Predict region with happiness by neural networks (backpropagation algorithm)

Jiamin Shang 001267391

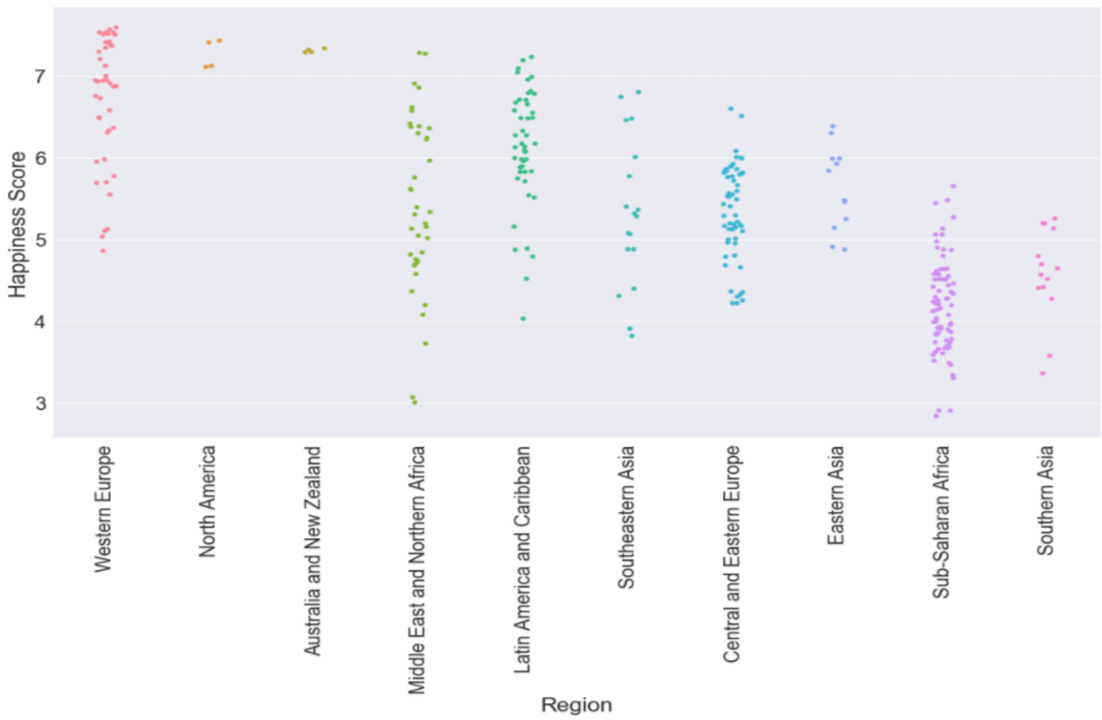
Abstract

To validating an existing machine learning algorithm in real-world contexts. I choose the algorithm learned in last semester to build a neural networks to predict the data from database I selected (World Happiness Report 2015,2016,2017). Can predict a country's region by analyse happiness scores with a fine accuracy.

Introduction

The happiness scores and rankings use data from the Gallup World Poll. The scores are based on answers to the main life evaluation question asked in the poll. The columns following the happiness score estimate the extent to which each of six factors – economic production, social support, life expectancy, freedom, absence of corruption, and generosity – contribute to making life evaluations higher in each country than they are in Dystopia, a hypothetical country that has values equal to the world's lowest national averages for each of the six factors.

According to the analysis in EDA. The happiness scores in all regions have pretty different distribution. So I combine the data of 2015 and 2016 to analyze and build the train data for neural networks to predict and validate the 2017's data.



Last semester, in info7250. I use java to build neural networks with backpropagation algorithm to implement gender recognition via voice. So, in this analysis, I just change my own algorithm code from java to python and do some necessary adjustment.

My core code include util functions, Neural network class, result encoding. My neural network is a basic 1 hidden layer model with 6 input parameters(I select 5 related factors according my analysis in EDA, drop some unnecessary factors) and 10 outputs (10 different regions in dataset).

The conclusion of EDA:

- The happiness score has positive linear relationship with other parameters
- Economy fmaily and health also have positive linear relationship with each other
- Generosity, Dystopia Residual is not correlated with any other factors.

Dataset

```
RangeIndex: 315 entries, 0 to 314
Data columns (total 13 columns):
Country                          315 non-null object
Region                           315 non-null object
Happiness Rank                   315 non-null int64
Happiness Score                  315 non-null float64
Economy (GDP per Capita)         315 non-null float64
Family                           315 non-null float64
Health (Life Expectancy)         315 non-null float64
Freedom                          315 non-null float64
Trust (Government Corruption)    315 non-null float64
Generosity                       315 non-null float64
Dystopia Residual                315 non-null float64
Year                             315 non-null int64
Region Code                      315 non-null int64
dtypes: float64(8), int64(3), object(2)
memory usage: 32.1+ KB
```

Algorithm and model

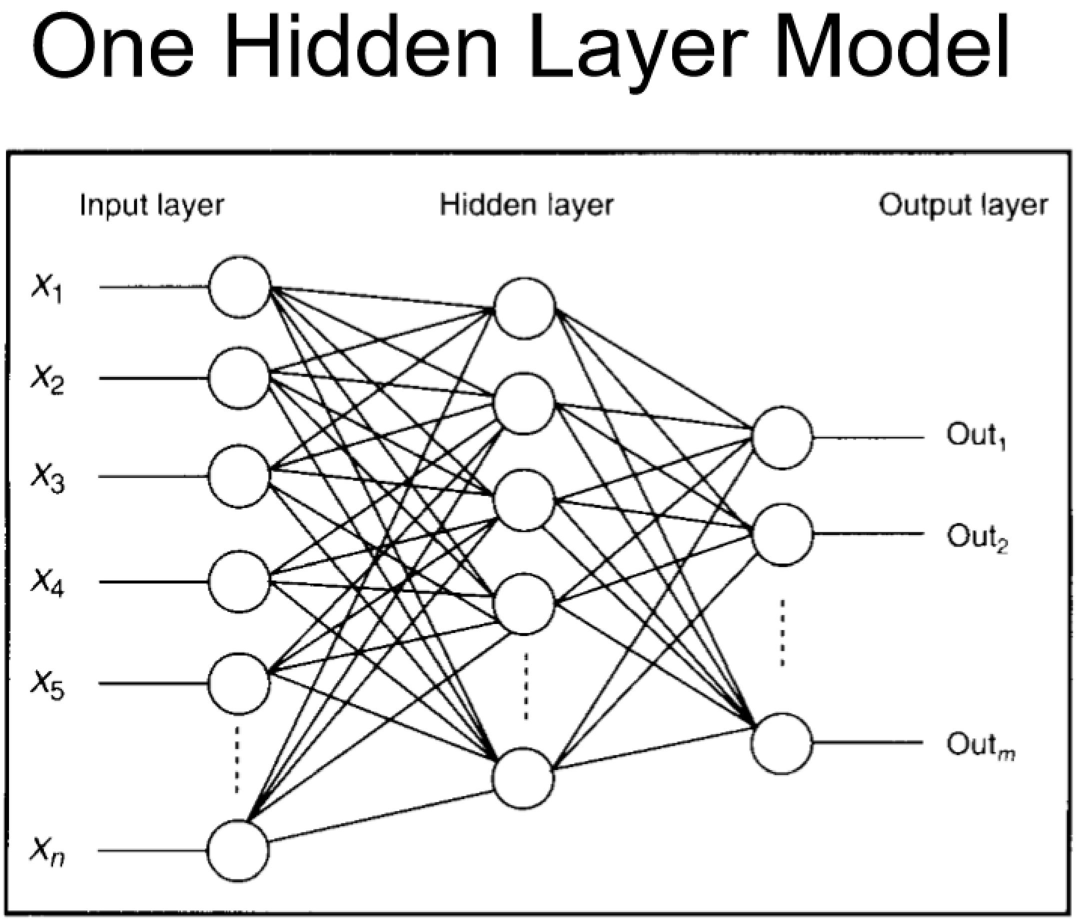
Backprop description

- Compares neural network computed outputs with the target output values
- Determines the magnitude and direction of the difference between actual and target values
- Then adjusts a neural network's weights and bias values so that the new outputs will be closer to the target values

- This process is repeated until the actual output values are close enough to the target values, or some maximum number of iterations has been reached
- Because computing the back-propagation hidden layer gradients requires the values of the output layer gradients, the algorithm computes "backwards"

Neural network

Weight values are stored in a particular order: input-to-hidden weights, followed by hidden -layer biases, followed by hidden-to-output weights, followed by output biases



Code with Documentation
define util function

```
def rand(a, b):
    return (b - a) * random.random() + a
```

```
def make_matrix(m, n, fill=0.0):
    mat = []
    for i in range(m):
        mat.append([fill] * n)
    return mat

def sigmoid(x):
    return 1.0 / (1.0 + math.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

def result_list(result):
    #resultlist = []
    max_item = max(result)
    i = 0
    m = 0
    for item in result:
        i += 1
        if item == max_item:
            m = i
    #resultlist.append(m)
    return m
```

Neural network with BP algorithms, to init the weights and biases matrixes, also define the predict, train, test and BP algorithms function

```
class BPNeuralNetwork:
    def __init__(self):
        self.input_n = 0
        self.hidden_n = 0
        self.output_n = 0
        self.input_cells = []
        self.hidden_cells = []
        self.output_cells = []
        self.input_weights = []
        self.output_weights = []
        self.input_correction = []
        self.output_correction = []

    def setup(self, ni, nh, no):
        self.input_n = ni + 1
        self.hidden_n = nh
        self.output_n = no
```

```
        # init cells
        self.input_cells = [1.0] * self.input_n
        self.hidden_cells = [1.0] * self.hidden_n
        self.output_cells = [1.0] * self.output_n
        # init weights
        self.input_weights = make_matrix(self.input_n, self.hidden_n)
        self.output_weights = make_matrix(self.hidden_n, self.output_n)
        # random activate
        for i in range(self.input_n):
            for h in range(self.hidden_n):
                self.input_weights[i][h] = rand(-0.2, 0.2)
        for h in range(self.hidden_n):
            for o in range(self.output_n):
                self.output_weights[h][o] = rand(-2.0, 2.0)
        # init correction matrix
        self.input_correction = make_matrix(self.input_n, self.hidden_n)
        self.output_correction = make_matrix(self.hidden_n, self.output_n)

    def predict(self, inputs):
        # activate input layer
        for i in range(self.input_n - 1):
            self.input_cells[i] = inputs[i]
        # activate hidden layer
        for j in range(self.hidden_n):
            total = 0.0
            for i in range(self.input_n):
                total += self.input_cells[i] * self.input_weights[i][j]
            self.hidden_cells[j] = sigmoid(total)
        # activate output layer
        for k in range(self.output_n):
            total = 0.0
            for j in range(self.hidden_n):
                total += self.hidden_cells[j] * self.output_weights[j][k]
            self.output_cells[k] = sigmoid(total)
        return self.output_cells[:]

    def back_propagate(self, case, label, learn, correct):
        # feed forward
        self.predict(case)
        # get output layer error
        output_deltas = [0.0] * self.output_n
        for o in range(self.output_n):
            error = label[o] - self.output_cells[o]
            output_deltas[o] = sigmoid_derivative(self.output_cells[o]) * error
```

```
        # get hidden layer error
        hidden_deltas = [0.0] * self.hidden_n
        for h in range(self.hidden_n):
            error = 0.0
            for o in range(self.output_n):
                error += output_deltas[o] * self.output_weights[h][o]
            hidden_deltas[h] = sigmoid_derivative(self.hidden_cells[h]) * error
        # update output weights
        for h in range(self.hidden_n):
            for o in range(self.output_n):
                change = output_deltas[o] * self.hidden_cells[h]
                self.output_weights[h][o] += learn * change + correct *
self.output_correction[h][o]
                self.output_correction[h][o] = change
        # update input weights
        for i in range(self.input_n):
            for h in range(self.hidden_n):
                change = hidden_deltas[h] * self.input_cells[i]
                self.input_weights[i][h] += learn * change + correct *
self.input_correction[i][h]
                self.input_correction[i][h] = change
        # get global error
        error = 0.0
        for o in range(len(label)):
            error += 0.5 * (label[o] - self.output_cells[o]) ** 2
        return error

    def train(self, cases, labels, limit=10000, learn=0.05, correct=0.1):
        for j in range(limit):
            error = 0.0
            for i in range(len(cases)):
                label = labels[i]
                case = cases[i]
                error += self.back_propagate(case, label, learn, correct)

    def test(self, traindata, result, testcases,testresult, ni, nh, no):
        self.setup(ni, nh, no)
        self.train(traindata, result, 10000, 0.05, 0.1)
        testout = []
        count = 0.0
        error = 0.0
        for case in testcases:
            #print(result_list(self.predict(case)))
            #print(self.predict(case))
```

```
            testout.append(result_list(self.predict(case)))
        for index in range(len(testout)):
            count += 1
            if testout[index] != testresult[index]:
                error +=1
        accuracy = (count - error)/count
        print("Number of test: "+ str(count))
        print("Number of error: " + str(error))
        print("Accuracy: " + str(accuracy))
        return accuracy
```

Encode the 10 results and generate traindata(2015+2016 data) and testdata(2017 data)

```
whrcp = whr
traininput = whrcp.loc[:, 'Happiness Score':'Dystopia Residual'].values
trainresult = whrcp.loc[:, 'Region Code'].values
resultx = []
for res in trainresult:
    restmp = []
    if res == 1:
        restmp = [1,0,0,0,0,0,0,0,0,0]
    elif res == 2:
        restmp = [0,1,0,0,0,0,0,0,0,0]
    elif res == 3:
        restmp = [0,0,1,0,0,0,0,0,0,0]
    elif res == 4:
        restmp = [0,0,0,1,0,0,0,0,0,0]
    elif res == 5:
        restmp = [0,0,0,0,1,0,0,0,0,0]
    elif res == 6:
        restmp = [0,0,0,0,0,1,0,0,0,0]
    elif res == 7:
        restmp = [0,0,0,0,0,0,1,0,0,0]
    elif res == 8:
        restmp = [0,0,0,0,0,0,0,1,0,0]
    elif res == 9:
        restmp = [0,0,0,0,0,0,0,0,1,0]
    elif res == 10:
        restmp = [0,0,0,0,0,0,0,0,0,1]
    resultx.append(restmp)
whr2017 = pd.read_csv('./world-happiness-report/2017.csv')
testinput = whr2017.loc[:, 'Happiness.Score':'Dystopia.Residual'].values
testresult = whr2017.loc[:, 'Region Code'].values
```

Start predict
net = BPNeuralNetwork()
output = net.test(traininput,resultx,testinput,testresult,8,12,10)

Results
I choose 2017's report as test data to test the trained model. The prediction is implemented

Result:

```
net = BPNeuralNetwork()
output = net.test(traininput,resultx,testinput,testresult,5,12,10)
Predict: Sub-Saharan Africa    In dataset: Sub-Saharan Africa
Predict: Sub-Saharan Africa    In dataset: Sub-Saharan Africa
Predict: Sub-Saharan Africa    In dataset: Southern Asia
Predict: Sub-Saharan Africa    In dataset: Sub-Saharan Africa
Predict: Sub-Saharan Africa    In dataset: Sub-Saharan Africa
Predict: Sub-Saharan Africa    In dataset: Sub-Saharan Africa
Predict: Sub-Saharan Africa    In dataset: Latin America and Caribbean
Predict: Sub-Saharan Africa    In dataset: Middle East and Northern Africa
Predict: Sub-Saharan Africa    In dataset: Sub-Saharan Africa
Predict: Sub-Saharan Africa    In dataset: Sub-Saharan Africa
Predict: Sub-Saharan Africa    In dataset: Sub-Saharan Africa
Predict: Sub-Saharan Africa    In dataset: Sub-Saharan Africa
Predict: Sub-Saharan Africa    In dataset: Sub-Saharan Africa
Predict: Middle East and Northern Africa    In dataset: Middle East and Northern Africa
Predict: Sub-Saharan Africa    In dataset: Sub-Saharan Africa
Predict: Sub-Saharan Africa    In dataset: Middle East and Northern Africa
Predict: Sub-Saharan Africa    In dataset: Sub-Saharan Africa
Number of test: 155.0
Number of error: 67.0
Accuracy: 0.567741935483871
```

Discussion
I set 2000 loop with 0.05 learn rate and 0.005 momentum to start the model
For a 10 outputs model, the accuracy is around 55%~62%. I think it's a fine accuracy.
The accuracy can be improved by use bigger dataset and add more layers. But because of the limitation of my dataset and my laptop, I only have 312 row of train data to train the model also can't calculate too many layers (take too much time).

Backpropagation can be very slow for large data sets. One weakness of backpropagation is that the algorithm is often extremely sensitive to values used for the learning rate and momentum. So, I also don't set it very small to accelerate the calculation.

Anyhow, the accuracy is acceptable. It can identify the person with the happiness data belong to which region. It also show the happiness' distribution

Conclusion

According to the EDA and prediction of neural networks model, the results show the happiness scores' relationship with economy, family, health and freedom definitely. Region always share the similar status. Thus, they have similar happiness scores distribution. So the regions can be predicted by these factors.

References

Data source https://www.kaggle.com/unsdsn/world-happiness
Algorithm and model Dino Konstantopoulos 2017 INFO 7250