Machine learning assignment report (521289S)

Name: Chen Haoyu

Student number: 2474045

Nickname: AlphaOulu

In this assignment design, I used outliers, PCA, Cross Validation and KNN meothods in my design.

Solution's classification accuracy

1) from my own testing 95.48%

2) from the testing server 95.9618%

In trainClassifier function

1. Preprocessing
   The removing mean normalized method was used to preprocess the data.
2. Outliers
   I assumed that the abnormal sample ratio is n, and delete those which have the biggest feature values according to the order.
   Tempts have been taken as below and n=0.04 performances best.

n	accuracy
0.06	0.9117
0.05	0.9121
0.04	0.9141
0.03	0.9099
0.02	0.8945

3. PCA.
   Function [U , S , ~] = svd (sigma), was used to get data PCA transformed, the U, S were calculated and past to

evaluateclassifier function by parameters.

4. SFFS and cross validation
   First, KNN parameters was settled. Then, I used SFFS to select the features, best_worst function was used to judge the feature selection, CV_score function was used to do cross validation and avoid over fitting.
5. Passing parameters
   Parameters calculated and to be past are listed here:
   'trasample', processed training samples,
   ' traclass', processed training classes,
   'best_fset', best_fset, selectes features,
   'tramean', Mean, mean of the training samples (for PCA),
   'KNN_params', KNN_params, KNN parameters,
   'svd_U', U, PCA parameters U,
   'svd_S', S, PCA parameters S.

In evaluateClassifier function

1. Preprocessing
   The removing mean normalized method was used to preprocess the samples.
2. PCA
   Get samples PCA transformed.
3. Classification
   Use KNN to classify the samples.

k	accuracy
51	0.9141
25	0.927
11	0.9467
9	0.9499
7	0.9548
5	0.9537

```
%% TITLE ***************************************************************
% *                                                                     *
% *                   521289S Machine Learning                          *
% *                  Programming Assignment 2016                        *
% *                                                                     *
% *   Author 1: << Chen Haoyu and 2474045 here >>                       *
% *                                                                     *
% *   NOTE: The file name for this file MUST BE 'classify.m'!           *
% *        Everything should be included in this single file.           *
% *                                                                     *
%
***********************************************************************
*

function classify()

end

function nick = getNickName()
nick = 'AlphaOulu';   % CHANGE THIS!
end

function parameters = trainClassifier(samples, classes )

%preprocessing
n_train = size (samples,1);
Mean = mean (samples);
samples = samples - repmat (Mean, n_train, 1);

%outliers

outratio=0.04;
outindex = OutlierRemover(samples, outratio);

samples(outindex,:) = [];
classes(outindex,:) = [];

%PCA
covsample=cov(samples);
[U , S , ~] = svd (covsample);
samples = samples * U *diag(1./sqrt(diag(S)));

% SFFS
% Create a validation set which will be a half of the train set

% Here we will perform feature selection and use 5-fold Cross-Validation
in
% order to estimate significance

KNN_params =  struct('K', 7, 'p', 2);
[res_vector, best_fset] = SFFS(@KNN_predict, KNN_params, 5, samples,
classes);
%
% Displaying the indices of selected features
selected = find(best_fset);
fmt = strcat(['Features selected: ' repmat('%d ',1, length(selected))
```

```
'\n']);
fprintf(fmt, selected);
% pass parameters
parameters =
struct('trasample',samples,'traclass',classes,'best_fset',best_fset,'tr
amean',Mean,'KNN_params',KNN_params,'svd_U',U,'svd_S',S);

end


function results = evaluateClassifier( samples, parameters )

%preprocessing
n = size (samples,1);
Mean = parameters.tramean;
samples = samples - repmat (Mean, n, 1);

U = parameters.svd_U;
S = parameters.svd_S;
%PCA
samples = samples * U *diag(1./sqrt(diag(S)));

%classification
trasample = parameters.trasample;
traclass = parameters.traclass;
best_fset = parameters.best_fset;
KNN_params = parameters.KNN_params;
results = KNN_predict(trasample(:, logical(best_fset)),traclass, samples(:,
logical(best_fset)), KNN_params);
end


function [results] = KNN_predict(X, y, X_test, params)
% L^p-distance-based KNN classifier
%
% Input:
% -------
% X: training data
% y: training labels
% X_test: testing data
%
% params: structure containing K and p parameters of KNN with L^p metric
%
% Output:
% -------
% results: vector of predictions
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
K = params.K;
p = params.p;

train_size = size(X, 1);
test_size = size(X_test, 1);

results = zeros(test_size, 1);
for j=1:test_size
```

```
    this = X_test(j, :);
    dists = sum(abs(X - repmat(this, train_size,1)).^p,2).^(1/p);
    [d I] = sort(dists,'ascend');
    y_train_srt = y(I);
    results(j) = mode(y_train_srt(1:K));
end
end

function [score] = CV_score(classifier, params, n_folds, X, y)
score = 0;
N = size(X, 1);
step = int64(N/n_folds);
for n=1:n_folds
    train_ind = [1:(step*(n-1)) (n)*step:N];
    test_ind = 1:N;
    test_ind(train_ind)=[];

    X_train = X(train_ind, :);
    y_train = y(train_ind, :);

    X_test = X(test_ind, :);
    y_test = y(test_ind, :);

    y_pred = classifier(X_train, y_train, X_test, params);
    score = score + sum(y_pred==y_test)/length(y_test);
end
score = score/n_folds;
end

function outindex = OutlierRemover(samples, outratio)


absample = abs(samples);

[numsam, dimsam] = size(samples);

numout = numsam * outratio;

N = zeros(numsam, dimsam);

for i = 1:dimsam
    [~,N(:,i)] = sort(absample(:,i),'descend');
end

tmp = 1;
outindex = unique (N(1:tmp,:));

while length(outindex) <= numout
    tmp = tmp + 1;
    outindex = unique (N(1:tmp,:));
end

end

function [res_vector, best_fset] = SFFS(classifier, params, n_folds, X, y)
```

```
    % Sequential Forward Floating Search Feature selection

    % Vector of used features
    fvector = zeros(1,size(X, 2));
    % How many features we are going to take (can be changed to any number
    % depending on the problem
    max_n_features = size(X, 2);
    % Initial dimension is one
    n_features = 1;

    best_result = 0;
    res_vector = zeros(1, max_n_features);
    backwards = 0;

    while(n_features <= max_n_features)
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
        % Step 1: Inclusion
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
        % Search for the best feature vector forwards
        [best_result_add, best_feature_add] = best_worst(classifier,
params,...
            X, y, n_folds, fvector, backwards);
        % Update the set of used features
        fvector(best_feature_add) = 1;

        % Save best result
        if(best_result < best_result_add)
            best_result = best_result_add;
            best_fset = fvector;
        end
        % Sometimes we might come back to the same number of selected
        % features. So, if the accuracy is better in the case when we came
        % back, then we will update the vector of results.

        if(best_result_add > res_vector(n_features))
            res_vector(n_features) = best_result_add;
        end

        % Print current result
        fprintf('Accuracy: %.4f, N_features: %d\n', res_vector(n_features),
int64(n_features))

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
        % Step 2: Exclusion
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%

        backwards = 1;
        while backwards
            % If the number of selected features is greater than 2, we try
            % to remove one the features. In the case when the number of
            % selected features is less than 2, we will go back to the
            % inclusion step
            if (n_features > 2)
```

```
            % Search the worst feature
            [best_result_rem, best_feature_rem] = best_worst(classifier,
params,...
                X, y, n_folds, fvector, backwards);

            % If better than before, step backwads and update results,
            % otherwise we will go to the inclusion step
            if(best_result_rem > res_vector(n_features-1) )
                fvector(best_feature_rem) = 0;
                n_features = n_features - 1;
                if(best_result < best_result_rem)
                    best_result = best_result_rem;
                    best_fset = fvector;
                end
                res_vector(n_features) = best_result_rem;
                fprintf('Accuracy: %.4f, N_features: %d\n',
res_vector(n_features), int64(n_features))
            else
                backwards = 0;
            end
        else
            backwards = 0;
        end
    end
    n_features = n_features +1;
  end
end

function [best_score, res_feature, best_feature_set] =
best_worst(classifier, params, X, y, n_folds, features, worst)
    % Function to find best or worst feature
    best_score = 0;
    res_feature = 0;
    best_feature_set = [];
    if worst == 0 % searching for the best feature, trying to add them one
by one
        for take=1:length(features)
            if features(take) == 0 % If the feature was not taken, try to add
it
                features(take) = 1;
                acc = CV_score(classifier, params, n_folds, X(:,
logical(features)), y);
                if acc > best_score
                    best_score = acc;
                    res_feature = take;
                end
                features(take) = 0;
            end
        end
    else % searching for the worst feature
        for take=1:length(features)
            if features(take) == 1 % If the feature was taken, try to remove
it
                features(take) = 0;
                acc = CV_score(classifier, params, n_folds, X(:,
logical(features)), y);
                if acc > best_score
                    best_score = acc;
```

```
                res_feature = take;
            end
            features(take) = 1;
        end
    end
end
```