基于Tensorflow的病害猪肉预测

一．实验环境：

实验环境是基于虚拟机VMware上安装的Ubuntn16.04,Python版本是2.7，Tensorflow版本是1.3。

二．实验步骤

1.数据预处理：首先用numpy.load将数据读入，通过对数据进行分析，发现训练数据一共是334条，特征向量为10000维的数据。其中119条数据为label=0的数据，即健康猪肉，另外215条数据为label=1的数据，即病死猪肉。然后进行数据预处理，采用numpy.vstack和numpy.hstack将数据预处理为tensorflow框架的输入格式。结果如下：

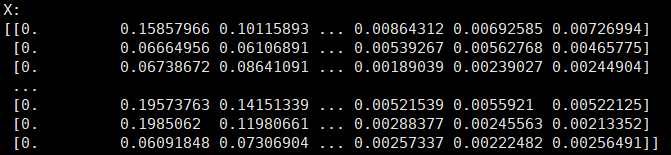


图1：预处理的特征向量

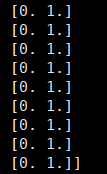
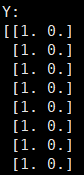


图2：预处理后标签Y

其中，标签为[1,0]的向量标记为label=0，即健康的猪肉，标签为[0，1]的向量标记为label=1，即病死的猪肉。

2.采用Tensorflow搭建神经网络框架：通过tf.Variable设置神经网络的连接权值weights和偏置biases。通过tf.reduce\_sum函数来指定损失函数为预测值和真实值的交叉熵，然后通过梯度下降优化器tf.GradientDescent来最小化交叉熵。确定学习率为0.001，训练轮次为4000轮，每轮训练8条数据。

3.实验结果：通过Tensorflow搭建的神经网络，设置每100轮输出结果，最终训练数据的准确率达到98.50%，测试数据的准确率达到85%。实验结果图如下（输出轮次较多，仅截取后10轮）：

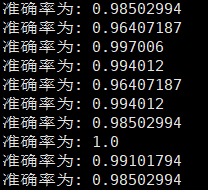


图3：训练数据准确率



图4：测试数据准确率

4.实验总结：通过上述实验，可以发现对于输入数据的特征向量为10000维的数据，神经网络仍可达到较好的拟合效果。对于测试数据为的准确率仅为85%，而训练数据的准确率高达98%，说明该网络对训练数据的学习过于深入，局部点陷入过拟合，导致鲁棒性较差。后期改进可以通过对损失函数加入惩罚项，即正则化的方式，还有采用随机梯度下降方式，并且对神经网络的连接参数进行一定的修改，可以一定程度上解决这个问题。

附录(实验代码)：

*# coding=utf-8*

**import** **tensorflow** **as** **tf**

**import** **numpy** **as** **np**

*# 训练数据*

data = np.load("train0.npy") *# m=119, n=10000*

data1 = np.load("train1.npy") *# m=215, n=10000*

X = np.vstack([data,data1])

y1 = np.zeros((119,1))

y1temp = np.ones((119,1))

y1 = np.hstack((y1temp,y1))

y2 = np.ones((215,1))

y2temp = np.zeros((215,1))

y2 = np.hstack((y2temp,y2))

Y = np.vstack((y1,y2))

*# 测试数据*

testdata0 = np.load("test0a.npy") *# m1= 20,n1= 10000*

testdata1 = np.load("test1a.npy") *# m1= 20,n1= 10000*

xt = np.vstack((testdata0,testdata1))

yt1 = np.zeros((20,1))

yt1temp = np.ones((20,1))

yt1 = np.hstack((yt1temp,yt1))

yt2 = np.ones((20,1))

yt2temp = np.zeros((20,1))

yt2 = np.hstack((yt2temp,yt2))

yt = np.vstack((yt1,yt2))

*# 设置权重weights和偏置biases作为优化变量，初始值设为0*

weights = tf.Variable(tf.zeros([10000, 2]))

biases = tf.Variable(tf.zeros([2]))

*# 构建神经网络模型*

x = tf.placeholder("float", [None, 10000])

y = tf.nn.softmax(tf.matmul(x, weights) + biases)*# 模型的预测值*

y\_real = tf.placeholder("float", [None, 2])*# 真实值*

*# 预测值与真实值的交叉熵*

cross\_entropy = -tf.reduce\_sum(y\_real \* tf.log(y))

*# 使用梯度下降优化器最小化交叉熵*

train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross\_entropy)

**print** "X:**\n**",X

**print** "Y:**\n**",Y

m, n = np.shape(X)

**print** "m=",m

**print** "n=",n

*# 开始训练*

init = tf.global\_variables\_initializer()

sess = tf.Session()

sess.run(init)

BATCH\_SIZE = 8

**for** i **in** range(4000):

start = (i\*BATCH\_SIZE) % 334

end = start + BATCH\_SIZE

sess.run(train\_step, feed\_dict={x: X[start:end], y\_real:Y[start:end]})

**if** i % 100 == 0:

correct\_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y\_real, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, "float"))

**print** "准确率为:",

**print** sess.run(accuracy, feed\_dict={x:xt, y\_real:yt})