EE7207 Assignment 1

Comparison of RBF Neural Network and Kernel SVM on Given Data

Gao Zhaoqi

School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore

Matric. No. G2001905J

GAOZ0012@e.ntu.edu.sg

1. RBF Neural Network

1.1 Algorithm description

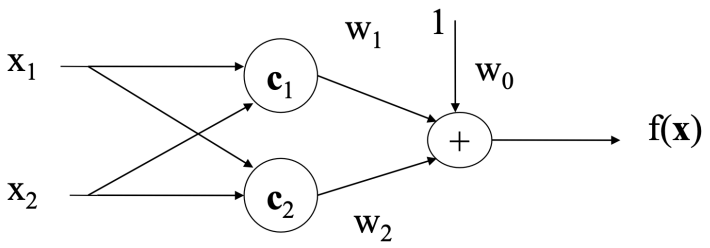

Figure. 1. Architecture of RBF Neural Network

a) Input layer

The input layer of RBF neural network distributes each data sample to every hidden layer neuron, it doesn't perform any computation. For example, each hidden layer neuron will receive the information of all dimensions from one training data sample, and computer the distance between this data sample between its neuron center.

b) Hidden layer

The hidden layer neurons will use a radial basis function to do the nonlinear mapping of the input data. A radial basis function usually has two parameters, center vector c

and width σ. Specifically, center vector c has the same dimension as input data and will compute the distance d between input data and neuron center:

$$d_i = \|x - c_i\|$$

Width σ determine the size of the receptive field. Gaussian function is a commonly used radial basis function:

$$g(d_i) = \exp\left(-\frac{d_i^2}{2\sigma^2}\right)$$

c) Output layer

Output layer can be simply regarded as a linear regression model which will computes the weighted sum of the hidden layer neurons outputs:

$$f(x) = \sum_{j=1}^{m} w_j o_j(x) + w_0$$

In which m is the number of hidden layer neurons, o_j is the output of the jth hidden neuron, w_0 is the weight of the bias term.

1.2 Parameters to Tune

For an RBF neural network training, we need to determine three parameters to train the model, which are:

a) Number of hidden layer neurons m

b) Centers of hidden layer neurons c

c) Width of radial basis function σ

Usually, we can decide these parameters by the following method: For number of hidden neurons m, we can decide by experiment. After we decide the number of hidden neurons c, we can use K-means algorithm to decide m number of centers of training data as the hidden neuron centers c. For radial basis function we can decide the width of the gaussian function by:

$$\sigma = \frac{d_{max}}{\sqrt{2m}}$$

In which d_{max} is the maximum centers of the chosen centers.

2. Kernel SVM

2.1 Algorithm Description

a) Definition

Support vector machine is commonly used as a classifier by using a discriminate function whose parameter is mainly determined by those data samples called support vectors, which are closest to the decision hyperplane.

support vectors——>max(margin of separation)——>decision hyperplane——>classificer

b) Discriminate function (classifier)

For linearly separable data, we can define the discriminate function (hyperplane) as:

$$g(x) = w^T x + b$$

The two classes classifier will work as the following decision rule:

$$\begin{cases} w^T + b \geq 1 & for\ d = +1 \\ w^T + b \leq -1 & for\ d = -1 \end{cases}$$

Where d is the class label, and the data samples which will satisfy the above equation with equality sign are support vectors.

c) Margin of separation

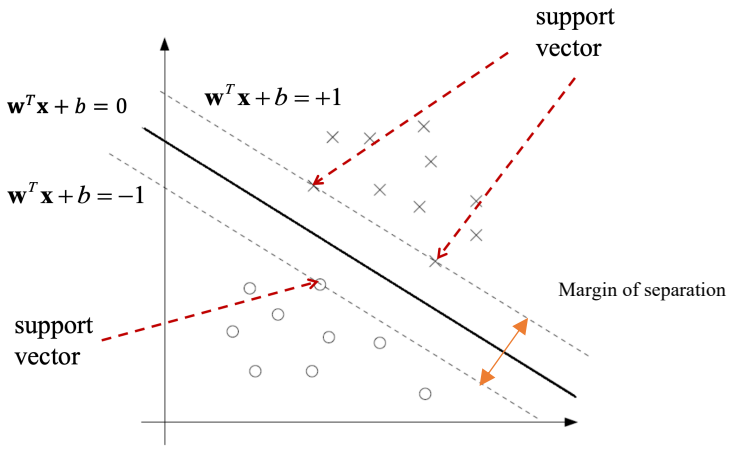

Figure. 2. Illustration of Support Vectors

The learning goal of SVM is to find the optimal hyperplane $w^T x + b = 0$ which will maximize the margin of separation which is the distance between two classed of support vectors.

To express the margin of separation with the function parameter, we firstly represent a data sample x by using an algebraic measure:

$$x = x_p + r\frac{w}{\|w\|}$$

Where x_p is the projection of x on the hyperplane, r is the distance from data sample x to the hyperplane, $\frac{w}{\|w\|}$ is the unite vector of $x - x_p$, which means $\overrightarrow{x - x_p} = r\frac{w}{\|w\|}$. This expression can be easily figure out by the figure showed below:

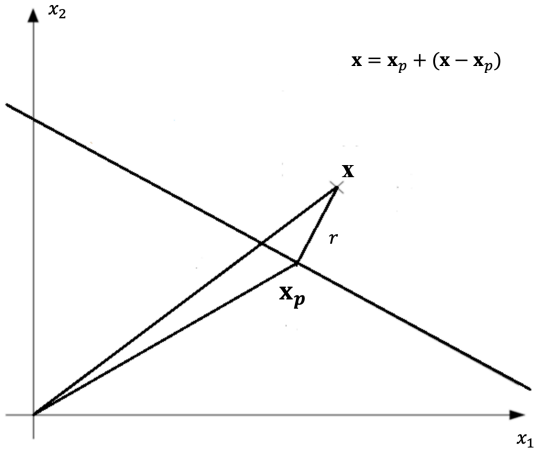

Figure. 3. Illustration of Algebraic Expression of Data Sample x

Thus, we can drive the following representation of the margin of separation by substituting the x in the discriminate function:

$$\begin{aligned} g(x) &= w^T x + b \\ &= w^T\left(x_p + r\frac{w}{\|w\|}\right) + b \\ &= w^T x_p + b + r\frac{w^T w}{\|w\|} \\ &= g(x_p) + r\|w\| \\ &= r\|w\| \end{aligned}$$

Therefore, the distance from the support vector to the optimal hyperplane is:

$$r = \frac{g(x^s)}{\|w\|} = \begin{cases} \frac{1}{\|w\|} & for\ d^s = 1 \\ \frac{-1}{\|w\|} & for\ d^s = -1 \end{cases}$$

So, the margin of separation between two class is:

$$\rho = \frac{2}{\|w\|}$$

d) Optimization problem

Maximizing the margin of separation is equal to minimize $\|w\|$, which is the Euclidean norm of the weight vector w. This is also equivalent to minimize the following function:

$$J(w) = \frac{1}{2}\|w\|^2 = \frac{1}{2}w^T w$$

Also, to correctly classify the training data, we must follow the decision rule of classifier, which can be expressed as:

$$d(i)[w^T x(i) + b] \geq 1 \quad for\ i = 1,2,\dots,N$$

Which is also equivalent to:

$$d(i)[w^T x(i) + b] - 1 \geq 0 \quad for\ i = 1,2,\dots,N$$

Since there are N number of training samples, we have N number of constraints.

So, we form an optimization problem with inequality constraint.

e) Lagrange multiplier

To solve this optimization problem, we can construct the Lagrange function:

$$J(w,b,a) = \frac{1}{2}w^T w - \sum_{i=1}^{N} a(i)(d(i)[w^T x(i) + b] - 1)$$

$$= \frac{1}{2}w^T w - \sum_{i=1}^{N} a(i)d(i)w^T x(i) - b\sum_{i=1}^{N} a(i)d(i) + \sum_{i=1}^{N} a(i)$$

Where $a(i)$ is called the Lagrange multiplier, and $a(i) \geq 0$

To minimize $J(w,b,a)$, the following partial derivative optimality conditions must be satisfied:

KKT condition

$$\begin{cases} \dfrac{\partial J(w,b,a)}{\partial w} = w - \displaystyle\sum_{i=1}^{N} a(i)d(i)x(i) = 0 \\ \dfrac{\partial J(w,b,a)}{\partial b} = -\displaystyle\sum_{i=1}^{N} a(i)d(i) = 0 \end{cases}$$

Therefore, we can know:

$$\begin{cases} w = \displaystyle\sum_{i=1}^{N} a(i)d(i)x(i) \\ \displaystyle\sum_{i=1}^{N} a(i)d(i) = 0 \end{cases}$$

So, by substituting above representation into $J(w,b,a)$, we have:

$$\text{Max} \qquad J(w,b,a) = Q(a) = \sum_{i=1}^{N} a(i) - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} a(i)a(j)d(i)d(j)x^T(i)x(j)$$

This is a problem only contains Lagrange multiplier $a(i)$ as the unknow, which subject to:

$$\begin{cases} a(i) \geq 0 \\ \sum_{i=1}^{N} a(i)d(i) = 0 \end{cases}$$

By solving this quadratic programing problem, we can get the optimum Lagrange multiplier value $a(i)$ which will be used to compute the optimal weight vector w and b.

f) Kernel function

For some data that can't be separated linearly, we can map them to some high dimensional space which the data can be separated with a linear hyperplane. Kernel function can help us perform this high dimensional projection operation without knowing the exact projection function.

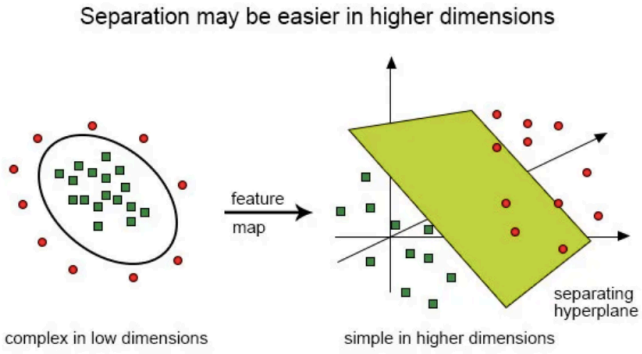

Figure. 3. Nonlinearly separable data in low dimensional space can be linearly separated in high dimensional space

Suppose we use $\varphi(x)$ to projection the original data into a high dimensional space (nonlinear transformation). Therefore, we can have the high dimensional linear hyperplane as:

$$\sum_{j=0}^{m} w_j\varphi_j(x) = w^T\varphi(x) = 0$$

Where m is the number of dimension after the projection, w_0 is the bias term and $\varphi(x) = [\varphi_0(x), \varphi_1(x), \dots, \varphi_m(x)]^T$ is the feature space after nonlinear mapping, $\varphi_0(x) = 1$.

From the optimality condition of linear SVM, we can know:

$$w = \sum_{i=1}^{N} a(i)d(i)\varphi[x(i)]$$

in which N is the number of training samples.

The optimization problem can also be expressed as:

$$J(w, b, a) = Q(a) = \sum_{i=1}^{N} a(i) - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} a(i)a(j)d(i)d(j)\varphi[x(i)]^T\varphi[x(j)]$$

Notice that $\varphi[x(i)]^T\varphi[x(j)]$ is a scalar value which can be represented by the inner product of vector $\varphi[x(i)]$ and $\varphi[x(j)]$.

Therefore, we define the kernel function as:

$$K[x(i), x(j)] = \langle \varphi(x(i)), \varphi[x(j)] \rangle = \varphi[x(i)]^T\varphi[x(j)]$$

Which is the inner product of the data point in the projection space and can be used to measure the similarity of these two data point $\varphi(x(i))$ and $\varphi[x(j)]$.

Gaussian kernel is a commonly used kernel function:

$$K[x(i), x(j)] = \exp\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right) = \exp\left(-\gamma\|x_1 - x_2\|^2\right)$$

We can see that kernel function can help us compute the inner product of the projection data in the low dimensional space without knowing the exact projection function.

Thus, we can used kernel function to express our optimization problem and classifier (discriminate function):

$$J(w, b, a) = Q(a) = \sum_{i=1}^{N} a(i) - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} a(i)a(j)d(i)d(j)K[x(i), x(j)]$$

x是任意input x(i)是训练后得到的support vector

$$g(x) = \sum_{i=1}^{N} a(i)d(i)K[x(i), x]$$

Where x is the input data in low dimensional space, $x(i)$ is the ith training data in low dimensional space.

Thus, after we solve the optimization problem and get the optimal Lagrange multiplier $a(i)$, we can use kernel function to compute the high dimensional decision hyperplane and construct our classifier which out knowing the exact projection function.

2.2 Parameters to Tune

Usually there are three parameters for kernel SVM training, which are:

a) Penalty parameter for those nonseparable data inside the margin C

b) Kernel function

c) Constant in kernel function γ

Usually C is default set as 1. Polynomial kernel or Gaussian (RBF) kernel can be used as the kernel function. γ can be set as $\gamma = \frac{1}{number\ of\ features}$

3. Model Training

3.1 Dataset Information

There are 330 number of training samples in the original dataset, each training sample has 33 features. These training data belongs to two classes which are labeled as 1 and -1. To better evaluate the performance of each model, the original training set is splitted into 3 parts by 8:2. Therefore, the new training set contains 264 samples, test set has 66 samples.

3.2 Parameter Setting

a) RBF neural network

Parameter	Method	Value
Hidden neurons	By trial	4
Neuron centers	K-means	-
Sigma	$d_{max}\sqrt{\frac{m}{2}}$	5.8211

b) Kernel SVM

Parameter	Method	Value
Penalty C	By trial	1
Kernel function	By trial	Gaussian kernel
Constant γ	$\frac{1}{number\ of\ features}$	0.0303

4. Accuracy on Data Classification

4.1 Conclusion

In this assignment, a sigmoid function is used as a classifier for RBF neural network, the output values that are bigger or equal to 0.5 are labeled as 1, vice versa. We can see that Kernel SVM has a better performance on both training set and test set.

4.2 Result

Dataset	RBF Neural Network	Kernel SVM
Training set (n=264)	88.63%	95.45%
Test set (n=66)	96.97%	98.48%

5. Prediction on Test Dataset

5.1 Conclusion

The test dataset contains 21 samples in total, the prediction results of RBF neural network and kernel SVM are presented below. By following the sequence of the test data samples, we can see that the data label 1 and -1 seems to be alternately appear, there are only two groups of data which are colored in red that have been labeled differently.

5.2 Result

No. of Samples	RBF Classification	Kernel SVM Classification

1	1	1
2	-1	-1
3	1	1
4	-1	-1
5	1	1
6	-1	-1
7	-1	1
8	-1	-1
9	1	1
10	-1	-1
11	-1	1
12	-1	-1
13	1	1
14	1	1
15	1	1
16	-1	-1
17	1	1
18	-1	-1
19	1	1
20	-1	-1
21	1	1

6. Appendix

6.1 Python Code for RBF Neural Network

```
import scipy.io
from scipy.spatial.distance import pdist, squareform
import numpy as np
from sklearn.cluster import KMeans

# Load Dataset
data = scipy.io.loadmat('data_train.mat')['data_train']
```

```
label = scipy.io.loadmat('label_train.mat')['label_train']

# Split dataset by the proportion of  8：2
data_train = data[0:int(data.shape[0] * 0.8), :]
label_train = label[0:int(data.shape[0] * 0.8), :]
data_test = data[int(data.shape[0] * 0.8):, :]
label_test = label[int(data.shape[0] * 0.8):, :]

# Setting hidden layer parameters
n_neurons = 4
centers = KMeans(n_clusters=n_neurons).fit(data_train).cluster_centers_   # hidden layer neurons
centers found by K-means
sigma = n_neurons * np.nanmax(squareform(pdist(centers))) / np.sqrt(2*n_neurons)

class RBFNN(object):
    def __init__(self, data_train, label_train, neurons, center, sigma):
        self.data_train = data_train
        self.label_train = label_train
        self.n_neurons = neurons
        self.centers = center
        self.sigma = sigma

    def hidden_layer(self, input_data):
        hidden_output = np.zeros((input_data.shape[0], self.n_neurons))
        for i in range(input_data.shape[0]):
            d = np.sum(np.power(np.tile(input_data[i], (self.n_neurons, 1)) - self.centers, 2),
axis=1)
            o = np.exp(-1 * d / (2 * self.sigma ** 2))
            hidden_output[i] = o
        # Adding bias unit
        fai = np.column_stack((np.ones((input_data.shape[0], 1)), hidden_output))
        return fai

    def output_layer(self, fai):
        w = self.train()
        return np.dot(fai, w)

    def train(self):
        fai = self.hidden_layer(self.data_train)
        w = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(fai), fai)), np.transpose(fai)),
                   self.label_train)   # use normal equation to compute the weight
        return w

def sigmoid(x):
```

```
    return 1 / (1 + np.exp(-x))

def classifier(x):
    prediction = sigmoid(x)
    prediction[prediction >= 0.5] = 1
    prediction[prediction < 0.5] = -1
    return prediction

def accuracy(data, label):
    acc = sum(data == label) / label.shape[0] * 100
    return acc

model = RBFNN(data_train, label_train, n_neurons, centers, sigma)
w = model.train()
test_fai = model.hidden_layer(data_test)
test_output = model.output_layer(test_fai)
test_prediction = classifier(test_output)
test_accuracy = accuracy(test_prediction, label_test)

train_fai = model.hidden_layer(data_train)
train_output = model.output_layer(train_fai)
train_prediction = classifier(train_output)
train_accuracy = accuracy(train_prediction, label_train)

print('test accuracy:', test_accuracy[0], '%', '| train accuracy:', train_accuracy[0], '%')

test = scipy.io.loadmat('data_test.mat')['data_test']
fai = model.hidden_layer(test)
output = classifier(model.output_layer(fai))
```

6.2 Python Code of Kernel SVM

```
import scipy.io
from sklearn import svm

data = scipy.io.loadmat('data_train.mat')['data_train']
label = scipy.io.loadmat('label_train.mat')['label_train']

data_train = data[0:int(data.shape[0] * 0.8), :]
label_train = label[0:int(data.shape[0] * 0.8), :]

data_test = data[int(data.shape[0] * 0.8):, :]
label_test = label[int(data.shape[0] * 0.8):, :]
```

```
model = svm.SVC(kernel='rbf')
model.fit(data_train, label_train)

test_prediction = model.predict(data_test).reshape(data_test.shape[0], -1)
test_accuracy = (test_prediction == label_test).sum() / test_prediction.shape[0]

train_prediction = model.predict(data_train).reshape(data_train.shape[0], -1)
train_accuracy = (train_prediction == label_train).sum() / train_prediction.shape[0]

print('test accuracy:', test_accuracy, '%', '| train accuracy:', train_accuracy, '%')

test = scipy.io.loadmat('data_test.mat')['data_test']
prediction = model.predict(test)
```