ANN人工神经网络分类算法python实现

安佑风险...
www.anyourmc.com

今天我们来看看ANN人工神经网络的分类算法python 实现。和前面几篇文章一样，重点在理解这些算法的设计思路。在实际运用中，可以拆解开自己设计。

先看看这套算法的设计流程：

1、随机生成一个包含输入层、隐藏层和输出层的数据结构，发明的人，称为网络结构。 值得注意的是，这里的网络结构设计是主观设计的，也就是超参，需要先设计好输入层的权重数量，隐藏层的权重数量，和输出层的权重数量。这里的随机主要是指权重的随机性。

2、选用适当的链接（激活）函数，将参与前面权重计算的数据转化为下一层的输入值。

3、通过反向传播，其实就是将输出值和期望值之间的误差乘以激活函数的导数，传回输入层，进而为权重的调整提供一个算法依据，目标函数为尽可能使得加权的误差平方和缩小。

4、权重调整后的网络结构达到一定条件后，可以用于预测。

下面来看看代码。数学原理和代码实现之间的距离就是对数据结构的熟悉和运算框架的搭建的差异。完全scratch 非常烧脑。

```
fromrandomimportseed

fromrandomimportrandom

frommathimportexp

definitialize_network(n_inputs,n_hidden,n_outputs):

network = list()

hidden_layer = [{'weights':[random() foriinrange(n_inputs+1)]} foriinrange(n_hidden)]

network.append(hidden_layer)

output_layer =[{'weights':[random() foriinrange(n_hidden+1)]} foriinrange(n_outputs)]

network.append(output_layer)

returnnetwork
```

层数量

```
seed(1)

network = initialize_network(2,1,2)

forlayerinnetwork:

print(layer)
```

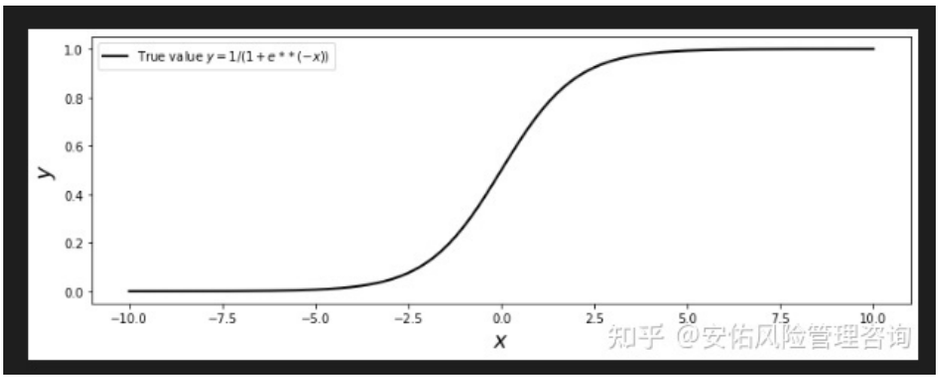
```
[{'weights': [0.13436424411240122, 0.8474337369372327, 0.763774618976614]}]
[{'weights': [0.2550690257394217, 0.49543508709194095]}, {'weights':
[0.4494910647887381, 0.651592972722763]}]
```

从运行结果可以看到，这是一个，两层的结构。权重的最后一项是偏差值的权重。可以看到例子中，变量的维度是2.

```
defactivate(weights,inputs):

activation = weights[-1]

foriinrange(len(weights)-1):

activation += weights[i]*inputs[i]

returnactivation#标量
```

```
deftransfer(activation):

return1/(1+ exp(-activation)) #标量
```

接着看到这里，这是运算，数据输入后，经过权重的线性变换后，还要经过激活函数的转化。这里采用的sigmoid 函数。



该函数的值域是[0,1]。实现代码如下：

```
x = np.linspace(-10,10,1000)

y_exact = 1/(1+np.exp(-x))

fig, ax = plt.subplots(figsize =(12,4))

ax.plot(x,y_exact,'k', lw=2,label = 'True value $y=1/(1+e**(-x))$')

ax.set_xlabel(r'$x$',fontsize =18)

ax.set_ylabel(r'$y$',fontsize =18)

ax.legend(loc =2)
```

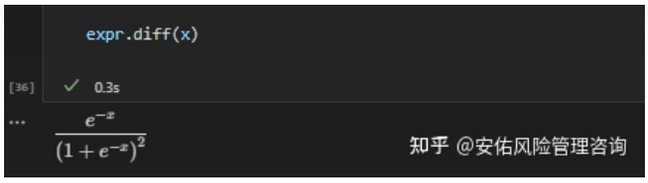
接下来，需要通过上面的计算，让输出层得到输出值。这里人为设计是两次sigmoid .如果需要修改第一次为tanh（x）,就需要在这个函数的第一层加入该函数的计算函数。

```
defforward_propagate(network,row):

inputs = row

forlayerinnetwork: # 先从输入层到其他层，其他层包括了隐藏层和输出层new_inputs = [] #记录输入层的output

forneuroninlayer:

activation = activate(neuron['weights'],inputs)

neuron['output']= transfer(activation)

new_inputs.append(neuron['output'])

inputs = new_inputs#在隐藏层和输出层进行input迭代returninputs#进入隐藏层和输出层的output
```

接下来是调整权重。

```
deftransfer_derivative(output) : #sigmoid 函数的斜率（导数）returnoutput*(1-output) # 导数因子分解，用y来表示
```

这个计算来自sigmoid 的导数。



通过因式分解很容易得到上面的计算公式。

下面进入计算最烧神经的部分。反向计算的代码。

defbackward_propagate_error (network,expected):

foriinreversed (range(len(network))):

layer = network[i] #倒序输出层，隐藏层，输入层

errors = list() #换层清零

ifi != len(network)-1: #这是第三步

forjinrange(len(layer)): # 输入层

error = 0

forneuroninnetwork[i+1]: #隐藏层和输出层的神经元

error += (neuron['weights'][j]*neuron['delta']) #隐藏层和输出层的第一个权重乘以delta

errors.append(error) # 记录误差else: #先计算这步

forjinrange(len(layer)):

neuron =layer[j] #隐藏层和输出层的神经元

errors.append(neuron['output']-expected[j]) #输出层误差

forjinrange(len(layer)): #这是第二步和第四步

neuron = layer[j]

neuron['delta']= errors[j]*transfer_derivative(neuron['output']) #在神经元中记录这个误差*导数 先记录的隐藏层和输出层，再记录输入层 returnnetwork

笔者标注了，先计算输出层的误差，再通过导数运算乘以误差，并通过delta 记录在网络中。

然后再返回输入层，调用输出层的delta 数据和权重数据，进行加权求和，记录为输入层的误差，再和输入层数据计算的导数相乘得到输如层的delta数据。

这为下一步调整权重做好了数据的准备。

inputs = row [:-1] #倒数第二个元素往前全选 输入层

ifi != 0: #不是输入层inputs = [neuron['output'] forneuroninnetwork[i-1]]

forneuroninnetwork[i]:

forjinrange(len(inputs)):

neuron['weights'][j] -= l_rate*neuron['delta']*inputs[j]

neuron['weights'][-1] -= l_rate*neuron['delta'] #假设bias的inputs为1

权重调整公式，不同的书上计算方法不一样，这里采用了一个l_rate ，也是人工设计的超参。下面看看设定为0.3的运算结果。

接下来，可以训练网络了。

deftrain_network(network,train ,l_rate,n_epoch,n_outputs):

forepochinrange(n_epoch):

sum_error = 0

forrowintrain :

outputs = forward_propagate(network,row)

expected = [0foriinrange(n_outputs)] #生成期望值根据期望值的个数先生成全为0的list

expected[row[-1]] = 1#再根据每行的标签值来修改，如果标签为0 ，则修改第一个元素，如果标签值为1，修改第二个元素，以此类推

sum_error += sum([(expected[i]-outputs[i])**2foriinrange(len(expected))]) # 计算各输出值的误差平方和

backward_propagate_error(network,expected)

update_weights(network,row,l_rate)print('>epoch=%d,lrate=%.3f,error =%.3f' %(epoch,l_rate,sum_error))

可以很明显看到，error 随着迭代逐渐减小。但是要减小到什么程度，算法中没有明确给出标准。这里需要注意在实际计算中，可能需要设定一个阈值。

最后就是预测。

```
defpredict(network,row):

outputs = forward_propagate(network,row)

returnoutputs.index(max(outputs)) #最大值的索引
```

整套算法中，数学难度不大，就不再后面推导数学公式。

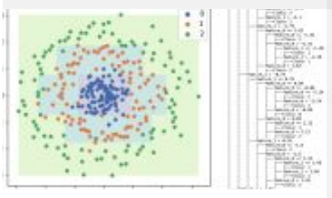
编辑于 2022-05-05 20:46

分类算法 机器学习算法 数学建模

文章被以下专栏收录

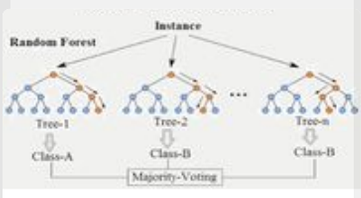
数据分析
数字经营，离不开数据分析

推荐阅读



ython手写决策树并应对过度以合问题

leeph... 发表于deeph...



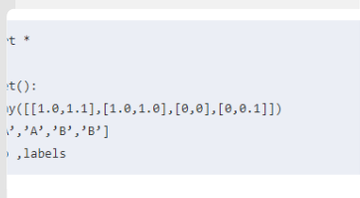
Python随机森林算法入门和性能评估

statr 发表于实战统计学

CNN可视化之类激活热力图 Grad-CAM

船长黑板报所有文章和代码的最新版本均在 Captain1986/CaptainBlackboard 维护，知乎不做维护 您的Star是对我的鼓励## 引言 卷积神经网络因为其在很多任务上效果很好但是...

船长 发表于船长黑板报



k-近邻算法的Python实现

Pytho... 发表于Pytho...

还没有评论

写下你的评论...

▲ 赞同 3 ▼ 添加评论 分享 喜欢 收藏 申请转载 ···