武汉大学计算机学院

课程论文

Average academic grades predictor-Based on students' background information

课程名称 商务智能

专业年级班级 2016 级 卓越一班

姓名 彭璇

学号 2016302580096

学期 2018-2019 学年 第一 学期

成绩 任课教师签名

1. Abstract

In traditional Chinese idioms, there are some interesting proverbs to relate one's success to some certain factors. For example: "一分耕耘一分收获", which emphasize the importance of diligence. Also, like "有其父必有其子", "虎父无犬子". These proverbs assert the role that ones' parental level play in the overall performance of their children. And the facts around us seem to constantly confirm these points. Over time, these points seem to have become the truth of everyone. However, this kind of opinion seems to lack of evidence from statistical field. We just judge it according to our personal experience.

Until recently, there emerges a popular dataset on Kaggle which is about the academic performance of 1000 students. Moreover, in the dataset, some important background information is given, like parental education level, race and lunch type. In a while it occurs to me that I can possibly use the data to verify the truth of the points mentioned above by exploiting the knowledge I have learned in BI class. So after consideration, I decide to make use of of Deep Neural Network(DNN, also MLP) to try to train and get a proper model to reveal the relationship between students' background information and their average grades level.

Keywords:

DNN, Decision Tree, Tensorflow, k-fold cross validation, one-hot encoding, SK-learn, Batch-Normalization, Dropout

2. Requirement Analysis

First, it's necessary to encode the literal data into a proper form that the computer can handle with.

After that, a well-trained model is needed. To achieve this, we need to take the encoded data as the input of the model. Then according to the calculated loss ,we can adjust the parameters of the model.

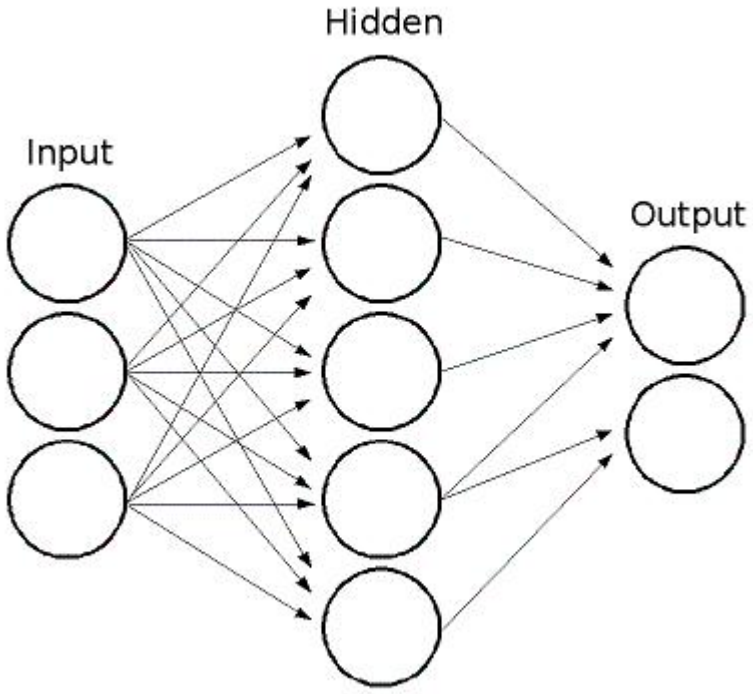
Finally, we need a separated testing dataset to reassure the effectiveness of the model. The training and testing process should be visualized.

Particularly, in the grades prediction scenario, due to the relatively small scale of dataset(only 1000 pieces of data in total), an appropriate training method is desperately needed to avoid under-fitting and finally get well-tuned model.

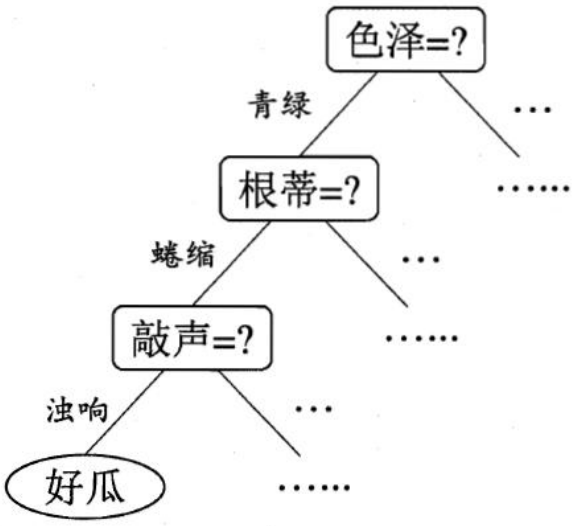
3. Solution

1. For the encoding method, we choose one-hot encoding. One-hot is suitable to be applied here because most of the background attributes like race are nominal attributes. And the method itself is, of course, intuitive, effective and convenient to use.

2. As for model:
considering that the input is literal words and sentences, so in this case CNN may not achieve the performance we expect.
Moreover, there is no prioritization in time and space for the data. So RNN is not suitable, either.
So we choose the most intuitive neural network, DNN(Multi-Layer Perceptron) for its capability to fit any function. Also, it's easier to get well-tuned.
Also, since we've learned Decision Tree in class. In some certain cases, decision tree can achieve a pretty good performance. So Decision Tree model is constructed in the experiment, as well.



(DNN Model Structure. 17 nodes in input layer, 32 nodes in first layer)



(Typical Decision Tree structure)

3. To achieve well-tuned model with relatively small scale of dataset, we apply the "k-fold cross validation" and "leave-one"method to the training process. During training, the total data is split into 100 parts and 99 of them are used as training dataset while the rest 1 part works as testing set. So after cross validation, we actually train the model for 100x1000 = 100,000 (times) in gross.

4. Running Environment And Dependencies

Python 3.6: Python is a popular computer language in machine learning for its convenient grammar and richness in libraries. In the construction of model, all the code are written in python.

Numpy: Numpy is the fundamental package for scientific computing with Python.

Tensorflow: TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks

Scikit-learn:also called SKlearn. Scikit-learn is a free machine learning math software library for the Python programming language. It features a lot various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN. We use the library from SKlearn to achieve cross-validation.

Anaconda: Anaconda is a free-to-use and open-source famous distribution of the Python and R programming languages for data science and machine

learning applications (large-scale data processing, predictive analytics, scientific computing), that aims to simplify package management and deployment

Matplotlib: matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy.

5. Data Set

Source: The author found the data on Kaggle(a data competition website) sporadically.

(see: https://www.kaggle.com/spscientist/students-performance-in-exams)

Structure:

The structure of original data is shown below:

Sex	Race	Parental education level	lunch type	Review	Math	Reading	Writing
female	group B	bachelor's degree	standard	none	72	72	74
female	group C	some college	standard	completed	69	90	88
female	group B	master's degree	standard	none	90	95	93
male	group A	associate's degree	free/reduced	none	47	57	44
male	group C	some college	standard	none	76	78	75
female	group B	associate's degree	standard	none	71	83	78
female	group B	some college	standard	completed	88	95	92
male	group B	some college	free/reduced	none	40	43	39
male	group D	high school	free/reduced	completed	64	64	67
female	group B	high school	free/reduced	none	38	60	50
male	group C	associate's degree	standard	none	58	54	52
male	group D	associate's degree	standard	none	40	52	43
female	group B	high school	standard	none	65	81	73
male	group A	some college	standard	completed	78	72	70
female	group A	master's degree	standard	none	50	53	58

1. There are 8 attributes in total. We take the first 5 attributes as the input. For the last 3 attributes which represents students' grades on certain subjects respectively, we add the 3 grades up and calculate the average grades, shown below:

72.666667
82.333333
92.666667
49.333333
76.333333
77.333333
91.666667
40.666667
65
49.333333
54.666667

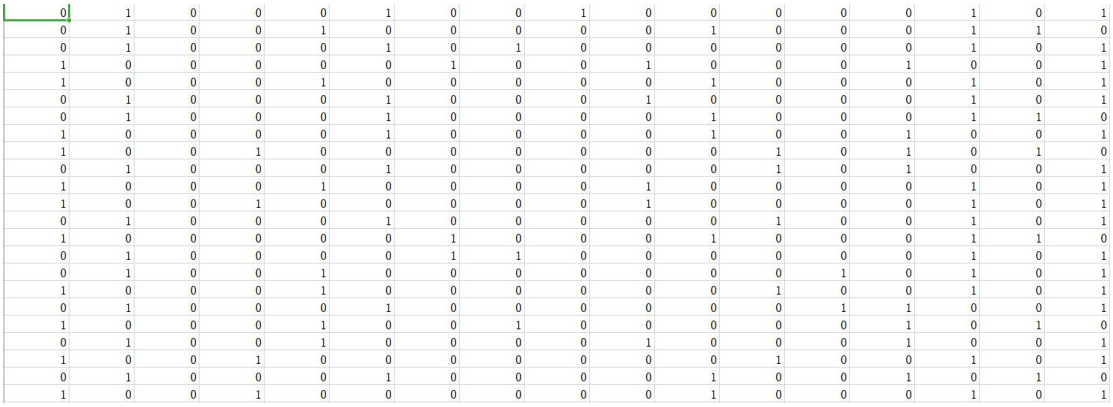
2. For the average grades, we classify them into two type:

if avg >=60, then it's classified into "passed", encoded as[1,0] as y.

Else, it will be classified as "failed", encoded as [0,1] as y.

In the whole 1000 pieces of data, there are 700 "passed" and 300 "failed".

3. Then we encode the input literal data into x with one-hot method, shown below:



6. Experiment Procedures

(1) Firstly, as we've mentioned in the "Data Set" part, the original data should be preprocessed and encoded.

In fact, at the very beginning, the y is classified into 5 categories: A(avg>=90),B(80<=avg<90),C(70<=avg<80),D(60<=avg<70),E(avg<60).However, may be limited by the number of data available, after a period of trying, the best accuracy we can get is only 50%, which is not so good for a 5 classification problem. So we turn to 2 classification.

(2) Then, build up the basic MLP structure. About the number of layers, I've tried 1,2,3,4(of course when n=1, it's perceptron algorithm or Logistic Regression). Only to find that 2 is the best number for layers in this scenario.

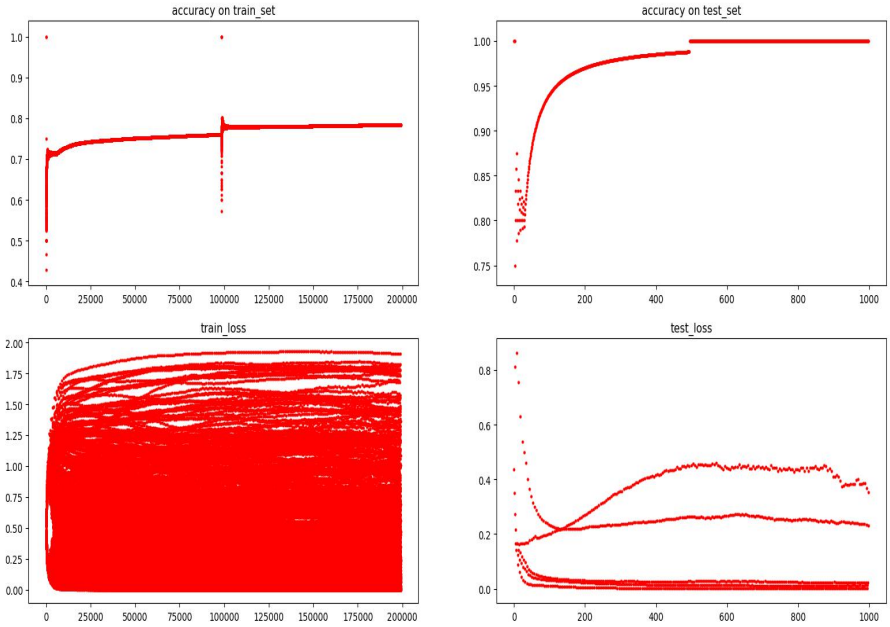
(3) For the number of neurons in each layer, in layer1 we have 32 neuron because this size is capable enough to deal with the input(length=17) and meanwhile stay time-saving. The layer2 have 2 neurons so that we can take the larger value between the 2 as the output of the model. After the 2 layers, a softmax activation function is used to normalize the output.

(4) Also, during the tuning of the DNN model, some other methods were once applied including **Batch Normalization** and **Dropout**. However, due to the relatively small scale of network size, these methods didn't improve the effectiveness as we may expect. So they're removed then.

As for the Decision Tree model, we directly make use of the library in Scikit-learn.

7. Result

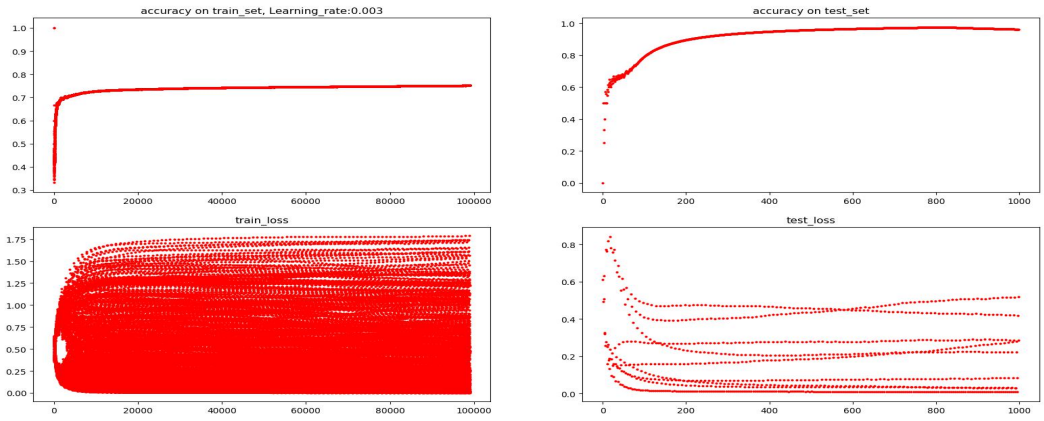
As is known to all, the training results of neural networks have a close relationship with the initialization of parameters. At the beginning stage, I used the 200-fold cross validation and tried to train the model for dozens of times. Soon there was a time that I achieve the best result I've got.



(train the model for 200,000 pieces of data, finally reached nearly 100% accuracy on test set)
(Here the batch_size =1(because of the simpleness of network, training one by one is viable) so the loss may shake greatly in the above picture.)

As we can see, finally we reached nearly 100% accuracy on the test set. And the way I calculate the accuracy is cumulative calculation. That is, to calculate the accuracy on the who 200,000 pieces of data. So a lot of erroneous prediction in the beginning stage of training comprise a large part of total training, which can explain why the accuracy on train set is only about 80%. But, admittedly, the 100% accuracy should be the result of over-fitting.

However...After this round of training, I forgot to save the model I've trained. So the code I've handed in is another version of model, whose performance is as listed below:



(Test accuracy converges to 96%)

To show the training effect intuitively, I save the tuned model and choose 5 pieces of data to verify it.

1	C
2	B
3	A
4	E
5	C

```
[[0.9235191  0.07648094]]
[[0.9377317  0.06226836]]
[[0.90594816 0.09405182]]
[[0.44229138 0.5577086 ]]
[[0.6236763  0.37632367]]
```

As we can see, the predictor gives the right predicting result for all 5 samples.

As for the Decision Tree, maybe because of the insufficient ability to fit the features in such a small scale of data and also the intrinsic drawback of Decision Tree. The result based on 100-fold test is not rather brilliant, but still shows some kind of relationship:

```
Acc on train:  0.7555555555555555
Acc on test:  0.8
```

(Result on decision tree)

All in all, the whole process shows there indeed exists strongly related relationship between students' academic performance and their background information.

8. Core codes

```
def load_data(x_path='../data/Processed_StudentsPerformance.csv',
              y_path='../data/AverageGradesLevel.txt'):
    data=[]
    level=[]
    print('start loading set.')
    reader_x = csv.reader(open(x_path))
    for row in reader_x:
        data.append(row)
    reader_y = open(y_path, 'rU')
    try:
        for line in reader_y:
            level.append(line)
    finally:
        reader_y.close()
```

(data read)

```
def gen(X_data,y_data,n_splits):
    for train_index, test_index in KFold(n_splits).split(X_data):
        X_train, X_test = X_data[train_index], X_data[test_index]
        y_train, y_test = y_data[train_index], y_data[test_index]
        yield X_train,y_train,X_test,y_test
```

(use Sklearn to realize a k-fold data set generator)

```
LAYER1_NODE = 32
def MLP(input,INPUT_NODE,OUTPUT_NODE):
    weight1 = tf.Variable(tf.truncated_normal([INPUT_NODE,LAYER1_NODE],stddev=0.1,dtype=float),name='w1')
    biases1 = tf.Variable(tf.truncated_normal([LAYER1_NODE],stddev=0.1,dtype=float),name='b1')
    weight2 = tf.Variable(tf.truncated_normal([LAYER1_NODE, OUTPUT_NODE], stddev=0.1, dtype=float),name='w2')
    biases2 = tf.Variable(tf.truncated_normal([OUTPUT_NODE], stddev=0.1, dtype=float),name='b2')
    result1 = tf.nn.relu(tf.matmul(input,weight1)+biases1)
    result2 = tf.nn.relu(tf.matmul(result1,weight2)+biases2)
    result = tf.nn.softmax(result2)
    return result
```

(DNN Model constructor)

```
#loss
loss = tf.reduce_mean(tf.reduce_sum(tf.square(output-y),axis=1))
#train_step
train_step = tf.train.GradientDescentOptimizer(LEARNING_RATE_BASE).minimize(loss)
#accuracy
correct_pred = tf.equal(tf.argmax(y,1),tf.argmax(output,1))
accuracy = tf.reduce_mean(tf.cast(correct_pred,tf.float64))
```

(define loss function, optimizer and accuracy calculator)

```
with tf.Session() as sess:
    init = tf.global_variables_initializer()
    sess.run(init)
    x_all,y_all = Data_act.load_data()
    for j in range(epoch):
        ...
        data_obj = Data_act.gen(x_all,y_all,epoch)
        x_train,y_train,x_test,y_test = next(data_obj)
        print(x_train.shape)
        for i in range(int(TRAIN_STEPS)):
            # xs = x[i]
            # ys = y[i]
            train_step.run({
                x: np.reshape(x_train[i],(1,-1)),
                y: np.reshape(y_train[i],(1,-1))
            })
            train_step_num += 1
            # steps+=1
            acc = accuracy.eval({
                x: np.reshape(x_train[i], (1, -1)),
                y: np.reshape(y_train[i], (1, -1))
            })
            c_acc = (acc + f_acc * (train_step_num - 1)) / train_step_num
            y1.append(c_acc)
            f_acc = c_acc
            los = loss.eval({
                x: np.reshape(x_train[i], (1, -1)),
                y: np.reshape(y_train[i], (1, -1))
            })
            y2.append(los)
            out = output.eval({
                x: np.reshape(x_train[i], (1, -1))
            })
```

(train model with session)

```
    saver = tf.train.Saver()
    model_path = "../model/model.ckpt"
    save_path = saver.save(sess, model_path)
```

(save the model)

```
sess = tf.Session()
saver = tf.train.import_meta_graph('model.ckpt.meta')
saver.restore(sess,tf.train.latest_checkpoint('./'))
graph = tf.get_default_graph()
w1 = graph.get_tensor_by_name("w1:0")
b1 = graph.get_tensor_by_name("b1:0")
w2 = graph.get_tensor_by_name("w2:0")
b2 = graph.get_tensor_by_name("b2:0")
```

(restore the model)

```
x,y = Data_act_tree.load_data()

X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.01,random_state=0)
tree = DecisionTreeClassifier()
tree.fit(X_train,y_train)
print("Acc on train: ",tree.score(X_train,y_train))
print("Acc on test: ",tree.score(X_test,y_test))
```

(DecisionTreeModel)

9. Instructions for running

To run the DNN training code, in anaconda environment, change the directory into the "*train*" directory, and "*activate tensorflow*". Type in "*python GradesPredict.py*" to train the model by oneself.

To run the tuned model, in anaconda environment, change the directory into the "*model*" directory, and "*activate tensorflow*". Type in "*python testModel.py*" to train the model by oneself. One can freely change the selected verification data in the codes to know more about the model.

To run the Decision Tree model, in anaconda environment, change the directory into the "*model*" directory, and "*activate tensorflow*". Type in "*python DecisionTree.py*" to train the model by oneself.

For ordinary/non-vitual environment, ignore the step of "*activate tensorflow*"

Consider the randomness of parameter initialization, one may takes several times to train before he could get an ideal predictor.

10. Discussion

In this experiment, the tuned model shows there is indeed some kind of relationship between children's performance in school and their background information. Typically, **the higher a child's parental education level is, the richer his/her family is(can be partly judged by children's lunch type, "free/reduced" lunch type means relatively poor), the better preparation the child made before test, the better**

overall grades he/she can achieve in the test.

The biggest reward I get from this experiment is , I get to know the delight hidden behind the boring data. Meanwhile, I improve my skills of many aspects like Data Mining and Deep Learning. I become more proficient on using python, tensorflow and some third-party machine learning libraries.

However, a pity here is, the scale of data is too small, so in the training process it's unavoidable for the model to get over-fit to some extent. Hope we can find better methods to improve the model in the future.