Neural & Fuzzy Systems Assignment 1

MU XINYI G2101432J

1.INTRODUCTION

The Radial Basis Function (RBF) network is a type of feed forward neural network. In RBF networks, the output of hidden layers are linear combination and fed to the output layer, therefore, the weight between the hidden layer and the output layer can be solved by the close-form function. RBF networks are commonly used for function approximation problems and can also be used for time series forecasting or classification.

The Support Vector Machine (SVM) is a type of generalized linear classifier that performs binary classification of data according to supervised learning. Its decision boundary is the maximum-margin hyperplane that solves the learning sample. SVM can perform non-linear classification through the kernel method and is one of the common kernel learning methods.

In this assignment, we need to train two neural networks classifiers to solve a two-class pattern classification problem. The first one is an RBF neural network classifier and the second one is a kernel SVM classifier.

The progress of how I obtained the predict labels will be shown first, and then I will compare and discuss these two classifiers' performance. The subfunctions are in the appendix.

2.METHODS

1)Data Preparation:

```
load('data_train.mat')
load('label_train.mat')

data_labels = [data_train,label_train];
data_labels = shuffling(data_labels);% exclude influence of data's sequence
% Devide training data into two parts randomly
[train_set,val_set,test_set] = divide_data(data_labels,20,0);
train_labels = train_set(:,end);
val_labels = val_set(:,end);
train_data = train_set(:,1:end-1);
val_data =  val_set(:,1:end-1);
```

Firstly, the data should be shuffled to exclude the influence of the data's sequence.

Secondly, a set-divide function is used for randomly picking up data from the dataset so the original training data set can be divided into a new training set and a validation set. The validation set is used for evaluating the performance of two classifiers, and adjusting parameters to select the one corresponding to the best model.

2) RBF Classifier:

```
HNeurons = 25;% number of neurons in the hidden layer
iter_limit = 100000;
E = 10^-5;
learning_rate = 0.1;
sigma = 8; % d_max * sqrt(2*m)
CT = som(train_data, HNeurons, learning_rate, E, iter_limit);% use SOM to find centers
Phi = CPhi(train_data,CT,sigma);
[rows, ~] = size(Phi);
Phi = [Phi, ones(rows,1)];
W = regression(Phi, train_labels);

% test on training set:
train_output = Phi * W;
Lminus = train_output<0;
Lplus = train_output>=0;
train_output(Lminus) = -1;
train_output(Lplus) = 1;
variation = train_output - train_labels;

accuracy_train = 1 - length(nonzeros(variation))/length(train_output)
```

```
accuracy_train = 0.9356
```

```
% test on validation set:
Phi = CPhi(val_data,CT,sigma);
[rows, ~] = size(Phi);
Phi = [Phi, ones(rows,1)];
val_output = Phi * W;
Lminus = val_output<0;
Lplus = val_output>=0;
val_output(Lminus) = -1;
val_output(Lplus) = 1;
variation = val_output - val_labels;

accuracy_val = 1 - length(nonzeros(variation))/length(val_output)
```

```
accuracy_val = 0.9242
```

```
load('data_test.mat')
Phi = CPhi(data_test,CT,sigma);
[rows, ~] = size(Phi);
Phi = [Phi, ones(rows,1)];
test_output = Phi * W;

test_output'
```

```
ans = 1×21
    0.7938   -0.1834    1.2371   -0.0721    1.2377   -1.2645    0.5436   -0.6179 ···
```

```
test_label = test_output;
Lminus = find(test_label<0);
Lplus = find(test_label>=0);
test_label(Lminus) = -1;
```

```
test_label(Lplus) = 1;

test_label'
```

```
ans = 1×21
     1    -1     1    -1     1    -1     1    -1     1    -1     1    -1     1 ···
```

A bi-polar activation function is used in the final layer, which means zero is the threshold.The performance on the validation set is almost equal to the training set, therefore, the classifier has good generalization and there is almost no overfitting occured.

3) SVM Classifier:

```
L = nominal(train_labels);
gamma = 1;% determined by trial
SVMModel=fitcsvm(train_data,L,'KernelFunction','rbf','BoxConstraint',1,'KernelScale',gamma);

% Accuracy on training set
train_label_pred = predict(SVMModel, train_data);
cp = classperf(nominal(train_labels), train_label_pred);

accuracy_train_svm = cp.CorrectRate
```

```
accuracy_train_svm = 0.9432
```

```
% Accuracy on validation set
val_label_pred = predict(SVMModel, val_data);
cp = classperf(nominal(val_labels), val_label_pred);

accuracy_val_svm = cp.CorrectRate
```

```
accuracy_val_svm = 0.9545
```

```
% Classification and output
load('data_test.mat')
test_label_pred = predict(SVMModel, data_test);

test_label_pred'
```

```
ans = 1×21 nominal 数组
     1      -1      1      -1      1      -1      1      -1      1      -1      1      -1      1      1      1
```

The accuracy of training set is close to the accuracy of validation set, which means the SVM classifier also performs well and no overfitting occured.

3.DISCUSSIONS

In this assignment, a two-class pattern classification problem is solved by using two classifier. In first classifier RBF, the accuracy of training set is 93.56% while the validation set's accuracy is 92.42%. These two values are close to each other, so the RBF classifier has a good generalization ability and there is almost no overfitting occured. As for the second one, namely the SVM classifier, the accuracy of training set is 94.32% and the

accuracy of validation set is 95.45%, which are both higher than the RBF classifier. The SVM classifier performs even better than the RBF classifier.

The predicting labels from two classifier are the same, which is the transpose of the following matrix:

$$[1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, 1, 1, -1, 1, -1, 1, -1, 1]$$

4.APPENDIX

1) Function which exclude the influence of data's sequence

```
function data= shuffling(data)
[n,~] = size(data);
data = data(randperm(n),:);
end
```

2) Function which divides a set of data into train, validation and test sets

The validation_ratio means the percentage of the data that is used as validation set. The test_ratio used above equals zero.

```
function [train_data, val_data, test_data]=divide_data(data, val_ratio,test_ratio)

[n,~] = size(data);
val_amount = round(n * val_ratio/100);
test_amount = round(n * test_ratio/100);

data_indizies = 1:n;
val_indizies = randperm(n,val_amount);% randomly select without repeating elements
val_data = data(val_indizies,:);
data_indizies = setdiff(data_indizies, val_indizies);
data = data(data_indizies,:);
[n,~] = size(data);

data_indizies = 1:n;
test_indizies = randperm(n,test_amount);
test_data = data(test_indizies,:);
data_indizies = setdiff(data_indizies,test_indizies);
train_data = data(data_indizies,:);
end
```

3) Self Organizing Map Function

Used for estimating the centers of RBF.

```
function W = som(data,HNeurons, learning_rate, E, iter_limit)
    [q,m] = size(data);
    n_0 = learning_rate;
    W = randn(HNeurons,m);
    W = W/norm(W);
    delta = E +1;
    iter = 1;
```

```
    data_copy = data;
    while delta > E && iter<iter_limit
        iter2 = mod(iter,q);
        if iter2 == 0
            data_copy = data;
        end
        i = randi([1,q-iter2],1,1);
        x = data_copy(i,:);
        [n_2,~] = size(data_copy);
        filter_i =setdiff(1:n_2,i);
        data_copy = data_copy(filter_i,:);
        X_mat = repmat(x,HNeurons,1)';
        x_dists_to_W = sqrt(sum((X_mat-W').^2,1));
        % determine the winning neuron
        [min_val,min_ind] = min(x_dists_to_W);
        eta_n = n_0 * exp(-iter/1000);
        w_winning = W(min_ind,:);
        % only the winning neuron is updated
        w_update = w_winning + eta_n *1 * min_val;
        delta = norm(w_winning-w_update);
        iter = iter +1;
    end
end
```