感知器原理

更新时间：2017-10-02 14:57:35

全站首页

总结自《统计学习方法》《机器学习（周志华）》以及相关博客

http://blog.csdn.net/mosbest/article/details/52029217

http://www.jianshu.com/p/3d316f380041

感知器PLA是一种最简单，最基本的线性分类算法（二分类）。其前提是数据本身是线性可分的。

模型可以定义为$f(x)=sign(wx+b)$，sign函数是阶跃函数，阈值决定取0或1。

模型选择的策略，利用经验损失函数衡量算法性能，由于该算法最后得到一个分离超平面，所以损失函数可以定义为$L(w,b)=-\sum y_i(wx+b)$，由于对于误分类点，yi和wx+b的正负属性相反，所以，所以加一个符号，来表征样例点与超平面的距离（此处不利用误分类点的个数的原因，由于不可导，不易优化）。

算法选择，最终的目标是求损失函数的最小值，利用机器学习中最常用的梯度下降GD或者随机梯度下降SGD来求解（相关优化算法的理解请自行百度）。

SGD算法的流程如下：输入训练集和学习率

1、初始化w0，b0，确定初始化超平面，并确定各样例点是否正确分类（利用yi和wx+b的正负性关系）；

2、随机在误分类点中选择一个样例点，计算L关于w和b在该点处的梯度值；

站内搜索

本版最新

1 o语言的变量、函数、Socks5代理服务器

2 09.27（10.02）

3 Codeforces 846 C Four Segments 前缀和 暴

4 vs2015和VC++6.0中while (scanf("%d", &x) !=

5 Python的SQLAlchemy和ORM

6 python多个变量赋值

7 Java JDBC->Mybatis

8 洛谷P2926 [USACO08DEC]拍头Patting Head

9 VMware Fusion 10序列号

10 WD My Cloud Ex2 Ultra下的SVN（Subversi

11 如何在地址栏（title标签里）和收藏夹里 加上

12 bzoj 1826 缓存交换

13 努比亚 Z17（Nubia NX563J） 解锁BootLoad

14 php中常用的字符串查找函数strstr()、strpos()

15 .NET 使用HttpWebRequest 伪造Request.Url

16 Mybatis mapping文件中 数据封装类使用内部

17 作业20170928—1代码规范，结对要求

18 课程作业02（关于java的几点讨论）

全站首页

3、更新w，b，按照如下方向 $w=w+\eta x_i y_i, b=b+\eta y_i$ ；

4、迭代运行，直到满足停止条件（限定迭代次数或者定义可接受误差最大值）；

可知，初值的选择，误分类点的选择顺序都影响算法的性能和运行时间。PLA是一个很基本的算法，应用场景很受限，只是作为一个引子来了解机器学习，后面有很多高级的算法，比如SVM和MLP，以及大热的deep learning，都是感知器的扩展。

此外，根据novikoff定理，可得到一些结论：1、存在|w|=1的超平面可以将案例分开，且 $y_i(\hat{w}\cdot\hat{x}_i)\geq\gamma$ ；2、 $k\leq(\frac{\max|\hat{x}_i|}{\gamma})^2$ 表示误分类次数；此处w和x是扩展后的样例数据，将权值和1分别连接到原来的w和b后形成一个大向量。

对于PLA，还有一个对偶问题，此处，简单介绍一下对偶问题相关的知识。

对偶问题：

每一个线性规划问题，我们称之为原始问题，都有一个与之对应的线性规划问题我们称之为对偶问题。原始问题与对偶问题的解是对应的，得出一个问题的解，另一个问题的解也就得到了。并且原始问题与对偶问题在形式上存在很简单的对应关系：目标函数对原始问题是极大化,对对偶问题则是极小化

原始问题目标函数中的收益系数（优化函数中变量前面的系数）是对偶问题约束不等式中的右端常数,而原始问题约束不等式中的右端常数则是对偶问题中目标函数的收益系数

原始问题和对偶问题的约束不等式的符号方向相反

原始问题约束不等式系数矩阵转置后即为对偶问题的约束不等式的系数矩阵
原始问题的约束方程数对应于对偶问题的变量数,而原始问题的变量数对应于对偶问题的约束方程数

该类最新

对偶问题的对偶问题是原始问题

总之他们存在着简单的矩阵转置，系数变换的关系。当问题通过对偶变换后经常会呈现许多便利，如约束条件变少、优化变量变少，使得问题的求解证明更加方便计算可能更加方便。

对偶问题中，此处将w和b看成是x和y的函数，w和b可表示为 $w=\sum \alpha_i y_i x_i; b=\sum \alpha_i y_i; \alpha_i = n_i \eta$ ，ni表示更新次数，模型 $f(x)=\mathrm{sign}(\sum \alpha_i y_i x_i + b); \alpha=(\alpha_1, \alpha_2 \ldots \alpha_N)^T$ ，算法流程如下：输入训练集，学习率

1、 $\alpha=0, b=0$ ；

2、随机选取误分类点对，并更新计算 α, b ，具体更新，依据上面的表达式；

3、直至没有误分类点，停止计算，返回相应的参数；

原始问题和对偶问题都是严格可收敛的，在线性可分的条件下，一定可以停止算法运行，会达到结果，存在多个解。

如果线性不可分，可以利用口袋算法，每次迭代更新错误最小的权值，且规定迭代次数。口袋算法基于贪心的思想。他总是让遇到的最好的线拿在自己的手上。。。 就是我首先手里有一条分割线wt，发现他在数据点(xn,yn)上面犯了错误，那我们就纠正这个分割线得到wt+1,我们然后让wt与wt+1遍历所有的数据，看哪条线犯的错误少。如果wt+1犯的错误少，那么就让wt+1替代wt，否则wt不变。 那怎样让算法停下来呢？？——我们就 自己规定迭代的次数 由于口袋算法得到的线越来越好（PLA就不一定了，PLA是最终结果最好，其他情况就说不准了），所以我们就 自己规定迭代的次数 。

以下是PLA简单的代码实现，转自http://blog.csdn.net/u014403897/article/details/45072829

```
#include<iostream>
#include <stdio.h>
```

全站首页

该类最早

本版最新

优秀作者推荐

站点信息

友情链接

马开东博客

马开东云搜索

最近活动

1212双12活动盛大开启，5折优惠惊喜不断

全站首页

```
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;
}
//以下函数为感知器算法真正函数，参数分别是特征个数，训练样本数，学习速率，迭代次数，i
void perception(int feature_num,int training_num,double a,int times,double** training_set,do
     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])
              }
          }
          for(int i=0;i<feature_num;i++){
              w[i]+=delta_w[i];
          }
          delete[] delta_w;
       }
```

全站首页

```
}
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;
}
```

全站首页

至此，第一个具体的算法PLA就已经总结完毕。

此文链接：http://makaidong.com/taojake-ML/55499_908179.html
转载请注明出处：感知器原理

来源：马开东云搜索（微信/QQ：420434200,微信公众号：makaidong-com）
欢迎分享本文，转载请保留出处！

【原文阅读】:http://www.cnblogs.com/taojake-ML/p/6097290.html