模式识别与智能系统系列课程

课程总结报告

学　　号：19053022

姓　　名：沈万成

日　　期：2020 年 7 月

第一部分 课程学习收获

从感知机到支持向量机

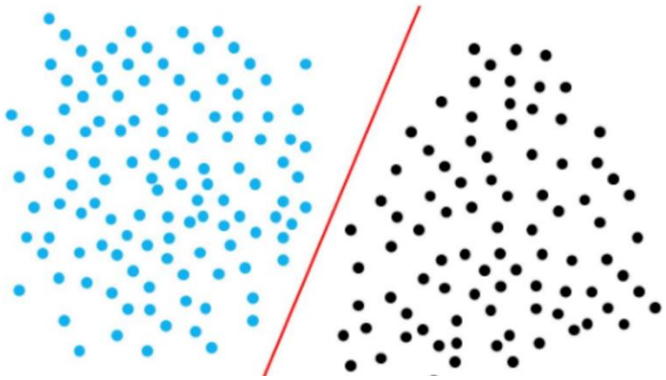

图 1 线性可分数据集示意图

线性可分数据集是指在数据集的某个特征空间中，存在一个超平面 S（线性超平面）可以把数据集中的正样本和负样本完全分开的数据集，这个超平面就可以认为是分类决策面，线性分类器的目的就是找到这个决策面。感知机和支持向量机都是非常经典的二分类器，这两种算法的思想都是在特征空间中寻找划分平面对数据集进行划分，但寻找划分平面的算法不同。

1. 感知机（perceptron）

1.1 感知机模型

感知机在 1957 年由 Rosenblatt 提出，是神经网络和支持向量机的基础。

假设数据集输入特征空间是$\mathcal{X} \subseteq \mathbb{R}^n$，n 为特征维数，输出空间是$\mathcal{Y} = \{+1, -1\}$。设$x = \{x_1, x_2, \cdots, x_n\} \in \mathcal{X}$是数据集中的一个样本的特征向量，输出$y \in \mathcal{Y}$表示它的类别，感知机的分类决策函数为：

$$
\begin{aligned}
& f(x) \\
& = sgn((\sum_{i=1}^{n} w_i x_i) - threshold) \\
& = sgn((\sum_{i=1}^{n} w_i x_i) + \underbrace{(-threshold)}_{w_0}\underbrace{(+1)}_{x_0}) \\
& = sgn(\sum_{i=0}^{n} w_i x_i) \\
& = sgn(w^T x)
\end{aligned}
\tag{1.1}
$$

ω（包含了偏置）是感知机的模型参数，不同的ω对应于特征空间中不同的决策超平面，怎样找到最好的参数ω就涉及到感知机的学习策略。

1.2 感知机学习策略

感知机的学习算法 Perceptron Learning Algorithm(PLA)采用“逐点修正”的方法。

输入：

训练数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$，其中，$x_i \in \mathcal{X}=\mathbb{R}^n$，$y_i \in \mathcal{Y}=\{-1,+1\}$，$i=1,2,\cdots,N$；

学习率 $\eta(0<\eta\leq 1)$

输出：

ω；感知机决策函数 $f(x)=sgn(w^T x)$

学习过程：

（1）初始化ω

（2）在训练集中选取一个样本(x_i,y_i)；

（3）如果$y_i(w^T x_i)\leq 0$，参数更新：

$$\omega_{t+1} \leftarrow \omega_t + \eta y_i x_i$$

（4）转至（2），遇到分类错误的样本就修正参数，直到所有的样本都被正确分类。

对上述这种学习策略进行直观的解释:

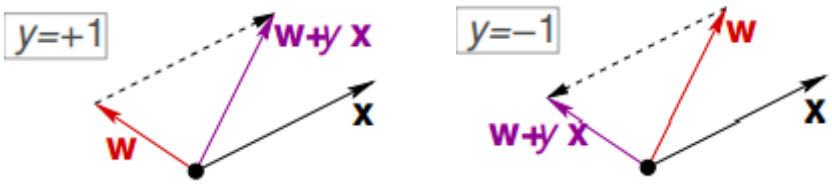

图 2 PLA 示意图

首先随机选择一条直线进行分类。然后找到第一个分类错误的点，如果这个点表示正类，被误分为负类，即 $w_t^T x_{n(t)} < 0$，那表示 ω 和 x 夹角大于 90 度，其中 ω 表示直线的法向量。所以，x 被误分在直线的下侧（相对于法向量，法向量的方向即为正类所在的一侧），修正的方法就是使 ω 和 x 夹角小于 90 度。通常做法是 $\omega_{t+1} \leftarrow \omega_t + y_{n(t)} x_{n(t)}$，$y_i$=1，如上图所示，一次或多次更新后的 $\omega + yx$ 与 x 夹角小于 90 度，能保证 x 位于直线的上侧，则对误分为负类的错误点完成了直线修正。

对于误分为正类的点做类似的处理，按照这种思想，使分离超平面向误分类点一侧移动，以减少误分类点与超平面的距离，直到超平面越过误分类点使其被正确分类。

1.3 PLA 的收敛性

PLA 的“逐点修正”方法有可能导致算法的不收敛，所以要证明经过有限次的迭代可以得到一个将训练数据完全正确分类的分类超平面和感知机模型。

对于线性可分的情况，存在一个完美权重 ω_f，使得 $y_n = sign(w_f^T x_n), \forall x_n \in X$，对于任意训练样本，有

$y_{n(t)}{w_f}^T x_{n(t)} \geq \min_n y_n {w_f}^T x_n > 0$，在 PLA 的更新过程中，若 ω_{t+1} 与 ω_f 越来越近，即 $<\omega_{t+1}, \omega_f>$ 越来越大，则证明 PLA 是有学习效果的。

$$
\begin{aligned}
&<\omega_{t+1}, \omega_f> \\
&={\omega_f}^T \bullet \omega_{t+1} \\
&={\omega_f}^T (\omega_t + y_{n(t)} x_{n(t)}) \\
&\geq {\omega_f}^T \bullet \omega_t + \min_n y_n {w_f}^T x_n \\
&> {\omega_f}^T \bullet \omega_t + 0 =<\omega_t, \omega_f>
\end{aligned}
\tag{1.2}
$$

所以 PLA 是有学习效果的

由公式（1.2），再令 $\rho = \min_n \frac{y_n w_f x_n}{\left\| w_f^T \right\|}, R = \max_n \left\| x_n \right\|$，由 Novikoff 定理可知，PLA 在训练集上的误分类次数 T 满足：$T \leq \left(\frac{R}{\rho} \right)^2$，所以经过有限次的搜索 PLA 可以找到将训练数据完全正确分开的分离超平面。

1.4 线性不可分数据集下的感知机

上一部分证明了在线性可分的情况下，PLA 是可以停下来并正确分类的，但对于非线性可分的情况，w_f 实际上并不存在，那么之前的推导并不成立，PLA 不一定会停下来。所以，PLA 虽然实现简单，但也有缺点，对于非线性可分的情况，我们可以把它当成是数据集中掺杂了一些噪声。

在非线性情况下，可以把学习的条件放松，即不苛求每个点都分类正确，而是容忍有错误点，取错误点的个数最少时的权重 w，这种感知机学习算法称为 Packet Algorithm，它的算法流程与 PLA 基本类似。

（1）初始化权重 w_0，计算出在这条初始化的直线中，分类错误点的个数。

（2）对错误点进行修正，更新 w，得到一条新的直线，再计算其对应的分类错误的点的个数，并与之前错误点个数比较，取个数较小的直线作为当前选择的分类直线。

（3）再经过 n 次迭代，不断比较当前分类错误点个数与之前最少的错误点个数比较，选择最小的值保存。直到迭代次数完成后，选取个数最少的直线对应的 w，即为我们最终想要得到的权重值。

1.5 感知机的代码实现

1、验证 PLA 和 pocket_PLA 算法在线性可分数据集上的执行时间和效果

1.1 设计简单的线性可分数据集（二维）

先指定一条直线-2+x1+2*x2=0,然后再直线两侧各取 10 个点作为训练样本，所取数据如下所示：

```
1   0.0 2.0 1
2   1.0 1.0 1
3   1.0 2.0 1
4   0.0 0.0 -1
5   1.0 0.0 -1
6   0.5 -1.3    -1
7   2.0 1.0 1
8   2.0 2.0 1
9   3.0 0.0 1
10  1.5 -0.5    -1
11  2.0 -1  -1
12  3.0 1.0 1
13  -0.5 1  -1
14  1 -1    -1
15  3.0 2.0 1
16  3.0 2.1 1
17  2.5 2.5 1
18  1.7 -0.5    -1
19  1.5 -1  -1
20  0.3 -0.3    -1
```

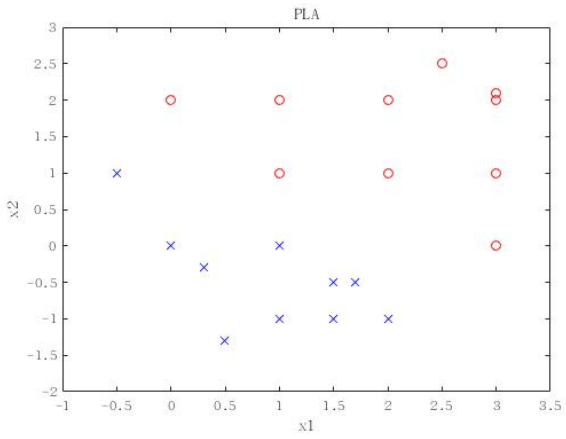

图 3 训练数据

1.2 使用 PLA 算法对训练数据进行分类

核心代码：

```
//PLA
void PLA(){
    int correctNum = 0;//当前正确样本数，=n 说明全部正确
    int index = 0;//当前正在计算第几个样本
    bool isFinished = 0;//结束标志
    while(isFinished == 0){
        if(trainingSet[index].output == sign(mult(weight,trainingSet[index].input,DEMENSION))) //sign(w*x)
            correctNum++;//当前样本无错，连续正确样本数+1
        else{           //correct mistake
            double temp[DEMENSION];
            mult(temp,trainingSet[index].input,DEMENSION,trainingSet[index].output);//y*x
            add(weight,temp,DEMENSION);//w(t+1) = w(t) + y*x
            step++;
            correctNum = 0;//出错后，正确数归零
            cout << "step" << step << ":" << endl
                 << "index = " << index << " is wrong." << endl;
        }
        if(index == n-1)
            index = 0;
        else
            index ++;
        if(correctNum == n) isFinished = 1;
    }
    cout << "Total step :" << step << endl;
}
```

图 4 PLA 训练 C++实现

算法结果：

Total step : 9

weights: [-3 2 2]

在原始数据中画出分类线，查看分类效果：

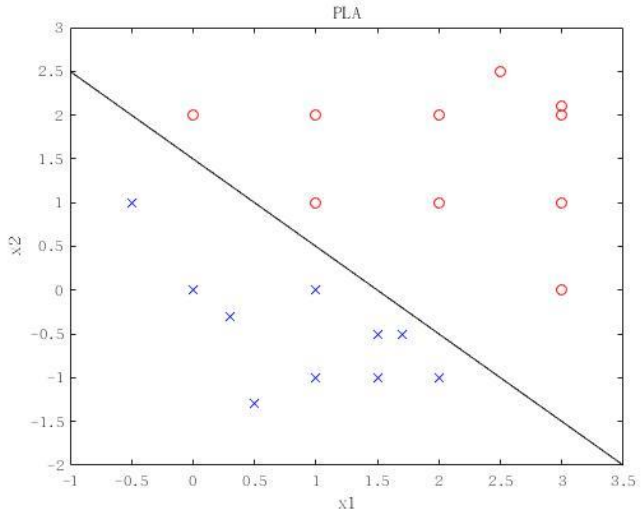

图 5 感知机分类效果

1.3 使用 pocket_PLA 算法对训练数据进行分类

核心代码：

```
//pocket_PLA
void pocketPLA(double *pocketweights,double *weights,int iteration){
    int index = 0;
    int iter= 1;
    int n = trainingSet.size();         //数据个数
    while(iter < iteration){             //iteration 迭代次数
        if(sign(multiply(trainingSet[index].x,weights)) != trainingSet[index].y){ //mistake
            double temp[DEMENSION];
            multiply(temp,trainingSet[index].x,trainingSet[index].y);              //xy
            for(int i=0;i<DEMENSION;i++)
                weights[i] += temp[i];  //w(t+1) = w(t) + y*x
            if(getErrorNum(weights,trainingSet) < getErrorNum(pocketweights,trainingSet)){
                for(int j = 0;j<DEMENSION;j++)
                    pocketweights[j] = weights[j]; //replace weights
            }
            iter++;
        }
        if(index == n-1)
            index = 0;
        else
            index++;
    }
}
```

图 6 pocket_PLA 训练 C++实现

设定迭代次数和 PLA 相同，即 9 次，算法结果：

final 错误个数: 1

pocket_weights: [-2 0.3 2]

不断增大迭代次数，直到最后的错误个数为 0（数据集线性可分），得到迭代次数最少为 12

final 错误个数: 0

pocket_weights: [-3 2.3 3]

分类效果：

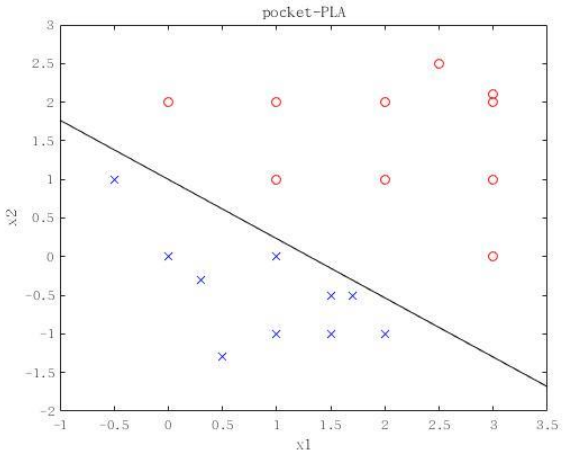

图 7 pocket_PLA 分类效果

1.4 算法比较

因为 pocket_PLA 迭代次数比 PLA 多，而且每次 pocket_PLA 迭代都要遍历数据集的所有点，显然 pocket_PLA 的执行时间比 PLA 多得多；

算法效果比较：通过比较两种算法求得的分类直线，离分类直线最近的点的距离，PLA 的更大一点，所以 PLA 的分类效果较 pocket_PLA 更好。

2.验证 pocket_PLA 算法在非线性可分数据集上的效果

2.1 设计简单的非线性可分数据集（二维)

对第一步的数据集做简单修改，添加一个不可二分的点：

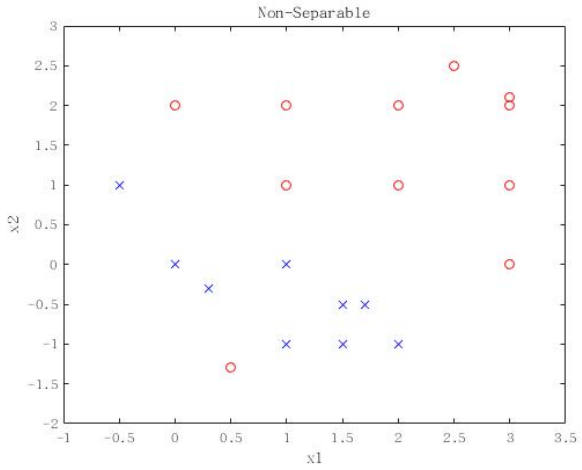

图 8 非线性数据集

2.2 算法结果

当算法错误个数最小时停止迭代：

min_step:17

final 错误个数: 1

pocket_weights: [-4 1.4 3.4]

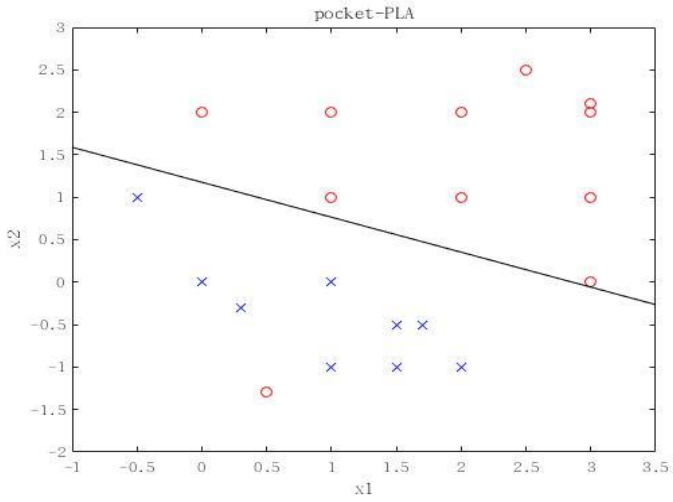

图 9 pocket_PLA 分类结果

2 支持向量机（SVM）

支持向量机也是在特征空间中寻找一个划分样本实例类别的超平面，这点和感知机相似，但是不同的是，支持向量机寻找出的超平面唯一且最优，而感知机是根据误分类点定义出的代价函数求得的超平面不唯一，包含多个，而且支持向量机支持在特征空间中寻找出非线性的平面(非线性支持向量机)。

感知机算法的思想是寻找拥有较少的错分训练样本数的判别函数未，但是这个判别函数未必就是一个好的判别函数，SVM 的思想是最大化分类间隔，使两类样本到分类面的最短距离之和最大。

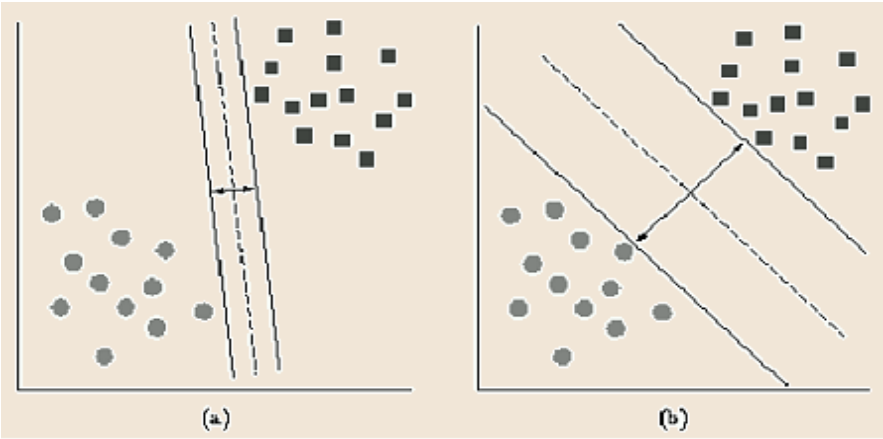

图 10 分类间隔示意图

2.1 线性支持向量机的数学模型

给定训练样本 $S=\{(x_i,y_i)\}_{i=1}^{n}, x_i \in \mathrm{X} \subseteq \mathbb{R}^d, y_i \in Y=\{-1,+1\}$

设所求的分类面表达式为：$g(x)=\tilde{w}^T x+\tilde{b}$，该分类面若能将训练样本线性分开，则：

$$\begin{cases} \tilde{w}^T x_i+\tilde{b}>0\ ,\ y_i=+1 \\ \tilde{w}^T x_i+\tilde{b}<0\ ,\ y_i=-1 \end{cases} \tag{2.1}$$

即：

$$y_i(\tilde{w}^T x_i + \tilde{b}) > 0 \tag{2.2}$$

上式是在无限个样本下成立的，在有限个样本下，存在$\varepsilon > 0$使得$y_i(\tilde{w}^T x_i + \tilde{b}) \geq \varepsilon$ ，即

$$y_i(w^T x_i + b) \geq 1 \ , \forall i \tag{2.3}$$

其中， $w = \frac{\tilde{w}}{\varepsilon}, b = \frac{\tilde{b}}{\varepsilon}$

样本x_i到分类面(w,b)的距离为：

$$\frac{\left| w^T x_i + b \right|}{\|w\|} = \frac{y_i\left(w^T x_i + b \right)}{\|w\|} \tag{2.4}$$

样本集 S 到分类面(w,b)的距离ρ为：

$$\rho = \min_{(x_i, y_i) \in S} \frac{y_i\left(w^T x_i + b \right)}{\|w\|} = \frac{1}{\|w\|} \tag{2.5}$$

因为 SVM 的思想是最大化分类间隔，所以优化目标为：

$$\begin{cases} \max\limits_{w,b} \dfrac{1}{\|w\|} \\ s.t. \ \ y_i(w^T x_i + b) \geq 1 \ , \forall i \end{cases} \tag{2.6}$$

对公式(2.6)的优化目标进一步改善，得到：

$$\begin{cases} \min\limits_{w,b} \dfrac{1}{2}\|w\|^2 \\ s.t. \ \ y_i(w^T x_i + b) \geq 1 \ , \forall i \end{cases} \tag{2.7}$$

公式(2.7)就是利用线性 SVM 求解线性分类面本质上需要求解的优化问题，这是一个典型的凸优化问题。

2.2 线性支持向量机的求解

（1）构造 Lagrange 函数

$$L(w,b;\alpha)=\frac{1}{2}\|\mathrm{w}\|^2-\sum_{i=1}^{n}\alpha_i[y_i(w^Tx_i+b)-1] \quad (\le\frac{1}{2}\|\mathrm{w}\|^2) \tag{2.8}$$

其中，$\alpha=[\alpha_1,\alpha_2,\cdots,\alpha_n]^T,\alpha_i\ge 0$为 Lagrange 乘子。

（2）KKT 条件

$$\begin{cases}\nabla_w L=w-\sum_{i=1}^{n}\alpha_i y_i x_i=0 & \Rightarrow\ w=\sum_{i=1}^{n}\alpha_i y_i x_i\\ \nabla_b L=-\sum_{i=1}^{n}\alpha_i y_i=0 & \Rightarrow\ \sum_{i=1}^{n}\alpha_i y_i=0\\ \forall i,\alpha_i[y_i(w^Tx_i+b)-1]=0 & \Rightarrow\ \alpha_i=0 \text{ 或 } y_i(w^Tx_i+b)=1\end{cases} \tag{2.9}$$

将(2.9)代入(2.8)化简：

$$\begin{aligned}L(w,b;\alpha)&=\frac{1}{2}\|\mathrm{w}\|^2-\sum_{i=1}^{n}\alpha_i[y_i(w^Tx_i+b)-1]\\&=\frac{1}{2}\|\mathrm{w}\|^2-\sum_{i=1}^{n}\alpha_i y_i w^Tx_i-\sum_{i=1}^{n}\alpha_i y_i b+\sum_{i=1}^{n}\alpha_i\\&=\frac{1}{2}\|\mathrm{w}\|^2-w^T\sum_{i=1}^{n}\alpha_i y_i x_i+\sum_{i=1}^{n}\alpha_i\\&=-\frac{1}{2}\|\mathrm{w}\|^2+\sum_{i=1}^{n}\alpha_i\\&=-\frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^Tx_j+\sum_{i=1}^{n}\alpha_i\end{aligned} \tag{2.10}$$

（3）转换为对偶问题求解

原始问题为：$\min_{\alpha}(\max_{w,b}L(w,b;\alpha))$

其对偶问题为：$\max_{\alpha}(\min_{w,b}L(w,b;\alpha))$，即：

$$\begin{cases}\max\limits_{\alpha} & -\frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j(x_i^Tx_j)+\sum_{i=1}^{n}\alpha_i\\ s.t. & \sum_{i=1}^{n}\alpha_i y_i=0,\ \alpha_i\ge 0,\forall i\end{cases} \tag{2.11}$$

为了方便求解，(2.11)的等价形式为：

$$\begin{cases} \min\limits_{\alpha} \ \dfrac{1}{2}\sum\limits_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j (x_i^T x_j) - \sum\limits_{i=1}^{n}\alpha_i \\ s.t. \ \ \sum\limits_{i=1}^{n}\alpha_i y_i = 0, \ \alpha_i \geq 0, \forall i \end{cases} \tag{2.12}$$

由(2.9)可得到：

$$\begin{cases} w = \sum\limits_{i=1}^{n}\alpha_i y_i x_i \\ b = y_i - \sum\limits_{j=1}^{n}\alpha_j y_j (x_j^T \bullet x_i) \end{cases}, \alpha_i > 0 \tag{2.13}$$

并且有 $\forall i, \alpha_i[y_i(w^T x_i + b) - 1] = 0$，对于取值不为 0 的 α_i，它将使得 $y_i(w^T x_i + b)=1$，这样的样本就是支持向量，所以 SVM 的解的表达式可以重写为：

$$w = \sum_{i=1}^{l} y_i \alpha_i x_i \Rightarrow w = \sum_{\text{支持向量}} y_i \alpha_i x_i \tag{2.14}$$

只要求解(2.12)这个 QP 问题，得到 $\alpha=[\alpha_1, \alpha_2, \cdots, \alpha_n]^T$，代入(2.13)，便可得到 SVM 的分类超平面及决策函数

$$f(x) = \text{sgn}(\sum_{\text{支持向量}} \alpha_i \, y_i x_i^T \bullet x + b) \tag{2.15}$$

2.3 线性不可分的情况

（1）数据集中含有少量非线性可分样本的情况

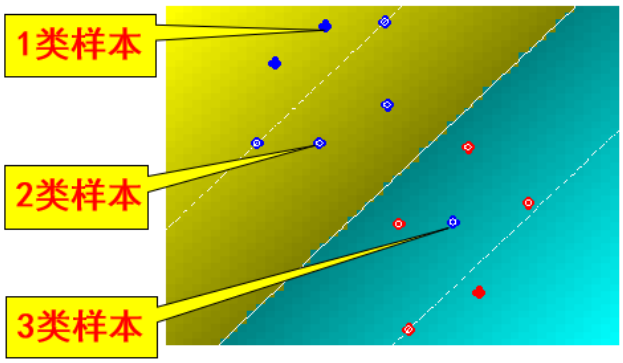

图 11 含有少量非线性可分数据的数据集

在含有少量非线性可分数据的数据集中，含有三类样本，1 类样本

距离分类超平面距离$d \geq \frac{1}{\|w\|}$，被正确分类，这是在完全线性可分数据集中的情况。2 类样本距离分类超平面距离$d < \frac{1}{\|w\|}$，被正确分类。3 类样本是被错误分类的样本。对于完全线性可分的数据集，是不会出现 2 类和 3 类样本的，处理这种情况，有两种解决思路：

①某些样本点不满足函数间隔大于等于 1 的约束条件，引入松弛变量ξ

$$y_i(w^T x_i + b) \geq 1 - \xi_i,\ \ \xi_i \geq 0 \tag{2.16}$$

②减少分类间距$\frac{1}{\|w\|}$

易知，①②是相互制约的，只能在它们之间权衡，对于每个松弛变量ξ_i，支付一个代价ξ_i，优化目标由原来的$\frac{1}{2}\|w\|^2$变为$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i,\ C \geq 0$，C 称为惩罚参数，C 越大，分类器越不能容忍错误，容易造成过拟合，C 越小，分类器越不在乎错误，容易造成欠拟合。

加入惩罚参数和松弛变量的 SVM 原始学习问题变成如下凸二次规划问题：

$$\begin{cases} \min\limits_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum\limits_{i=1}^{N}\xi_i \\ s.t.\ \ y_i(w^T x_i + b) \geq 1 - \xi_i, \xi_i \geq 0, \forall i \end{cases} \tag{2.17}$$

与线性可分 SVM 类似，通过 Lagrange 函数，将(2.17)转化为对偶问题：

$$\begin{cases} \min\limits_{\alpha} \ \frac{1}{2}\sum\limits_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j (x_i^T x_j) - \sum\limits_{i=1}^{n}\alpha_i \\ s.t. \ \ \sum\limits_{i=1}^{n}\alpha_i y_i = 0,\ 0 \le \alpha_i \le C, \forall i \end{cases} \tag{2.18}$$

类似的，通过求解对偶问题从而得到原始问题的解，进而确定分类超平面和决策函数。

（2）完全线性不可分的数据集

对于完全线性不可分的数据集，上述方法就不适用了，为了对这类数据集进行分类，可以使用非线性变换将原始数据集的特征空间变换为线性可分的，从而使用线性分类器进行分类，这类分类器被称为“广义线性分类器”，这种非线性变换实际上是一个特征映射，将数据集原始空间映射到一个线性可分的高维空间。

定义一个特征映射$\phi(\bullet)$，将原始空间$\mathcal{X}$映射到$\mathcal{F}$：

$$\phi(\bullet): \mathcal{X} \to \mathcal{F} \tag{2.19}$$

SVM 决策函数为：

$$h(x) = w^T \phi(x) + b \tag{2.20}$$

待学习凸二次规划问题为：

$$\begin{cases} \min\limits_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum\limits_{i=1}^{N}\xi_i \\ s.t. \ \ y_i(w^T\phi(x_i) + b) \ge 1 - \xi_i, \xi_i \ge 0, \forall i \end{cases} \tag{2.21}$$

虽然在理论上知道特征映射$\phi(\bullet)$可以将原来线性不可分的原始数据映射到某个高维空间，使其变得线性可分，但是寻找这样的特征映射是非常困难的，特征映射$\phi(\bullet)$的确定往往需要相当高的技巧和相当专业的领域知识，特征映射$\phi(\bullet)$的计算可能会相当复杂，怎样选择合适的$\phi(\bullet)$也无法确定。

所以使用核函数$K(x_i,x_j)$隐式的构建这种映射:

$$K(x_i,x_j)=<\phi(x_i),\phi(x_j)>=\phi(x_i)^T\bullet\phi(x_j) \tag{2.22}$$

这样，在学习和预测时只定义核函数$K(x_i,x_j)$，特征映射$\phi(\bullet)$是隐性定义的，无需知道其具体形式，直接计算$K(x_i,x_j)$比较容易。

通过 Lagrange 函数，将(2.21)转化为对偶问题：

$$\begin{cases}\min\limits_{\alpha} \ \frac{1}{2}\sum\limits_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j(\phi(x_i)^T\bullet\phi(x_j))-\sum\limits_{i=1}^{n}\alpha_i \\ s.t. \ \ \sum\limits_{i=1}^{n}\alpha_i y_i=0,\ 0\le\alpha_i\le C,\forall i\end{cases} \tag{2.23}$$

$$\Downarrow$$

$$\begin{cases}\min\limits_{\alpha} \ \frac{1}{2}\sum\limits_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j K(x_i,x_j)-\sum\limits_{i=1}^{n}\alpha_i \\ s.t. \ \ \sum\limits_{i=1}^{n}\alpha_i y_i=0,\ 0\le\alpha_i\le C,\forall i\end{cases} \tag{2.24}$$

只有核函数$K(x_i,x_j)$是正定核函数数，(2.23)才能等价为(2.24)。

此时，SVM 的决策函数为：

$$\begin{aligned} f(x)&=\mathrm{sgn}(w^T\phi(x)+b) \\ &=\mathrm{sgn}(\sum_{\text{支持向量}}\alpha_i\, y_i<\phi(x_i),\phi(x)>+b) \\ &=\mathrm{sgn}(\sum_{\text{支持向量}}\alpha_i\, y_i K(x_i,x)+b)\end{aligned} \tag{2.25}$$

常用的核函数有：

多项式核函数：$k(x_i,x_j)=(1+x_i\cdot x_j)^d$

高斯核函数：$k(x_i,x_j)=\exp(-\left\|x_i-x_j\right\|^2/2\sigma^2)$

sigmoid 核函数：$k(x_i,x_j)=\tanh(k\cdot<x,x_i>+\theta)$

核函数可以单个使用，也可以多种核函数组合使用：

$$K(x,x')=\sum_{p=1}^{m}\gamma_p K_p(x,x') \tag{2.26}$$

其中，$\gamma_p \geq 0$，$K_p(x,x')$是核函数。

2.4 SVM 的编程实现

基于 OpenCV3 提供的机器学习库函数实现简单的 SVM 分类，代码和分类效果如下：

代码截图：

```
void test_opencv_svm_simple()
{
    // two class classifcation
    // reference: opencv-3.3.0/samples/cpp/tutorial_code/ml/introduction_to_svm/introduction_to_svm.cpp
    const int width{ 512 }, height{ 512 };
    Mat image = Mat::zeros(height, width, CV_8UC3);

    const int labels[] { 1, -1, -1, -1 };
    const float trainingData[][2] {{ 501, 10 }, { 255, 10 }, { 501, 255 }, { 10, 501 } };

    Mat trainingDataMat(4, 2, CV_32FC1, (float*)trainingData);
    Mat labelsMat(4, 1, CV_32SC1, (int*)labels);

    Ptr<ml::SVM> svm = ml::SVM::create();
    svm->setType(ml::SVM::C_SVC);

    svm->setKernel(ml::SVM::LINEAR);
    svm->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER, 10, 1e-6));

    svm->train(trainingDataMat, ml::ROW_SAMPLE, labelsMat);

    // Show the decision regions given by the SVM
    Vec3b green(0, 255, 0), blue(255, 0, 0);
    for (int i = 0; i < image.rows; ++i) {
        for (int j = 0; j < image.cols; ++j) {
            Mat sampleMat = (Mat_<float>(1, 2) << j, i);
            float response = svm->predict(sampleMat);

            if (response == 1)
                image.at<Vec3b>(i, j) = green;
            else if (response == -1)
                image.at<Vec3b>(i, j) = blue;
        }
    }

    // Show the training data
    int thickness{ -1 };
    int lineType{ 8 };
    circle(image, Point(501, 10), 5, Scalar(0, 0, 0), thickness, lineType);
    circle(image, Point(255, 10), 5, Scalar(255, 255, 255), thickness, lineType);
    circle(image, Point(501, 255), 5, Scalar(255, 255, 255), thickness, lineType);
    circle(image, Point(10, 501), 5, Scalar(255, 255, 255), thickness, lineType);

    // Show support vectors
    thickness = 2;
    lineType = 8;
    //
    double C = svm->getC();
```

```
    vector<float> svm_alpha = {0};
    vector<float> svm_svidx;
    float  svm_rho;
    double decesion = svm->getDecisionFunction(0, svm_alpha, svm_svidx);
    svm_rho = decesion;
    Mat usv = svm->getUncompressedSupportVectors();
    Mat SV = svm->getSupportVectors();
    cout << "usv = " << usv << endl
         << "SV = " << SV << endl
         << "C = " << C << endl
         << "svm_alpha = " << svm_alpha.size() << "..." << svm_alpha[0] << endl
         << "svm_svidx = " << svm_svidx.size() << "..." << svm_svidx[0] << endl
         << "svm_rho = " << svm_rho << endl;

    for (int i = 0; i < usv.rows; ++i) {
        const float* v = usv.ptr<float>(i);
        circle(image, Point((int)v[0], (int)v[1]), 6, Scalar(128, 128, 128), thickness, lineType);
    }

    imwrite("result_svm_simple.png", image);
    imshow("SVM Simple Example", image);
    waitKey(0);
}
```

分类效果：

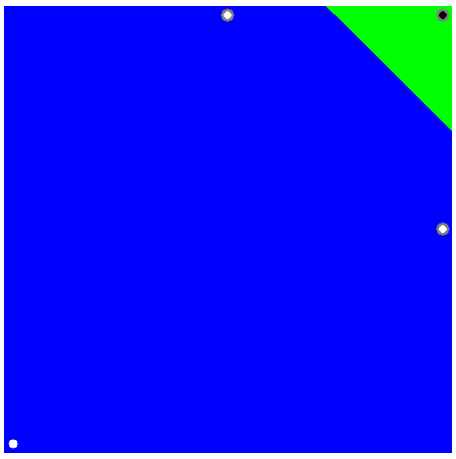

图 12 SVM 分类效果图

3.贝叶斯决策（Bayes）

以 SVM 为代表的分类方法并不能能解决所有的模式识别问题，对于只有正样本没有负样本的数据集，无法使用经典的 SVM，对于只能观察到一些零星线索的情况，SVM 也无法对它们进行融合，这就需要使用贝叶斯决策方法进行判断和决策。

贝叶斯决策方法是基于贝叶斯定理与特征条件独立假设的分类方法。对于给定的数据集，首先基于特征条件独立假设学习输入输出的联合概率分布，然后基于此模型，对给定的输入 x，利用贝叶斯定理求出后验概率最大的输出 y。

3.1 贝叶斯公式

条件概率公式：

$$P(x|y)=\frac{P(xy)}{P(y)},\ p(y)>0 \tag{3.1}$$

全概率公式：

$$P(A)=\sum_{i=1}^{n}P(B_i)P(A|B_i) \tag{3.2}$$

贝叶斯公式：

$$P(B_i|A)=\frac{P(A|B_i)P(B_i)}{\sum_{j=1}^{n}P(B_j)P(A|B_j)}=\frac{P(A|B_i)P(B_i)}{P(A)} \tag{3.3}$$

$P(B_i|A)$ 为**后验概率**，表示结果事件 A 发生后，各不相容的条件 B_i 存在的概率，它是在结果出现后才能计算得到的，因此称为“后验”；

$P(A|B_i)$ 为**类条件概率**，表示在各条件 B_i 存在时，结果事件 A 发生的概率。

$P(B_i)$为**先验概率**，表示各不相容的条件B_i出现的概率，它与结果A是否出现无关，仅表示先验知识或主观推断。

所以，贝叶斯公式给出了根据出现的先验概率和类条件概率，计算一个结果出现时导致这个结果的各个原因存在的概率，也就是实现了逆概率推理的过程。

3.2 贝叶斯决策方法

3.2.1 问题表示：

分类类别$w_i, i=1,2,\cdots,c$

样本特征向量$x=[x_1,x_2,\cdots,x_d]\in\mathbb{R}^d$

先验概率：$P(w_i)\quad\sum_{i=1}^{c}P(w_i)=1$

后验概率：$P(w_i\mid X=(x_1,x_2,\cdots,x_d))=\dfrac{P(X\mid w_i)P(w_i)}{\sum_{j=1}^{c}P(w_j)P(X\mid w_j)}=\dfrac{P(X\mid w_i)P(w_i)}{P(X)}\ (3.4)$

3.2.2 基于最小错误率的贝叶斯判别

基于最小错误率的贝叶斯分类决策规则为：

当$P(w_i\mid X)=\max\limits_{1\le j\le c}P(w_j\mid X)$时，判决$X\in w_i$，由$(3.4)$，$P(X)$对于每一类都是相等的，所以在分类决策时，只有分子项起作用，分类决策规则可以写为：

若，$P(X\mid w_i)P(w_i)=\max\limits_{1\le j\le c}[P(X\mid w_j)P(w_j)]$，判决$X\in w_i$，这其实是最大后验概率的的思想，采用最大后验概率判别时的分类错误率为：

$$P(e)=\int_{-\infty}^{+\infty}P(error,x)dx=\int_{-\infty}^{+\infty}P(error\mid x)P(x)dx$$

而

$$P(error \mid x)=\sum_{i=1}^{c} P(w_i|X)-\max_{1\le j\le c} P(w_j \mid X)，$$

当 $P(error \mid x)$ 取得最小值的时候， $P(e)$ 也取得了最小值，所以最大后验概率分类器与最小错误率分类器是等价的。

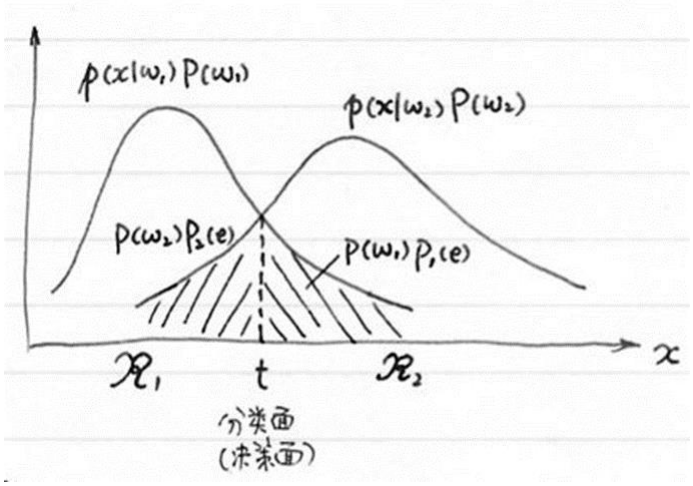

图 13 最小错误率判别准则的几何解释

3.2.3 基于最小风险的贝叶斯判别

最小错误率贝叶斯分类器非常简单，在依据新的信息来修正先验概率，减小做分类决策的错误率时会非常有效，但当我们使用贝叶斯分类器时，仅仅考虑识别错误率低是不够的，还应当把我们所采取的分类决策所带来的后果考虑进去，这就是“最小风险贝叶斯分类器”。

假定有一个模式样本 x， x 属于 w_i 类的概率为 $P(w_i \mid x)$，若样本 x 实际是属于 w_j 类的，而分类器将其分类到 w_i 类，相应产生的损失记为 $L_{i,j}$，则期望损失为：

$$r_j(x)=\sum_{i=1}^{M} L_{i,j}P(w_i \mid x)=\frac{1}{P(x)}\sum_{i=1}^{M} L_{i,j}P(x \mid w_i)P(w_i)\propto \sum_{i=1}^{M} L_{i,j}P(x \mid w_i)P(w_i) \quad (3.5)$$

在大多数模式识别问题中，正确判别的损失为 0，而对所有的错误判别，损失相同，故此有所谓 0-1 假设，此时有

$$
\begin{aligned}
r_j &= \sum_{i=1}^{M} L_{i,j} P(w_i \mid x) = \sum_{i=1}^{M} (1-\delta_{i,j}) P(w_i \mid x) \\
&= \sum_{i=1}^{M} P(w_i \mid x) - P(w_j \mid x) \\
&= 1 - P(w_j \mid x)
\end{aligned} \tag{3.6}
$$

因此，在 0-1 假设下，最小风险判别与最小错误率判别是等价的。

4 聚类(clustering)

聚类是无监督学习的一种典型方法，它通过确定合适的分类准则，使得同类的样本尽可能聚集在一起，不同类的样本尽可能分开。

4.1 相似性度量

如何确定两个样本输入同类涉及到样本间的相似性度量，常用的度量方法是距离度量：

街区距离： $D(X,Z)=\| X-Z \|_1=\sum_{i=1}^{n}| x_i-z_i |$

欧氏距离： $D^2(X,Z)=\| X-Z \|_2^2=(X-Z)^T(X-Z)$

切比雪夫距离： $D(X,Z)=\| X-Z \|_\infty=\max_i | x_i-z_i |$

两个样本距离越近相似性越高。

4.2 K-means 聚类

算法流程：

输入：训练数据 $\{x_1,x_2,\cdots,x_n\}$；簇的个数 K

初始化：指定 K 个聚类中心， $\mu_1,\mu_2,\mu_3,\cdots,\mu_K$

迭代：

①给每个样本点分类，选择最近的聚类中心 μ_k

$$k=\arg\min_{k=\{1,2,\cdots,K\}}\|x_i-x_k\| \tag{4.1}$$

②更新 μ_k， C_k 为第 k 类样本点的集合

$$\mu_k=\frac{1}{|C_k|}\sum_{j\in C_k}X_j \tag{4.2}$$

③重复①②，直到收敛

4.3 K-means 聚类的编程实现

基于 OpenCV3 提供的机器学习库函数实现简单的 K-means 聚类，代码和聚类效果如下：

代码截图：

```
void kmeans_test()
{
    const int MAX_CLUSTERS = 5;
    Scalar colorTab[] =
    {
        Scalar(0, 0, 255),
        Scalar(0,255,0),
        Scalar(255,100,100),
        Scalar(255,0,255),
        Scalar(0,255,255)
    };//最多5类,对应5种颜色

    Mat img(500, 500, CV_8UC3);
    RNG rng(12345);

    while(1)
    {
        int k, clusterCount = rng.uniform(2, MAX_CLUSTERS+1);//生成 {2,3,4,5} 中一个数...k只是定义,没有初始化
        int i, sampleCount = rng.uniform(1, 1001);//样本数
        Mat points(sampleCount, 1, CV_32FC2), labels;//points是CV_32FC2类型,所以每个坐标是个Vec2f,Scalar(a,b)来赋值
        clusterCount = MIN(clusterCount, sampleCount);
        std::vector<Point2f> centers;//聚类中心

        /* generate random sample from multigaussian distribution */
        for( k = 0; k < clusterCount; k++ )
        {
            Point center;
            center.x = rng.uniform(0, img.cols);
            center.y = rng.uniform(0, img.rows);
            Mat pointChunk = points.rowRange(k*sampleCount/clusterCount,
                                             k == clusterCount - 1 ? sampleCount :
                                             (k+1)*sampleCount/clusterCount);//每次取出样本的sampleCount/clusterCount个
            rng.fill(pointChunk, RNG::NORMAL, Scalar(center.x,center.y), Scalar(img.cols*0.05,img.rows*0.05));
        }

        randShuffle(points, 1, &rng);//打乱行

        double compactness = kmeans(points,                     //待聚类数据
                                    clusterCount,               //簇数
                                    labels,                     //labels表示每一个样本的类的标签，是一个整数，从0开始的索引整数,是簇数.
                                    TermCriteria(TermCriteria::EPS+TermCriteria::COUNT, 10, 1.0),
                                    3,                          //聚类3次,取结果最好的那次,
                                    KMEANS_PP_CENTERS,          //MEANS_PP_CENTERS:centers 初始化方法
                                    centers);                   //聚类中心
                                                                //返回 紧密度

        img = Scalar::all(0);
        Mat img2 = img.clone();
        for( i = 0; i < sampleCount; i++ )
        {
            int clusterIdx = labels.at<int>(i);
            Point2f ipt = points.at<Point2f>(i);
            //Point2f p(ipt.250 + clusterIdx*10);
            circle( img, ipt, 2, colorTab[clusterIdx], FILLED, LINE_AA );
            circle(img2,ipt,2,Scalar::all(255));
        }//显示聚类后的数据
        for (i = 0; i < (int)centers.size(); ++i)
        {
            Point2f c = centers[i];
            //Point2f p(c,250 + i*10);
            circle( img, c, 40, colorTab[i], 1, LINE_AA );
        }//聚类中心
        cout << "Compactness: " << compactness << endl;

        imshow("Before_cluster",img2);
        imwrite("Before_cluster.jpg",img2);
        imshow("After_cluster", img);
        imwrite("After_cluster.jpg", img);

        char key = (char)waitKey();
        if( key == 27 || key == 'q' || key == 'Q' ) // 'ESC'
            break;
    }
}
```

聚类效果：

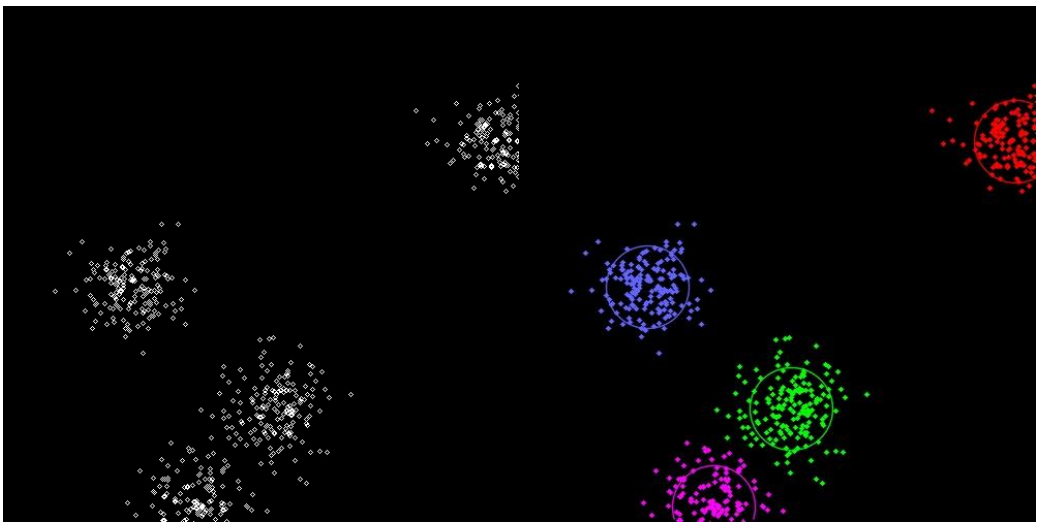

图 14 聚类前后对比图

5.PCA 降维

降维主要用于数据分析，也可以用于监督学习的预处理，降低特征维数从而减少计算量。降维也可以帮助发现高维数据中的统计规律。主成分分析法（PCA）是一种典型的线性降维方法，它的基本思想是利用较少的新变量代替原来较多的旧变量，这些新变量是原始数据通过在方差较大的方向上投影获得的。原来线性相关的数据，经过一个正交变换后变成若干个线性无关的新变量，方差表示在新变量上信息的大小，按照方差大小排序，将新变量依次称为第一主成分、第二主成分等。

下图是对主成分分析法的一个直观解释：

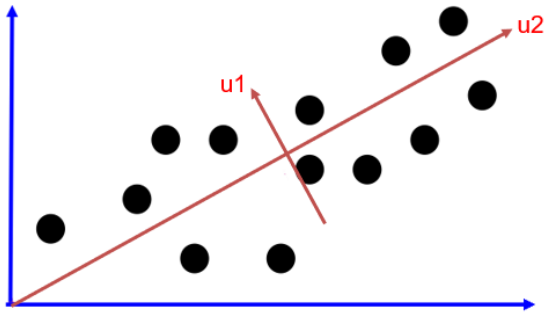

图 15 待降维数据

如图 13 所示，原始数据在 u2 方向上方差最大，u2 便是其一个主成分。

5.1 主成分的确定

假设原始高维数据为 $X \in \mathbb{R}^d$，定义一个投影矩阵 $U \in \mathbb{R}^{d \times k}$ 将其降到 k 维数据 $Z \in \mathbb{R}^k$（$d >> k$），即：

$$Z = U^T X \tag{5.1}$$

这个 k 就表示 k 个投影方向。

PCA 的基本思想是寻找原始数据方差最大的k个方向，方差计算公式为;

$$D=\frac{1}{N}\sum_{i=1}^{N}\left\|X_i-\bar{X}\right\|^2 \tag{5.2}$$

其中，$\bar{X}=\frac{1}{N}\sum_{i=1}^{N}X_i$是原始数据均值。

找到方差大的方向对样本X_i进行投影，假设投影方向为u_1，(5.2)转化为：

$$\begin{aligned} D' &= \frac{1}{N}\sum_{i=1}^{N}(u_1^T X_i - u_1^T \bar{X})^2 \\ &= \frac{1}{N}\sum_{i=1}^{N}u_1^T(X_i-\bar{X})(X_i-\bar{X})u_1 \\ &= u_1^T[\frac{1}{N}\sum_{i=1}^{N}(X_i-\bar{X})(X_i-\bar{X})^T]u_1 \\ &= u_1^T \bullet S \bullet u_1 \end{aligned} \tag{5.3}$$

其中，$S=\frac{1}{N}\sum_{i=1}^{N}(X_i-\bar{X})(X_i-\bar{X})^T \in \mathbb{R}^{d\times d}$为协方差矩阵。

原始寻找方差最大的方向即可转化为下面这个优化问题：

$$\begin{aligned} &\max_{u_1} \ u_1^T \bullet S \bullet u_1 \\ &s.t. \ \ u_1^T u_1 = 1 \end{aligned} \tag{5.4}$$

构建拉格朗日函数求解(5.4)：

$$L(u_1;\lambda)=u_1^T \bullet S \bullet u_1 + \lambda(-u_1^T u_1 + 1) \tag{5.5}$$

令$\frac{\partial L}{\partial u_1}=2Su_1-2\lambda u_1=0$，可以得到：

$$Su_1=\lambda u_1 \Rightarrow u_1^T S u_1 = \lambda u_1^T u_1 = \lambda \tag{5.6}$$

由(5.6)，为了使目标函数$u_1^T \bullet S \bullet u_1$最大，只需要使$\lambda$最大，可以看到，$\lambda$是$S$的特征值，$u_1$是$S$的特征值$\lambda$对应的特征向量，所以取$S$前$k$大哥特征值对应的特征向量就是方差前$k$大的投影方向，最后所求投影

矩阵为：

$$U=[u_1,u_2,\cdots,u_k]\ \ (\lambda_1>\lambda_2>\cdots>\lambda_k) \tag{5.7}$$

除了使用求解S特征值λ的方法进行主成分分析，还可以使用奇异值分解法来进行加速。

奇异值分解是矩阵的一种分解方法：

对于$\forall X\in\mathbb{R}^{n\times d}$，可以分解为

$$X=U\sum V^T \tag{5.8}$$

其中，

$$\Sigma^T=\Sigma=diag(\sigma_1,\sigma_2,\cdots\sigma_n),\ \ \sigma_1>\sigma_2>\cdots>\sigma_n$$
$$VV^T=I$$
$$V=[V_1,V_2,\cdots,V_n]$$

对原始数据进行去中心化处理后，$\bar{X}$=0，此时的协方差矩阵为：

$$\begin{aligned} S&=\frac{1}{N}X^TX \\ &\overset{SVD}{=}\frac{1}{N}(V\sum U^T)(U\sum V^T) \\ &=\frac{1}{N}V\sum{}^2V^T \end{aligned} \tag{5.9}$$

又有：

$$
\begin{aligned}
S \bullet V_1 &= \frac{1}{N} V \Sigma^2 V^T \bullet V_1 \\
&= \frac{1}{N} V \begin{pmatrix} \sigma_1^2 & & & \\ & \sigma_2^2 & & \\ & & \ddots & \\ & & & \sigma_k^2 \end{pmatrix} \begin{pmatrix} V_1^T V_1 \\ V_2^T V_1 \\ \vdots \\ V_k^T V_1 \end{pmatrix} \\
&= \frac{1}{N} V \begin{pmatrix} \sigma_1^2 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \\
&= \frac{\sigma_1^2}{N} V_1
\end{aligned}
\tag{5.10}
$$

所以，V_1就是S的特征向量，$\frac{\sigma_1^2}{N}$就是V_1对应的特征值，所以按照σ_1^2进行排序得到的$V=[V_1,V_2,\cdots,V_n]$中取前k大个就是 PCA 需要的投影矩阵$U=[V_1,V_2,\cdots,V_k]$。

累计贡献率：前k个主成分共有多大的综合能力，用这k个主成分的方差和在全部方差中所占比重表示，即

$$
\frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{n} \sigma_i^2} \tag{5.11}
$$

PCA 总结：

优点：①无参数②易于计算③无局部最优

缺点：①只能得到样本分布的线性主方向，改进方式是使用引入核方法的 KPCA

②计算效率低下

5.2 PCA 近似重建

将原始数据使用 PCA 降维后可以减少特征的维数，在数据传输时

占用了更少的带宽，在数据接收端需要对接收到的数据进行重建。

PCA 近似重建是(5.1)的逆过程：

$$Z \in \mathbb{R}^k \rightarrow \tilde{X} \in \mathbb{R}^d \tag{5.12}$$

重建的 $\tilde{X} = \sum_{i=1}^{k} (X^T u_i) u_i = \sum_{i=1}^{k} Z_i u_i$

5.3 PCA 的编程实现

基于 OpenCV3 提供的机器学习库函数实现简单的 PCA 降维和近似重建，代码和降维效果如下：

代码截图：

```
void pca_test()
{
    float A[ 120 ]={
        1.5 , 2.3 , 1.5 , 2.3 , 1.5 , 2.3 ,
        3.0 , 1.7 , 3.0 , 1.7 , 3.0 , 1.7,
        1.2 , 2.9 , 1.2 , 2.9 , 1.2 , 2.9 ,
        2.1 , 2.2 , 2.1 , 2.2 , 2.1 , 2.2 ,
        3.1 , 3.1 , 3.1 , 3.1 , 3.1 , 3.1 ,
        1.3 , 2.7 , 1.3 , 2.7 , 1.3 , 2.7 ,
        2.0 , 1.7 , 2.0 , 1.7 , 2.0 , 1.7 ,
        1.0 , 2.0 , 1.0 , 2.0 , 1.0 , 2.0 ,
        0.5 , 0.6 , 0.5 , 0.6 , 0.5 , 0.6 ,
        1.0 , 0.9 , 1.0 , 0.9 , 1.0 , 0.9 ,
        1.5 , 2.3 , 1.5 , 2.3 , 1.5 , 2.3 ,
        3.0 , 1.7 , 3.0 , 1.7 , 3.0 , 1.7,
        1.2 , 2.9 , 1.2 , 2.9 , 1.2 , 2.9 ,
        2.1 , 2.2 , 2.1 , 2.2 , 2.1 , 2.2 ,
        3.1 , 3.1 , 3.1 , 3.1 , 3.1 , 3.1 ,
        1.3 , 2.7 , 1.3 , 2.7 , 1.3 , 2.7 ,
        2.0 , 1.7 , 2.0 , 1.7 , 2.0 , 1.7 ,
        1.0 , 2.0 , 1.0 , 2.0 , 1.0 , 2.0 ,
        0.5 , 0.6 , 0.5 , 0.6 , 0.5 , 0.6 ,
        1.0 , 0.9 , 1.0 , 0.9 , 1.0 , 0.9 };

    Mat DataMat = Mat::zeros( 20, 6, CV_32FC1 );

    //将数组A里的数据放入DataMat矩阵中
    for (int i = 0;i < DataMat.rows;i++)
    {
        float* p = DataMat.ptr<float>(i);
        for (int j = 0;j < DataMat.cols;j++)
        {
            p[j] = A[i*6+j];
        }
    }

    cout << "A = \n" << DataMat << endl;
    // OPENCV PCA
    /*
     * PCA(InputArray data, InputArray mean, int flags, int maxComponents = 0);//maxComponents:主成分个数
     * PCA(InputArray data, InputArray mean, int flags, double retainedVariance);//retainedVariance:主成分比重
    */

    PCA pca(DataMat, noArray(), PCA::DATA_AS_ROW,1);//保留80% 或者指定个数

    cout << "(pca)eigenvalues =  \n " << pca.eigenvalues << endl;
    cout << "(pca)eigenvectors = \n" << pca.eigenvectors << endl;
    cout << "(pca)means = \n" << pca.mean << endl;

    Mat eigenvalues;//特征值
    Mat eigenvectors;//特征向量
    /* ...*/

    Mat means = repeat(pca.mean,DataMat.rows/pca.mean.rows,DataMat.cols/pca.mean.cols);
    cout << "means: \n" << means << endl;

    //DataMat -= means;
    /* ...*/

    Mat projectMat = pca.project(DataMat);

    /* ...*/

    cout << "投影后: \n" << projectMat << endl;
    cout << "type: " << projectMat.type() << endl;
    Mat backproMat = pca.backProject(projectMat);
    cout << "逆投影: \n" << backproMat << endl;
}
```

降维和重建结果：

```
(pca)eigenvalues =
 [2.761358]
(pca)eigenvectors =
[0.4352051, 0.37938085, 0.43520534, 0.3793807, 0.43520522, 0.37938064]
(pca)means =
[1.6700001, 2.01, 1.6700001, 2.01, 1.6700001, 2.01]
投影后:
[0.10810643;
 1.3836447;
 0.39930728;
 0.77766162;
 3.107605;
 0.30224022;
 0.07802905l;
 -0.88614398;
 -3.1323509;
 -2.1381004;
 0.10810643;
 1.3836447;
 0.39930728;
 0.77766162;
 3.107605;
 0.30224022;
 0.07802905l;
 -0.88614398;
 -3.1323509;
 -2.1381004]
type: 5
逆投影:
[1.7170485, 2.0510135, 1.7170485, 2.0510135, 1.7170485, 2.0510135;
 2.2721694, 2.5349283, 2.2721696, 2.5349281, 2.2721696, 2.5349281;
 1.8437806, 2.1614895, 1.8437808, 2.1614895, 1.8437806, 2.1614895;
 2.0084424, 2.3050299, 2.0084426, 2.3050299, 2.0084424, 2.3050299;
 3.0224457, 3.1889658, 3.0224464, 3.1889653, 3.0224459, 3.1889651;
 1.8015366, 2.1246641, 1.8015367, 2.1246641, 1.8015366, 2.1246641;
 1.7039587, 2.0396028, 1.7039587, 2.0396028, 1.7039587, 2.0396028;
 1.2843457, 1.6738139, 1.2843455, 1.6738141, 1.2843456, 1.6738141;
 0.30678499, 0.82164603, 0.30678421, 0.82164651, 0.3067846, 0.82164669;
 0.73948789, 1.1988456, 0.73948735, 1.198846, 0.73948785, 1.1988461;
 1.7170485, 2.0510135, 1.7170485, 2.0510135, 1.7170485, 2.0510135;
 2.2721694, 2.5349283, 2.2721696, 2.5349281, 2.2721696, 2.5349281;
 1.8437806, 2.1614895, 1.8437808, 2.1614895, 1.8437806, 2.1614895;
 2.0084424, 2.3050299, 2.0084426, 2.3050299, 2.0084424, 2.3050299;
 3.0224457, 3.1889658, 3.0224464, 3.1889653, 3.0224459, 3.1889651;
 1.8015366, 2.1246641, 1.8015367, 2.1246641, 1.8015366, 2.1246641;
 1.7039587, 2.0396028, 1.7039587, 2.0396028, 1.7039587, 2.0396028;
 1.2843457, 1.6738139, 1.2843455, 1.6738141, 1.2843456, 1.6738141;
 0.30678499, 0.82164603, 0.30678421, 0.82164651, 0.3067846, 0.82164669;
 0.73948789, 1.1988456, 0.73948735, 1.198846, 0.73948785, 1.1988461]
```

第二部分 课程考核

1.考题分析

1.1 问题重述

1) 任务：基于图像的路口识别

图 16 路口指示图

2) 要求：请训练一个分类器，输入为手机在校园主干道路口附近拍摄的图像，输出为该图像拍摄地点所处的路口编号。

3) 路口编号：

表 1 路口编号

A	B	C	D	E
1	2	3	4	5
F	G	H	I	else
6	7	8	9	0

4)

5) 注意点：

a) 每个路口的测试图片将来自不同的拍摄方向，可能有不同的图像比例，可能雨天拍摄，甚至还可能夜间拍摄；

b) 所谓的 else，可能是其它的路口，可能不是路口场景，甚至可能连道路场景都不是。

1.2 问题分析

从考题要求出发，需要设计一个具有路口分类作用的模式识别算法，涉及到分类，那就一定要对图片进行特征提取，直接使用神经网络进行特征提取，基于神经网络有三种解决思路：

1) 使用图像分类的方法，训练一个分类网络，输入一张待测图片，直接输出它所属的路口编号；

2) 使用图像检索的方法，训练一个检索网络，输入一张待测图片，从 gallery 数据集中检索出与其最相似的 TOP K 张图片，选择其中出现次数最多的路口作为它的路口编号；

3) 使用目标检测的方法，训练一个检测网络，输入一张待测图片，在图片中框出路口的位置，并给出路口的编号。

在实际的算法设计中，由于时间有限，成功实现了第一种思路，性能较好；第二种方法虽然也实现了但是性能不佳；由于标注工作需要时间太长，并没有实现第三种思路。

下就简单介绍数据采集过程和已经实现的两种算法。

2.数据采集和预处理

在完成了整个模式识别考试并听了部分同学的报告后，才意识到

在使用的网络模型相近时，一个模式识别算法性能的好坏很大程度上会被使用的训练数据集的好坏牵制。

2.1 数据收集

由于从不同的方向拍摄到的路口图片差异性很大，类内方差大于类间方差，所以前期考虑了两种不同的数据采集方法，都是为了减少类内方差：

（1）360°全景图

图 17 全景图示例

这种 360°全景图可以把路口各个方向都包括进来，没有从不同方向单独拍摄带来的类内差异过大的问题，但是拍摄设备只有手机，在实际拍摄过程中无法拍摄出完整的全景图而且畸变严重，并且分类模型训练方式也有问题，但是如果有更好的获取这种全景图的方式，可以将它用于图像检索，输入待测试图片后，在全景图上进行划窗，可以检索到与之最为相似的路口图片。

（2）分方向拍摄

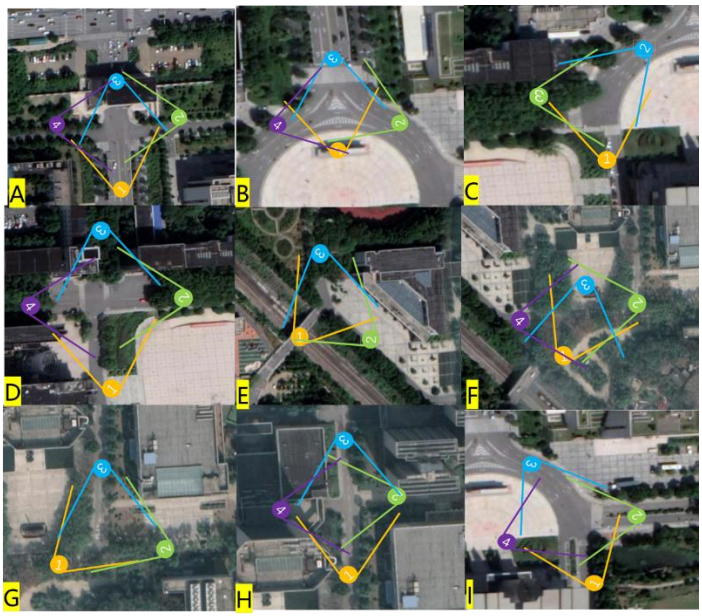

图 18 不同路口拍摄方向示意图

既然每个路口从不同方向单独拍摄图片会带来类内差异过大的问题，那么可以把差异较大方向拍摄的图片单独分成一类，具体分类情况见表 2，最后把 9 类路口细分成了 33 个单独的小类。

表 2 路口具体细分情况

A1-A4	B2-B4	C1-C3	D1-D4	E1-E3
1	2	3	4	5
F1-F4	G1-G3	H1-H4	I1-I4	else
6	7	8	9	0

在拍摄时间和拍摄情景上，从白天到傍晚到夜晚都有，夜间图片只拍摄了光线条件较好的，同时也有晴天、阴天、雨天不同天气状况的图片，有路口空旷的，也有路口处有大量车辆和行人的图片，力图做到光照和情景的多样性。

具体图片采集时，采用先拍摄视频后插帧选取照片的方式，再对

模糊、遮挡过分严重的图片进行剔除，最终选用了 3 万多张照片作为训练数据，每一类照片数量相近，大概有 1000 张照片，防止出现样本数量不均衡的情况。

2.2 图像增强

在正式训练之前，对数据集采用以下方法进行图像增强：

(1)随机长宽比裁剪+颜色抖动

在测试时，不同拍摄设备的照片会有长宽比不同、颜色空间不同等问题，这样处理可以增强算法对不同拍摄设备的鲁棒性。

（2）随机旋转

应对不同的拍摄角度。

（3）随机遮挡

应对测试图片路口被不同物体遮挡的情况。

图像增强处理后的图片：

表 3 图像增强示意图

3.基于 ResNet 的路口分类

如果使用图像分类的思路来解决路口识别问题，那么算法整体流程如下图所示：

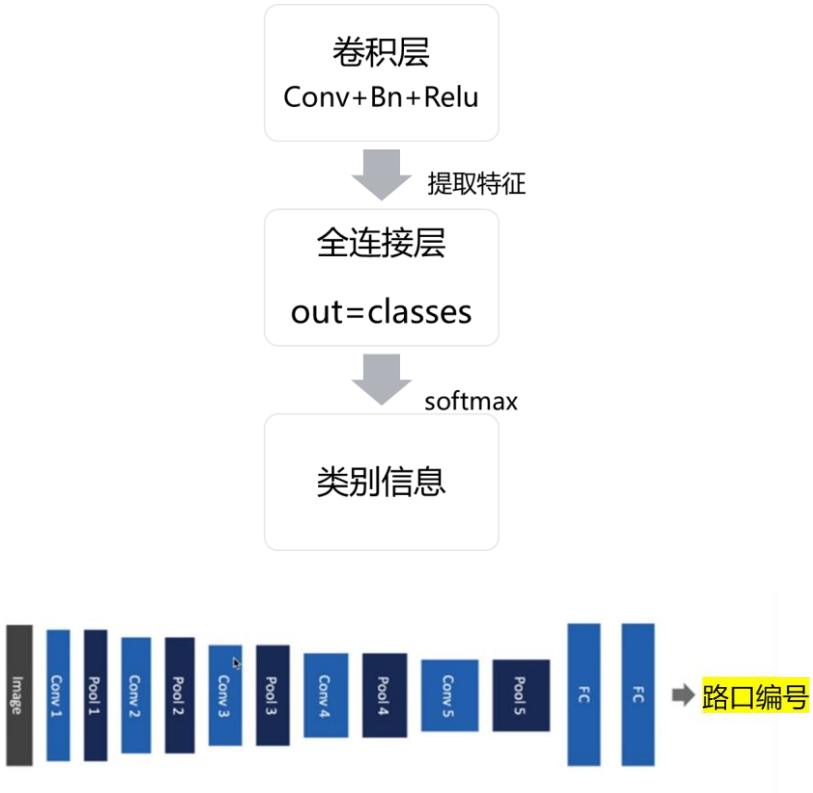

图 19 图像分类流程图

目前的图像分类算法基本上都是使用卷积神经网络自动提取图像特征，使用全连接层来分类，不同的图像分类算法主要不同在使用的卷积神经网络的不同，本次考试选用了“简单实用”的 ResNet-101 作为特征提取网络，下对 ResNet 做简单的介绍。

3.1 ResNet 简介

随着神经网络层数的加深，网络在训练集上的准确度会出现饱和甚至下降，这不是因为过拟合的原因，而是深层网络发生了退化现象，ResNet 使用残差学习思想解决网络的退化现象，允许原始输入信息直接传到后面的层中，在网络中加入 shortcut，构建所谓的残差块，残差块示意图如下：

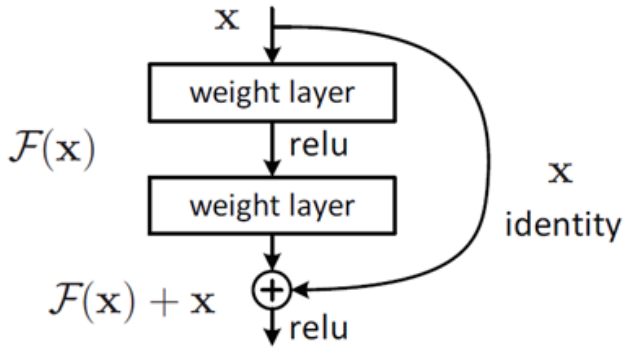

图 20 残差学习单元

根据上图，残差块具有两层结构：.

$$F = W_2\sigma(W_1x) \tag{3.1}$$

其中σ代表非线性函数 ReLU，然后通过一个 shortcut，和第 2 个 ReLU，获得输出 y：

$$y = F(x,\{W_i\}) + x \tag{3.2}$$

即把浅层网络的输出直接加到深层网络的输出，将输出分成$F(x)+x$两部分，其中F是x的函数，实际上是对于x的补充，这样就把任务从根据x映射成一个新的y转为了根据x求x和y之间的差距，这是一个相对更加简单的任务。这个简单的加法并不会给网络增加额外的参数和计算量，同时却可以大大增加模型的训练速度、提高训练效果，并且当模型的层数加深时，这个简单的结构能够很好的解决退化问题。

残差模块会明显减小模块中参数的值，从而让网络中的参数对反向传导的损失值有更敏感的响应能力，虽然在根本上没有解决回传损失小的问题，但却让参数减小，相对而言增加了回传损失的效果，产生了一定的正则化作用。

残差块最重要的作用就是改变了前向和后向信息传递的方式，前

向过程中帮助网络中的特征进行恒等映射，在反向过程中帮助传导梯度，让更深的模型能够成功训练。

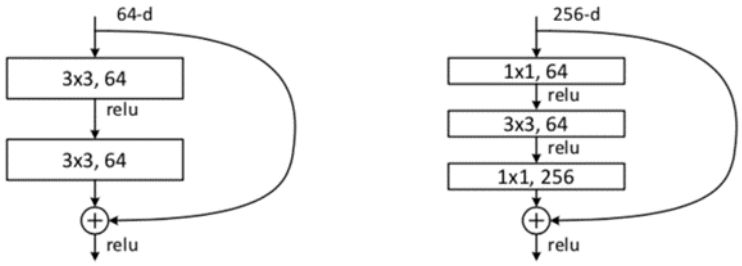

图 21 两种形式的残差块

在 ResNet 网络结构中会用到两种残差模块，如上图所示。

第一种是以两个 3×3 的卷积网络串接在一起作为一个残差模块（building block），这种两层的残差学习单元中包含两个相同输出通道数的 3×3 卷积。

另外一种是 1×1、3×3、1×1 的 3 个卷积网络串接在一起作为一个残差模块（bottleneck design），第一个 1x1 的卷积把 256 维 channel 降到 64 维，最后通过 1x1 卷积恢复，整体上使用的参数数目为：1x1x256x64 + 3x3x64x64 + 1x1x64x256 = 69632，而不使用 bottleneck 的参数数目为两个 3x3x256 的卷积，即 3x3x256x256x2 = 1179648，差了 16.94 倍，加入 1×1 的卷积核可以减小计算成本，层数虽然怎加了，但是计算成本没有增加，所以这种三层结构可以把网络拓展成更深的模型。

另外，如果有输入、输出维度不同的情况，我们可以对 x 做一个线性映射变换维度，再连接到后面的层。

通过上述分析，普通直连的卷积神经网络和 ResNet 的最大区别在于，ResNet 有很多旁路的支线将输入直接连到后面的层，使得后面的

层直接学习残差。

传统的卷积层或全连接层在信息传递时，或多或少会存在信息丢失、损耗等问题。ResNet 在某种程度上解决了这个问题，通过直接将输入信息绕道传到输出，保护信息的完整性，整个网络则只需要学习输入、输出差别的那一部分，简化学习目标和学习难度。

表 4 不同网络结构的 ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

表 4 中给出了五种不同深度的 ResNet，分别是 18 层，34 层，50 层，101 层和 152 层，所有的网络都分成 5 部分，分别是：conv1，conv2_x，conv3_x，conv4_x，conv5_x。即 ResNet 共有 5 组卷积，第一组卷积的输入大小是 224x224，第五组卷积的输出大小是 7x7，缩小了 32 倍，每次缩小 2 倍，总共缩小 5 次，且每次都是在每组卷积的第一层上把 stride 设置为 2。

ResNet 应用于图像分类任务时使用卷积层对输入图片进行特征提取，把最后的全连接层 1000 维特征输出用于分类任务，在本次考试中，分类数目为 33 类，所以直接将最后的全连接层由 1000 层改为 33 层进行模型的训练和预测。

3.2 模型训练和测试结果

3.2.1 模型训练

在具体的模型训练过程中，batch_size 设为 32，损失函数选用交叉熵，优化器选用 Adam，初始学习率设为 0.00001，每个 10Epoch 学习率降为原来的 1/10,共训练 40 个 Epoch。训练结果如下图所示：

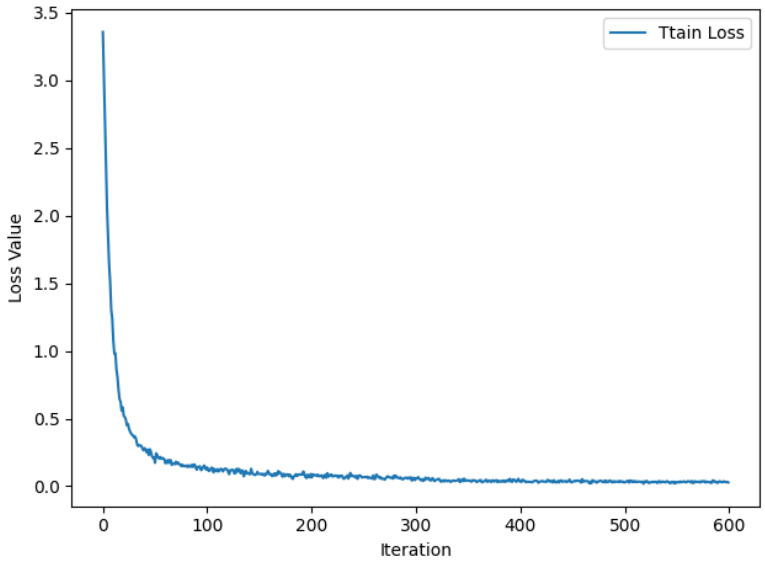

图 22 训练集 loss 下降图

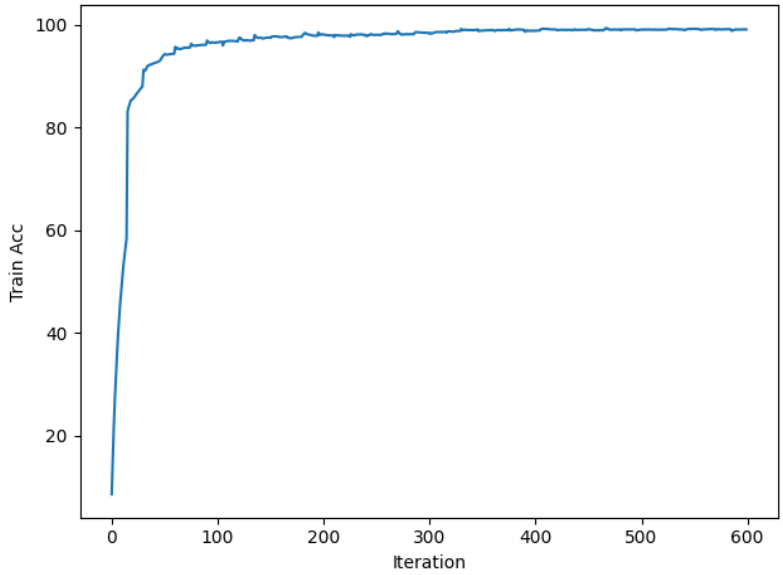

图 23 训练集准确度上升

经过 40 个 epoch，训练损失降至 0.02，训练准确率达到 98%。

3.2.2 考试测试结果分析

图 24 分错的测试样本

在正式考试中，采用上述基于 ResNet-101 训练出来的图像分类模型对测试路口图片进行分类，置信度小于 0.4 的置为 else，0 类，测试正确率为 89%，共分错 11 张图片，其中有 5 张是夜间图片，有 4 张是 else 图片，另有两张不具有代表性，夜间图片共有 10 张，准确率为 50%，else 图片有 10 张，准确率为 60%，模型对于夜间图片和 else 类图片的分辨能力不足。

4.基于度量学习的路口识别

虽然使用分类模型解决了考试中的路口识别问题，但是如果想把路口识别推广开来，不只是识别 9 个路口，想要识别全国所有的路口，那么就不能使用分类模型来做，而且分类模型无法准确处理 else 类别问题，这就体现出了分类模型的两个弊端：

（1）如果分类类别数过大，会导致网络参数过多，训练缓慢

（2）网络可以分类的类别数是固定不变的，如果改变分类类别数，比如加入一个新类，那么就需要重新定义网络模型，从头训练一遍。

解决办法就是不去学习类别信息，而是使用度量学习去学习距离信息，学习图片之间的相似性。

基于度量学习设计图像检索算法的流程如下图所示：

图 25 图像检索流程

图像检索和图像分类的不同在于在网络中不使用全连接层，直接使用卷积层输出的特征向量在 gallery 数据集（标准数据集）中进行最近邻搜索，选出与其最相似的 TOP K 张图片，选择其中出现次数最多的类别作为输入图片的类别信息。

下就简单介绍一下度量学习和在度量学习中经常使用的三元组损失函数。

4.1 度量学习简介

度量学习又被称为距离度量学习或者相似度学习，旨在学习出两张图片的相似性。

度量学习（metric learning）研究如何在一个特定的任务上学习一

个距离函数，使得该距离函数能够帮助基于近邻的算法（k-NN、k-means 等）取得较好的性能。深度度量学习（deep metric learning）是度量学习的一种方法，它的目标是学习一个从原始特征到低维稠密的向量空间（称之为嵌入空间，embedding space）的映射，使得同类对象在嵌入空间上使用常用的距离函数计算的距离比较近，而不同类的对象之间的距离则比较远。

首先定义一个映射：

$$f(x):\mathbb{R}^F \to \mathbb{R}^d \tag{4.1}$$

将图片从原始域映射到特征域，之后再定义一个距离度量函数：

$$D(x,y):\mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R} \tag{4.2}$$

计算两个特征向量之间的距离，传统的度量学习的工作主要集中在设计距离度量函数上，因为可供使用的图像特征描述子很少，在深度学习出现之后，度量学习的重点转移到了设计网络的度量损失，距离度量函数一般使用欧式距离或者余弦距离。

度量学习通过最小化网络的度量损失，来寻找一个最优的映射 $f(x)$，使得正样本对的距离尽可能小，负样本对之间的距离尽可能大，可以看成是在特征空间进行聚类。

4.2 度量学习常用的损失函数

（1）对比损失（Contrastive loss）

对比损失的输入是两个样本组成的样本对，用于训练 Siamese（孪生）网络，每一对训练图片都有一个标签 y，y=1 表示两张图片属于同类（正样本对），反之 y=0，表示两张图片属于异类（正样本对）为

负样本对。

对比损失可以表示为：

$$L_c = yd_{a,b}^2 + (1-y)(\alpha - d_{a,b})_+^2 \tag{4.3}$$

其中 $(z)_+ = \max(z,0)$。

当输入正样本对时，$d_{a,b}$ 逐渐减小，同类图片会在特征空间形成聚类。反之，当网络输入负样本对时，$d_{a,b}$ 会逐渐变大，直到超过设定的阈值 α，α 是预先设定的超参数，表示不同类样本之间的距离应超过该 值。通过最小化损失函数，可以使正样本对之间距离逐渐变小，负样本对之间距离逐渐变大。

（2）三元组损失（Triplet loss）

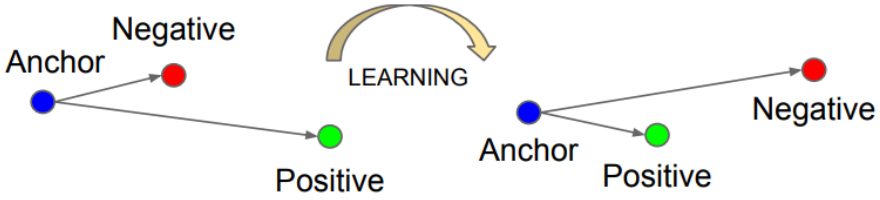

图 26 三元组损失示意图

三元组损失是一种被广泛应用的度量学习损失，之后的大量度量学习方法也是基于三元组损失演变而来。顾名思义，三元组损失需要输入三张图片。和对比损失不同，一个输入的三元组（Triplet）包括一对正样本对和一对负样本对。

三张图片分别命名为锚点图片(Anchor) a、正样本图片(Positive)p 和负样本图片(Negative) n 。图片 a 和图片 p 为一对正样本对，图片 a 和图片 n 为一对负样本对。

三元组损失的学习目的是让正样本对间的距离小于负样本对间的

距离，两者之间相差一个设定的阈值α：

$$d_{a,p}+\alpha<d_{a,n} \tag{4.4}$$

则三元组损失表示为：

$$L_t=(d_{a,p}-d_{a,n}+\alpha)_+ \tag{4.5}$$

但是原版的三元组损失只考虑正负样本对之间的相对距离，而并没有考虑正样本对之间的绝对距离，这对于图像检索没有太大影响，但是在一些问题上，比如目标跟踪，需要确认前后帧是否为同一个目标，这就需要一个比较可靠的绝对距离作为保证。

为此提出改进三元组损失(Improved triplet loss)：

$$L_{it}=d_{a,p}+(d_{a,p}-d_{a,n}+\alpha)_+ \tag{4.6}$$

这样就可以保证网络不仅能够在特征空间把正负样本推开，也能保证正样本对之间的距离很近。

（3）难样本采样三元组损失(Triplet loss with batch hard mining, TriHard loss)

传统的三元组随机从训练数据中抽样三张图片，这样的做法虽然比较简单，但是抽样出来的大部分都是简单易区分的样本对。如果大量训练的样本对都是简单的样本对，就非常不利于网络学习到更好的参数。于是有学者提出了一种基于训练批量(Batch)的在线难样本采样方法——TriHard Loss。

TriHard 损失的核心思想是：

对于每一个训练 batch，随机挑选 P 个类别，每类随机挑选 K 张不同的图片，即一个 batch 含有 P×K 张图片。之后对于 batch 中的

每一张图片 a ，我们可以挑选一个最难的正样本和一个最难的负样本和 a 组成一个三元组。

TriHard 损失可以表示为：

$$L_{th}=\frac{1}{P\times K}\sum_{a\in A}(\max_{p\in A}d_{a,p}-\min_{n\in B}d_{a,n}+\alpha)_{+} \tag{4.7}$$

其中α是人为设定的阈值参数。TriHard 损失会计算 a 和 batch 中的每一张图片在特征空间的欧式距离，然后选出与 a 距离最远（最不像）的正样本 p 和距离最近（最像）的负样本 n 来计算三元组损失。通常 TriHard 损失效果比传统的三元组损失要好。

本次实验选用 TriHard 损失训练度量学习模型。

4.3 度量学习模型训练和测试结果

基于 ResNet-101 提取特征，选用 TriHard 损失作为度量损失函数，交叉熵损失函数作为分类损失，两者相加作为总损失，度量模型和分类模型联合训练。

训练结果如下图所示：

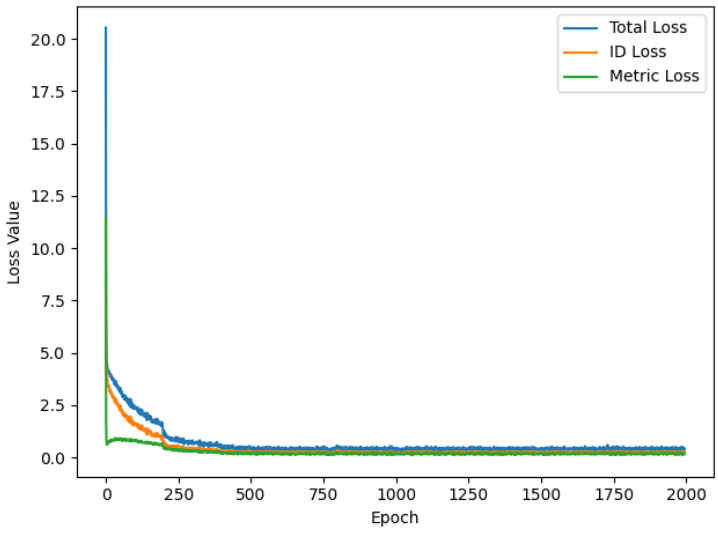

图 27 训练集损失下降图

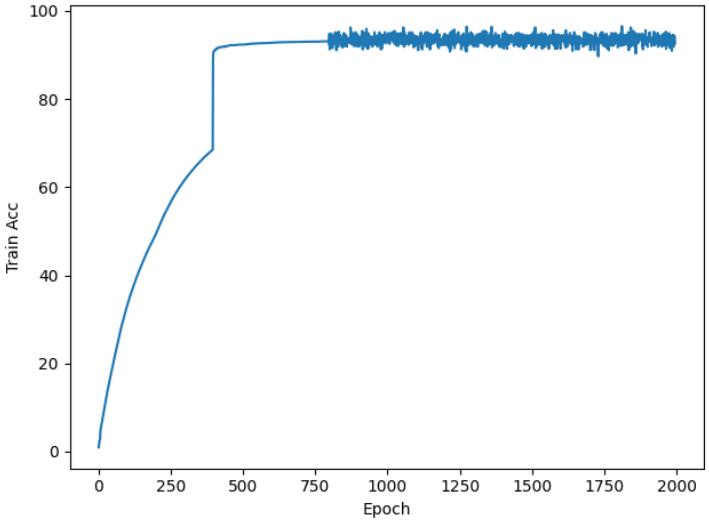

图 28 训练集分类准确率上升图

由于度量损失需要独特的采样方式，不利于分类模型的训练，训练完成后，总体损失降至 1 左右后便无法继续下降，分类准确率也只能达到 90%，由于没有很好的改进训练方法，基于度量学习进行路口识别最终在测试集上的准确率只能达到 56%。

5.关于 Else 类的思考

在测试样本中含有未知类样本的识别问题被称为开集识别问题，如果使用闭集识别系统，系统将错误地将来自未知类的测试样本识别为属于已知闭合集类之一，正确率下降。

考试准备阶段有三种应对开集识别的思路：

（1）使用异常检测领域的相关技术手段，比如 OneClassSvm，IForest，Auto Encoder 等等，把训练集中没有的类别数据当作异常；

（2）使用聚类的方法，设定相关的阈值，如果新的样本距每个类别的中心距离都超过了阈值，则认为该样本是其他类别。

这种方法实际上就是基于度量学习的路口识别，从表征学习和度

量学习学习到的特征空间可以明显看出，表征学习学出的特征空间只有可分性并不具有可判别性，而度量学习学出的特征空间具有明显的聚类特性。

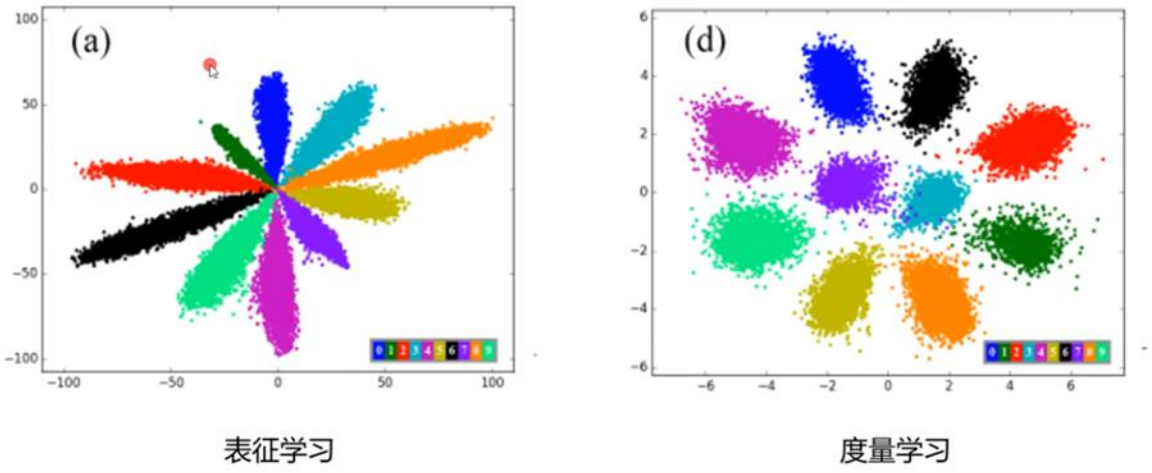

图 29 表征学习和度量学习特征空间对比图

（3）分类网络输出置信度低的认为是其他类别。

实际考试时使用的是第三种方法，置信度阈值设为 0.4，低于 0.4 的设为 else 类。

第三部分 意见和建议

由于新冠肺炎疫情，模式识别课程无法进行线下授课，只能采取线上 MOOC 课程+线下讲座的形式来完成这门课程，估计这种教学形式也只会在今年出现一次，以后还会是像以前一样采用线下教学的形式，所以对课程的意见和建议只能基于几次线下课程来提，有片面、不到位的地方还请吴老师见谅。

（1）在课程考核的最后希望老师可以把往届做的比较好的同学的方法进行一个总结，并分享之前同学比较新奇独特的想法，在听到这些独特或优秀的想法后，在同学们之间一定会发生思维的碰撞，对培养创新思维有很大的作用；

（2）无论是之前课程过程中布置的实践作业还是最后的课程考试，都是采用离线测试的形式，但是这样做就无法让我们体会真正的模式识别应用的完整过程，因为它缺失了最后一环：落地应用，在从算法到实际落地使用的过程中也会遇到各种各样的问题，所以我建议以后老师可以在实践作业中选择一个要求同学使用相关的硬件进行具体实现，可采取自愿原则，课程考核时也可以报名汇报在将算法硬件具体实现中的经验和教训，这样的同学同样也可以拿到 A。

以上就是我对模式识别课程的一些建议，非常感谢吴涛老师最后坚持进行课下讲座教学，虽然时间短暂，也让我受益匪浅，注意到了之前学习过程中有所忽视的地方，让我对模式识别有了一个更加清晰深刻的认识。