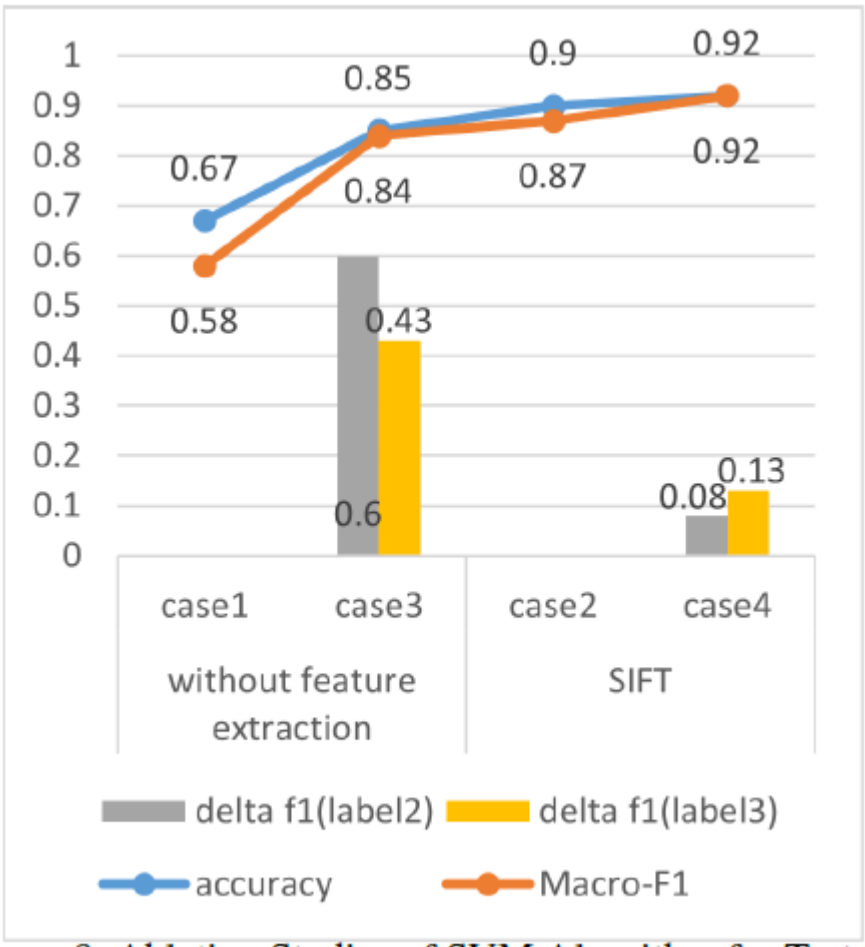


Figure 9. Ablation Studies of SVM Algorithm for Test Set

(1) Introduction

本来也想试一下用完SIFT之后再进行SVM多分类。可惜的是代码途中一直报错且未查找出原因，故只能简单描述下大致的流程。

关键步骤为：

a) Scale-space extrema detection : 在不同尺度上检测图像中的局部极值。这是通过在不同尺度上使用高斯滤波器对图像进行卷积，然后通过对卷积后得到的图像进行差分，来获得尺度空间中的表示方法。

b) Keypoint localization： 通过将二次函数拟合到尺度空间极值并丢弃对比度低或边缘对齐差的关键点来完成的

c) Orientation assignment： 根据局部邻域中主导梯度方向为每个关键点分配方向

d) Keypoint descriptor： 计算每个关键点的描述符，该描述符基于其局部邻域中的梯度大小和方向。描述符是一个固定长度的向量，编码局部特征的外观和空间布局。

(2) Progress

当在图像中提取中SIFT特征后就可以将其作为SVM分类器的输入进行预测。在进行预测前需要保证SIFT特征向量拥有相同的长度，可以用"词袋"的办法(bag of visual words). 先使用K-means 聚类的办法将每一类图片的SIFT特征聚类为K类，构成该类的visual vocabulary；对于训练集中的每一张图片，统计vocabulary中K个word的“词频”，得到相应的直方图，将直方图作为样本向量即可构建SVM的训练数据和测试数据。

2. 通过交叉验证选取SVM最优参数

在R的e1071 package中发现有函数tune.svm可以做svm的最优参数选择，损失惩罚函数C以及核函数的参数都是支持向量机中的重要参数。可通过交叉验证的方式确定参数。tune.svm函数可自动实现10折交叉验证，并给出预测误差最小时的参数值。

利用k折交叉验证进行参数选择的原理是：将C和gamma按一定步长进行取值组合，在不同的(C，gamma)组合下，将训练集样本分成k组，一组作为验证的数据样本，其余k-1组则用来作为训练的数据，每一组数据都轮流着作为验证数据样本，这样在一组（C,gamma）组合下，就需要进行k次计算，把这k次计算的模型测试准确率score的均值作为这组（C,gamma）下模型的得分。这样的话就能够得到不同（C，gamma）组合下模型的得分，取得分最高的那组（C,gamma）即可，如果碰到有相同分的，一般考虑参数C,取C小的，因为在保证模型的准确率的情况下，C越小模型容错率大，可以避免过拟合，若C也相同，则取先出现的那一组gamma

由于数据量过大，在运行6小时后仍未出结果，于是将数据量从60000缩减到5000；可能会有准确率上比较明显的下降。

```
set.seed(4)
pre_train1=pre_train[1:5000,]
label_train1=label_train[1:5000]

length(pre_train1)
rbf.tune<-tune.svm(pre_train1,label_train1,
                        kernel = "radial",
                        gamma = c(1/64,0.1,0.5,3),
                        cost = c(0.1,2,3,4))
rbf.tune$best.performance    ##loss:0.0394
rbf.tune$best.parameters #gamma=0.015625=1/64,cost=4;
best_rbf<-rbf.tune$best.model #最佳模型
tune.test<-predict(best_rbf,newdata = pre_test[1:5000,])
table(tune.test,label_test[1:5000])
confusionMatrix(as.factor(tune.test),as.factor(label_test[1:5000]))
```

结果如下：

```
rbf.tune$performances
      gamma cost  error  dispersion
1  0.015625  0.1 0.0856 0.015826841
2  0.100000  0.1 0.7898 0.026856822
3  0.500000  0.1 0.8922 0.016369348
4  3.000000  0.1 0.8934 0.015973589
5  0.015625  2.0 0.0408 0.009150592
6  0.100000  2.0 0.1872 0.030069549
7  0.500000  2.0 0.8072 0.022943409
8  3.000000  2.0 0.8918 0.016067911
9  0.015625  3.0 0.0402 0.009114092
10 0.100000  3.0 0.1872 0.030069549
11 0.500000  3.0 0.8072 0.022943409
12 3.000000  3.0 0.8918 0.016067911
13 0.015625  4.0 0.0394 0.008694826
14 0.100000  4.0 0.1872 0.030069549
15 0.500000  4.0 0.8072 0.022943409
16 3.000000  4.0 0.8918 0.016067911
```

可知当gamma=1/64,cost=4的时候error最小，此时为最优参数。

利用最佳模型进行预测和准确率测试结果为：

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2   3   4   5   6   7   8   9
         0 434   0   2   0   0   2   9   0   2   5
         1   0 542   0   0   0   0   2   4   0   0
         2   0   2 449   1   1   1   3  12   1   3
         3   0   3  18 469   0  44   2  13  25   8
         4   1   0   2   0 396   1   5   3   1   3
         5   1   0   0   0   0 208   1   0   0   0
         6   4   0   1   0   6   4 419   0   0   0
         7   0   0   2   1   0   0   1 387   0   1
         8  17  24  54  28  41 190  19  24 460  48
         9   3   0   2   1  56   6   1  69   0 452

Overall Statistics

               Accuracy : 0.8432
                 95% CI : (0.8328, 0.8532)
    No Information Rate : 0.1142
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8257

 Mcnemar's Test P-Value : NA
```

准确率为84.32%。相比之前60000的数据量下PCA+rbf_SVM 78%的准确率，可以说是提升了。

五、致谢

Reference book:

[1]Gareth James; Daniela Witten; Trevor Hastie; Robert Tibshirani.An Introduction to Statistical Learning: with Applications in R[M].Springer New York, NY.2021.

[2]Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning

https://neptune.ai/blog/how-to-use-google-colab-for-deep-learning-complete-tutorial

https://medium.com/swlh/setting-up-google-colab-for-cnn-modeling-55b5208599c4

https://zhuanlan.zhihu.com/p/31886934

https://blog.csdn.net/u012874151/article/details/45457085

此外，感谢所有朋友在本人代码遇到困难时提供的无偿帮助。

特别鸣谢：一位不愿意透露姓名的好友分享很多学习网站和帖子，受益匪浅

感谢Frey的精神支持