South China University of Technology

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

The Course Report of Artificial Intelligence

姓名：	李可欣
学号：	201530611999
指导老师：	宋恒杰
班级：	15 级 3 班
日期：	2018/1/18

一、 论文原文

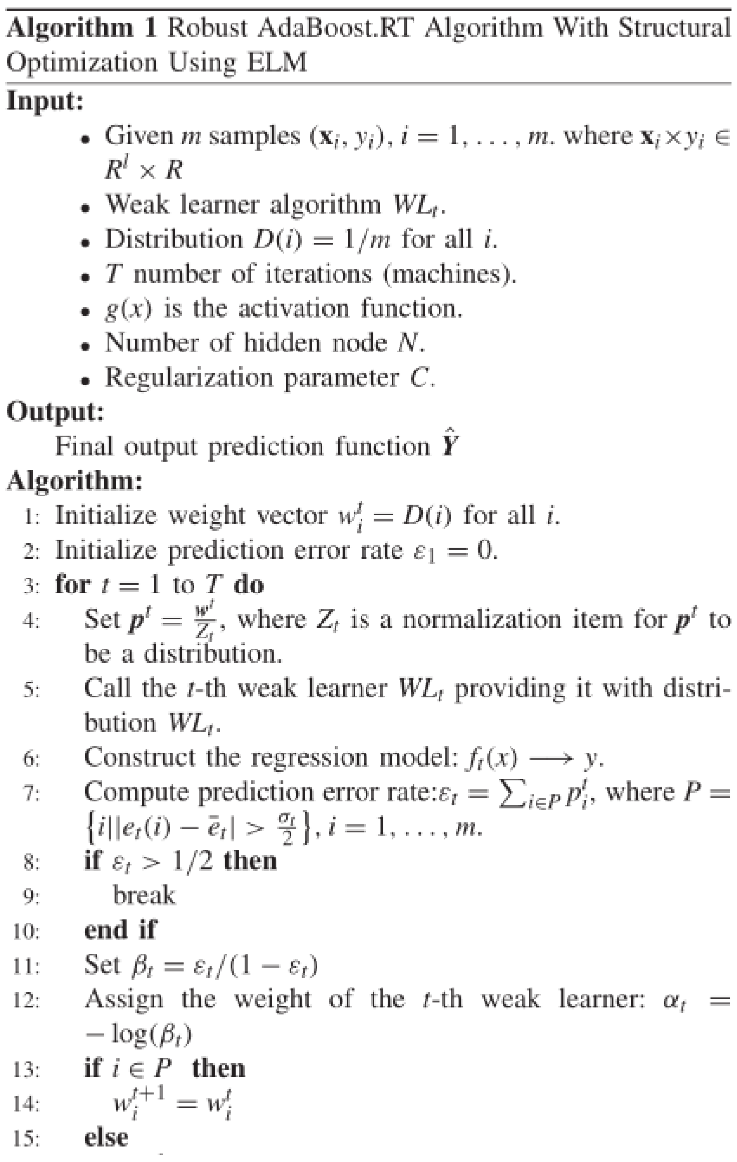
Peng-Bo Zhang and Zhi-Xin Yang,“A Novel AdaBoost Framework With Robust Threshold and Structural Optimization，”in *IEEE Trans. Cybernetics*, vol. 48, no. 1, Jan. 2018. (原文见 AdaBoost.pdf)

二、 翻译

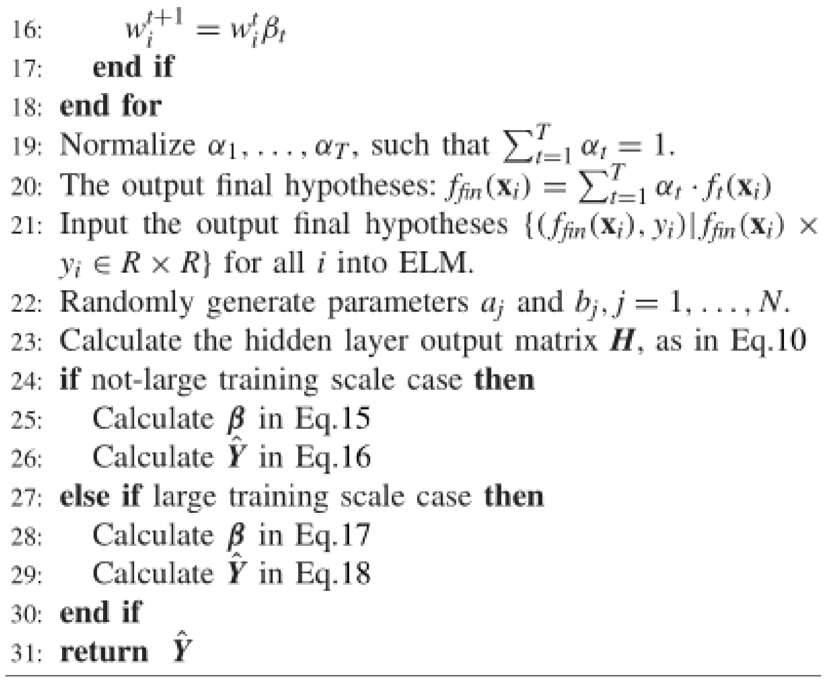
论文翻译见，翻译.pdf。

三、 算法设计及实现

1. 算法设计



```
Algorithm 1 Robust AdaBoost.RT Algorithm With Structural Optimization Using ELM
Input:
    • Given $m$ samples $(\mathbf{x}_i, y_i)$, $i = 1, \dots, m$. where $\mathbf{x}_i \times y_i \in R^l \times R$
    • Weak learner algorithm $WL_t$.
    • Distribution $D(i) = 1/m$ for all $i$.
    • $T$ number of iterations (machines).
    • $g(x)$ is the activation function.
    • Number of hidden node $N$.
    • Regularization parameter $C$.
Output:
    Final output prediction function $\hat{\boldsymbol{Y}}$
Algorithm:
 1: Initialize weight vector $w_i^t = D(i)$ for all $i$.
 2: Initialize prediction error rate $\varepsilon_1 = 0$.
 3: for $t = 1$ to $T$ do
 4:     Set $\boldsymbol{p}^t = \frac{w^t}{Z_t}$, where $Z_t$ is a normalization item for $\boldsymbol{p}^t$ to be a distribution.
 5:     Call the $t$-th weak learner $WL_t$ providing it with distribution $WL_t$.
 6:     Construct the regression model: $f_t(x) \longrightarrow y$.
 7:     Compute prediction error rate:$\varepsilon_t = \sum_{i \in P} p_i^t$, where $P = \left\{ i \middle| |e_t(i) - \bar{e}_t| > \frac{\sigma_t}{2} \right\}$, $i = 1, \dots, m$.
 8:     if $\varepsilon_t > 1/2$ then
 9:         break
10:     end if
11:     Set $\beta_t = \varepsilon_t/(1 - \varepsilon_t)$
12:     Assign the weight of the $t$-th weak learner: $\alpha_t = -\log(\beta_t)$
13:     if $i \in P$ then
14:         $w_i^{t+1} = w_i^t$
15:     else
```



```
16:         $w_i^{t+1} = w_i^t \beta_t$
17:     **end if**
18: **end for**
19: Normalize $\alpha_1, \dots, \alpha_T$, such that $\sum_{t=1}^{T} \alpha_t = 1$.
20: The output final hypotheses: $f_{fin}(\mathbf{x}_i) = \sum_{t=1}^{T} \alpha_t \cdot f_t(\mathbf{x}_i)$
21: Input the output final hypotheses $\{(f_{fin}(\mathbf{x}_i), y_i) | f_{fin}(\mathbf{x}_i) \times y_i \in R \times R\}$ for all $i$ into ELM.
22: Randomly generate parameters $a_j$ and $b_j, j = 1, \dots, N$.
23: Calculate the hidden layer output matrix $\boldsymbol{H}$, as in Eq.10
24: **if** not-large training scale case **then**
25:     Calculate $\boldsymbol{\beta}$ in Eq.15
26:     Calculate $\hat{\boldsymbol{Y}}$ in Eq.16
27: **else if** large training scale case **then**
28:     Calculate $\boldsymbol{\beta}$ in Eq.17
29:     Calculate $\hat{\boldsymbol{Y}}$ in Eq.18
30: **end if**
31: **return** $\hat{\boldsymbol{Y}}$
```

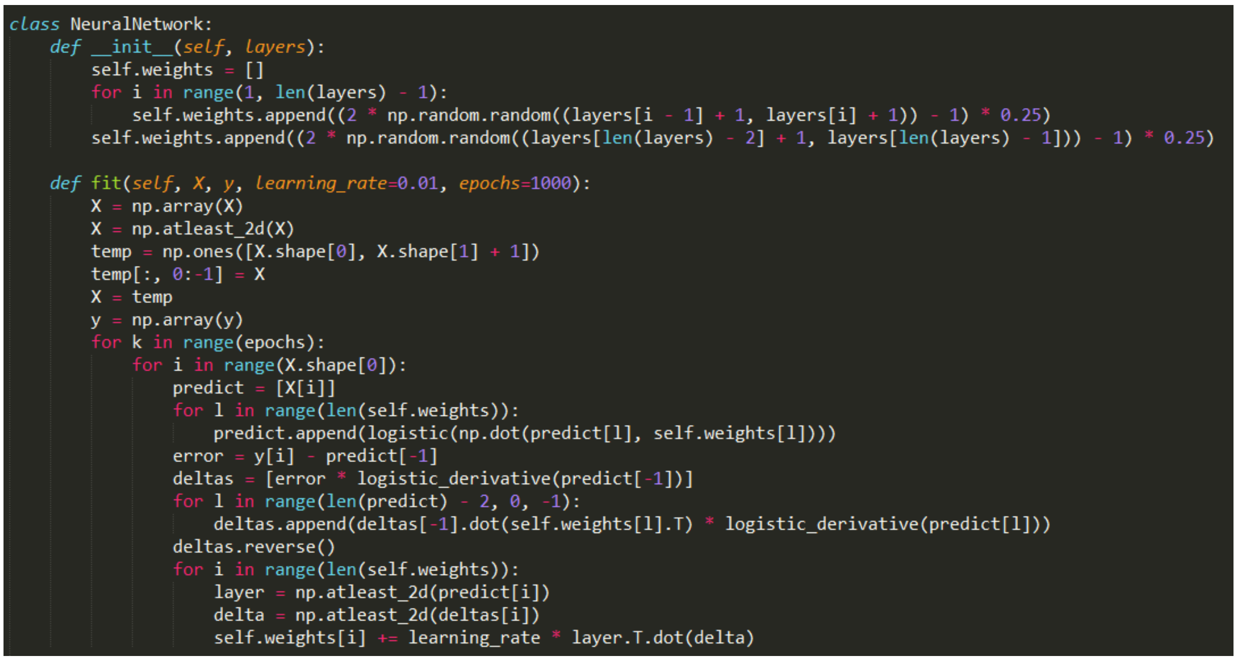
2. 实现

此次课程任务中，一共实现了包括 Back Propagation，AdaBoost，Extreme Learning Machine 三个核心算法。其中 BP 算法作为 AdaBoost 的弱学习器；ELM 作为进一步优化 AdaBoost 结构的优化方法。实现语言为 Python。

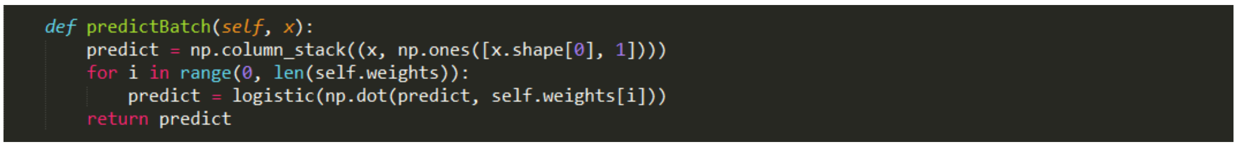
算法实现主要代码截图：

（详见 BP.py，AdaBoost.py，ELM.py）

- **Back Propagation**

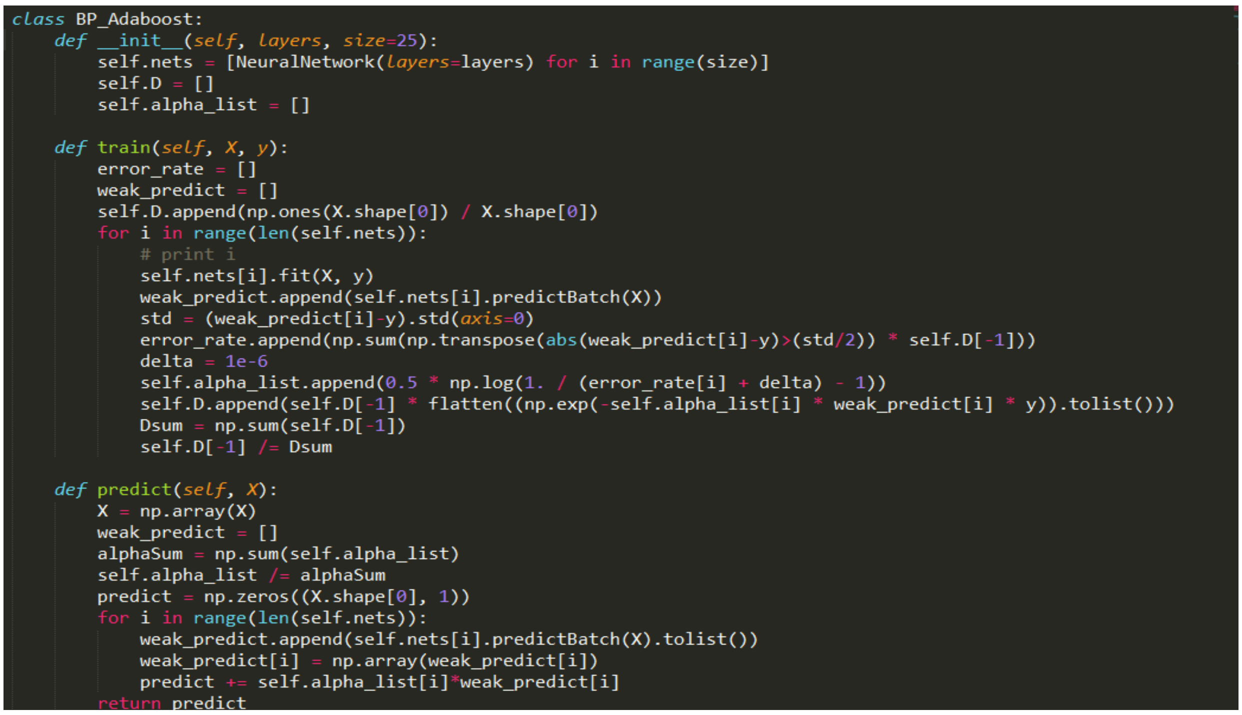


```python
class NeuralNetwork:
    def __init__(self, layers):
        self.weights = []
        for i in range(1, len(layers) - 1):
            self.weights.append((2 * np.random.random((layers[i - 1] + 1, layers[i] + 1)) - 1) * 0.25)
        self.weights.append((2 * np.random.random((layers[len(layers) - 2] + 1, layers[len(layers) - 1])) - 1) * 0.25)

    def fit(self, X, y, learning_rate=0.01, epochs=1000):
        X = np.array(X)
        X = np.atleast_2d(X)
        temp = np.ones([X.shape[0], X.shape[1] + 1])
        temp[:, 0:-1] = X
        X = temp
        y = np.array(y)
        for k in range(epochs):
            for i in range(X.shape[0]):
                predict = [X[i]]
                for l in range(len(self.weights)):
                    predict.append(logistic(np.dot(predict[l], self.weights[l])))
                error = y[i] - predict[-1]
                deltas = [error * logistic_derivative(predict[-1])]
                for l in range(len(predict) - 2, 0, -1):
                    deltas.append(deltas[-1].dot(self.weights[l].T) * logistic_derivative(predict[l]))
                deltas.reverse()
                for i in range(len(self.weights)):
                    layer = np.atleast_2d(predict[i])
                    delta = np.atleast_2d(deltas[i])
                    self.weights[i] += learning_rate * layer.T.dot(delta)
```



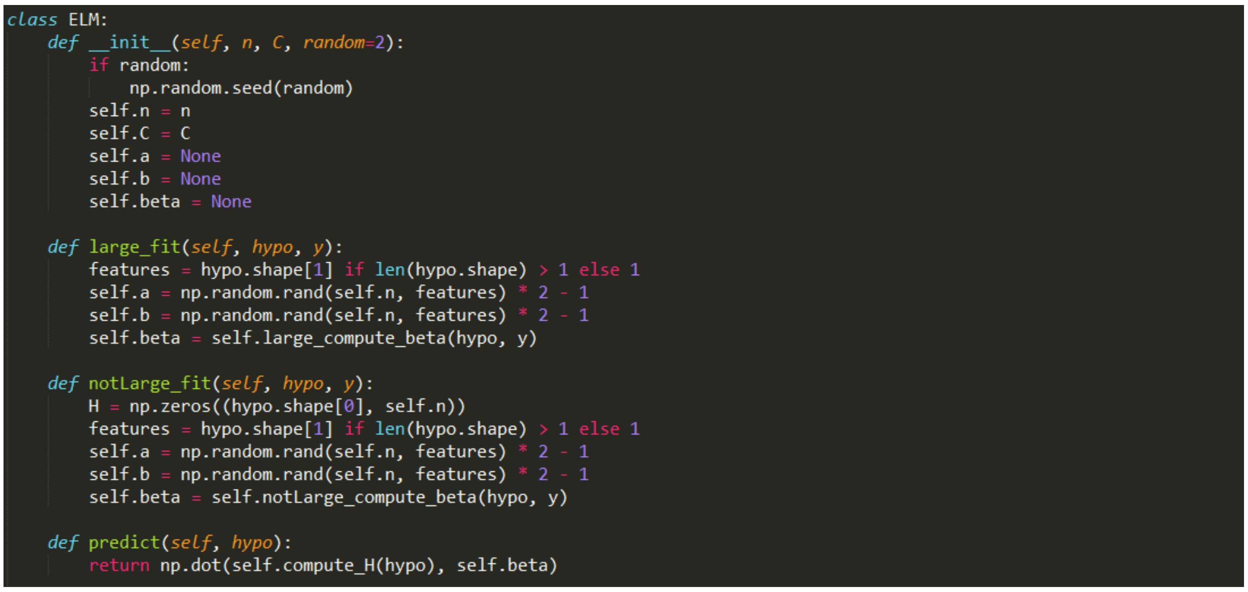
```python
    def predictBatch(self, x):
        predict = np.column_stack((x, np.ones([x.shape[0], 1])))
        for i in range(0, len(self.weights)):
            predict = logistic(np.dot(predict, self.weights[i]))
        return predict
```

- **AdaBoost**

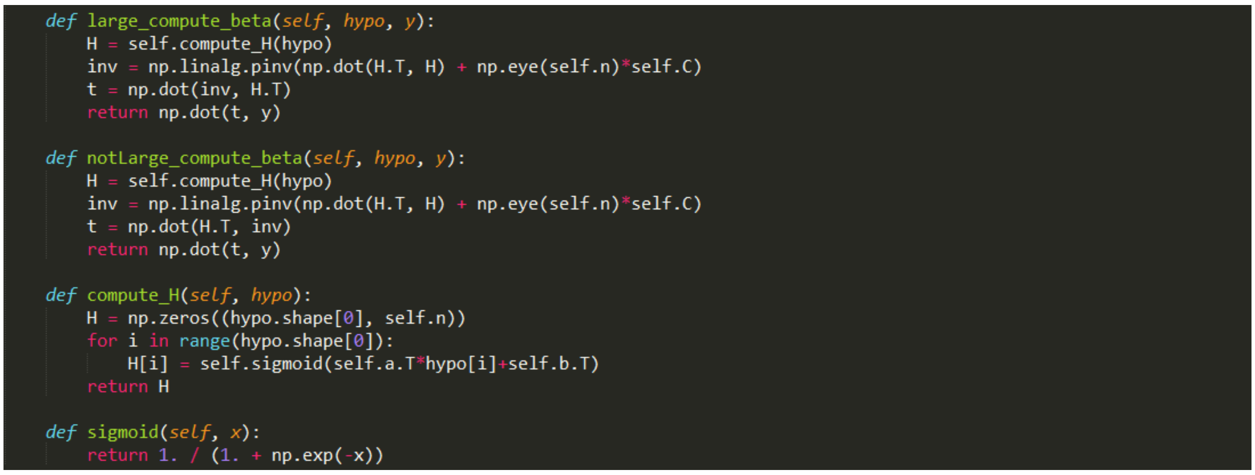


```python
class BP_Adaboost:
    def __init__(self, layers, size=25):
        self.nets = [NeuralNetwork(layers=layers) for i in range(size)]
        self.D = []
        self.alpha_list = []

    def train(self, X, y):
        error_rate = []
        weak_predict = []
        self.D.append(np.ones(X.shape[0]) / X.shape[0])
        for i in range(len(self.nets)):
            # print i
            self.nets[i].fit(X, y)
            weak_predict.append(self.nets[i].predictBatch(X))
            std = (weak_predict[i]-y).std(axis=0)
            error_rate.append(np.sum(np.transpose(abs(weak_predict[i]-y)>(std/2)) * self.D[-1]))
            delta = 1e-6
            self.alpha_list.append(0.5 * np.log(1. / (error_rate[i] + delta) - 1))
            self.D.append(self.D[-1] * flatten((np.exp(-self.alpha_list[i] * weak_predict[i] * y)).tolist()))
            Dsum = np.sum(self.D[-1])
            self.D[-1] /= Dsum

    def predict(self, X):
        X = np.array(X)
        weak_predict = []
        alphaSum = np.sum(self.alpha_list)
        self.alpha_list /= alphaSum
        predict = np.zeros((X.shape[0], 1))
        for i in range(len(self.nets)):
            weak_predict.append(self.nets[i].predictBatch(X).tolist())
            weak_predict[i] = np.array(weak_predict[i])
            predict += self.alpha_list[i]*weak_predict[i]
        return predict
```

- **Extreme Learning Machine**



```python
class ELM:
    def __init__(self, n, C, random=2):
        if random:
            np.random.seed(random)
        self.n = n
        self.C = C
        self.a = None
        self.b = None
        self.beta = None

    def large_fit(self, hypo, y):
        features = hypo.shape[1] if len(hypo.shape) > 1 else 1
        self.a = np.random.rand(self.n, features) * 2 - 1
        self.b = np.random.rand(self.n, features) * 2 - 1
        self.beta = self.large_compute_beta(hypo, y)

    def notLarge_fit(self, hypo, y):
        H = np.zeros((hypo.shape[0], self.n))
        features = hypo.shape[1] if len(hypo.shape) > 1 else 1
        self.a = np.random.rand(self.n, features) * 2 - 1
        self.b = np.random.rand(self.n, features) * 2 - 1
        self.beta = self.notLarge_compute_beta(hypo, y)

    def predict(self, hypo):
        return np.dot(self.compute_H(hypo), self.beta)
```



```python
    def large_compute_beta(self, hypo, y):
        H = self.compute_H(hypo)
        inv = np.linalg.pinv(np.dot(H.T, H) + np.eye(self.n)*self.C)
        t = np.dot(inv, H.T)
        return np.dot(t, y)

    def notLarge_compute_beta(self, hypo, y):
        H = self.compute_H(hypo)
        inv = np.linalg.pinv(np.dot(H.T, H) + np.eye(self.n)*self.C)
        t = np.dot(H.T, inv)
        return np.dot(t, y)

    def compute_H(self, hypo):
        H = np.zeros((hypo.shape[0], self.n))
        for i in range(hypo.shape[0]):
            H[i] = self.sigmoid(self.a.T*hypo[i]+self.b.T)
        return H

    def sigmoid(self, x):
        return 1. / (1. + np.exp(-x))
```

四、实验与结果

此次实验，一共在七个数据集上对提出算法进行验证。得出算法的实验结果将与另外三个比较算法在相同数据集上进行对比。其中，七个数据集包括六个与论文相同的，来自 UCI 数据集的数据集；一个是在机器学习课程回归算法实验中用到的数据集。比较算法包括 BP Neural Network，Robust AdaBoost，以及 ELM。

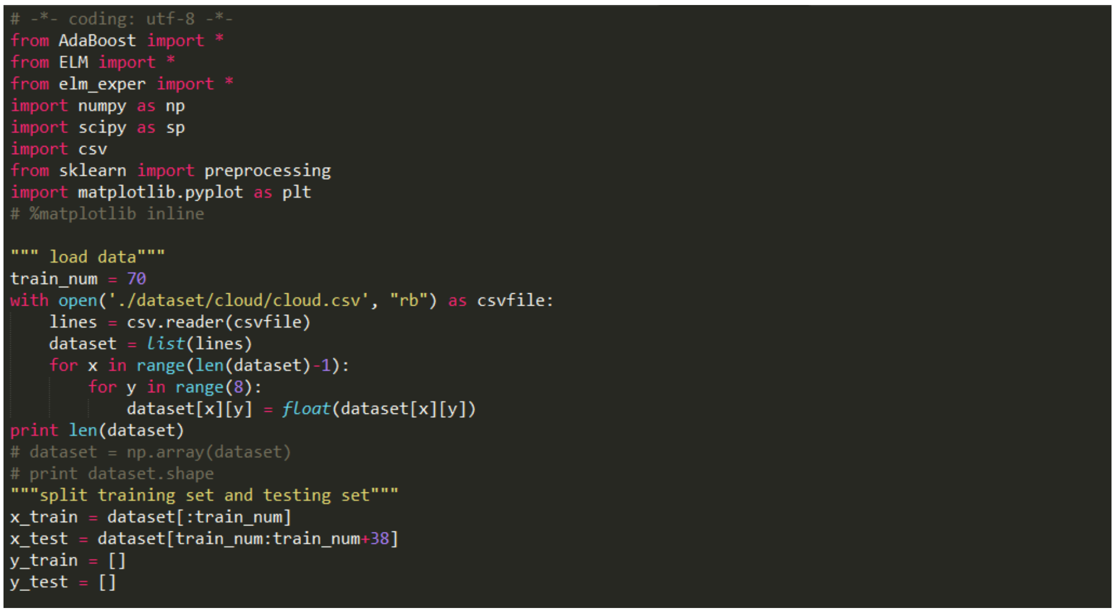
实验数据集

Dataset	Training Data	Testing Data	Features

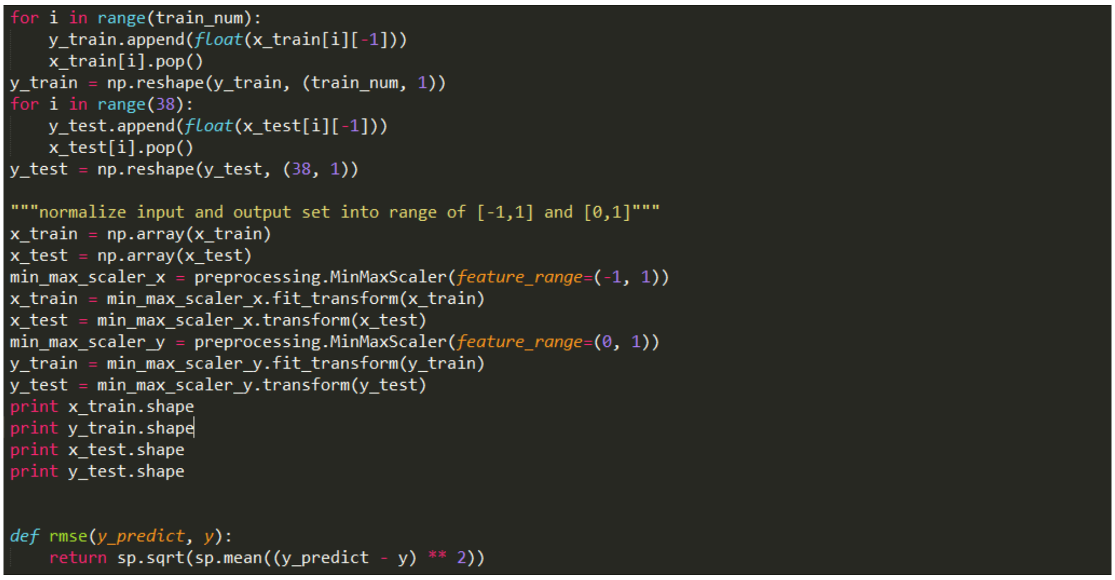
Cloud	70	38	9
Housing	354	152	13
Auto-MPG	320	72	7
Computer hardware	110	99	7
Breast Cancer	114	80	32
Abalone	3050	1127	7
LVST	80	45	308

1.　实验主要代码截图

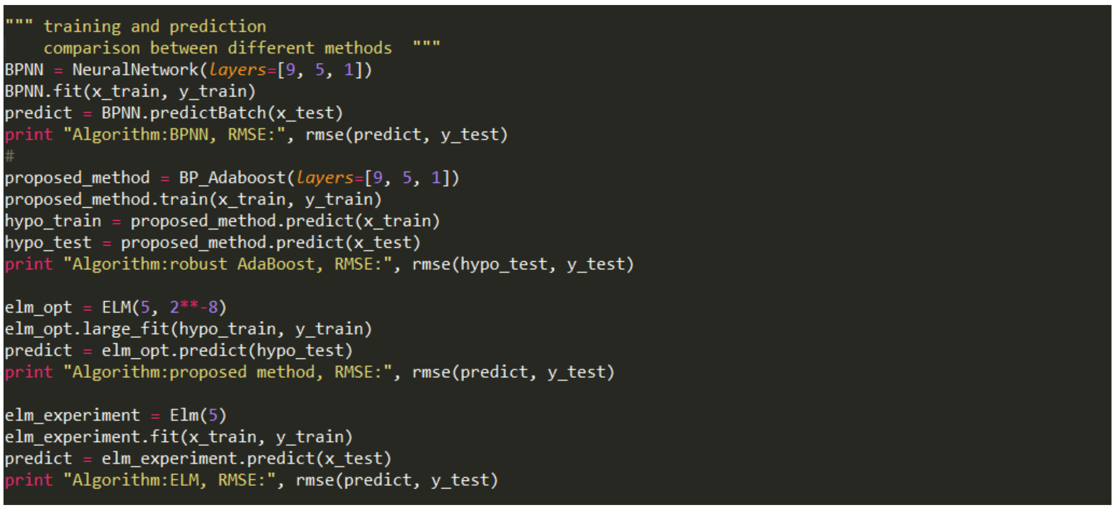
（详见 cloud_train.py，Housing_train.py，AutoMPG_train.py，Computer Hardware.py，BreastCancer.py，Abalone_train.py，LVST.py）



```
# -*- coding: utf-8 -*-
from AdaBoost import *
from ELM import *
from elm_exper import *
import numpy as np
import scipy as sp
import csv
from sklearn import preprocessing
import matplotlib.pyplot as plt
# %matplotlib inline

""" load data"""
train_num = 70
with open('./dataset/cloud/cloud.csv', "rb") as csvfile:
    lines = csv.reader(csvfile)
    dataset = list(lines)
    for x in range(len(dataset)-1):
        for y in range(8):
            dataset[x][y] = float(dataset[x][y])
print len(dataset)
# dataset = np.array(dataset)
# print dataset.shape
"""split training set and testing set"""
x_train = dataset[:train_num]
x_test = dataset[train_num:train_num+38]
y_train = []
y_test = []
```

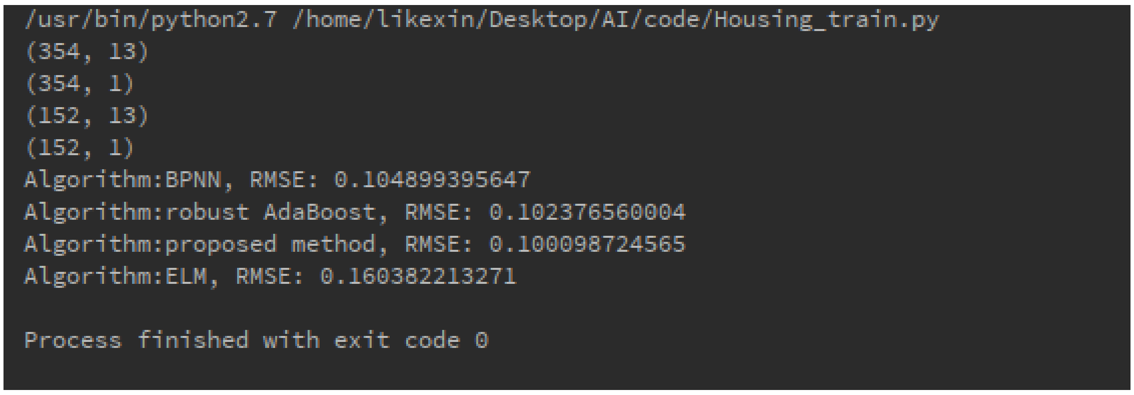


```python
for i in range(train_num):
    y_train.append(float(x_train[i][-1]))
    x_train[i].pop()
y_train = np.reshape(y_train, (train_num, 1))
for i in range(38):
    y_test.append(float(x_test[i][-1]))
    x_test[i].pop()
y_test = np.reshape(y_test, (38, 1))

"""normalize input and output set into range of [-1,1] and [0,1]"""
x_train = np.array(x_train)
x_test = np.array(x_test)
min_max_scaler_x = preprocessing.MinMaxScaler(feature_range=(-1, 1))
x_train = min_max_scaler_x.fit_transform(x_train)
x_test = min_max_scaler_x.transform(x_test)
min_max_scaler_y = preprocessing.MinMaxScaler(feature_range=(0, 1))
y_train = min_max_scaler_y.fit_transform(y_train)
y_test = min_max_scaler_y.transform(y_test)
print x_train.shape
print y_train.shape
print x_test.shape
print y_test.shape


def rmse(y_predict, y):
    return sp.sqrt(sp.mean((y_predict - y) ** 2))
```

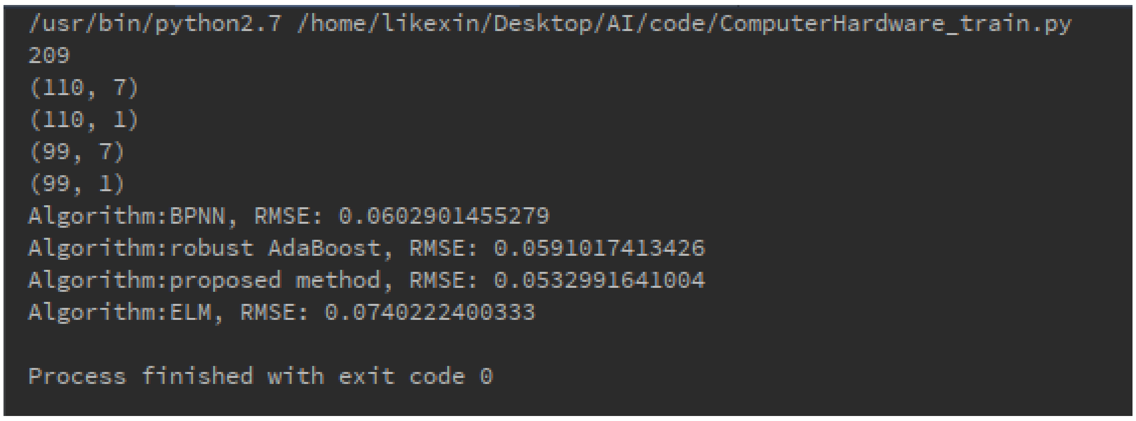


```python
""" training and prediction
    comparison between different methods  """
BPNN = NeuralNetwork(layers=[9, 5, 1])
BPNN.fit(x_train, y_train)
predict = BPNN.predictBatch(x_test)
print "Algorithm:BPNN, RMSE:", rmse(predict, y_test)
#
proposed_method = BP_Adaboost(layers=[9, 5, 1])
proposed_method.train(x_train, y_train)
hypo_train = proposed_method.predict(x_train)
hypo_test = proposed_method.predict(x_test)
print "Algorithm:robust AdaBoost, RMSE:", rmse(hypo_test, y_test)

elm_opt = ELM(5, 2**-8)
elm_opt.large_fit(hypo_train, y_train)
predict = elm_opt.predict(hypo_test)
print "Algorithm:proposed method, RMSE:", rmse(predict, y_test)

elm_experiment = Elm(5)
elm_experiment.fit(x_train, y_train)
predict = elm_experiment.predict(x_test)
print "Algorithm:ELM, RMSE:", rmse(predict, y_test)
```

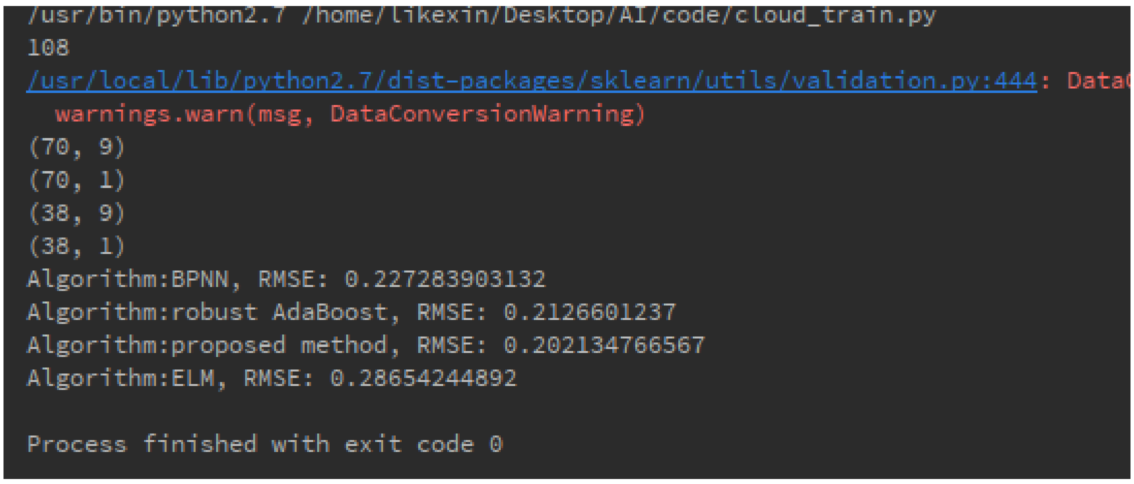
2. 实验结果截图



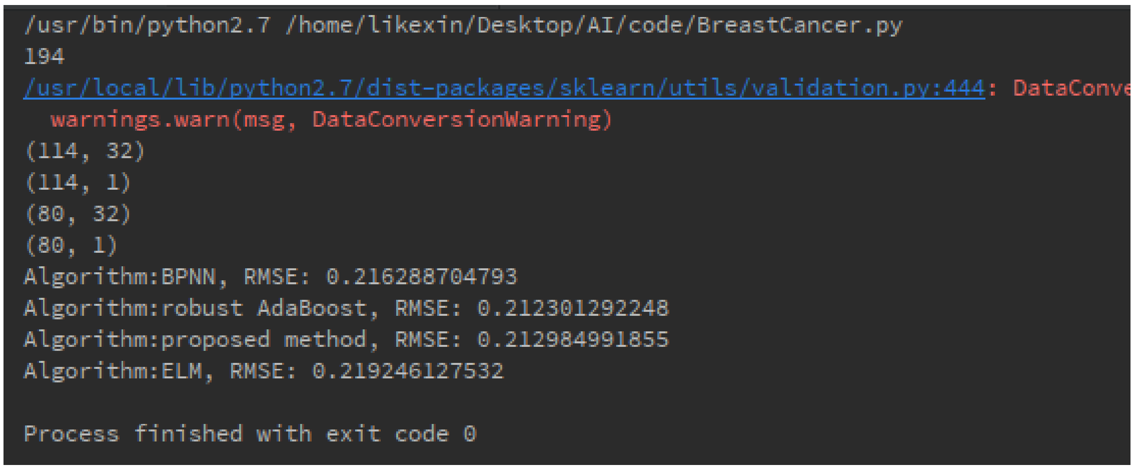
```
/usr/bin/python2.7 /home/likexin/Desktop/AI/code/Housing_train.py
(354, 13)
(354, 1)
(152, 13)
(152, 1)
Algorithm:BPNN, RMSE: 0.104899395647
Algorithm:robust AdaBoost, RMSE: 0.102376560004
Algorithm:proposed method, RMSE: 0.100098724565
Algorithm:ELM, RMSE: 0.160382213271

Process finished with exit code 0
```



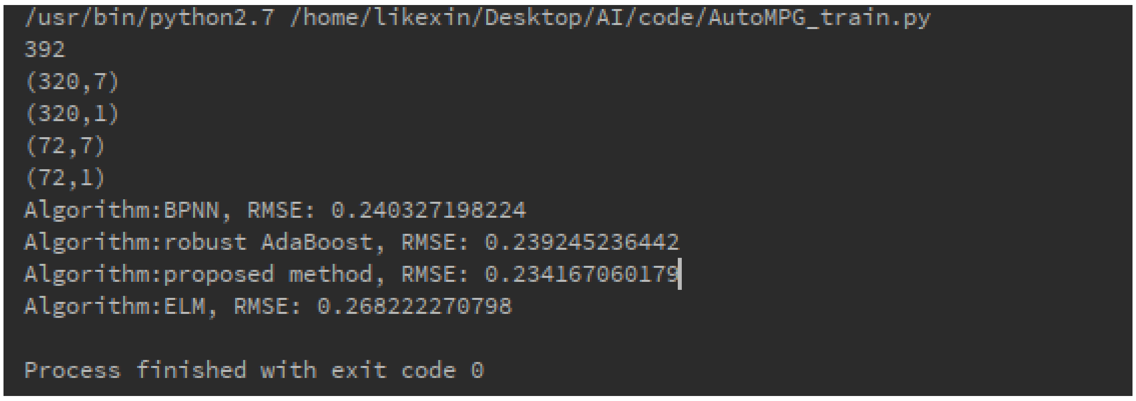
```
/usr/bin/python2.7 /home/likexin/Desktop/AI/code/ComputerHardware_train.py
209
(110, 7)
(110, 1)
(99, 7)
(99, 1)
Algorithm:BPNN, RMSE: 0.0602901455279
Algorithm:robust AdaBoost, RMSE: 0.0591017413426
Algorithm:proposed method, RMSE: 0.0532991641004
Algorithm:ELM, RMSE: 0.0740222400333

Process finished with exit code 0
```



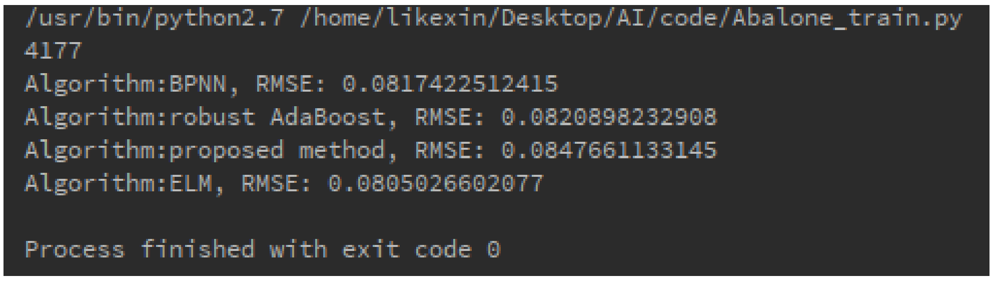
```
/usr/bin/python2.7 /home/likexin/Desktop/AI/code/cloud_train.py
108
/usr/local/lib/python2.7/dist-packages/sklearn/utils/validation.py:444: Data
  warnings.warn(msg, DataConversionWarning)
(70, 9)
(70, 1)
(38, 9)
(38, 1)
Algorithm:BPNN, RMSE: 0.227283903132
Algorithm:robust AdaBoost, RMSE: 0.2126601237
Algorithm:proposed method, RMSE: 0.202134766567
Algorithm:ELM, RMSE: 0.28654244892

Process finished with exit code 0
```



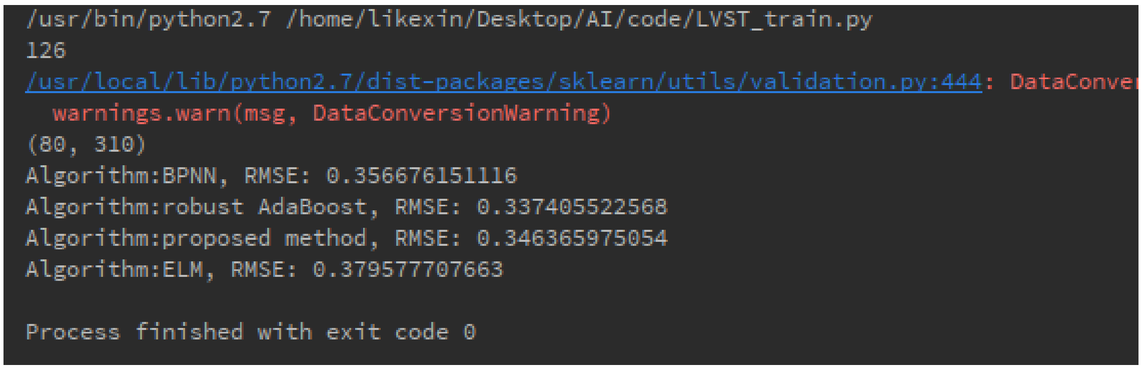
```
/usr/bin/python2.7 /home/likexin/Desktop/AI/code/BreastCancer.py
194
/usr/local/lib/python2.7/dist-packages/sklearn/utils/validation.py:444: DataConve
  warnings.warn(msg, DataConversionWarning)
(114, 32)
(114, 1)
(80, 32)
(80, 1)
Algorithm:BPNN, RMSE: 0.216288704793
Algorithm:robust AdaBoost, RMSE: 0.212301292248
Algorithm:proposed method, RMSE: 0.212984991855
Algorithm:ELM, RMSE: 0.219246127532

Process finished with exit code 0
```



```
/usr/bin/python2.7 /home/likexin/Desktop/AI/code/AutoMPG_train.py
392
(320,7)
(320,1)
(72,7)
(72,1)
Algorithm:BPNN, RMSE: 0.240327198224
Algorithm:robust AdaBoost, RMSE: 0.239245236442
Algorithm:proposed method, RMSE: 0.234167060179
Algorithm:ELM, RMSE: 0.268222270798

Process finished with exit code 0
```



```
/usr/bin/python2.7 /home/likexin/Desktop/AI/code/Abalone_train.py
4177
Algorithm:BPNN, RMSE: 0.0817422512415
Algorithm:robust AdaBoost, RMSE: 0.0820898232908
Algorithm:proposed method, RMSE: 0.0847661133145
Algorithm:ELM, RMSE: 0.0805026602077

Process finished with exit code 0
```



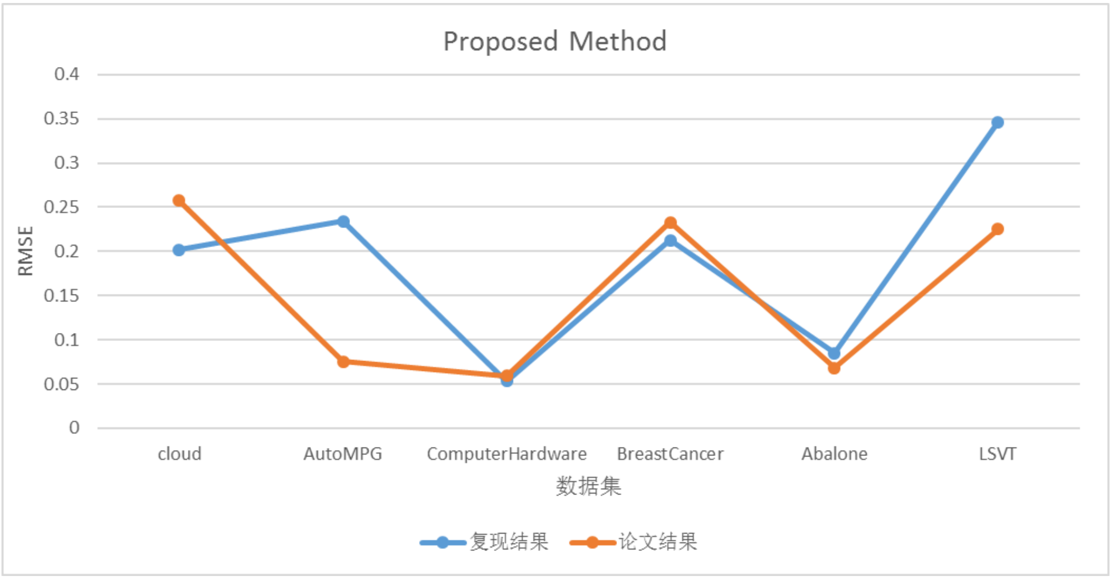
```
/usr/bin/python2.7 /home/likexin/Desktop/AI/code/LVST_train.py
126
/usr/local/lib/python2.7/dist-packages/sklearn/utils/validation.py:444: DataConve
  warnings.warn(msg, DataConversionWarning)
(80, 310)
Algorithm:BPNN, RMSE: 0.356676151116
Algorithm:robust AdaBoost, RMSE: 0.337405522568
Algorithm:proposed method, RMSE: 0.346365975054
Algorithm:ELM, RMSE: 0.379577707663

Process finished with exit code 0
```

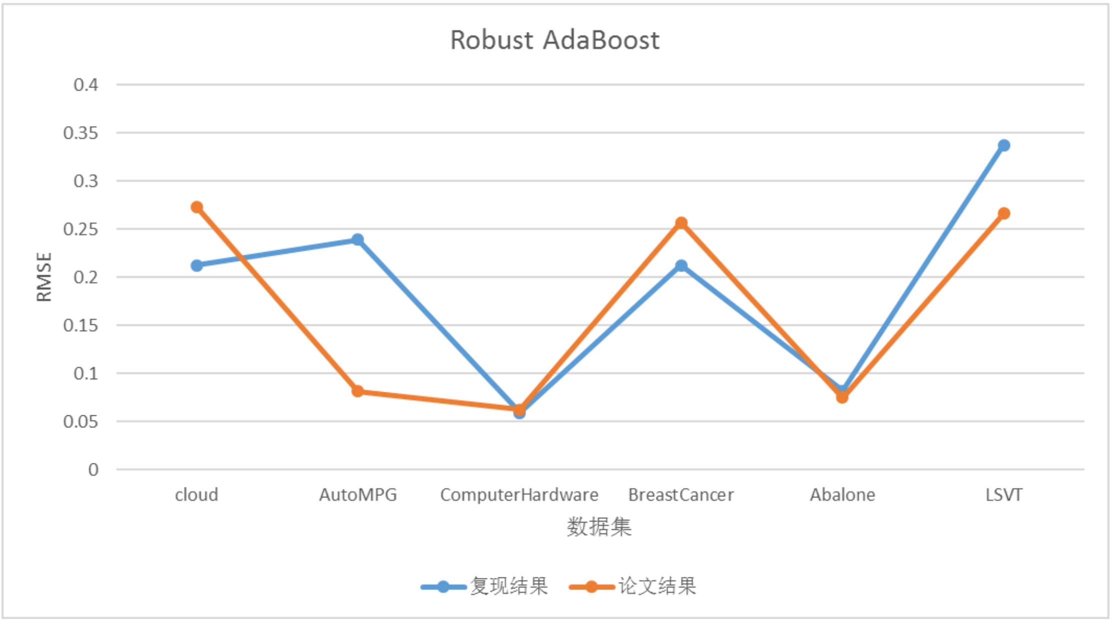
3. 衡量 RMSE 值的实验结果

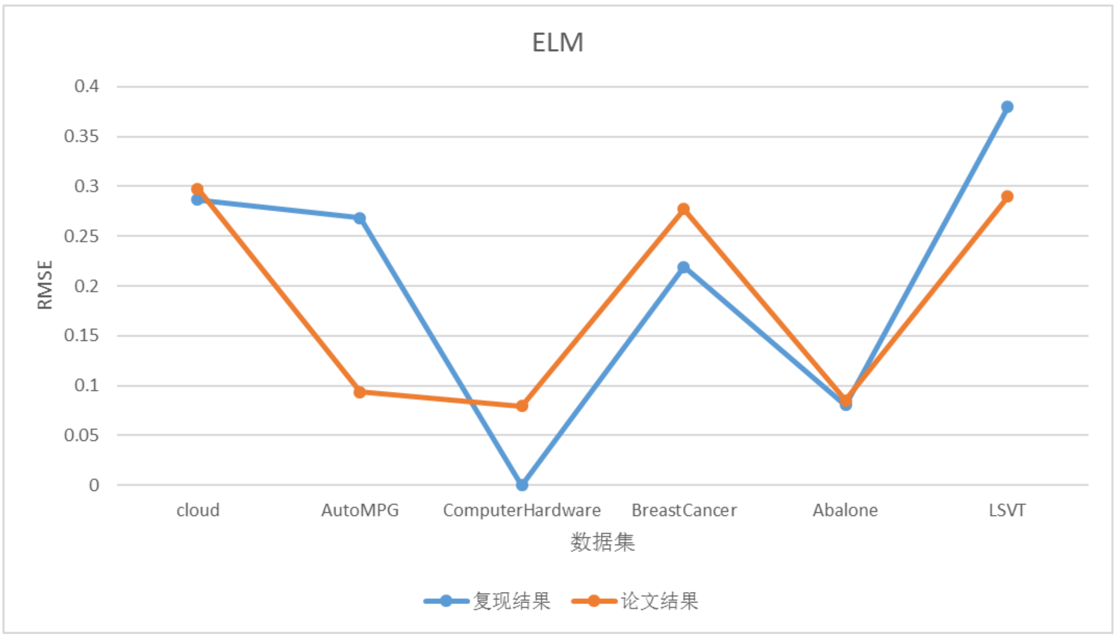
Dataset	Proposed Method	Adaboost	ELM	BPNN
Cloud	0.202134767	0.212660124	0.286542449	0.227283903
Housing	0.100098725	0.10237656	0.160382213	0.104899396
Auto-MPG	0.23416706	0.239245236	0.268222271	0.240327198
Computer hardware	0.053299164	0.059101741	0.07402224	0.060290146
Breast Cancer	0.212984992	0.212301292	0.219246128	0.216288705
Abalone	0.084766113	0.082089823	0.08050266	0.081742251
LVST	0.346365975	0.337405523	0.379577708	0.356676151

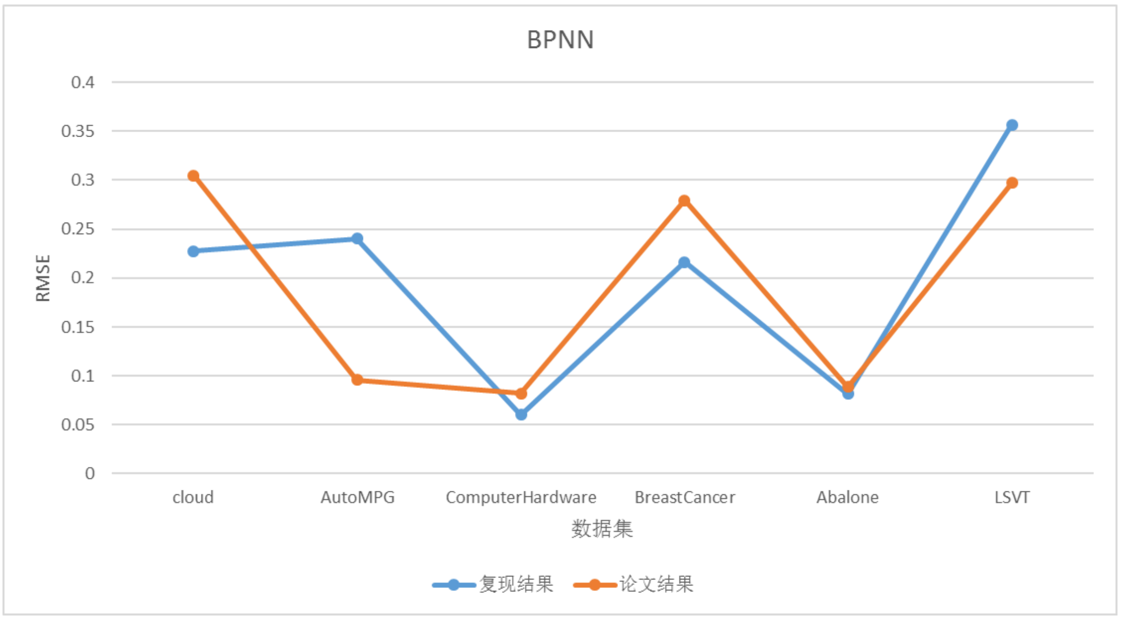
4. 复现结果与论文结果的比较

在与论文中使用相同的数据集上，对四个方法进行结果比较。比较结果如下图所示。可以看出，复现结果与论文结果基本相近。且在 67% 的数据集上，复现结果有更优的表现。



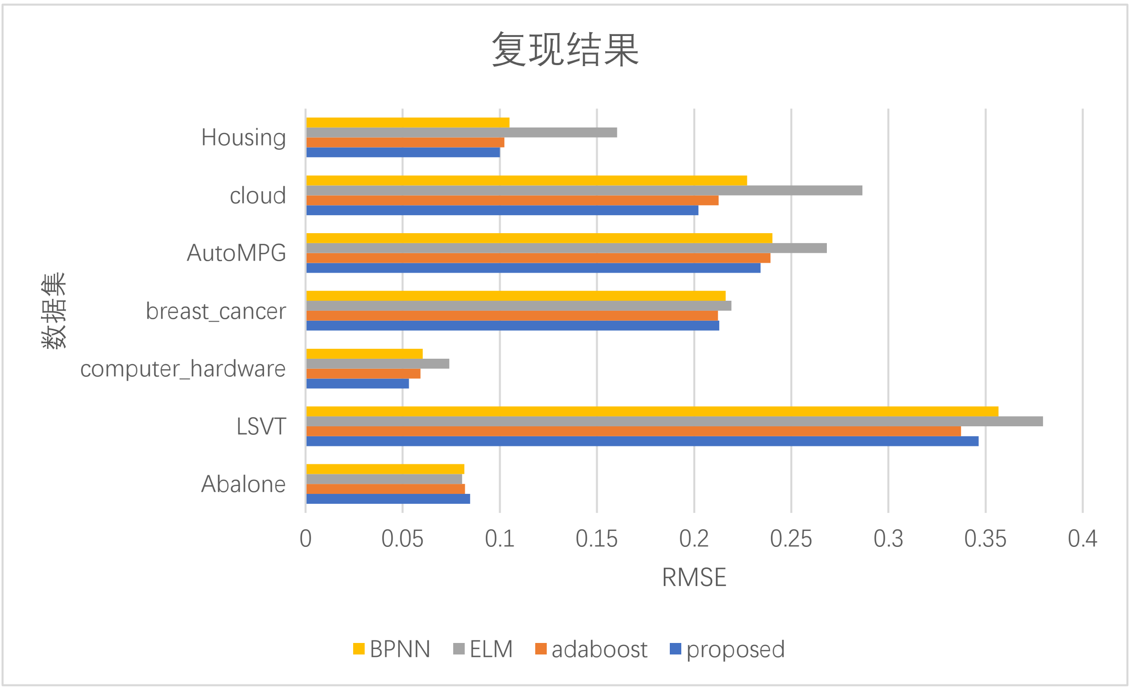






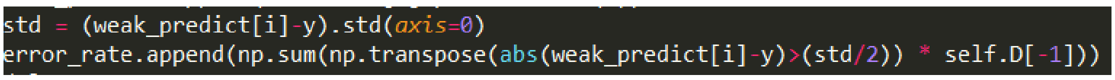
5. 复现结果分析

复现结果如下图所示。可以看出，在 71%的数据集上，Proposed Method 的表现都要略微优于其他三个比较算法；剩余数据集上(Breast Cancer，Abalone)，四个方法的表现几乎一致。然而，复现结果显示提出方法总体表现虽然是最优的，但它并没有带来显著的优化能力。



五、总结

在机器学习的课程上接触了 **Adaboost** 算法，在实验中对于截止阈值的选择是按照实验说明的要求设定的，当时虽然对它有些困惑但也并没有对它给予过多的关注。在这次的课程任务中，通过对论文的学习，对 **AdaBoost** 阈值的确定方法有了新的认识——弱学习器在 **m** 个样本上的预测值的统计特点被用以自适应。



```
std = (weak_predict[i]-y).std(axis=0)
error_rate.append(np.sum(np.transpose(abs(weak_predict[i]-y)>(std/2)) * self.D[-1]))
```

另外，对 **BP** 算法和 **ELM** 算法的实现，也让我对于神经网络也有了更加具体的认识。

整个课程任务虽然没有想象中的困难，但从论文选择、论文研读和翻译，再到算法的理解和实现，最后进行复现实验再与论文结果相比较，整个过程也耗费了一定的时间与精力。能自己完整地完成整个过程中的所有任务，其实是非常高兴的，我也在其中体会到了很大的乐趣，对未来进行进一步的学习充满了信心和期待。