ECE 662 Homework 2

1 Question 1

From the class, we learned that separation hyperplane can be drawn between two classes by maximizing below cost function (Fisher's Linear Discriminant function.)

$$J(w) = \frac{w^T S_B w}{w^T S_w w}$$

If we find w_0 which maximizes above cost function, w_0 happens to be $w_0 = S_w^{-1}(\mu_1 - \mu_2)$.

However, some students raised a possibility of using another equation for the separation plane which looks like below.

$$J(w) = w^T S_B w$$

We can think of this case as $S_w = I$, then w_0 which maximizes the cost function is $w_0 = \mu_1 - \mu_2$.

To see how these two different formula separate two classes, I did an experiment with randomly generated data whose class, mean, covariance are known. Firstly, I generated sample data from the known mean, covariance for two different classes. I used below mean and covariance for two classes.

$$\mu_1 = \begin{bmatrix} -2 \\ 2 \end{bmatrix}, \mu_2 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$$

$$\Sigma_1 = \Sigma_2 = \begin{bmatrix} 5.5225 & 1.4695 \\ 1.4695 & 1.4775 \end{bmatrix}$$

Then, I generated 1,000 samples for each class and made a scatter plot (Figure 1).

Using these samples from two different classes, I applied two different $J(w)$ to the randomly generated data to get a separation plane.

Firstly, figure 2 is showing a projection plane (green line in this case) over scatter plot when $J(w) = w^T S_B w$ is used as a cost function. Figure 3 shows the histogram of sample data which are projected onto the projection line. As you can in the figure 3, there is a lot of overlap between two classes, even though two classes are well separated in the scatter plot (Figure 1).

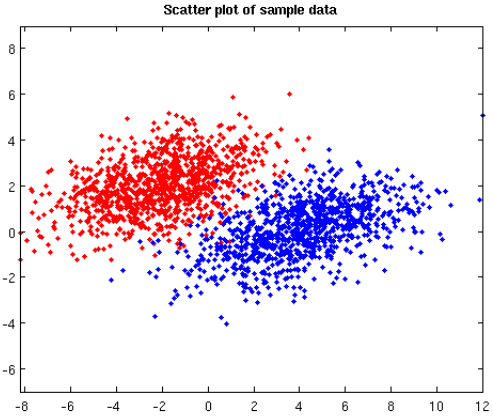

Figure 1: Scatter plot of sample data

Next, I used $J(w) = \frac{w^T S_B w}{w^T S_w w}$ as a cost function, and computed projection plane which is shown in figure 4. All the sample data are projected onto projection line and this is shown in figure 5. As you can see in the figure 5, there is a lot less overlap between two classes compared to the figure 3.

Based on this experiments, I was able to observe that cost function $J(w) = \frac{w^T S_B w}{w^T S_w w}$ is much more robust than $J(w) = w^T S_B w$.

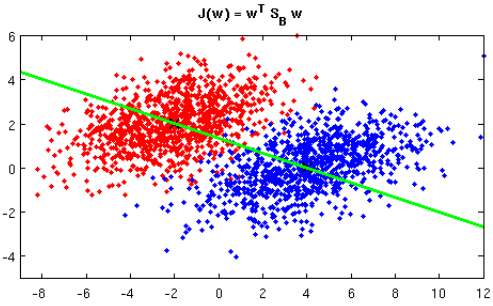

Figure 2: Projection plane with $J(w) = w^T S_B w$

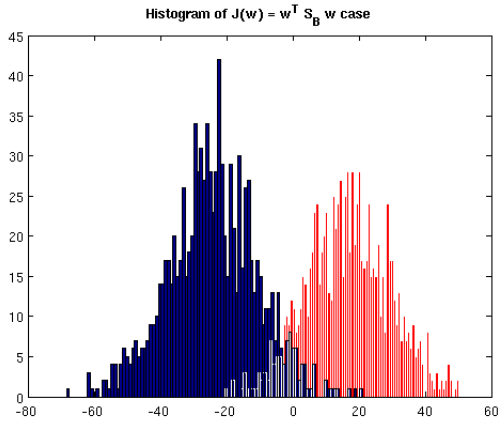

Figure 3: Histogram of projected data

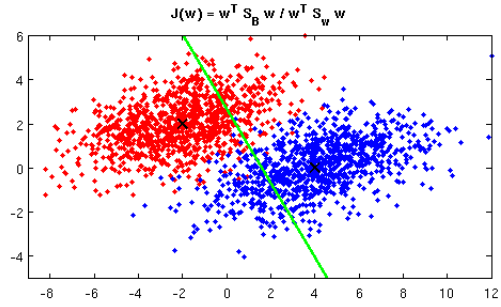

Figure 4: Projection plane with $J(w) = w^T S_B w$

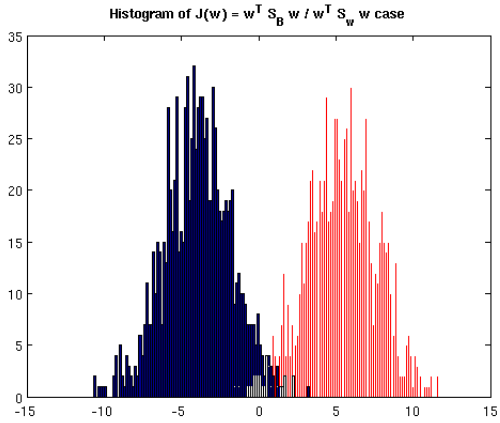

Figure 5: Histogram of projected data

2 Question 2

I downloaded the training data and testing data from the LIBSVM website (http://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/). I decided to use a1a data. It has two different classes in the data. The number of training data is 1,605 and the number of testing data is 30,956. The dimension of training and testing data is 123.

I used LIBSVM (A library for Support Vector Machines) as SVM (Support Vector Machine) classifier (http://www.csie.ntu.edu.tw/ cjlin/libsvm/) and FANN (Fast Artificial Neural Network) as ANN (Artificial Neural Network) classifier.

The data I downloaded from the web page are in the format for the LIBSVM package, so there was no problem for LIBSVM to read training and testing data. However, FANN uses different types of input data format, so I need to write my own python script which converts LIBSVM input data into FANN input data format. You can see this code in the appendix.

2.1 Support Vector Machine

In case of LIBSVM, I downloaded the source code from the web page and compiled on my laptop (Linux machine). After the compile, I was able to use two binary file, "svm-train" for the training classifier using training data, "svm-predict" for the classification of testing data using the classifier built by "svm-train".

I started the experiment with default options.

```
jinha@jinha-laptop:~/EE662/hw2/data$ svm-train a1a.train
*
optimization finished, #iter = 495
```

```
nu = 0.460268
obj = -673.031393, rho = -0.628569
nSV = 754, nBSV = 722
Total nSV = 754

jinha@jinha-laptop:~/EE662/hw2/data$ svm-predict a1a.test
a1a.train.model a1a.test.predict
Accuracy = 83.5864% (25875/30956) (classification)
```

With default option, 25875 out of 30956 testing data are classified correctly which is around 83 percent accuracy. RBF Kernel is being used for the default.

There is another tool "easy.py" which is python script. This script file read in the training data, and do the scaling and optimal parameter search for C and γ.

```
jinha@jinha-laptop:~/EE662/hw2/SupportVectorMachine/libsvm/libsvm-2.85/tools$
python easy.py /home/jinha/EE662/hw2/data/a1a.train
/home/jinha/EE662/hw2/data/a1a.test
Scaling training data...
Cross validation...
Warning: empty z range [81.8069:81.8069], adjusting to [80.9888:82.625]
Notice: Cannot contour non grid data. Please use "set dgrid3d".
Warning: empty z range [81.8069:81.8069], adjusting to [80.9888:82.625]
Notice: Cannot contour non grid data. Please use "set dgrid3d".
Best c=8192.0, g=3.0517578125e-05 CV rate=83.8006
Training...
Output model: a1a.train.model
Scaling testing data...
Testing...
Accuracy = 83.8287% (25950/30956) (classification)
```

The optimal parameters were found to be $C = 8192$ and $\gamma = 3.05 \times 10^{-5}$. Using these parameters, 25950 out of 30956 testing data are classified correctly. Compared to the previous experiment, I found out that the accuracy has increased a little bit, but not much.

2.2 Artificial Neural Network

In case of FANN, FANN libraries is provided for C language. Using these libraries, I wrote two C programs; 1) A program "train" trains classifier using training data. This program accepts training data file, trained model output file, number of input, number of output, number of layers, number of hidden neurons, desired error, maximum epoch, and number of epoch between report as input arguments. 2) A program "fann_predict" computes accuracy of the classifier using testing data. This program accepts testing data file and trained model output file as input arguments. This program reads in testing data and for every testing data, it computes output value. If the output value is greater

than 0, it is classified to be 1, otherwise, it is classified to be -1. Using these program, I did several experiments.

2.2.1 Experiment 1

I trained the classifier with number of layers = 3, number of hidden neurons = 10, desired error = 0.001, maximum epoch number = 1000, and report result every 100 epochs. Below is the result from this experiments.

```
jinha@jinha-laptop:~/EE662/hw2/fann$ ./train fann.a1a.train
fann.a1a.net 123 1 3 10 0.001 1000 100
Max epochs     1000. Desired error: 0.0010000000.
Epochs            1. Current error: 0.2550733089. Bit fail 1605.
Epochs          100. Current error: 0.0815767571. Bit fail 342.
Epochs          200. Current error: 0.0484785624. Bit fail 181.
Epochs          300. Current error: 0.0927885696. Bit fail 244.
Epochs          400. Current error: 0.0407177471. Bit fail 118.
Epochs          500. Current error: 0.0239799526. Bit fail 90.
Epochs          600. Current error: 0.0225784816. Bit fail 90.
Epochs          700. Current error: 0.0227248091. Bit fail 90.
Epochs          800. Current error: 0.0213591419. Bit fail 93.
Epochs          900. Current error: 0.0207666326. Bit fail 89.
Epochs         1000. Current error: 0.0201974604. Bit fail 96.

jinha@jinha-laptop:~/EE662/hw2/fann$ ./fann_predict fann.a1a.test
fann.a1a.net
Testing finished.
Number of test data = 30956
Number of correctly classified data = 24371
Accuracy = 78.727872 percent
```

As you can see above, 24371 out of 30956 testing data are classified correctly which is about 78.7 percent accuracy.

2.2.2 Experiment 2

I increased the number of hidden neurons to be 100 with all the other options are same as experiment 1.

```
jinha@jinha-laptop:~/EE662/hw2/fann$ ./train fann.a1a.train
fann.a1a.net 123 1 3 100 0.001 1000 100
Max epochs     1000. Desired error: 0.0010000000.
Epochs            1. Current error: 0.2534945309. Bit fail 1605.
Epochs          100. Current error: 0.0843408108. Bit fail 354.
Epochs          200. Current error: 0.0574599430. Bit fail 246.
Epochs          300. Current error: 0.0480635688. Bit fail 188.
Epochs          400. Current error: 0.0412287638. Bit fail 160.
```

```
Epochs          500. Current error: 0.0339034423. Bit fail 127.
Epochs          600. Current error: 0.0292778723. Bit fail 117.
Epochs          700. Current error: 0.0272603929. Bit fail 98.
Epochs          800. Current error: 0.0255394652. Bit fail 106.
Epochs          900. Current error: 0.0234575570. Bit fail 84.
Epochs         1000. Current error: 0.0229356531. Bit fail 87.

jinha@jinha-laptop:~/EE662/hw2/fann$ ./fann_predict fann.a1a.test
fann.a1a.net Testing finished.
Number of test data = 30956
Number of correctly classified data = 24588
Accuracy = 79.428867 percent
```

It took much longer to train classifier than the experiment 1, but I was not able to see much improvement on error. Using this classifier, the accuracy increases a little.

2.2.3 Experiment 3

I increased the maximum number of epoch to be 3000, and set the number of hidden neurons to be 20, and all the other options are same as experiment 1.

```
jinha@jinha-laptop:~/EE662/hw2/fann$ ./train fann.a1a.train
fann.a1a.net 123 1 3 20 0.001 3000 100
Max epochs     3000. Desired error: 0.0010000000.
Epochs            1. Current error: 0.2468506992. Bit fail 1605.
Epochs          100. Current error: 0.0662000477. Bit fail 256.
Epochs          200. Current error: 0.0283776037. Bit fail 111.
Epochs          300. Current error: 0.0190245137. Bit fail 90.
Epochs          400. Current error: 0.0165658966. Bit fail 81.
Epochs          500. Current error: 0.0154185938. Bit fail 80.
Epochs          600. Current error: 0.0149685284. Bit fail 80.
Epochs          700. Current error: 0.0148450714. Bit fail 80.
Epochs          800. Current error: 0.0147421118. Bit fail 82.
Epochs          900. Current error: 0.0147388419. Bit fail 82.
Epochs         1000. Current error: 0.0146861197. Bit fail 82.
Epochs         1100. Current error: 0.0145523343. Bit fail 84.
Epochs         1200. Current error: 0.0144924037. Bit fail 78.
Epochs         1300. Current error: 0.0144513045. Bit fail 80.
Epochs         1400. Current error: 0.0144520355. Bit fail 84.
Epochs         1500. Current error: 0.0145255607. Bit fail 80.
Epochs         1600. Current error: 0.0144326407. Bit fail 76.
Epochs         1700. Current error: 0.0144779198. Bit fail 80.
Epochs         1800. Current error: 0.0144425528. Bit fail 84.
Epochs         1900. Current error: 0.0155108096. Bit fail 86.
Epochs         2000. Current error: 0.0143694002. Bit fail 86.
Epochs         2100. Current error: 0.0143982098. Bit fail 84.
```

```
Epochs         2200. Current error: 0.0143587971. Bit fail 84.
Epochs         2300. Current error: 0.0143050291. Bit fail 72.
Epochs         2400. Current error: 0.0143002253. Bit fail 70.
Epochs         2500. Current error: 0.0142810391. Bit fail 80.
Epochs         2600. Current error: 0.0142837940. Bit fail 82.
Epochs         2700. Current error: 0.0143083390. Bit fail 86.
Epochs         2800. Current error: 0.0143467411. Bit fail 80.
Epochs         2900. Current error: 0.0142409699. Bit fail 80.
Epochs         3000. Current error: 0.0142384712. Bit fail 76.

jinha@jinha-laptop:~/EE662/hw2/fann$ ./fann_predict fann.a1a.test
fann.a1a.net Testing finished.
Number of test data = 30956
Number of correctly classified data = 24536
Accuracy = 79.260886 percent
```

As you can see from above result, error has decreased a little, but the accuracy of classifier didn't change much.

2.3 Comparison between SVM and ANN

I implemented Support Vector Machine and Artificial Neural Network algorithm and used same training and testing data to see the performance of two classifiers. Support Vector Machine showed the maximum of 83.83 percent accuracy, and Artificial Neural Network showed the maximum of 79.43 percent accuracy.

3 Question 3

For this question, the same data from question 2 are used.

3.1 Parzen window

I tried to implement the Parzen window classifier with rectangular shape window, but it didn't work well. This is because the training and testing data are in the binary format which means all the values of each dimension is either 0 or 1. If I apply parzen window with rectangular shape window, I end up classifying correctly only when there exists an exactly identical training data. Otherwise, it will fail to classify to either class.

3.2 K-nearest neighbor

K-Nearest Neighbor algorithm is implemented in MATLAB environment. 30956 testing data are classified using KNN classifier with various K value. The results are summarized in the below table.

K	Number of correctly classified data	Accuracy (Percent)
3	24972	80.67
5	25291	81.70
7	25392	82.03
9	25482	82.32
11	25519	82.44

As you can see from above table, the accuracy of KNN classifier increases as K increases. However, it took more time to finish classification as K increases.

3.3 The nearest neighbor

The nearest neighbor classifier is the special case of K-nearest neighbor classifier. We can just set $K = 1$ and use KNN algorithm. Below is the result of the nearest neighbor classifier.

K	Number of correctly classified data	Accuracy (Percent)
1	24280	78.43

As you can see from above table, the accuracy of the nearest neighbor classifier is lower than KNN classifier. However, the time it takes to finish classification is much less than KNN classifier.

3.4 Comaprison among three approaches

K-nearest neighbor classifier takes more time as K value increases, and the accuracy of K-nearest neighbor classifier increases as K value increases. The nearest neighbor classifier is the special case of K-nearest neighbor classifier, that is $K = 1$. The nearest neighbor classifier showed lower accuracy compared to the K-nearest neighbor classifier, but it took less time to classify all the testing data.

4 Appendix

4.1 Converting input data format from LIBSVM to FANN

```
#!/usr/bin/python

# Syntax: convert.py inputFile nData nFeature nOutput
import sys
import string
import Numeric

# Open input file
inFile = open(sys.argv[1], 'r')

# Print out the first line
print sys.argv[2], sys.argv[3], sys.argv[4]
```

```
for line in inFile:
    # Initialize feature vector to all zero
    feature = Numeric.zeros(int(sys.argv[3]))
    sline = string.split(line)

    # Iterate through and update the value for the feature
    for num in sline[2:]:
        snum = string.split(num,":")
        feature[int(snum[0])-1] = float(snum[1])

    # Print out the feature vector
    for a in feature:
        print a,

    # Print out the output
    print
    print sline[0]

# Close file
inFile.close()
```

4.2 ANN train source code

```
#include <stdio.h>
#include <stdlib.h>
#include "fann.h"

int main(int argc, char *argv[])
{
  /* Syntax: train trainFile outFile numInput numOutput numLayers
         numNeuronsHidden desiredError maxEpoch epochBetweenReport */
  unsigned int num_input = atoi(argv[3]);
  unsigned int num_output = atoi(argv[4]);
  unsigned int num_layers = atoi(argv[5]);
  unsigned int num_neurons_hidden = atoi(argv[6]);
  float desired_error = atof(argv[7]);
  unsigned int max_epochs = atoi(argv[8]);
  unsigned int epochs_between_reports = atoi(argv[9]);

  struct fann *ann =
    fann_create_standard(num_layers, num_input, num_neurons_hidden, num_output);

  fann_set_activation_function_hidden(ann, FANN_SIGMOID_SYMMETRIC);
  fann_set_activation_function_output(ann, FANN_SIGMOID_SYMMETRIC);
```

```
  fann_train_on_file(ann, argv[1], max_epochs, epochs_between_reports, desired_error);

  fann_save(ann, argv[2]);
  fann_destroy(ann);

  return 0;
}
```

4.3 ANN prediction source code

```
/*
** fann_predict.c
**
** Made by (Jinha Jung)
** Login   <jinha@jinha-laptop.ecn.purdue.edu>
**
** Started on  Thu Mar 27 17:30:56 2008 Jinha Jung
** Last update Sun May 12 01:17:25 2002 Speed Blue
*/

#include <stdio.h>
#include <stdlib.h>
#include "floatfann.h"

/* Syntax
   fann_predict testFile netFile */
int main(int argc, char *argv[])
{
  int lengthTestData;
  int numCorrect = 0;
  int class;
  float temp;
  int i;
  fann_type *calc_out;

  /* Create FANN structure */
  struct fann *ann = fann_create_from_file(argv[2]);
  struct fann_train_data *testData = fann_read_train_from_file(argv[1]);

  lengthTestData = fann_length_train_data(testData);

  //printf("Length of Test data = %d\n", lengthTestData);

  for (i = 0; i < lengthTestData; i++) {
        /* Initialize the result class */
```

```
        class = 0;
        /* Go through fann */
        calc_out = fann_run(ann, testData->input[i]);
        /* If result is positive class = 1, otherwise class=-1 */
        if (calc_out[0] > 0) {
          class = 1;
        } else {
          class = -1;
        }
        //printf("%d data is classified to %d\n", i, class);

        /* Check whether it is correctly classified */
        if (class == testData->output[i][0]) {
          numCorrect = numCorrect + 1;
          //printf ("Number of correct = %d\n", numCorrect);
        }
  }

  printf("Testing finished.\n");
  printf("Number of test data = %d\n", lengthTestData);
  printf("Number of correctly classified data = %d\n", numCorrect);
  printf("Accuracy = %f percent\n",
                (float)numCorrect/(float)lengthTestData*100);

  fann_destroy_train(testData);
  fann_destroy(ann);

  return 0;

}
```

4.4 KNN source code

```
%*****************************************************************
%    ECE 662 - Pattern Recognition
%    Homework #2, Prob #3
%    Desc : Implementation of KNN
%    Name : KyoHyouk Kim & Jinha Jung
%*****************************************************************
clear all; clc; format long g;
numTrain = 1605;
numTest = 30956;
nDim = 123;
K = 5;
numCorrect = 0;
```

```
% load training data & test data
% load trainData.mat
fid1 = fopen('/data/Class/2008/Spring/ECE_662_Pattern_Recognition/hw2/data/a1a.train');
fid2 = fopen('/data/Class/2008/Spring/ECE_662_Pattern_Recognition/hw2/data/a1a.test');

% initialization of matrix for input training & test data
trainData = zeros(numTrain, nDim);


% initialization of matrix for class
trainLabel = zeros(numTrain,1);

% make training sample matrix from the given LIBSVM sample data set
for i=1:numTrain
    line = fgets(fid1);
    % set up class
    class = str2double(line(1:2));
    trainLabel(i,1) = class;
    idx = findstr(line,':');
    for j=1:size(idx,2)
        if(line(idx(j)-1) ~= ' ' && line(idx(j)-2) == ' ')
            tmp = str2double(line(idx(j)-1:idx(j)-1));
            trainData(i,tmp) = 1;
        elseif(line(idx(j)-2) ~= ' ' && line(idx(j)-3) == ' ')
            tmp = str2double(line(idx(j)-2:idx(j)-1));
            trainData(i,tmp) = 1;
        elseif(line(idx(j)-3) ~= ' ' && line(idx(j)-4) == ' ')
            tmp = str2double(line(idx(j)-3:idx(j)-1));
            trainData(i,tmp) = 1;
        end
    end
end

% make test sample matrix from the given LIBSVM sample data set
for i=1:numTest
    % Initialize the testData before the process
    testData = zeros(1, nDim);
    line = fgets(fid2);
    % Read the label of test data
    testLabel = str2double(line(1:2));
    idx = findstr(line,':');
    % Read the feature value from test data
    for j=1:size(idx,2)
        if(line(idx(j)-1) ~= ' ' && line(idx(j)-2) == ' ')
            tmp = str2double(line(idx(j)-1:idx(j)-1));
            testData(1,tmp) = 1;
```

```
        elseif(line(idx(j)-2) ~= ' ' && line(idx(j)-3) == ' ')
            tmp = str2double(line(idx(j)-2:idx(j)-1));
            testData(1,tmp) = 1;
        elseif(line(idx(j)-3) ~= ' ' && line(idx(j)-4) == ' ')
            tmp = str2double(line(idx(j)-3:idx(j)-1));
            testData(1,tmp) = 1;
        end
    end

    % KNN
    testResult = wkKnn(trainData', trainLabel', testData', K);

    % Check if it is correctly classified.
    if (testLabel == testResult)
        numCorrect = numCorrect + 1;
    end
end

fprintf('\nNumber of testing data = %d\n', numTest);
fprintf('Number of correctly classified testing data = %d\n', numCorrect);
fprintf('Accuracy = %f\n', numCorrect/numTest*100);

fclose(fid1);
fclose(fid2);
```