机器学习与模式识别课程报告

题目：使用 TensorFlow 构建 Logistic 回归分类器

组　　员：王倩妮

班　　级：交通 2015-02 班

学　　号：2015112956

任课教师：郝 莉

2018 年 4 月 24 日

一、关于 TensorFlow

TensorFlow 是谷歌基于 DistBelief 进行研发的第二代人工智能学习系统，其命名来源于本身的运行原理。Tensor（张量）意味着 N 维数组，Flow（流）意味着基于数据流图的计算，TensorFlow 为张量从流图的一端流动到另一端计算过程。TensorFlow 是将复杂的数据结构传输至人工智能神经网中进行分析和处理过程的系统。因此 TensorFlow 适合进行深度学习工作，如神经网络的构建。

TensorFlow 作为第三方库，与 Python 语言有类似之处，但在某些方面存在差异，比如其数据流图的特征，使其需要先将变量初始化后再使用 sess.run 进行运算。由于 TensorFlow 有数据流图（Graph）的概念，我们未来也可以通过 Tensorboard 查看数据的流动方式。

二、Logistic 回归问题描述

利用 Logistic 回归方法，依据影响房子的居住的因素来对于房子是否有人居住进行分类。

三、数据准备

3.1 数据描述

数据来源于 UCI 机器学习库。数据包含以下项目：

名称	类型	描述
date time	非特征	year-month-day hour:minute:second
Temperature	特征	in Celsius
Relative Humidity	特征	%
Light	特征	in Lux
CO2	特征	in ppm
Humidity Ratio	特征	Derived quantity from temperature and relative humidity, in kgwater-vapor/kg-air
Occupancy	标签	0 or 1, 0 for not occupied, 1 for occupied status

3.2 数据预处理

首先读入数据，查看数据的缺失状况，经验证发现，数据未存在缺失状况。

接下来去除与分类无关的时间数据，选取"Temperature","Humidity", "Light", "CO2", "HumidityRatio"5 项作为特征。

随后将数据集随机按照 3/4 与 1/4 的比例划分为训练集特征 X_train、测试集特征

X_test、训练集标签 y_train、测试集标签 y_test。

3.3 数据可视化

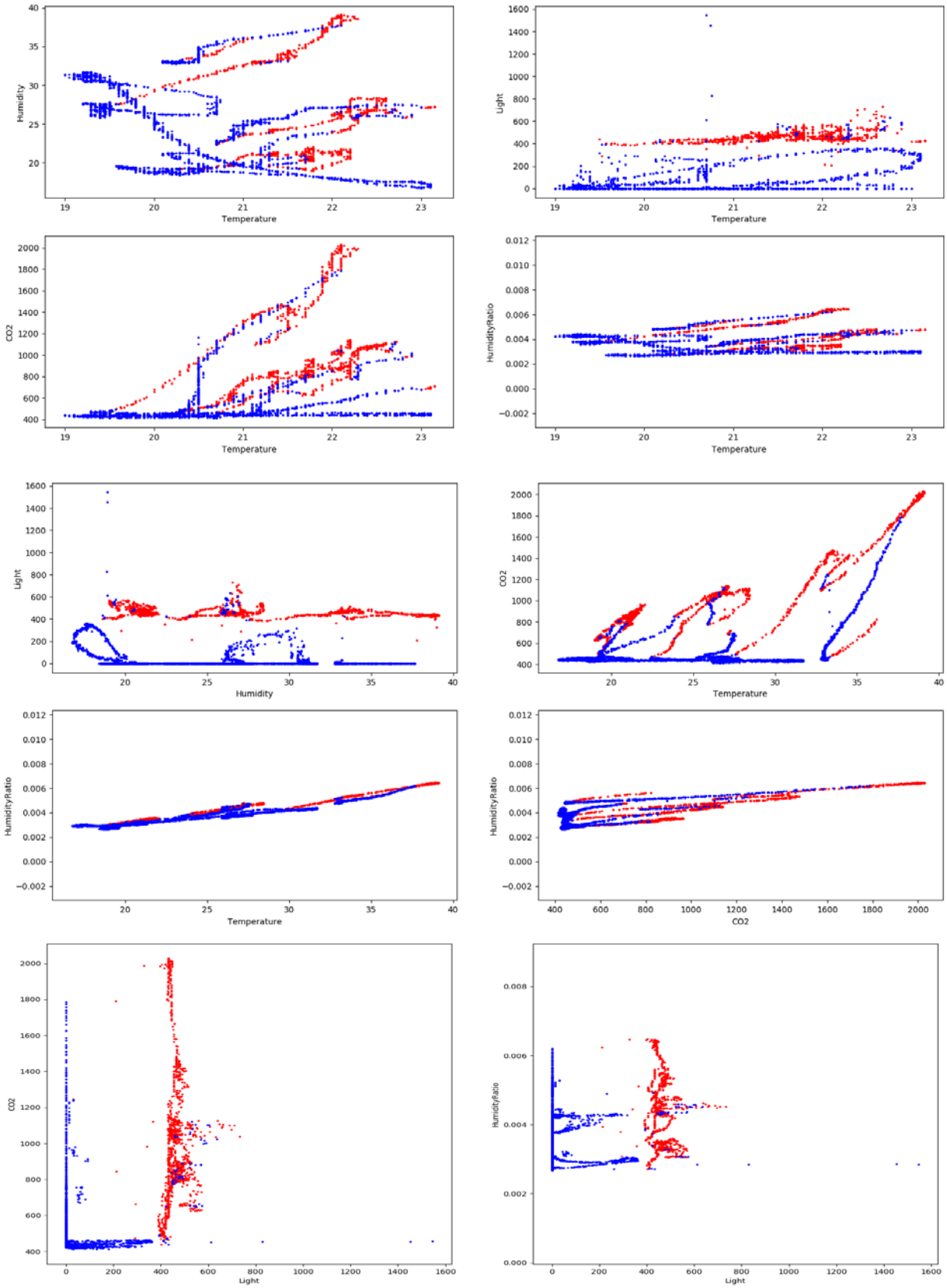

图 1 各特征值散点图

四、利用 TensorFlow 进行建模

4.1 TensorFlow 构建一般步骤

1) 定义特征
2) 标签 one-hot 编码
3) 参数设置(learning_rate、training_epochs、batch、W、b、x、y)
4) 代价函数定义
5) 梯度下降定义
6) 准确率定义
7) 变量初始化(通过 sess.run)
8) 数据流图构建及将数据输入构建好的数据流图进行训练(通过 sess.run)

4.2 使用 TensorFlow 构建 Logistic 回归分类器

TensorFlow 是基于 Python 等平台的第三方库，因此存在一些专用函数，为了简化编码过程，在初始时通过 import tensorflow as tf 导入此库，之后通过 tf+函数名进行调用。

- One-hot 编码

利用 TensorFlow 进行机器学习务必将标签转化为 one-hot 形式，one-hot 形式与类别数目有关，如划分为两类，则为[0,1]与[1,0]，如划分为四类，则为[1,0,0,0]; [0,1,0,0]; [0,0,1,0]; [0,0,0,1]。由于 TensorFlow 默认的处理数据类型为 int32 或 float32，因此需要在进行 one-hot 转换前进行一步转换。

```
y_train=tf.to_int32(y_train)#需要一步类型转换
y_test=tf.to_int32(y_test)#需要类型转换
y_train = tf.concat(axis=1,values=[1 - y_train, y_train])
y_test = tf.concat(axis=1,values=[1 - y_test, y_test])
```

- 机器学习参数设置

在机器学习过程进行前一般会进行参数设置。设置 learning_rate 学习率，其概念类似梯度下降中的步长，设置过大可能跨过最小值区域，设置过小可能在循环步骤结束后依然离最小值较远。设置循环次数 training_epochs，以确定进行多少次优化步骤，如果设置过小则可能还未收敛，如果设置过大可能循环步骤太多，增加时空开销。此外也可以使用“batch”的方法进行梯度下降的处理，这种方法可以在一定程度上提高效率和准确率。

此外需在此步骤中利用 placeholder 设置 x、y，此步骤的作用是固定住了 x 与 y 的模式。

```
learning_rate = 0.001
training_epochs = 50
batch_size = 100
display_step = 1
n_samples = X_train.shape[0]
n_features = 5
```

```
n_class = 2
x = tf.placeholder(tf.float32, [None, n_features])
y = tf.placeholder(tf.float32, [None, n_class])
```

- 模型参数

输入 Logistic 函数的值可以看做是特征向量与系数 W 矩阵的乘积与偏移量 b 的累加，因此此处定义 W 与 b，并按照线性的形式定义预测值，输入 Logistic 函数的过程在代价函数定义中实现。

```
W = tf.Variable(tf.zeros([n_features, n_class]))
b = tf.Variable(tf.zeros([n_class]))
pred = tf.matmul(x, W) + b
```

- 代价函数定义

梯度下降算法是以降低代价函数为目标的，因此代价函数的定义尤为重要，一般来讲在 TensorFlow 中一般采用交叉熵的定义方式，针对 Logistic 回归，交叉熵的定义公式为：

$$\mathrm{j}(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log h_\theta\left(x^{(i)}\right) + (1-y^{(i)})\log(1-h_\theta\left(x^{(i)}\right))\right]$$

当然在 TensorFlow 中已预设了内置函数可以直接调用。

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred,labels=y))
```

- 梯度下降定义

此处调用函数定义梯度下降优化器。

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

- 准确率定义

准确率定义分为统计正确个数与计算准确率两个步骤。

```
correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

- 变量初始化

利用 TensorFlow 进行数据流图建立时，需通过变量初始化步骤使上述变量定义发挥作用。

```
init = tf.initialize_all_variables()
```

- 数据流图构建及将数据输入构建好的数据流图中进行训练

数据流图需要通过 sess.run 进行运行，利用 for 循环遍历 50 次执行优化，并计算每个 batch 的平均准确率，在这一过程中要把值“喂”给 x、y 在 placeholder 固定住的模式。

```
with tf.Session() as sess:
    sess.run(init)
    for epoch in range(training_epochs):
        avg_cost = 0
        total_batch = int(n_samples / batch_size)
        for i in range(total_batch):
            _, c = sess.run([optimizer, cost],
                            feed_dict={x: X_train[i * batch_size : (i+1) * batch_size],
                                       y: y_train[i * batch_size : (i+1) * batch_size, :].eval()})
            avg_cost = c / total_batch
        plt.plot(epoch+1, avg_cost, 'co')
```

```
        if (epoch+1) % display_step == 0:
            print("Epoch:", "%04d" % (epoch+1), "cost=", avg_cost)
print("Optimization Finished!")
print("Testing Accuracy:", accuracy.eval({x: X_train, y:y_train.eval()}))
```

五、模型结果

5.1 Logistic 回归分类结果

设定循环步骤为 50 步，输出每次优化参数后的代价（cost）散点图结果如下图所示，可见 cost 呈现逐渐降低并在较低水平波动的趋势。50 步循环后，准确率达 97.3473%。

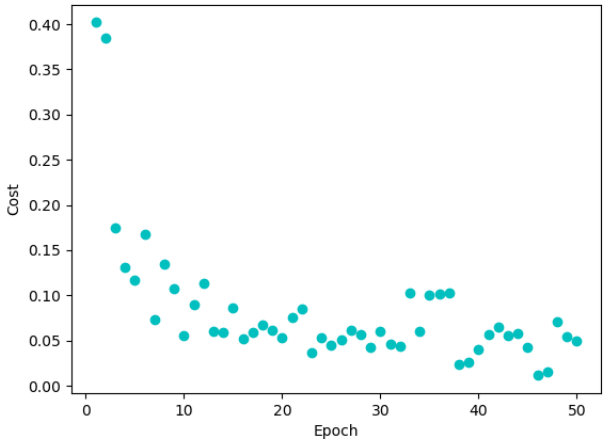

图 2 cost 变化散点图

六、优缺点分析与心得体会

使用 TensorFlow 进行 Logistic 回归运行速度较慢，每一步优化都需要等待较长时间，且学习率等参数的设定需要多次尝试，否则很难得到理想结果，且易出现类似 cost=nan 或 cost 波动的情况。

不过 TensorFlow 在神经网络构建方面大大简化了神经网络构建的难度，且可以通过 Tensorboard 较为清晰地展示数据流与结构，可以在未来的学习过程中多多尝试。

注：本案例来自于 https://blog.csdn.net/u010099080/article/details/53054519 ，由于使用该方法进行疝气病马生死分类效果不够理想（正确率 60%左右），因此借助本案例原始数据集进行以上测试。