# Machine Learning---PNN

## 引言

这篇将介绍machine learning中另一个较为经典的一个算法。之前介绍的几个算法（Backpropagation、LMS……）这些算法有一个明显的特点就是，它们都需要先训练算法，都会通过训练数据集调整算法的内部参数。

这次介绍的算法就摆脱了这种过程。

## PNN算法基本介绍

### 1.算法思想

PNN(Probabilistic Neural Networks)是一种较为出色的分类算法。

算法利用已分类的数据集，对未分类数据进行计算，得到“激活值”（Activation），在找出利用Winner Take All的原则，找到分类。

对于如何计算激活值下面会进行详细介绍。

### 2.激活值计算公式

隐藏层节点激活值，这里的隐藏层也便是样本集合（已分类的数据集）。

公式（1）

上诉中是隐藏层第i个节点的激活值， 便是已分类的数据集，F表示带分类的数据。

输出端节点的激活值。

公式（2）

其中便是第j个分类的输出端激活值；N表示样本数据（已分类）的数量；便是上面计算的隐藏层节点的激活值；是光滑参数。

### 3.算法流程

对于这个算法的流程也是比较好理解：

1.遍历类别；

2.遍历类别中的样本数据；

3.利用公式（1）对隐藏层节点激活值进行计算

4.利用公式（2）对当前分类，计算其激活值；

5.对所有类别激活值，进行比较，找出最大的，并返回类别序号。

## 算法实现

在这里提供这个算法的实现代码：

int pnn\_classify()

{

double output[CLASSNUM];

for(int c = 0 ; c < CLASSNUM ; ++c)

{

output[c] = 0.0;

for(int e = 0 ; e < EXAMPLENUM ; ++e)

{

double h = 0.0;

for(int d = 0 ; d < DIMENSIONALITY ; ++d)

{

h += example[d] \* dataset[c].example[e].f[d];

}

output[c] = exp((h-1.0)/pow(sigma,2));

}

output[c] = output[c] / (double)EXAMPLENUM;

}

//winner take all

double maxActivation = 0.0;

int iCMax = 0;

for (int c = 0 ; c < CLASSNUM ; ++c)

{

if(maxActivation < output[c])

{

iCMax = c;

maxActivation = output[c];

}

}

return iCMax;

}

## 总结

由于笔者不是专门研究人工智能方面，所以在写这些文章的时候，肯定会有一些错误，也请谅解，上面介绍中有什么错误或者不当地方，敬请指出，不甚欢迎。

如果有兴趣的可以留言，一起交流一下算法学习的心得。

声明：本文章是笔者整理资料所得原创文章，如转载需注明出处，谢谢。