School of Data and Computer Science, Sun Yat-sen University

E14 BP Algorithm (C++/Python)

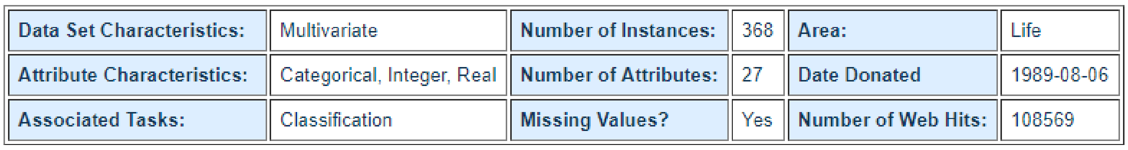
18364066 Lu Yanzuo

December 8, 2020

Contents

1 Horse Colic Data Set

The description of the horse colic data set (http://archive.ics.uci.edu/ml/datasets/Horse+Colic) is as follows:



Data Set Characteristics:	Multivariate	**Number of Instances:**	368	**Area:**	Life
Attribute Characteristics:	Categorical, Integer, Real	**Number of Attributes:**	27	**Date Donated**	1989-08-06
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	108569

We aim at trying to predict if a horse with colic will live or die.

Note that we should deal with missing values in the data! Here are some options:

- Use the feature’ s mean value from all the available data.
- Fill in the unknown with a special value like -1.
- Ignore the instance.
- Use a mean value from similar items.
- Use another machine learning algorithm to predict the value.

2 Reference Materials

1. Stanford: **CS231n: Convolutional Neural Networks for Visual Recognition** by Fei-Fei Li,etc.
   - Course website: http://cs231n.stanford.edu/2017/syllabus.html
   - Video website: https://www.bilibili.com/video/av17204303/?p=9&tdsourcetag=s_pctim_aiomsg
2. **Machine Learning** by Hung-yi Lee
   - Course website: http://speech.ee.ntu.edu.tw/~tlkagk/index.html
   - Video website: https://www.bilibili.com/video/av9770302/from=search
3. A Simple neural network code template

```
# -*- coding: utf-8 -*
import random
import math


# Shorthand:
# "pd_" as a variable prefix means "partial derivative"
# "d_" as a variable prefix means "derivative"
# "_wrt_" is shorthand for "with respect to"
# "w_ho" and "w_ih" are the index of weights from hidden to output
   layer neurons and input to hidden layer neurons respectively

class NeuralNetwork:
    LEARNING_RATE = 0.5
    def __init__(self, num_inputs, num_hidden, num_outputs,
       hidden_layer_weights = None, hidden_layer_bias = None,
       output_layer_weights = None, output_layer_bias = None):
    #Your Code Here

    def init_weights_from_inputs_to_hidden_layer_neurons(self,
       hidden_layer_weights):
    #Your Code Here

    def
       init_weights_from_hidden_layer_neurons_to_output_layer_neurons
       (self, output_layer_weights):
    #Your Code Here

    def inspect(self):
        print('------')
        print('* Inputs: {}'.format(self.num_inputs))
        print('------')
        print('Hidden Layer')
        self.hidden_layer.inspect()
        print('------')
        print('* Output Layer')
        self.output_layer.inspect()
        print('------')

    def feed_forward(self, inputs):
        #Your Code Here
    # Uses online learning, ie updating the weights after each
        training case
    def train(self, training_inputs, training_outputs):
        self.feed_forward(training_inputs)

        # 1. Output neuron deltas
        #Your Code Here
        # dE/dz

        # 2. Hidden neuron deltas
        # We need to calculate the derivative of the error with
            respect to the output of each hidden layer neuron
        # dE/dy  = Σ dE/dz  * dz/dy  = Σ dE/dz  * w
        # dE/dz  = dE/dy  * dz /d
        #Your Code Here

        # 3. Update output neuron weights
        # dE /dw   = dE/dz  * dz /dw
        # Δw =   * dE /dw
        #Your Code Here

        # 4. Update hidden neuron weights
        # dE /dw  = dE/dz  * dz /dw
        # Δw =   * dE /dw
        #Your Code Here

    def calculate_total_error(self, training_sets):
        #Your Code Here
        return total_error

class NeuronLayer:
    def __init__(self, num_neurons, bias):
        # Every neuron in a layer shares the same bias
        self.bias = bias if bias else random.random()

        self.neurons = []
        for i in range(num_neurons):
            self.neurons.append(Neuron(self.bias))

    def inspect(self):
        print('Neurons:', len(self.neurons))
        for n in range(len(self.neurons)):
            print(' Neuron', n)
            for w in range(len(self.neurons[n].weights)):
                print('  Weight:', self.neurons[n].weights[w])
            print('  Bias:', self.bias)

    def feed_forward(self, inputs):
        outputs = []
        for neuron in self.neurons:
            outputs.append(neuron.calculate_output(inputs))
        return outputs

    def get_outputs(self):
        outputs = []
        for neuron in self.neurons:
            outputs.append(neuron.output)
        return outputs

class Neuron:
    def __init__(self, bias):
        self.bias = bias
        self.weights = []

    def calculate_output(self, inputs):
```

```
#Your Code Here

def calculate_total_net_input(self):
#Your Code Here

# Apply the logistic function to squash the output of the neuron
# The result is sometimes referred to as 'net' [2] or 'net' [1]
def squash(self, total_net_input):
#Your Code Here

# Determine how much the neuron's total input has to change to
   move closer to the expected output
#
# Now that we have the partial derivative of the error with
   respect to the output (dE/dy ) and
# the derivative of the output with respect to the total net
   input (dy /dz ) we can calculate
# the partial derivative of the error with respect to the total
   net input.
# This value is also known as the delta ( ) [1]
#   = dE/dz  = dE/dy  * dy /dz
#
def calculate_pd_error_wrt_total_net_input(self, target_output):
#Your Code Here

# The error for each neuron is calculated by the Mean Square
   Error method:
def calculate_error(self, target_output):
#Your Code Here

# The partial derivate of the error with respect to actual
   output then is calculated by:
# = 2 * 0.5 * (target output - actual output) ^ (2 - 1) * -1
# = -(target output - actual output)
#
# The Wikipedia article on backpropagation [1] simplifies to the
    following, but most other learning material does not [2]
# = actual output - target output
#
# Alternative, you can use (target - output), but then need to
   add it during backpropagation [3]
#
# Note that the actual output of the output neuron is often
   written as y  and target output as t  so:
# = dE/dy  = -(t  - y )
def calculate_pd_error_wrt_output(self, target_output):
#Your Code Here

# The total net input into the neuron is squashed using logistic
    function to calculate the neuron's output:
# y  =   = 1 / (1 + e^(-z ))
# Note that where   represents the output of the neurons in
   whatever layer we're looking at and   represents the layer
   below it
#
# The derivative (not partial derivative since there is only one
    variable) of the output then is:
# dy /dz  = y  * (1 - y )
def calculate_pd_total_net_input_wrt_input(self):
#Your Code Here

# The total net input is the weighted sum of all the inputs to
   the neuron and their respective weights:
# = z  = net  = x w  + x w  ...
#
# The partial derivative of the total net input with respective
```

```
        to a given weight (with everything else held constant) then
        is:
    # = dz /dw  = some constant + 1 * x w ^(1-0) + some constant ...
        = x
    def calculate_pd_total_net_input_wrt_weight(self, index):
    #Your Code Here

# An example:

nn = NeuralNetwork(2, 2, 2, hidden_layer_weights=[0.15, 0.2, 0.25,
   0.3], hidden_layer_bias=0.35, output_layer_weights=[0.4, 0.45,
   0.5, 0.55], output_layer_bias=0.6)
for i in range(10000):
    nn.train([0.05, 0.1], [0.01, 0.99])
    print(i, round(nn.calculate_total_error([[[0.05, 0.1], [0.01,
       0.99]]]), 9))
```

3 Tasks

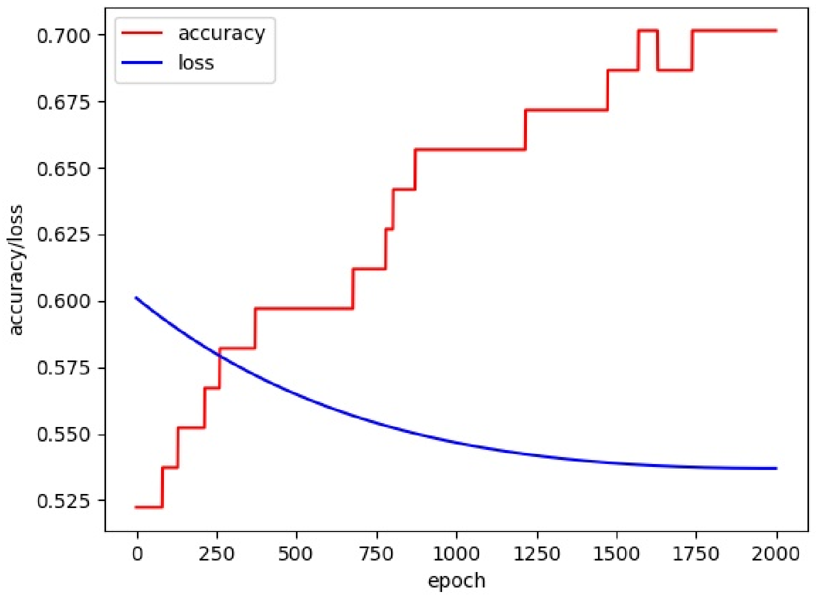
- Given the training set `horse-colic.data` and the testing set `horse-colic.test`, implement the BP algorithm and establish a neural network to predict if horses with colic will live or die. In addition, you should calculate the accuracy rate.
- Please submit a file named `E14_YourNumber.pdf` and send it to `ai_2020@foxmail.com`
- Draw the training loss and accuracy curves
- (optional) You can try different structure of neural network and compare their accuracy and the time they cost.

4 Codes and Results

本次实验要求我们应用课上学习到的反向传播 BP 算法知识，手动实现一个神经网络对 Horse Colic database 数据集进行分类。考虑到数据集的分类标签有三个，分别是 lived, died 和 was euthanized，常用于二分类的 logistic 回归已经不太适合使用，于是我将最后一层改为 softmax 回归，softmax 输出的值分别是不同类别的概率，损失函数使用交叉熵 CrossEntropy Loss Function。

观察数据集可以发现，一方面该数据集包含条目数量比较少，训练集只有 300 个样本而测试集只有 60 多个样本，另一方面标签的分配也十分不均匀，分类为 lived 的样本明显较多，且缺失值特别多竟达到了百分之三十的比例，这直接导致了我在后续调整网络结构的时候始终无法达到一个高效的分类结果，最终采用了两个全连接层，第一个全连接层后面添加 sigmoid 激活函数，第二个全连接层后面则是 softmax 函数用于分类。

经过多次和反复的实验，我发现由于数据集过小且质量不佳，分类结果受初始化影响非常非常大，我这里采取的是 xavier 初始化来初始化全连接层的 W 参数，偏置项 b 初始化为全零，在缺失值数据的处理方法是直接使用该属性的平均值，下面展示的是一次比较良好的 loss 下降过程图，loss 虽然一直在下降且逐渐收敛，但是分类的精度并不理想，但考虑到本次实验的着重点是 BP 算法，从 loss 的下降来看我手动求导的若干公式是正确的，BP 算法也运行正常。



```
import time
from math import sqrt


from tqdm import tqdm
import numpy as np
import matplotlib.pyplot as plt


def load_data(path):
attr, label = [], []
```

```
with open(path, "r") as f:
for line in f:
if not line.strip(): break
line_split = line.strip().split()
for i in range(len(line_split)):
if line_split[i] == "?": line_split[i] = "-1"

if int(line_split[22]) == -1: continue
label_onehot = np.zeros(3)
label_onehot[int(line_split[22])-1] = 1
label.append(label_onehot)

line_split = [float(x) for x in line_split]
attr.append([])
attr[-1].extend(line_split[0:2])
attr[-1].extend(line_split[3:22])
attr[-1].append(line_split[23])
attr[-1].append(line_split[27])

attr = np.array(attr, dtype=np.float64)
for i in range(attr.shape[1]):
nonzero = np.where(attr[:,i] != -1)
data = attr[:,i][nonzero[0]]
mean_value = np.mean(data)
zero = np.where(attr[:,i] == -1)
for j in zero[0]:
attr[j][i] = mean_value

mean = np.mean(attr, axis=0).reshape(1,-1)
var = np.var(attr, axis=0).reshape(1,-1)
attr = (attr - mean) / var

label = np.stack(label)
```

```
return attr, label


class SoftmaxRegression(object):
def __init__(self, lr):
self.lr = lr
self.w1 = np.random.uniform(low=-1.0, high=1.0, size=(13, 23)) * sqrt
   (13/(13+23))
self.b1 = np.zeros((13, 1))
self.w2 = np.random.uniform(low=-1.0, high=1.0, size=(3, 13)) * sqrt
   (13/(3+13))
self.b2 = np.zeros((3, 1))

self.z1 = np.zeros((13,1))
self.y_hat = np.zeros((3,1))
self.loss = np.zeros(1)

def forward(self, x):
assert x.shape == (1,23)
z1 = np.matmul(self.w1, x.T) + self.b1
sigmoid_z1 = 1 / (1 + np.exp(-z1))
self.z1 = sigmoid_z1

assert z1.shape == (13,1)
z2 = np.matmul(self.w2, self.z1) + self.b2

assert z2.shape == (3,1)
exp_z = np.exp(z2)
sum_exp_z = np.sum(exp_z)
self.y_hat = exp_z / sum_exp_z
assert self.y_hat.shape == (3,1)
# print(self.y_hat)
return np.argmax(self.y_hat)
```

```
def backward(self, x, y):
assert y.shape == (1,3)

gt = np.where(y == 1)[0][0]
self.loss = -np.log(self.y_hat[gt])[0]

y_hat_y = self.y_hat - y.T
y_hat_y_z1 = np.matmul(y_hat_y, self.z1.T)
w2_y_hat_y = np.matmul(self.w2.T, y_hat_y)
sigmoid_w2_y_hat_y = w2_y_hat_y * (1 - w2_y_hat_y)
w2_y_hat_y_x = np.matmul(sigmoid_w2_y_hat_y, x)

assert y_hat_y.shape == (3,1)
assert y_hat_y_z1.shape == (3,13)
assert w2_y_hat_y.shape == (13,1)
assert w2_y_hat_y_x.shape == (13,23)
self.w2 -= self.lr * y_hat_y_z1
self.b2 -= self.lr * y_hat_y
self.w1 -= self.lr * w2_y_hat_y_x
self.b1 -= self.lr * sigmoid_w2_y_hat_y

def test(model:SoftmaxRegression, attr, label):
correct = 0
for i in range(len(attr)):
predict = model.forward(attr[i].reshape(1,-1))
if label[i][predict] == 1:
correct += 1
# print(predict)
return correct / len(attr)

if __name__ == "__main__":
EPOCH = 2000
LR = 1e-6
```

```
np.random.seed(int(time.time()))
train_attr, train_label = load_data("horse-colic.data")
test_attr, test_label = load_data("horse-colic.test")

model = SoftmaxRegression(lr=LR)

loss_list = []
accu_list = []
for i in range(EPOCH):
loss_sum = 0.0
for j in range(len(train_attr)):
model.forward(train_attr[j].reshape(1,-1))
model.backward(train_attr[j].reshape(1,-1), train_label[j].reshape(1,-1)
   )
loss_sum += model.loss

accuracy = test(model, test_attr, test_label)
loss_list.append(loss_sum / len(train_attr))
accu_list.append(accuracy)
print("EPOCH {}/{} - accuracy:{:.3f} loss:{:.13f}".format(i+1, EPOCH,
   accuracy, loss_sum / len(train_attr)))

plt.plot(range(EPOCH), accu_list, color="red")
plt.plot(range(EPOCH), loss_list, color="blue")
plt.xlabel("epoch")
plt.ylabel("accuracy/loss")
plt.legend(["accuracy", "loss"])
plt.savefig("result.jpg")
```