模式识别与机器学习实验报告

150325 班
15141009
郑琦

May 3, 2018

实验一、线性分类器及感知器

1.1 实验介绍

Linear perceptron is one of the simplest learning algorithms for a two-class classifier. Given a set of data points in d-dimensions, belonging to two classes, and , the algorithm tries to find a linear separating hyper-plane between the samples of the two classes. If the samples are in one, two or three dimensions, the separating hyperplane would be a point, line or a plane respectively. The specific algorithm that we look into is a special case of a class of algorithms that uses gradient descent on a carefully defined objective function to arrive at a solution.

1.2 实验原理

Assume that the samples of the two classes are linearly separable in the feature space. i.e., there exists a plane $G(x) = W^T X + \omega_{n+1}$, where $W \in R^n$ and $X \in R^n$, such that all samples belonging to the first class are on one side of the plane, and all samples of the second class are on the opposite side. If such planes exist, the goal of the perceptron algorithm is to learn any one such plane, given the data points. Once the learning is completed and the plane is determined, it will be easy to classify new points in the future, as the points on one side of the plane will result in a positive value for $G(X) = W^T X + w_{n+1}$, while points on the other side will give a negative value.

According to the principle of perceptron learning, the weight vector W R can be extended to $W = (w_1, w_2, \dots w_{n+1})^T \in R^{n+1}$ and the feature vector may be extended to $X = (x_1 x_2 \dots x_n 1)^T \in R^{n+1}$also, thus the plane of classification can be expressed as $G(X) = \widehat{W^T}\widehat{X} = 0$. The learning rule (algorithm) for the update of weights is designed as

$$\widehat{W}(t+1) = \widehat{W}(t) + \frac{1}{2}\eta\widehat{X}(t) - \widehat{X}(t)sgn[\widehat{W}^t(t)\widehat{X}(t)] = \begin{cases} W(t) & \widehat{W}^T(t)\widehat{X}(t) > 0 \\ W(t) + \eta\widehat{X}(t) & otherwise \end{cases}$$

where η is the learning rate which may be adjusted properly to improve the convergence efficiency of the learning course.

1.3 实验目标

The goals of the experiment are as follows:

(1) To understand the working of linear perceptron learning algorithm.

(2)To understand the effect of various parameters on the learning rate and convergence of the algorithm.

(3)To understand the effect of data distribution on learnability of the algorithm.

1.4 Contents and Procedure

Stage 1:

(1) According to the above principle and theory in section 1.2, design and compile the programme codes of perceptron learning toward the linear classification of two classes.

(2) Create a linearly separable pattern dataset with more than 50 samples for each one of two classes.

(3) Initialize the weight vector W (0) and select a proper value for the learning rate $\eta \in (0, 1]$.

(4) Run your programme with your dataset and record the final result, pay attention to the number of iterations for convergence.

Stage 2:

Repeat the above procedure (2) (4) in stage 1 for the different datasets with varying amount of separation between two classes of patterns. Note down your observations.

Stage 3:

Study the effect of varying the learning rate η with different amounts of separability.

Stage 4:

Explore the questions in the previous section and design experiments to answer these questions. Complete and submit an experiment report about all experiment results with comparative analysis and a summary of experiences in this experiment study.

1.4 实验代码

编写 C++ 代码对线性分类器进行模拟实验，代码如下：

```
#include<iostream>
#include <stdio.h>
using namespace std;
double hypothose(double w[],int feature_num,double*
    training_set){
     double sum=0;
     for(int i=0;i<feature_num;i++){
          sum+=w[i]*training_set[i];
     }
     if (sum>0) return 1;
     else return 0;
}
void perception(int feature_num,int training_num,double a
    ,int times,double** training_set,double w[]){
```

```
    int dimentions=feature_num+1;
    while(times--){
        double* delta_w=new double[feature_num];
        for(int i=0;i<feature_num;i++){
            delta_w[i]=0;
        }
        for(int i=0;i<training_num;i++){
            for(int j=0;j<feature_num;j++){
                delta_w[j]+=(training_set[i][
                    feature_num]-hypothose(w,
                    feature_num,training_set[i]))*
                    training_set[i][j]*a;
            }
        }
        for(int i=0;i<feature_num;i++){
            w[i]+=delta_w[i];
        }
        delete[] delta_w;
    }
}
int main(){
    int feature_num,training_num,times;
    double a;
    freopen("in.txt","r",stdin);
    while(cin>>feature_num>>training_num>>a>>times){
        double** training_set=new double*[ training_num];
        for(int i=0;i<training_num;i++){
            training_set[i]=new double[training_num+2];
        }
        double* w=new double[feature_num+1];
        for(int i=0;i<training_num;i++){
            training_set[i][0]=1;
        }
        for(int i=0;i<training_num;i++){
            for(int j=1;j<=feature_num+1;j++){
                cin>>training_set[i][j];
            }
        }
        for(int i=0;i<=feature_num;i++){
            cin>>w[i];
        }
        perception(feature_num+1,training_num,a,times,
            training_set,w);
        for(int i=0;i<feature_num;i++){
            cout<<w[i]<<' ';
        }
```

```
        cout<<w[feature_num]<<endl;
        delete[] w;
        for(int i=0;i<training_num;i++){
            delete[] training_set[i];
        }
        delete[] training_set;
    }
    return 0;
}
```

1.5 实验分析

本次试验的主要目的是践行学习过的线性分类器中的一个内容：感知器。这是一种直接得到完整的线性判别函数 $g(x) = \omega^T x + \omega_0$ 的方法，英文为 preception。本次试验首先的内容是先将原本的向量进行增广，将向量 x 增加一维，且增加的一维取为常数，定义

$$y = [1, x_1, x_2, \ldots, x_d]^T$$

定义广义的权向量

$$\alpha = [\omega_0, \omega_1, \omega_2, \ldots \omega_d]^T$$

线性判别函数变为：

$$g(y) = \alpha^T y$$

决策规则变为：如果 g(y)>0, $y \in \omega_1$, 如果 g(y)<0,$y \in \omega_2$

再引入样本集可分性的概念。

队医一组样本 $y_1, y_2, y_3, \ldots$, 根据这样的决策规则，可以得到解向量和解区。但是考虑到可能由噪音而带来的误差，提出了余量的概念，希望不取靠近边缘的解。20 世纪 50 年代 Rosenblatt 提出了感知器准则函数：

$$J_P(\alpha) = \sum(-\alpha^T y_k)$$

要求其最小化，可以使用梯度下降的方法迭代求解，这也是这种方法的核心要义：迭代。第一步的 α_0 取多少值其实没有关系，可以通过一步一步迭代的方法逐步逼近希求的解向量

$$\alpha(t+1) = \alpha(t) - \rho_t \Delta J_p(\alpha)$$

迭代修正的公式为：

$$\alpha(t+1) = \alpha(t) + \rho_t \sum_{\alpha^T y_k \leq 0} y_k$$

重复这个过程，直到对所有的样本而言都有 $\alpha(t)^t y_i > 0$ 成立，得到解向量。对于步长 η 而言，η 决定了收敛的速度。不论样本书目和维数如何，只要解区存在，总可以经过有限步数的迭代得到一个解向量。在下面列出一个实例的表格说明：

训练样本	$W(K)^T x$	修正式	修正后的权值W(K+1)	迭代次数
x_1 1 0 1 1	+	W(0)	1 1 1 1	1
x_2 0 1 1 1	+	W(0)	1 1 1 1	
x_3 1 1 0 1	+	$W(0)-x_3$	0 0 1 0	
x_4 0 1 0 1	0	$W(1)-x_4$	0 -1 1 -1	
x_1 1 0 1 1	0	$W(2)+x_1$	1 -1 2 0	2
x_2 0 1 1 1	+	W(3)	1 -1 2 0	
x_3 1 1 0 1	0	$W(3)-x_3$	0 -2 2 -1	
x_4 0 1 0 1	-	W(4)	0 -2 2 -1	
x_1 1 0 1 1	+	W(4)	0 -2 2 -1	3
x_2 0 1 1 1	-	$W(4)+x_2$	0 -1 3 0	
x_3 1 1 0 1	-	W(5)	0 -1 3 0	
x_4 0 1 0 1	-	W(5)	0 -1 3 0	
x_1 1 0 1 1	+	W(5)	0 -1 3 0	4
x_2 0 1 1 1	+	W(5)	0 -1 3 0	
x_3 1 1 0 1	-	W(5)	0 -1 3 0	
x_4 0 1 0 1	-	W(5)	0 -1 3 0	

Figure 1: example caption

通过一定样本的学习，我们就获得了我们经过学习过后的 $\alpha(t)$。我试着改变了几次 η 的值，发现收敛的速率有所改变，但是当 η 值比较小的时候，收敛速度是比较慢的。当 η 比较大的时候（但是始终保持 η 值在 0 1 之间），我们会发现随着一个样本的加入，改变的幅度比较大，虽然速度比较快，但是不容易收敛。
对于不同的样本集而言，所要求的精度是不同的。根据不同的样本选择不同的 η 值，可以改变收敛的速度以及精度，达到预想效果，这就是线性感知器的关键所在。

电子版 github 地址： https://github.com/Raphael-Sanzio/homework1.git