INF 4300 - Exercise for Friday 05.11.2010
Using data efficiently

Goal

This exercise aims to further improve your skills with using Matlab and additional toolboxes for pattern recognition. The main topics of this exercise is feature evaluation, selection and performance evaluation. Furthermore, the topic of linear feature extraction (PCA and LDA) will be studied. Please read the **Stuff to learn** and try out the commands there, before you start doing the exercises. Also, this exercise assumes you have completed the previous exercise.

Stuff to learn

PRTools

Familiarize yourself with classfication and feature selection using PRTools. Make sure you understand the concepts of and are able to classify data using Gaussian ML classification (see:`help ldc` and `help qdc`). Make sure you know what a confusion matrix is and how you plot a scatterplot. Make/choose a simple example and plot a classification boundary on a scatterplot. Compare the results using different classification rules. Look at the help files of the different feature selection commands `featureval`, `featrank`, `featselb`, `featself`, `featseli`, `featselm`, `featselo`, and `featselp`.

Feature Evaluation

The routine `featureval` can be used to evaluate feature sets according to a criterion. For a given dataset, it returns either a distance between the classes in the dataset or a classification accuracy. In both cases it means that large values means good separation. Load the dataset `biomed`. How many features does this dataset have? How many possible subsets of two features can be made from this dataset? Make a script which loops through all possible subsets of two features and that creates for each combination a new dataset b. Use `featureval` to evaluate b using the Euclidean distance, the Mahalanobis distance and the leave-one-out error for the one-nearest neighbour rule.

Find, for each of the three criteria, the two features that are selected by individual ranking (use `featseli`), by forward selection (use `featself`) and by the above procedure that finds the best combination of two features. Compute for each set of two features the leave-one-out error for the one-nearest neighbour rule by `testk`.

Classifier evaluation and error estimation

The following routines are useful to know for evaluation of classifiers:

`testc` test a dataset on a trained classifier
`crossval` train and test classifiers by cross validation
`cleval` classifier evaluation by computing a learning curve
`gendat` split a given dataset at random into a training set and a test set.
`confmat` calculate a confusion matrix for your classifier

A simple example of the generation and use of a test set is the following:

Load the `mfeat_kar` dataset, consisting of 64 Karhunen-Loeve coefficients measured for 10*200 written digits ('0' to '9'). A training set of 50 objects per class (i.e. a fraction of 0.25 of 200) can be generated by:

```
>> a = mfeat_kar
MFEAT KL Features, 2000 by 64 dataset with 10 classes: [200 ... 200]
>> [trainset,testset] = gendat(a,0.25)
MFEAT KL Features, 500 by 64 dataset with 10 classes: [50 ... 50]
MFEAT KL Features, 1500 by 64 dataset with 10 classes: [150 ... 150]
```

50 × 10 objects are stored in trainset, the remaining 1500 objects are stored in testset. Train the linear normal densities based classifier and test it:

```
>> w = ldc(trainset);
>> testset*w*testc
```

Compare the result with training and testing by all data:

```
>> a*ldc(a)*testc
```

which is probably better for two reasons. Firstly, it uses more objects for training, so a better classifier is obtained. Secondly, it uses the same objects for testing as well a for training, by which the test result is definitely positively biased. Because of that, the use of separate sets for training and testing has to be preferred.

Exercises

Performance estimates

Write one or more m-functions that evaluate confusion matrices and report overall error, average error normalized according to class size, precision, recall and kappa.

Learning curves introduction

A learning curve displays the estimated performance as a function of the number of training examples. An easy to use routine for studying the learning curve of a classifier on a given dataset is `cleval`:

```
>> a = gendatb([30 30])
>> e = cleval(a,ldc,[2 3 5 10 20],3)
```

This generates at random training sets of sizes [2 3 5 10 20] per class out of the dataset a and trains the classifier `ldc`. The remaining objects are used for testing (so in this example the set a has to contain more than 20 objects per class). This is repeated 3 times and the resulting errors are averaged and returned in the structure e. This is ready made for plotting the so called learning curve by:

```
>> plotr(e)
```

An entire classification system (optional - but probably useful in the future)

In this exercise a classification system will be developed for classifying objects that belong to four different categories: ring, nut-6 (6-sided), nut-4 (4-sided), and bolt. A vision

system is available that acquires images of these objects. Using some digital image processing techniques (not part of this project) the images are segmented. After that, each imaged object is represented by a connected component in the resulting binary (logical) image. Figure 1 shows already segmented images containing rings, nuts-6, nuts-4 and bolts. These images are available for training and evaluation, thus providing us with a labeled dataset of 121 objects per class.

The classification will be based on so-called normalized Fourier descriptors. Which describe the shapes of the contours of the objects. The basic idea to consider the contour as a periodic curve in 2D which can be represented by a Fourier series. The Fourier descriptors are the coefficients of the Fourier series and may be used to create descriptors that are invariant to rotation and scale. The software provided with the project can produce many descriptors per objects. The goal of your design is to find a classifier that strives for minimal error rate.

The software that is provided within this project can calculate up to 64 descriptors denoted by Z_k, where k ranges from -31 up to $+32$. The descriptors are normalized such that they are independent from the orientation and the size. However, Z_0 and Z_1 should not be used, because Z_0 does not depend on the shape (but rather on the position) and Z_1 is always one (because it is used for the normalization). The given Matlab function, `ut_contourfft`, offers the possibility to calculate only a selection of the available descriptors.

Each image in Figure 1 shows the segments of 121 objects. Thus, extraction of the boundary of each segments, and subsequent determination of the normalized Fourier descriptors yields a training set of $4 \times 121 = 484$ labelled vectors, each vector having 62 elements. An image can be transformed into a set of measurement vectors with the following fragment of code:

```
fdlist = [-31:-1 2:32]; % exclude Z0 and Z1
imrings = imread('rings.tif'); % open and read the image file
figure; imshow(imrings); title('rings');
[BND,L,Nring,A] = bwboundaries(imrings,8,'noholes');% extract the
boundaries
FDS = ut_contourfft(BND,'fdlist',fdlist,'nmag'); % calculate the FDs
Zrings = zeros(Nring,length(fdlist)); % allocate space
for n=1:Nring
     Zrings(n,:) = FDS{n}'; % collect the vectors
end
```

Note that the function `bwboundaries` is part of the image processing toolbox in matlab so you need a computer that has access to this toolbox to be able to import your data. Likewise pieces of code are needed to get the measurement vectors from the other classes. The filenames of the four images are: `rings.tif`, `nuts6.tif`, `nuts4.tif` and **`bolts.tif`.** The function `ut_contourfft` accompanies the images. These images and function are convenienty stored under the `~inf4300/prdatasets/` path.

a) Write the code needed to import the data into Matlab and then into PRTools format. Hint: use repmat to create the array with labels. For instance, `repmat('ring',[Nring 1])` creates an array of `Nring` entries containing the string `'ring'`.

b) Use 5-fold cross validation to estimate the performance of the kNN classifier for k=3 when the classifier is using the ten Fourier descriptors Z2,Z3,…,Z11 or some other subset

of features.

c) Compute a confusion matrix which gives an impression which classes are likely to be confused. Evaluate precision, recall and kappa of your confusion matrix.

d) Use the cell array approach and crossvalidation to compare several of the classifiers you know on one half of the dataset. Keep the other half for testing after training is finished. How does the crossvalidation results on the training half compare to the classification results on the testing half. Discuss.

Simple feature selection

First we will create an artificial dataset for which we know there are just a few informative features. Find out what the characteristics of `gendatd` are and create a dataset containing 40 objects. Next, we rotate this dataset clockwise around the origin 45_. We do this by multiplying the dataset by the rotation matrix:$R = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

Make a new dataset b from the old dataset by multiplying it with this rotation matrix (note the sizes of the matrices, and be careful in which order you multiply the matrices!). Check your results by making a scatterplot of the two datasets. Finally add 4 extra non-informative features to dataset b. Use the procedure `gendats` to make two classes which are exactly on top of each other (see help files). Check the size of the new dataset (it should be 40×6 now!) and make scatterplots of features $(1,2), (1,3)$ and $(4,5)$.

Given this artificial dataset, would you prefer to use forward or backward feature selection? Any ideas on choice of criteria J to be used? Try forward and backward selection on this dataset using different criteria.
To find out which features have been selected, extract the feature indices from the mapping w by: `+w` Do you find the correct features in both cases? What results do you get by individual feature selection?

Feature Selection

Load the `glass` dataset. Rank the features by the sum of the Mahalanobis distances, usingindividual selection (`featseli`), forward selection (`featself`) and backward selection (`featselb`). The selected features can be retrieved from the mapping `w` by:

```
>> w = featseli(a,'maha-s');
>> getdata(w)
```

Compute for each feature ranking an error curve for the linear gaussian classifier by `clevalf`.

```
>> rand('seed',1); e = clevalf(a*w,ldc,[],[],5)
```

The random seed is reset to make the results for different feature sequences `w` comparable. The command `a*w` reorders the features in dataset a according to `w`. In `clevalf`, the classifier is trained by a randomly subsampled version of the given dataset. The remaining objects are used for testing. This is repeated 5 times. All results are stored in a structure e that can be visualised by `plotr(e)`.
Plot the result for the three feature sequences obtained by the three selection methods in a single figure by `plotr`. Compare this error plot with a plot of the `'maha-s'` criterion value

as a function of the feature size (use `feateval`).

Custom feature evaluation (very optional)

It is possible to make `feateval` use other measures of distance (for example Bhattacharrya or divergence) by a bit of creative Matlab programming. (Hint: you have two options; modify the `feateval` routine or wrap your distance measure into a "'classifier"' mapping)

Feature scaling

Besides classifiers that are hampered by the amount of features, some classifiers are sensitive to the scaling of the individual features. This can be studied by an experiment in which the data is good and one in which the data is badly scaled.
In relation with sensitivity to badly scaled data, we have three types of classifiers:

1. classifiers that are scaling independent
2. classifiers that are scaling dependent, but that can compensate badly scaled data by large training sets.
3. classifiers that are scaling dependent, that cannot compensate badly scaled data by large training sets.

First, generate a training set of 400 points for two normally distributed classes with common covariance matrix, as follows:

```
>> a = gauss(400,[0 0; 2 2],eye(2))
```

Prepare another dataset b by scaling down the second dimension of dataset a as follows:

```
>> x = +a; x(:,2) = x(:,2).*0.01; b = setdata(a,x);
```

Study the scatter plot of a and b (e.g. `scatterd(a)`) and note the difference when the scatterplot of b is scaled properly (`axis equal`).
Which of the following classifiers belong to which type (1,2 or 3)?:
`nmc knnc([],1),qdc`
It may help if you plot the decision boundaries in the scatter plots of a and b and play with the training set size.

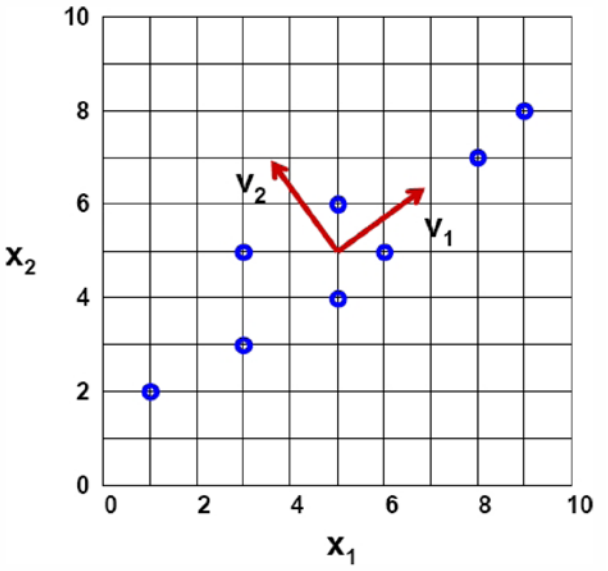
Verify your answer by the following experiment: Generate an independent test set c and compute the learning curves (i.e. an error curve as function of the size of the training set) for each of the classifiers. Use training sizes of 5,10,20,50,100 and 200 objects per class. Plot the error curves. Use `scalem` for scaling the features on their variance. For a fair result, this should be computed on the training set b and applied to b as well as to the test set c:

```
>> w = scalem(b,'variance'); b = b*w; c = c*w;
```

Compute and plot the learning curves for the scaled data as well. Which classifier(s) are independent of scaling? Which classifier(s) can compensate bad scaling by a large training set?

Principal component analysis

Given a dataset $x = (x1, x2) = (1,2),(3,3),(3,5),(5,4),(5,6),(6,5),(8,7),(9,8)$ compute the principal components. Use for example the covariance estimate $\Sigma_x = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^T(x_i - \mu)$. Eigenvalues are zeros of the characteristic equation $\Sigma_x v = \lambda v \Rightarrow |\Sigma_x - \lambda I| = 0$



Eigenvalues should become $\lambda 1 = 9.34$, $\lambda 2 = 0.41$ and eigenvectors $\begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix} = [0.81, 0.59]^T, \begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix} = [-0.59, 0.81]^T$

PCA and KLM

Familiarize yourself with `pca` and `klm`. The latter performs a Karhunen-Loeve mapping - which is another name for Principal Components. However the two methods differ in the way they estimate the covariance matrix, `klm` uses the average covariance matrix whereas `pca` uses the total data covariance matrix. What is the difference, and what results does it give?

More complex feature extraction

Take a simple 2-dimensional dataset, for instance `gendats`. Compute the covariance matrix of the data, using `c = cov(+x)`, and finally compute the eigenvectors, by `[v,d] = eig(c)`. Now make a scatterplot of the data. Can you interpret the results you get from `eig`? Map your data onto these new axis and make a scatterplot of the new dataset. Is it what you expected? Use the command `pca` and compare the results with your results.

Generate the dataset `a = gendatd(20,20,2,10)`. Compare the datasets you obtain from pca and `klm`.

Use the command `iris` to load a classification problem. Study the dataset. Use the command `fisherm` to perform a LDA on the dataset. Select the two best features using PCA/KLM, LDA and feature selection methods. Compare performances of the different methods on a testset. (Hint: To create a subset of the data to obtain a test set - use `gendat`)