Assignment 4

Machine Learning, Summer term 2014, Ulrike von Luxburg

To be discussed in exercise groups on May 12-14

Exercise 1 (Rewriting the Fisher criterion for LDA, 2 points) Show that the Fisher criterion

$$J(w) = \frac{\langle w, m_+ - m_- \rangle^2}{\sigma_{w,+}^2 + \sigma_{w,-}^2}$$

can be rewritten as

$$J(w) = \frac{\langle w, C_B w \rangle}{\langle w, C_W w \rangle}.$$

See the lecture notes for the meaning of m_+, m_-, $\sigma_{w,+}^2$, $\sigma_{w,-}^2$, C_B and C_W.

Read `prepare04.pdf` (available on the course webpage) for an example of how to perform linear discriminant analysis (LDA) in MATLAB.

Exercise 2 (Digit classification with LDA, 1+2 points) We want to use LDA for written digit classification on the USPS dataset from Assignment 1 (do not forget to convert the data type from `uint8` to `double`).

(a) Use `ClassificationDiscriminant.fit` to train a LDA classifier for digit 2 versus 9. Use `imagesc` (see Assignment 1) to plot the learned weights in a 16x16 image. Here you do not need to change the colormap.

(b) Use your learned weights(`.Linear`) and the constant weight(`.Const`) to predict labels of the test data for digits 2 and 9. Report the 0-1 loss of your prediction and compare it with the performance of the kNN classifier for $k = 5$.

Multiclass classification: So far, all classification algorithms we have seen were designed to deal with two class (binary) classification problems. In many real world applications (e.g. digit recognition), our data has several classes. A common approach to solve such problems is to build a multiclass classifier from several binary classifiers. Assume we have k classes $C_1, ..., C_k$. Usually we follow one of the two strategies:

- One vs. All: A single classifier is trained per class to distinguish that class from all other classes. Prediction is then performed by predicting using each binary classifier, and choosing the prediction with the highest confidence score (in LDA, $w_i x + b_i$ is the score for class i).
- One vs. One: For each pair of classes we construct a binary classifier ($c(c-1)/2$ classifiers in total). Usually, classification of an unknown pattern is done according to the maximum voting, where each of $c(c-1)/2$ classifiers votes for one class.

Exercise 3 (Multiclass classification with LDA, 2+3 points)

(a) Implement the One vs. All strategy for LDA classifier to classify digits 1, 2, 3 and 4 from USPS dataset (use the complete dataset — available on the course webpage). Report the test error.

(b) Implement the One vs. One strategy for LDA classifier to classify digits 1, 2, 3 and 4 from USPS dataset (use the complete dataset — available on the course webpage). Report the test error.

Exercise 4 (Complexity of multiclass classification, 2+1 points) You have a multiclass classification problem with n training points and k classes. Assume that each class has the same number of training points. A binary classifier `myClassifier` is given which requires $c(m_1^2 + m_2^2)$ operations to learn the classifier (where m_i is the number of training points in class i and c is a constant).

(a) Compare the training time of multiclass classification with *One vs. All* and *One vs. One* strategies using `myClassifier`. Which strategy is faster?

(b) Does the situation change if `myClassifier` only requires $c(m_1 + m_2)$ operations for training?

Cross validation (m-fold): In many machine learning algorithms we need to choose parameters of our learning algorithm. The regularization parameter λ in ridge regression or the number of neighbors k in kNN classification are examples of such parameters.
In *m-fold cross-validation* the original sample is randomly partitioned into m (roughly) equally sized subsamples. Of the m subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $m - 1$ subsamples are used as training data. The cross-validation process is then repeated m times (the folds), with each of the m subsamples used exactly once as the validation data. The m results from the folds then can be combined to produce a single estimation of the performance of the classifier under consideration.
Hence, we have the following scheme for choosing a parameter out of a finite set Λ of possible parameter values:

1. Split the data into m equally sized groups.
2. FOR $\lambda \in \Lambda$, $i = 1, \ldots, m$ DO
   (a) Select group i to be the validation set and all other $(m - 1)$ groups to be the training set.
   (b) Train the model with parameter λ on the training set, evaluate on the validation set and store the test error.
3. Select the parameter which performed best in the m folds (usually, one considers the average test error).

Exercise 5 (Parameter selection by the training error, 1 point) A naive approach for choosing the learning parameter would be to select the one which minimizes the training error. Explain why this is not a good idea in ridge regression and kNN classification.

Exercise 6 (Cross validation in ridge regression, 3 points) Use your ridge regression code (or the one available on the course webpage) and the training data in `dataRidge.mat` from Assignment 3. Choose the regularization parameter from $\lambda \in \{2^{-i} | i = -15, -14, ..., 7, 8\}$. Use 10-fold cross validation to select the regularization parameter with best performance (test error with L2 loss). To partition the data the MATLAB command `cvpartition` might be helpful.

MACHINE LEARNING- ASSIGNMENT 4

VICTOR BERNAL ARZOLA

Exercise 1

The Fisher criterion is

$$J(w) = \frac{\langle w, m_+ - m_- \rangle^2}{\sigma_{w,+}^2 + \sigma_{w,-}^2} \tag{0.1}$$

if we manipulate the numerator

$$= \frac{w^T (m_+ - m_-)(m_+ - m_-) w}{\sigma_{w,+}^2 + \sigma_{w,-}^2}$$

using the definition of between-class covariance

$$= \frac{w^T C_B w}{\sigma_{w,+}^2 + \sigma_{w,-}^2}$$

or

$$= \frac{\langle w, C_B w \rangle}{\sigma_{w,+}^2 + \sigma_{w,-}^2} \tag{0.2}$$

recalling the definition for the denominator of 0.1 we have

$$\sigma_{w,+}^2 = \frac{1}{n_+} \sum_{\{i:Y_i=+1\}} (\langle w, X_i \rangle - \langle w, m_+ \rangle)^2$$

or

$$\sigma_{w,+}^2 = \frac{1}{n_+} \sum_{\{i:Y_i=+1\}} \left(w^T (X_i - m_+)\right)^2$$

that can be written as

$$\sigma_{w,+}^2 = \frac{1}{n_+} \sum_{\{i:Y_i=+1\}} \left(w^T (X_i - m_+)(X_i - m_+)^T w\right) \tag{0.3}$$

using a similar procedure for the other class we get

$$\sigma_{w,+}^2 + \sigma_{w,-}^2 = w^T \left[\frac{1}{n_+} \sum_{\{i:Y_i=+1\}} \left((X_i - m_+)(X_i - m_+)^T\right) + \frac{1}{n_-} \sum_{\{i:Y_i=-1\}} \left((X_i - m_-)(X_i - m_-)^T\right) \right] w$$

recalling the definition of within-class covariance

$$\sigma_{w,+}^2 + \sigma_{w,-}^2 = \langle w^T, C_W w \rangle \tag{0.4}$$

Finally 0.1 0.30.4 lead to

$$J(w) = \frac{\langle w, C_B w \rangle}{\langle w^T, C_W w \rangle}$$

Exercise 4

Complexity of multi-class classification. You have a multi class classification problem with N training points and K classes. Assume that each class has the same number of training points.

Output	One vs one	One vs All
# Classifiers	$K(K-1)/2$	K
# Operations per classifier	$c\left(\left(\frac{N}{K}\right)^2+\left(\frac{N}{K}\right)^2\right)$	$c\left(\left(\frac{N}{K}\right)^2+\left(N-\frac{N}{K}\right)^2\right)$
Training Time	$K(K-1)c\left(\frac{N}{K}\right)^2$	$Kc\left(\left(\frac{N}{K}\right)^2+\left(N-\frac{N}{K}\right)^2\right)$

TABLE 1. Time Complexity Multi-class approaches (a)

For a classifier that takes $c\left(m_1^2+m_2^2\right)$ operations to learn One vs All takes more time.

Output	One vs one	One vs All
# Classifiers	$K(K-1)/2$	K
# Operations per classifier	$c\left(\left(\frac{N}{K}\right)+\left(\frac{N}{K}\right)\right)$	$c\left(\left(\frac{N}{K}\right)+\left(N-\frac{N}{K}\right)\right)$
Training Time	$K(K-1)c\left(\frac{N}{K}\right)$	KcN

TABLE 2. Time Complexity Multi-class approaches (b)

For a classifier that takes $c(m_1+m_2)$ operations to learn One vs All takes more time.

EXERCISE 5

A model has over fit the training data when its test error is larger than its training error (by at least some small amount). This means you model will fail to generalize. This is why training set error is a poor predictor of hypothesis accuracy for new data (generalization).

REMARKS

The polynomial regression model can be expressed in matrix form in terms of a design matrix X, a response vector Y, a parameter vector a, and a vector ε of random errors. Then the model can be written as a system of linear equations:

$$Y = Xa + \varepsilon$$

$$\begin{bmatrix} Y_1 \\ : \\ Y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 ... & x_1^p \\ 1 & x_2 ... & x_2^p \\ 1 & x_n ... & x_n^p \end{bmatrix} \begin{bmatrix} a_1 \\ : \\ a_n \end{bmatrix}$$

The i^{th} row of X and Y will contain the x and y value for the i^{th} data sample. The vector of estimated polynomial regression coefficients can be found with least squares. This is the unique least squares solution as long as X has linearly independent columns.

Table of Contents

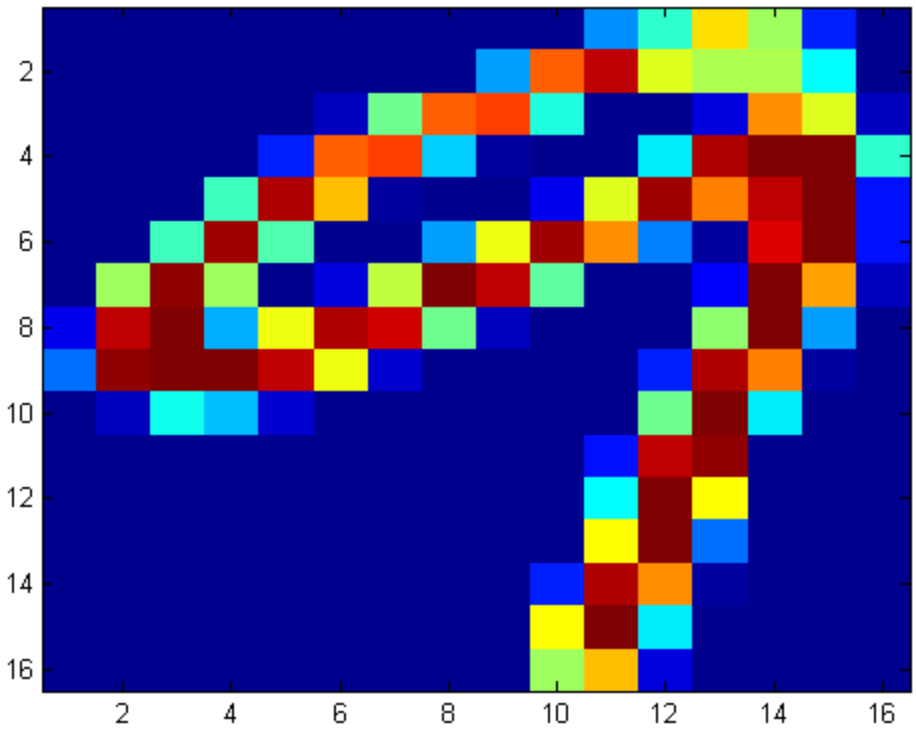
..... 1
Part (a) 1
Part (b) Label Prediction LDA for test data 3
Compare LDA with a KNN classifier with K=5 4
Cross validation (10-fold) for KNN 4
Exercise 6 Cross validation (10-fold) for KNN 5

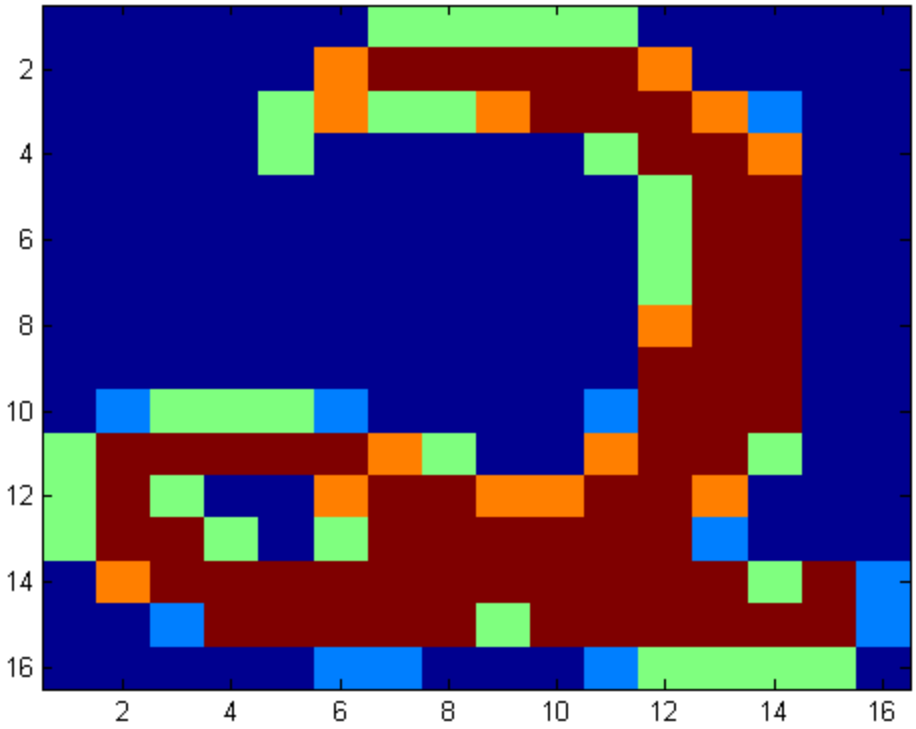

```
% Machine Learning Exercise 2

% The USPS dataset contains grayscale handwritten digit images scanned from
% envelopes by the U.S. Postal Service.
% The images are of size 16 x 16 (256 pixel) with pixel values
% in the range 0 to 255. We have 10 classes f1; 2; :::; 9; 0g.
% The training data has 500 images, stored in a 500 x 256 Matlab matrix
%(usps_train.mat). The training label is a 500 x 1 vector
% revealing the labels for the training data.
% There are 200 test images for evaluating your algorithm
% in the test data (usps_test.mat).
```

Part (a)

```
clear all; clc; close all;
load usps_train
load usps_test

% trList Find labels of data corresponding to 2 and 9
trList = find(train_label==2 | train_label==9);
% Train data on this indexs
x_train = double(train_data(trList,:));
y_train = double(train_label(trList));

%Create a linear classifier
cls = ClassificationDiscriminant.fit(x_train,y_train);
% Uses LDA (Hyperplane K+Lx=Y)

%b
K = cls.Coeffs(1,2).Const;
%w  256x1
L = cls.Coeffs(1,2).Linear;

%weights = reshape(L,16,16);
%imagesc((weights));

% A nine?
figure(1)
weights = reshape(x_train(100,:),16,16);
imagesc((weights));
```

```
% A two?
figure(2)
weights = reshape(x_train(2,:),16,16);
imagesc((weights));
```





Part (b) Label Prediction LDA for test data

```
%There are 200 test images for evaluating
%your algorithm in the test data (usps_test.mat).


testList = find(test_label==2 | test_label==9);
x_test = double(test_data(testList,:)); % 40 x 256
y_test = double(test_label(testList));

% [1 x_test][K;L]=Y
x_test_modified = [ones(size(x_test,1),1) x_test];% add offset
coeff = [K ; L];

y_predicted = x_test_modified * coeff ;
class_2 = find(y_predicted >=0);
class_9 = find(y_predicted <0);
y_predicted(class_2) = 2;
y_predicted(class_9) = 9;

%=== Test the classifier error loss01===
err = loss01(y_predicted, y_test)

%[y_predicted, y_test]
```

```
err =

    0.0750
```

Compare LDA with a KNN classifier with K=5

knnClassify(train_data,train_label,test_data,k)

```
k_values=5;

% Knn prediction
predTest(:,1)=knnClassify(x_train,y_train,x_test,k_values);

%=== Test the classifier error loss01===

errTest(:,1)=loss01(predTest(:,1),y_test)


%=========================================================================
```

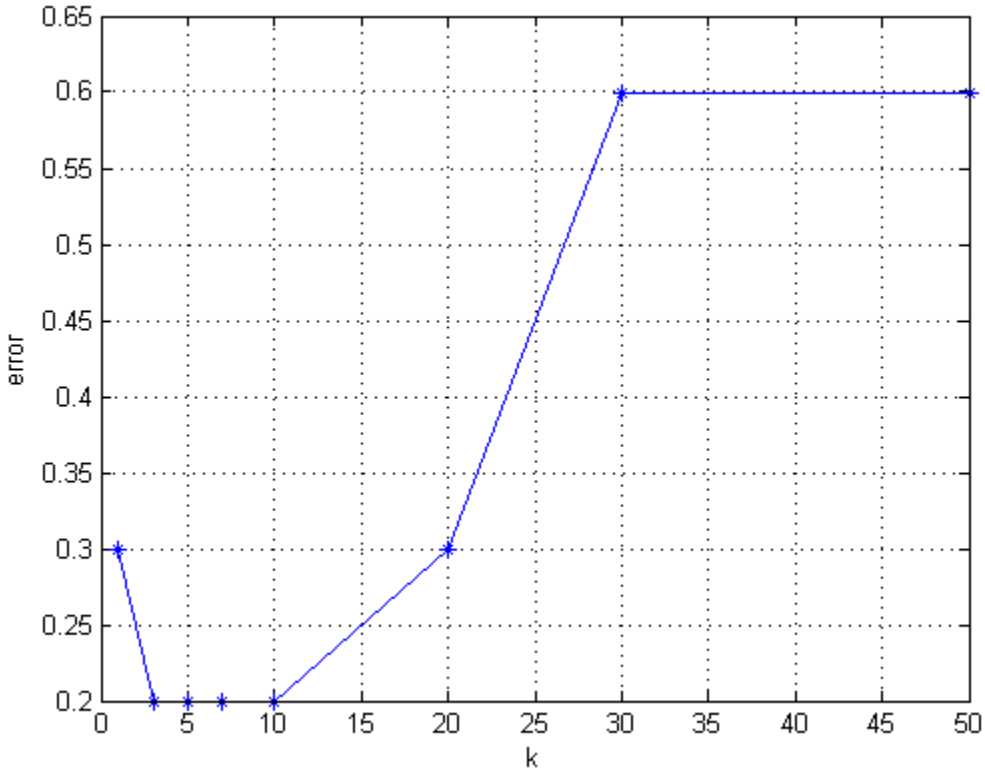
```
errTest =

    0.0250
```

Cross validation (10-fold) for KNN

Select the parameter which performed best in the m folds (usually, one considers the average test error).
Set the seed

```
rng(123)

% Y train (the labels) has a length 100, and we will do partitions of 90/10

k=[1,3,5,7,10,20,30,50];
CVO = cvpartition(y_train,'k',10);
err = zeros(CVO.NumTestSets,1);

for j=1:length(k)
for i = 1:CVO.NumTestSets
    trIdx = find(CVO.training(i)); % 90 indexes respect to the train data
    teIdx = find(CVO.test(i));% 10 indexes respect to the train data
    % Cross validation we test against a portion of the same train data
    ytest = knnClassify(x_train(trIdx,:),y_train(trIdx,:),x_train(teIdx,:),k(j));

    err(i) = loss01(ytest,y_train(teIdx));
end
% Average error
cvErr(j) = sum(err)/sum(CVO.TestSize);
end
```

```
plot(k,100*cvErr,'*-')
xlabel('k')
ylabel('error')
grid on
```



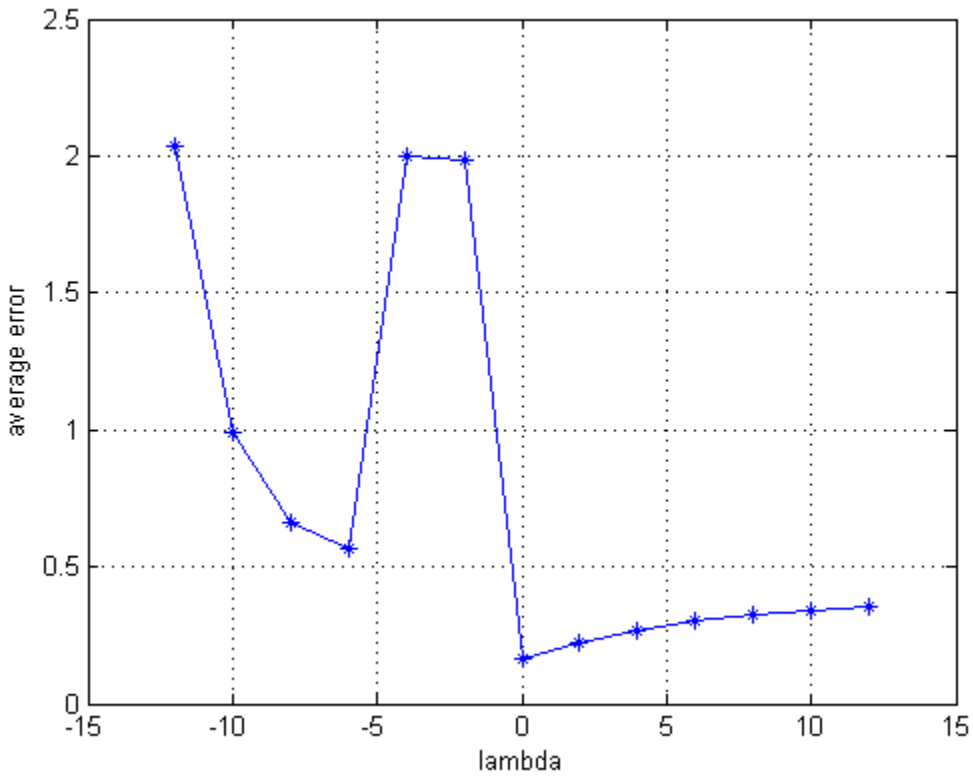
Exercise 6 Cross validation (10-fold) for KNN

Select the parameter which performed best in the m folds (usually, one considers the average test error). Set the seed

```
clc; clear all; close all;
load dataRidge.mat
rng(123)

% Buld the polynomial basis
% https://en.wikipedia.org/wiki/Polynomial_regression#Matrix_form_and_calculation_
% y_data= x_poly * a'  , where a is a vector of coefficients to be computed
for i=0:3
x_poly(:,i+1)= x_train.^i;
end

% Y train (the labels) has a length 100, and we will do partitions of 90/10
p=15; % polynomial basis degree
lambda=[-12,-10,-8,-6,-4,-2,0,2,4,6,8,10,12];
CVO = cvpartition(y_train,'k',10);
err = zeros(CVO.NumTestSets,1);
```

```
for j=1:length(lambda)
for i = 1:CVO.NumTestSets
    trIdx = find(CVO.training(i)); % 90 indexes respect to the train data
    teIdx = find(CVO.test(i));% 10 indexes respect to the train data
    % Cross validation we test against a portion of the same train data
    % named test
    weight= RidgeLLS(x_poly(trIdx,:),y_train(trIdx,:),lambda(j));
    ytest=x_poly(teIdx,:)*weight;

    err(i) =  sqrt((ytest-y_train(teIdx))'*(ytest-y_train(teIdx)));
end
% Average error
cvErr(j) = sum(err)/sum(CVO.TestSize);
end

figure(11)
plot(lambda,cvErr,'*-')
grid on
xlabel('lambda')
ylabel('average error')
```

Warning: One or more folds do not contain points from all the groups.



Published with MATLAB® 7.14