F14-16601: Machine Learning
Homework 8 Report

Dawei Wang
daweiwan@andrew.cmu.edu

November 24, 2014

Code of Conduct Declaration

- I did not receive any help whatsoever from anyone in solving this assignment.
- I did not give any help whatsoever to anyone in solving this assignment.

Experiment 1: Figure 1 are the top five eigenfaces. Figure 2 are the reconstructed images of Kawamura, with the number of reduced dimensions and the squared reconstruction errors as captions.

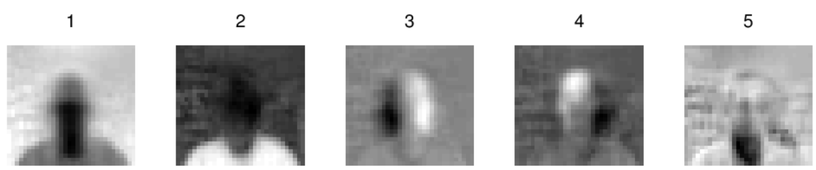

Figure 1: Top 5 Eigenfaces

Figure 2: Reconstructed Kawamura

Experiment 2: Here is the result:

Table 1: `svmtrain` with PCA

Trial	`all_test1.list`	`all_test2.list`	Total Time
Original	89.9281% (125/139)	83.1731% (173/208)	0.320113s
50D-PCA	89.2086% (124/139)	84.1346% (175/208)	0.0132291s
150D-PCA	87.7698% (122/139)	85.0962% (177/208)	0.0513999s
200D-PCA	88.4892% (123/139)	84.1346% (175/208)	0.0662739s

So performing PCA before SVM doesn't necessarily result in higher accuracies. This makes partial sense intuitively since SVM performs well when handling high-dimensional data, and we shouldn't have to reduce the dimensionality of the data beforehand. Nevertheless, with PCA the execution did take less time - with some slightly compromised accuracies. This could be one advantage.

Code: `ex1.m` is for the first experiment. `ex2.m` the second.

ex1.m

```
% read the images and save them in a matrix.

fid = fopen('all.list'); faces = [];
while !feof(fid)
  f = imread(fgetl(fid));
  faces = [faces; double(f(:)')];
end

% note that we have far more features than samples, it is more computationally feasible
% to only find out the singular value decomposition for the covariance matrix instead of the original.
[u, s, d] = svd(faces);

% extract the first five eigenfaces, scale them into the range of [0, 255]
% and plot them in a nice row with their indices as titles.
for index = 1: 5
  ef = d(:, index); scaled = uint8((ef - min(ef)) ./ (max(ef) - min(ef)) * 256);
  subplot (1, 5, index, 'align'); imshow (reshape(scaled, size(f))); title(num2str(index));
end

% project kawamura into the new space - though we don't need all the scores but for now -
% let's compute everything and extract some of them in the loop. then scale and plot it,
% don't forget to add the mean and compute the reconstruction error and display it.

reduced_kawamura = (faces(1, :) - mean(faces(1, :))) * d;
% reduced_kawamura = center(faces(1, :)) * d;
index = 0;
for n = 50: 50: 600
  kawamura = reduced_kawamura(:, 1: n) * d(:, 1: n)' + mean(faces(1, :));
  scaled = uint8((kawamura - min(kawamura)) ./ (max(kawamura) - min(kawamura)) * 256);
  subplot (3, 4, ++index, 'align'); imshow (reshape(scaled, size(f)));
  title (sprintf('%d (%.3f)', n, sum(sumsq(kawamura - faces(1, :)))));
end
```

ex2.m

```
% extract the images from the file specified by file,
% with proper labels set by parsing the filename.
function [labels, instances] = extract(file)
  fid = fopen(file); instances = labels = [];
  while !feof(fid)
    path = fgetl(fid);
    instances = [instances; double(imread(path)(:)')];
    labels = [labels; any(strfind(path, 'sunglasses'))];
  end
end

% scale the features as required by the libsvm svmtrain method.
function [scaled] = scale(instances, lb, ub)
  scaled = (instances - repmat(lb, size(instances, 1), 1)) ./ ...
    repmat(ub - lb, size(instances, 1), 1);
end

```

```
% this is the beginning of this script.
% include the libsvm interface for matlab/octave.
addpath('libsvm-3.20/matlab/');

% train the support vector machine.
[train_labels, train_instances] = extract('all_train.list');
ub = max(train_instances); lb = min(train_instances);
scaled_train_instances = scale(train_instances, lb, ub);
[test1_labels, test1_instances] = extract('all_test1.list');
scaled_test1_instances = scale(test1_instances, lb, ub);
[test2_labels, test2_instances] = extract('all_test2.list');
scaled_test2_instances = scale(test2_instances, lb, ub);

% generate some baseline classification accuracies...
fprintf('Baseline:\n'); tic;
model = svmtrain(train_labels, scaled_train_instances, '-q␣-c␣100␣-g␣0.01');
predicted1_label = svmpredict(test1_labels, scaled_test1_instances, model);
predicted2_label = svmpredict(test2_labels, scaled_test2_instances, model); toc;

% now apply principal component analysis to see if the performance gets better...
% apparently we're supposed to use the same basis vectors... or the scores don't make sense.
[all_labels, all_instances] = extract('all.list');
[~, ~, d] = svd(all_instances);
reduced_train_instances = center(scaled_train_instances) * d;
ub = max(reduced_train_instances); lb = min(reduced_train_instances);
reduced_train_instances = scale(reduced_train_instances, lb, ub);
reduced_test1_instances = scale(center(scaled_test1_instances) * d, lb, ub);
reduced_test2_instances = scale(center(scaled_test2_instances) * d, lb, ub);

fprintf('Reduced␣to␣50␣features...\n'); tic;
model_pca50 = svmtrain(train_labels, reduced_train_instances(:, 1: 50), '-q␣-c␣500␣-g␣0.07');
svmpredict(test1_labels, reduced_test1_instances(:, 1: 50), model_pca50);
svmpredict(test2_labels, reduced_test2_instances(:, 1: 50), model_pca50); toc;

fprintf('Reduced␣to␣150␣features...\n'); tic;
model_pca150 = svmtrain(train_labels, reduced_train_instances(:, 1: 150), '-q␣-c␣100␣-g␣0.16');
svmpredict(test1_labels, reduced_test1_instances(:, 1: 150), model_pca150);
svmpredict(test2_labels, reduced_test2_instances(:, 1: 150), model_pca150); toc;

fprintf('Reduced␣to␣200␣features...\n'); tic;
model_pca200 = svmtrain(train_labels, reduced_train_instances(:, 1: 200), '-q␣-c␣100␣-g␣0.07');
svmpredict(test1_labels, reduced_test1_instances(:, 1: 200), model_pca200);
svmpredict(test2_labels, reduced_test2_instances(:, 1: 200), model_pca200); toc;
```