Indian Institute of Information Technology, Vadodara

Intro to AI(CS308)

Mr. Parth Gupta

Name - Dilip Puri
ID - 201351014

Assign 02

Deadline -

---

1 Implementing Perceptron

In the Perceptron are going to classify data whether it is belongs to particular class or not so we have given dataset. Dataset consists features and their original belonging class.
For this problem we have given IRIS dataset there are three classes of in this dataset but we are going implement binary classification so either particular instance belongs to class or not?
Dataset have four features and class in name:

X = [petal_length petal_width sepal_length sepal_width class]
Classes: [Setosa Verginica Vermicolor]

Now we are going to introduce **Weight vector** which will control the classification of instance of dataset.

$$\mathbf{w} = [w_1 \ w_2 \ w_3 \ w_4]$$

Our classification function will be like this:

Belongs to class if $\sum_{i=1}^{d} w_i * x_i > threshold$
Not belongs to class if $\sum_{i=1}^{d} w_i^T * x_i < threshold$

so our formula be

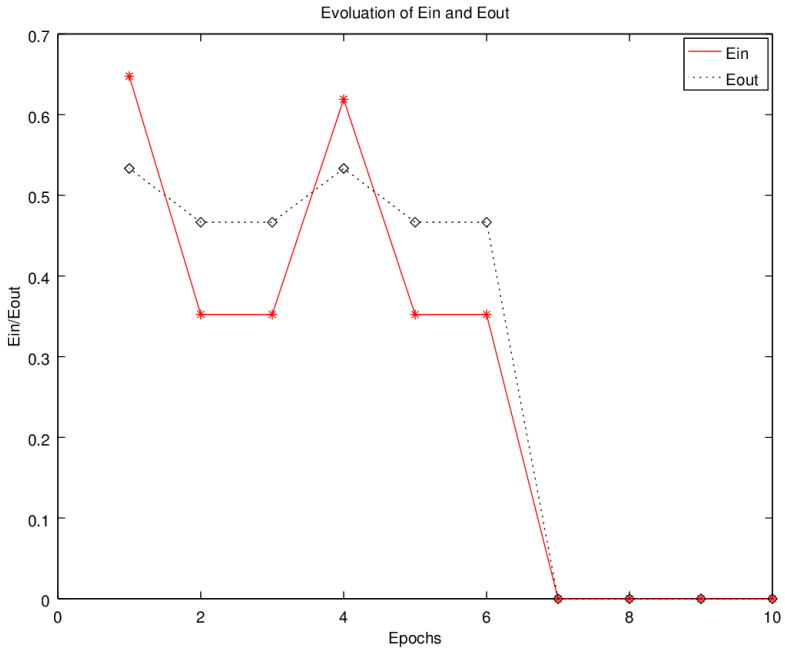
$$H(x) = sign((\sum_{i=1}^{d} w_i * x_i) + b)$$

Where $x_1 \cdots x_4$ our features of vector x;
$H(x) = +1$ means "belongs to class"
$H(x) = -1$ means "not belongs to class"
Note :- Our class is **"Setosa"**
The weights are $w_1 \cdots w_d$, and the threshold is determined by the bias term b.

Now the learning of the Perceptron is Weight so the algorithm will determine what w should be. Our learning algorithm will find this w using a simple iterative method. And we have update rule for w is if we have $y(t) \neq sign(w(t) * x(t))$ $w(t+1) = w(t) + y(t) * x(t)$

Results and Graphs

```
# Created by Octave 3.8.1, Fri Feb 05 02:38:27 2016 IST <dilip@dilip-notebook-pc>
# name: datasize
# type: matrix
# rows: 1
# columns: 2
 150 5


# name: train_test_ratio
# type: matrix
# rows: 1
# columns: 2
 0.7 0.3


# name: weight_initial
# type: matrix
# rows: 1
# columns: 4
 100 100 100 100


# name: weight_final
# type: matrix
# rows: 1
# columns: 4
 15.90000000000041 227.6000000000001 -335.2000000000003 -86.89999999999988


# name: err_in
# type: matrix
# rows: 1
# columns: 10
 0.6476190476190476 0.3523809523809524 0.3523809523809524 0.6190476190476191
    0.3523809523809524 0.3523809523809524 0 0 0 0


# name: err_out
# type: matrix
# rows: 1
# columns: 10
 0.5333333333333333 0.4666666666666667 0.4666666666666667 0.5333333333333333
    0.4666666666666667 0.4666666666666667 0 0 0 0
```




```
# name: weight_initial
1,1,1,1
# name: weight_final
81.50000000000041,248.7000000000001,-399.4000000000007,-182.4000000000003

# name: weight_initial
10,10,10,10
# name: weight_final
90.50000000000045,257.7000000000001,-390.4000000000007,-173.4000000000003

# name: weight_initial
100,100,100,100
# name: weight_final
15.90000000000041,227.6000000000001,-335.2000000000003,-86.89999999999988

# name: weight_initial
500,500,500,500
# name: weight_final
-91.99999999999956,388.6999999999998,-328.5000000000002,179.899999999999


# name: weight_initial
1000,1000,1000,1000
# name: weight_final
-206.5,631.2999999999992,-385.2000000000013,478.7000000000052
```

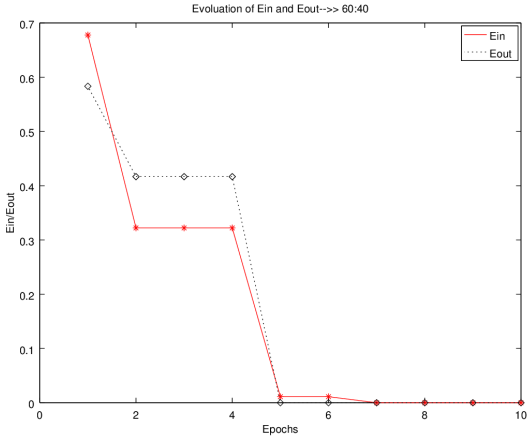


Figure 1: Train:Test::60:40

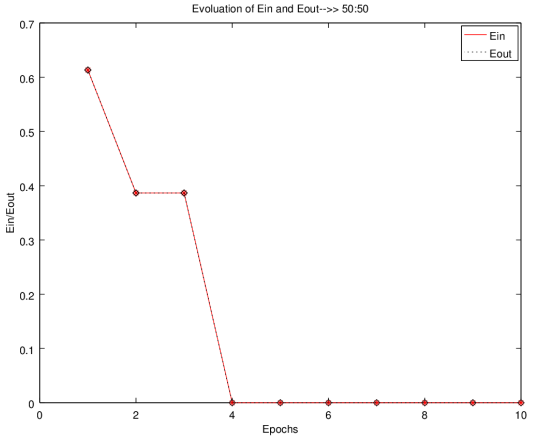


Figure 2: Train:Test::50:50

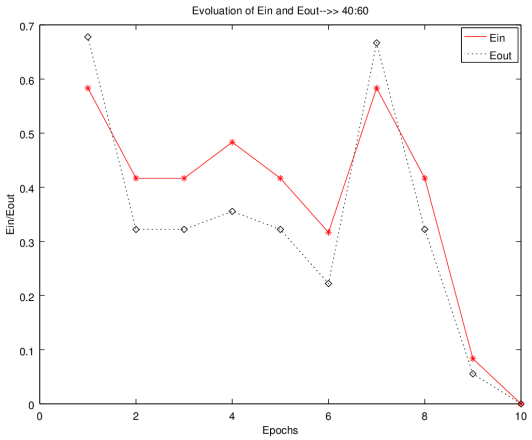


Figure 3: Train:Test::40:60

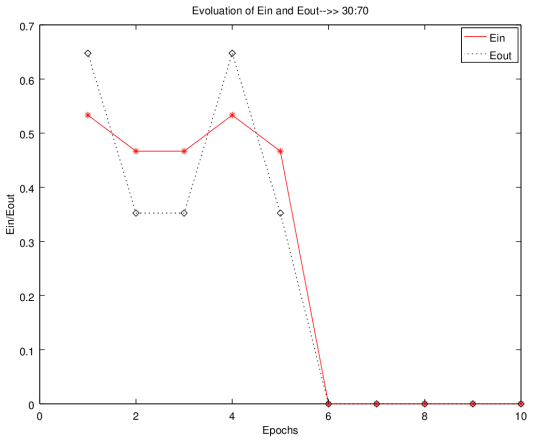
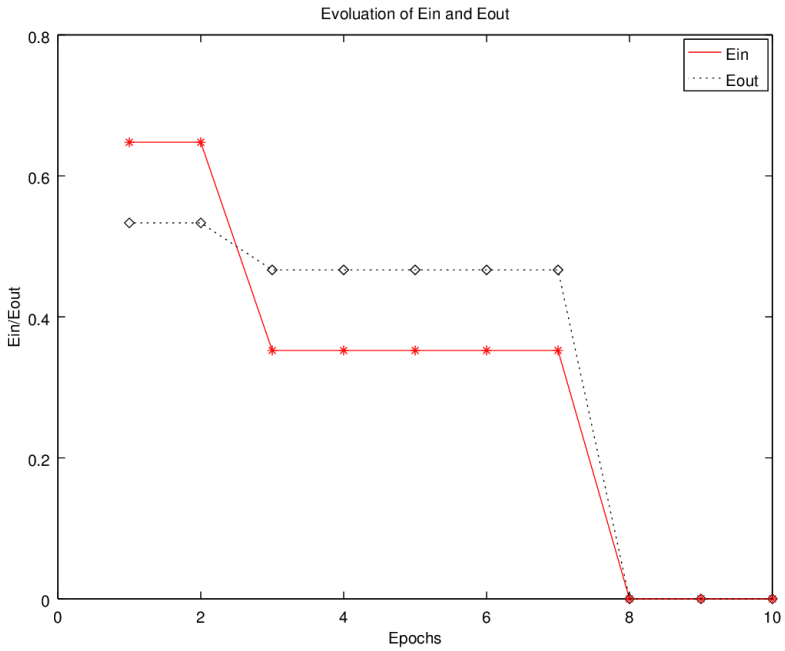


Figure 4: Train:Test::30:70

2 Pocket Algorithm



3 Regression and Nonlinear Regression

```
#weight final in regression
0.02582718291452129 0.4060820001850484 -0.4283487369024855 -0.08903770168733846

#weight final in nonlinear regression
-0.207193526479319 1.391279626104991 -0.5851351232399695 -1.134849748736726
    -0.02688609765220644
-0.08030810262686866 -0.09542183006176314 0.05210398582006542 0.01360351395127204
    -0.1148776084771055
0.0942798842068459 0.3558923732283532
```