Machine Learning
Assignment 3
Due to **08/2/2016.**

General Instructions:

The solution should be formatted as a report (printed) and running code should be included in a digital form. The solution can be done in pairs independently to other groups. Identical (or very similar solutions) are not allowed!

Problem 1

In this problem we will perform face recognition. The following method makes use of PCA, Multiple Discriminative Analysis, and Nearest Neighbor classifier.
Faces recognition is a high-dimensional problem, thus before applying a classifier, it's useful to reduce the dimension of the samples. Multiple Discriminative Analysis (MDA) finds a low-dimensional discriminative sub-space. Note, that the highest possible dimension of the sub-space is *c*-1 where *c* is the number of classes (people in our case).
MDA finds W which minimizes $\frac{|W^t S_b W|}{|W^t S_w W|}$, where S_b and S_w are the between and within scatter matrices (as defined in the lecture notes). The problem is that the dimension of the samples *d* (in our case $d=99^2$) is usually much larger than the number of training samples *N* (in our case *N*=100), thus S_w is singular. To solve this problem, we first use PCA to reduce the dimension of samples to *N*-*c*, and then apply MDA to reduce it to *c*-1.

PCA:
Let us consider a set of *N* samples $\{x_1,...,x_N\}$ in n-dimensional feature space. Let the scatter matrix S be defined as

$$S = \sum_{i=1}^{N} (x_i - \mu)(x_i - \mu)^T$$

where μ is the mean image of all samples. In PCA, the optimal projection *W* is chosen to maximize the determinant of the total scatter matrix of the projected samples, i.e.

$$W_{PCA} = \arg\max_W \left|W^T S\, W\right| = \left[w_1 \quad ... \quad w_{N-c}\right]$$

Where $\{w_i\}$ are the d-dimensional eigenvectors of S corresponding to the top eigenvalues. Note, that computing W_{PCA} requires solving eigen problem of dimension $d=99^2$. We will implement a more efficient solution:

Let *A* denote a matrix of training samples organized as columns. Instead of computing the eigen problem for $S = AA^t$, which has dimension *d* x *d*, we solve the eigen value

decomposition of $S' = A^t A$ of a much smaller dimension N x N. Let Z be the diagonal matrix of N-c highest eigenvalues of S' and V be the matrix of the N-c corresponding eigenvectors (as columns). We compute W_{PCA} as follows:

$$W_{PCA} = AVZ$$

MDA:

$$W_{MDA} = \arg\max_{W} \frac{|W^t W_{PCA}^t S_b W_{PCA} W|}{|W^t W_{PCA}^t S_w W_{PCA} W|}$$

The final projection matrix is obtained by

$$W_{opt} = W_{MDA} W_{PCA}^T .$$

Data:

download the Matlab file FaceData.mat from moodle, which contains the following data:
trainData is a 100 x 9801 matrix of 100 training examples (each example is a row). It contains two examples per subject; the number of people is 50 (2 images * 50 people =100 samples). The samples of the same person appear one after another.
testData is a 50 x 9801 matrix of 50 test examples (each example is a row). It contains one sample per subject.
Note: if you want to view the face images using Matlab image toolbox commands, you should resize the vectors to matrices (for example, to view the first training sample you do
q=reshape(trainData(1, :), 99,99);
imagesc(q);

Instructions:

Implement face recognition module using the above training and test data. Report the classification rate as the number of correct classifications divided by the number of test samples (=50).

Normalization Step

- Normalize each sample to zero mean and unit variance.
- Subtract the mean of training samples from each sample

Construction of low-dimensional basis:

- Normalization step for training samples
- Compute W_{PCA} as described above.

- Compute W_{opt} as described above.

Training Step

- Normalization Step for training samples
- Compute model for each person as the average of its two training samples.
- Project the model to W_{opt}. Output the resulting matrix of models.

Classification Step

- Normalization Step for test samples
- Project the test samples to W_{opt}
- Apply nearest neighbor to classify all test samples into *c*=50 classes, using the models from the training step.
- Output the classification rate.

Provided code:

Download the provided code Face.zip from moodle.

Directory Face contains three files:

`runFaceRec.m` which runs the system. No implementation is needed.

`FisherFaces.m` is a function that should construct a low dimensional basis W_{opt}.

It inputs `Nxd` matrix of training data for N=100 images (2 per person); d is the dimension of the features, representing an image.

You should compute W_{opt} as described above (in the code I named it F) and the mean of the training set `mn`, which are saved at the end of the function.

`recogTest.m` is a function that should perform training and classification.

`trainData` is `Nxd` matrix of training data for N=100 images (2 per person); `d` is the dimension of the features, representing an image (note that the derivation assumes that the matrix is **dxN**)

`testData` is `nxd` matrix of test data for n=50 images (1 per person); d is the dimension of the features, representing an image.

`suc` is the number of correct classifications (when the index of the test image is equal to the predicted index)

Problem 2

Use linear regression to predict a city-cycle fuel consumption in miles per gallon from 3 multivalued discrete and 4 continues attributes.

Attribute Information:

1. mpg: continuous –the value to be predicated
2. cylinders: multi-valued discrete
3. displacement: continuous
4. horsepower: continuous
5. weight: continuous
6. acceleration: continuous
7. model year: multi-valued discrete
8. origin: multi-valued discrete

Data: You'll find the data in the file in regdata.mat in moodle. This data is part of UCI Machine Learning Repository and can be downloaded from http://archive.ics.uci.edu/ml/datasets/Auto+MPG . The original data contains the name of the car, which was removed from regdata**.**mat **.**

a) Predict mpg (the first column of the data) based on the other columns. Use a linear regression model to predict the mpg, using squared error as the criterion to minimize. In other words

$$y = f(x, \hat{w}) = \hat{w}_0 + \sum_{i=1}^{13} \hat{w}_i x_i ,$$

where

$$\hat{w} = \arg\min_w \sum_{t=1}^{n} (y_t - f(x_t, w))^2 ,$$

here y_t are the mpg values, x_t are attribute vectors, and n is the number of training examples.

Instructions:

a) Write a function that learns the prediction model and outputs the mean training error (i.e. average squared-error over all training samples) and mean test error.
b) Use part of the data set as a training set, and the rest as a test set. Turn in the mean squared training and test errors for each of the following training set sizes: 10, 50, 100, 200.

Hint: permute the order of the samples before dividing the data into training and test sets (the data is ordered in some way, thus using the first rows as training and the rest as test will not generalize well.)

c) Write your conclusions about the experiments from b).

Problem 3

In this problem we will classify digits from MNIST, constructing multi-class classifiers using SVM in one-against-all manner. We will test Linear SVM and kernel RBF SVM and use k-fold cross-validation to find the best parameters that these classifiers need.

Linear SVM classifier:

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

Kernel SVM classifier with RBF kernel:

$$f(\mathbf{x}) = \text{sgn}\Big(\sum_{i \in SV} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\Big),$$

where $K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{u} - \mathbf{v}\|^2}{\gamma}\right)$ and γ is the kernel parameter.

Instructions:

Download LIBSVM from http://www.csie.ntu.edu.tw/~cjlin/libsvm/
It includes Matlab interface. Use `svmtrain` and `svmpredict` library functions for training and testing SVM models.

Note that `svmtrain` requires the training data to include first the positive samples and then the negative sample. You should rearrange your training data (and labels) that way.

Multi-class classifier in one-against-all manner:

Training: Learn SVM classifier for each class by using the samples of this class as positive examples and all samples of other classes as negative examples. Let $f_i(x)$ be the model, learned for class i.

Classification: Given a test sample x, score it using models of all classes $s_i = f_i(x), \forall i$.

Classify *x* with the label of class j if $s_j = \max_i s_i$.

K-fold cross validation:

The main script provides you with several values for each parameter. You should use k-fold cross validation on the training data, to find the set of parameters that gives you the best cross-validation rate averaged over all classes. In our experiments we use k=5. You should run separate cross-validation for linear svm (to find best C) and for kernel svm (to find the best γ and C). You should get about 0.77 (77%) cross-validation rate for linear and about 0.86 (86%) for the kernel.

Download the data and a partial implementation from moodle in SVM.zip

Data: Partial MNIST data (images of handwritten digits 0-9 and their labels)

Provided code:

Directory SVM contains all the files you need (except for LIBSVM and the data).
`runSVM.m` is the main script. Look at it, but no extra code is required there.

`trainLinearSVM.m` should run the training and cross validation of linear SVM for all classes.
The inputs to this function are:
`X` –the training data of size Nxd, N is the number of training samples, d is the number of features;
`Y` – the vector of labels for the training data
`K` – the number of folds in cross-validation
`allC – is a 1x2` array of values for the C-parameter, from which you choose the best value using cross-validation.

The outputs are:
`Models` – 10x1 array of structures which are the outputs of the 10 SVM trainings (one per class) using all training examples with the C found by the cross-validation.
`cv_acc` - the average classification rate obtained by cross-validation, corresponding to the best parameter .

You should write your implementation of one-against-all training and k-fold cross-validation of linear SVM. Use `svmtrain` from LIBSVM for training the SVM models.

`trainRBFSVM.m` should run the one-against-all training and cross validation of RBF kernel SVM for all classes.

The inputs to this function are:
`X` –the training data of size `Nxd`, N is the number of training samples, `d` is number of features;

`Y` – vector of labels for the training data
`K` – number of folds in cross-validation
`allC` – is a 1x2 array of values for the C-parameter
`allG` – is a 1x3 array of values for the gamma parameter of the RBF kernel.

The outputs are:
`models` - 10x1 array of structures which are the outputs of the 10 SVM training (one per class) using all training examples with the C and gamma found by the cross-validation.
`cv_acc` - average classification rate obtained from cross validation corresponding to the best parameter .

You should write your implementation of one-against-all training and k-fold cross-validation of RBF kernel SVM. Note that you should find a set of C and gamma that provides the best cross-validation rate on average over all classes. Use svmtrain from LIBSVM for training the SVM.

`testSVM.m` is the function for one-against-all classification using SVM.

Inputs to this function are:
`models` – 10x1 array of structures which are the outputs of the 10 SVM trainings (one per class)

`X` - test data of size nxd, n is the number of test samples, d is number of features;
`Y` – vector of labels for the test data. Use it for computing the accuracy of classification.

The output is `acc` which is the average test classification rate, computed as the number of correct classification divided by the number of tested examples.

You should write your implementation of the one-against-all classification. Use `svmpredict` to obtain the scores of the binary classifications. This function works with all types of kernel (including linear), thus you don't need to distinguish between linear and kernel SVM.