KNN的简单实践，我们可以使用CIFAR-10的图像，图像本身是32x32x3的数据,我们可以使用matlab或者c++或者python等语言实现这一目标，我们先采用伪代码的形式来完成这一目标。首先回顾下算法的核心思想，找出k个最近邻，然后采权值投票的形式作为分类结果，简单的投票可以认为每一个近邻的权值都是1。（顺便说一句这一章是我在去领证的火车上完成的C:\Users\云峰\Desktop\20151219225021.png，弄机器学习的人应该一起逗比下，不然会被代码无聊死的）。

我们实现下面几个简单的函数

函数读取图像将图像拉直成向量：

Function LoadImageData（image\_path）

{

Matrix image = ReadImage(image\_path);//读取图像

image = image.reshape(1,32x32x3);//将图像拉直成一个向量

image = image./255.0;//将图像的每个像素变成float防止数据溢出。

return image;

}

函数计算两个向量之间的距离:

Function CaculateDistance(image1,image2)

{

distance = Distance(image1,image2);

return distance;

}，

Distance是一个伪函数，这是一个计算两个向量之间距离的函数，因为在具体实践中，我们可能会采用各种不同的距离函数，比如欧式距离和街区距离，甚至余弦距离，几种不同的方式都可能导致我们使用的方案不同。

这里给大家实现这三种距离

欧式距离

Function EuclideanDistance(vector1,vector2)

{

vector\_minus = abs((vector1-vector2));

distanceL2 = sum(vector\_minus\*vector\_minus);

distance\_euclidean = sqrt(distanceL2);

return distance\_euclidean;

}

街区距离

Function ManhattanDistance(vector1,vector2)

{

vector\_minus = abs((vector1-vector2));

distance\_manhattan = sum(vector\_minus);

return distance\_manhattan;

}

余弦距离

Function CosDistance(vector1,vector2)

{

vector\_normalize1 = normalize(vector1);

vector\_normalize2 = normalize(vector2);

vector\_normalize\_minus=1- vector\_normalize1\*vector\_normalize2;

return vector\_normalize\_minus;

}

排序函数：

Function Sort(input\_distance\_vector\_pair , output\_distance\_vector\_pair)

{

output\_distance\_vector\_pair = parse\_sort(input\_distance\_vector\_pair);

}

选取前K个近邻函数

Function SelectTopK (k , input\_distance\_vector\_pair , output\_distance\_vector\_pair)

{

For(int i=0;i<k;i++)

{

output\_distance\_vector\_pair.push(input\_distance\_vector\_pair[i]);

}

}

到此为止我们的函数已经实现完毕，现在我们来构建整个系统，我们来写一些辅助功能的代码。

Function ListAllImageInMemory(image\_path\_root , list\_vectors)

{

child\_list = list(image\_path\_root);

foreach(current\_path in childlist)

{

If( Isdirectory( current\_path))

{

ListAllImageInMemory(current\_path,list\_vector)

}

else if(IsImageFile(current\_path))

{

tmp\_vector = LoadImageData(current\_path)

label = GetLabelFromPath(current\_path);

tmp\_vector\_pair = {tmp\_vector, label };

list\_vectors.push(tmp\_vector\_pair);

}

else

{

}

}

}

这样我们的所有数据就已经加载到内存中了，然后我们对外提供一个服务，这个服务来完成新来的图像进行分类的算法，这服务的单次服务内容如下：

Function Service(k , input\_image, output\_label)

{

all\_distance\_pair;

foreach (current\_pair in list\_vectors)

{

current\_distance = CaculateDistance(image, current\_pair.vector);

distance\_pair={current\_pair.Label, current\_distance };

all\_distance\_pair.push(distance\_pair);

}

Sort(all\_distance\_pair,all\_distance\_pair);

SelectTopK(k,all\_distance\_pair,output\_final\_pair);

output\_label = Ticket(output\_final\_pair , vector\_weight);

}

权重投票函数：

Function Ticket(output\_final\_pair , vector\_weight)

{

pair\_score;

position(0);

foreach pair in output\_final\_pair

if pair.label in pair\_score

pair\_score[pair.label] += vector\_weight [position]

end

position++;

end

};

这样整个KNN的从理论到实践的转化就有了全部的支撑。到现在为止我们已经有了伪代码了，我们现在来看下真正的代码应该是什么样子的。我们先来一段c++的代码来实现以上函数，我们需要使用一些辅助的库来实现这一过程。这里我们使用几个库来完成这一过程，我们会使用Opencv，boost，poco等开源库来完成这一工程代码。

下面我们就开始完成我们的代码吧。

­­