网页文本类型识别以及根据网页链接关系的改进

学号-20142112280-谢非

2016 年 12 月 21 日

目录

1 实验目的

1. 熟悉了解具体神经网络的实现
2. 了解应用向量空间模型进行文本特征抽取的原理

2 实验要求

1. 应用向量空间模型 (VSM) 以及 BP 神经网络实现对网页类型的预测
2. 力求分类正确率的提升

3 实验环境

实验代码使用 python3.5 实现，主要的 python 库有处理网页的 BeatifulSoup 和用于数组计算的 numpy.

4 实验原理

4.1 网页元数据处理

首先将每个网页的内容分词后计算频次，统计出一个字典结构，键值是单词，值是出现的频次. 在依次统计每个网页的同时，也把所有的网页的单词，频次统计出来.

1. 根据淘汰率，淘汰掉所有网页中出现次数较多的单词，默认淘汰率为 0.02
2. 总单词和词频在经过淘汰过滤之后，根据设置的文本特征向量维度，从中从大到小依次获取文本特征向量属性值（即对应单词）
3. 这实际上是第一次网页数据处理，之后要获取训练数据和测试数据还需要根据现有的总体文本向量属性来依次从每个网页数据中提取特征向量.

最终，将由网页数据中提取出来文本属性的特征向量，即 BP 神经网络的输入参数.

4.2 BP 神经网络的构造

[1] 从网上查找资料，并在学习 numpy 库的具体使用的同时，实现了一个简单的神经网络. 默认学习速率是 0.1, 循环执行 10000 遍.

```
class NeuralNetWork:

    def __init__(self, layers, activation, activation_deriv):
        "init a neural network"

        # config activation
        self.activation = activation
        self.activation_deriv = activation_deriv

        # init weights by layers
        self.weights = []
        for i in range(1, len(layers) - 1):
            self.weights.append(
                (2 * np.random.random((layers[i - 1] + 1, layers[i] + 1)) - 1) * 0.25
            self.weights.append(
                (2 * np.random.random((layers[i] + 1, layers[i + 1])) - 1) * 0.25)

    def train(self, trainData, trainOut, learinRate=0.1, epochs=10000):
        ''' training my net args '''
        trainData = np.atleast_2d(trainData)
        tempData = np.ones([trainData.shape[0], trainData.shape[1] + 1])
```

[1] 由于对神经网络的了解并不是特别深入，此次只实现了一个三层的神经网络 (觉得似乎变一变就可以扩展成多层了)，输入层节点数可根据文本特征向量的长度变化，隐藏层节点可以自由设置，输出层节点也可自由变化，不过实际训练数据时输出向量长度为 7

```
        tempData[:, 0:-1] = trainData
        trainData = tempData
        trainOut = np.array(trainOut)

        for times in range(epochs):
            # get a random inputData
            inputFlag = np.random.randint(trainData.shape[0])
            inputData = [trainData[inputFlag]]

            for curLayer in range(len(self.weights)):
                inputData.append(self.activation(
                    np.dot(inputData[curLayer], self.weights[curLayer])))
            error = trainOut[inputFlag] - inputData[-1]
            deltas = [error * self.activation_deriv(inputData[-1])]
            for l in range(len(inputData) - 2, 0, -1):
                deltas.append(deltas[-1].dot(self.weights[l].T)
                              * self.activation_deriv(inputData[l]))
            deltas.reverse()

            for i in range(len(self.weights)):
                layer = np.atleast_2d(inputData[i])
                delta = np.atleast_2d(deltas[i])
                self.weights[i] += learinRate * layer.T.dot(delta)
```

5 实验过程中发现的问题

实际测试过程中发现输出数据普遍偏向向量 [0,1,0,0,0,0,0] (类别为 other), 经过探索发现, 8282 个网页中存在 3159 个 other 类别的网页, 7 类数据中 1 类占据了过大的比例, 个人认为这可能是测试前期正确率低下的原因.

6 实验测试效果

经过超参的多次修改 [2] , 得到最高的测试正确率为 85.48% 输出附件见附件
此时神经网络维度为 500*40*7, 学习速率为 0.1, 循环次数 10000, 文本高频词淘汰率为 0.01

7 添加网页关系因素后的测试结果

实验后期, 检查网页中超链接的关系后觉得数据提供的网页关系并不十分明确, 于是转而采取了另一种类别关系的附加方法. 在训练过程中, 直接在已知类别的情况下检索网页内文本 " 分类 " 关键字 [3]的数量, 对该网页的整体权重属性进行相应增加 (具体实现时即添加了类别关键字出现的频次).

```
def setAddWeight(self,page):
    kindPattern=re.compile(self.kind)
    # 查找类别关键字的数目, 在生成网页文本特征向量时会添加到权重中
    self.addWeight=len(re.findall(kindPattern,page))
```

经过超参的多次修改, 得到最高的测试正确率为 86.22% [4]
测试输出附件 此时神经网络维度为 500*35*7, 学习速率为 0.1, 循环次数 10000, 文本高频词淘汰率为 0.01

8 实验总结

此次实验过程中, 我对神经网络的具体实现过程有了更深的理解, 了解到许多机器学习, 深度学习方面的知识. 对于文本特征的提取, 有了比较系统的认识. 实验过程中对于神经网络训练过程中的参数调整还不是很了解 [5], 同时觉得实验数据并不是很合理, 期待以后有时间自已做一个比较全面的网页内容分类程序.

[2] 似乎超参也可以让它自己学习
[3] 标示类别
[4] 实际测试输出见附件 test4.txt
[5] 实际测试时就是感觉改变了某些参数之后正确率提高, 便尝试继续修改该参数