Exercise 6: Classification Problems

In this exercise you will be asked to develop several tools relevant for classification problems. The data set that will guide us concerns the classification of radar returns from the ionosphere. It can be downloaded from `http://www.mbfys.kun.nl/~tom/files/ionosphere.dat`.

This radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere.

Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this database are described by 2 attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal.

The database consists of 351 instances (rows in the file), each made up of 34 inputs (the first 34 columns) and 1 output (the last column) specifying good (1) or bad (0) returns. We will use the first 200 instances for fitting different models. The remaining 151 instances will be used for independent testing of the performance of these models.

Principal Component Analysis

It is difficult to visualize the data in 34 dimensions. A popular method for dimension reduction is "principal component analysis". The basic idea is to make a linear projection $\mathbf{x}'$ of the original $\mathbf{x}$, i.e.,

$$\mathbf{x}' = Q\mathbf{x} \, ,$$

such that we keep as much "information" as we can. For visualization purposes, $\mathbf{x}'$ is usually taken to be two-dimensional, that is, we will make a projection from 34 to 2 dimensions and the projection matrix Q is 2 by 34. Now it can be argued that the best projection matrix consists of (any linear combination of) the two eigenvectors of the 34 by 34 input covariance matrix C with components

$$C_{kl} = \frac{1}{n-1} \sum_{i=1}^{n} (x_{ki} - \bar{x}_k)(x_{li} - \bar{x}_l) \, ,$$

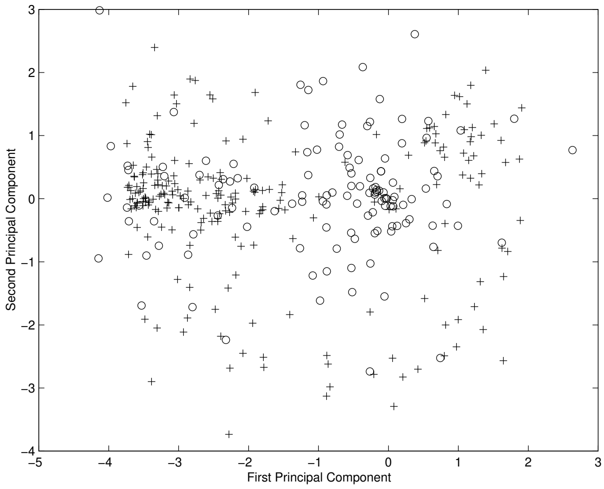
that correspond to the largest eigenvalues. (Note that if the different inputs correspond to different entities, it would have been better to first rescale all inputs to have mean zero and, for example, standard deviation 1.)

a. Build a program called `pca.m` that takes as its first argument an $n_{\text{inp}} \times n$ matrix of input values `x` and as its second argument the number `k` of principal components (2 for the above example), and returns the projection of `x` (called $\mathbf{x}'$ above) onto the first `k` principal components.

b. Plot the projections onto the first two principal directions against each other. Use different symbols for the "good" and "bad" radar returns. You can use all 351 instances. The result should look like Figure 1.

Logistic Regression

We will first apply logistic regression based on the two-dimensional projections of the original inputs. This enables us to visualize the results. Below we will apply logistic regression on the unprojected original inputs. Here and in the following, build your models on the first 200 instances and use the remaining 151 instances for testing.



You can use the function `solve_log.m`, that calls the function `error_log`, for applying logistic regression. Both can be downloaded from the course website:
`http://www.mbfys.kun.nl/~tom/files/solve_log.m` and
`http://www.mbfys.kun.nl/~tom/files/error_log.m`.

c. Use the function `solve_log.m` (with `lambda` set to zero) to solve the logistic regression problem on the first 200 instances. What is the percentage of incorrectly classified training/test instances?

d. Compute the decision boundary and plot it in the figure made under (b).

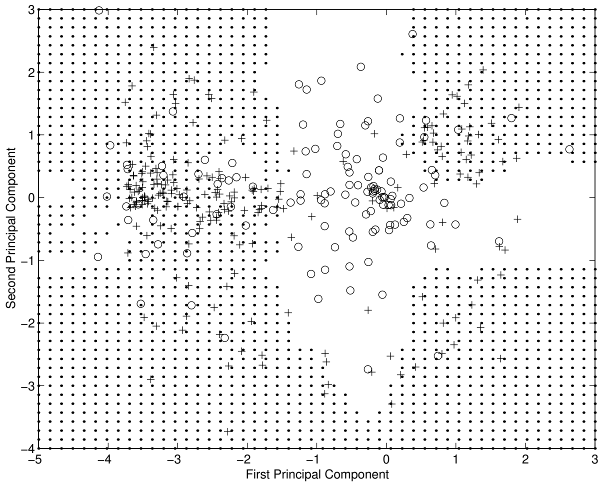
Nearest Neighbor Classification

Logistic regression is just one tool for solving classification problems. Another popular and relatively straightforward one is nearest neighbor classification. The idea is as follows. Given a data set of n instances and a new input vector $\mathbf{x}$, find the input $\mathbf{x}_i$ closest to the $\mathbf{x}$ and classify $\mathbf{x}$ as y_i. The straightforward generalization is k-nearest neighbor classification: find the k input vectors closest to $\mathbf{x}$ and use majority voting. For example, if $k = 3$, the new vector is classified as "good" if two or more of the three closest inputs have class "good" and classified as "bad" otherwise.

The function to be used is called `knnclass.m`, which can be downloaded from `http://www.mbfys.kun.nl/~tom/files/knnclass.m`. It takes as inputs `xtest` (an $n_{\text{inp}} \times n_{\text{test}}$ matrix of test inputs), `xtrain` (an $n_{\text{inp}} \times n_{\text{train}}$ matrix of training inputs), `ytrain` (an $1 \times n_{\text{train}}$ vector of corresponding outputs), and `k` (the number of nearest neighbors) and returns `ypred` (an $1 \times n_{\text{test}}$ vector of predicted outputs).

e. Compute and plot the classification error (percentage misclassified instances) for k-nearest neighbor classification for different k, e.g., ranging from 1 to 29 in steps of 2 (why?). Present the performance on both the training and the test set in one plot. What would you say is the the optimal number of nearest neighbors in this case?

f. Visualize the best solution obtained with the nearest neighbor algorithm. For example, take a grid of points and classify each of them using the nearest neighbor classifier and give different colors to the points that are classified as "good" and "bad". You should get something like Figure 2 (here only the grid points classified as "good" are colored). Do the same for the 1-nearest neighbor classifier. Can you explain the difference?



Classification Based on the Raw Inputs

Now we go back to the original data set and apply both logistic regression and the nearest neighbor classifier based on all raw inputs, i.e., based on the 34 by 200 dimensional matrix of training inputs.

g. Apply logistic regression based on all raw inputs and evaluate the training and test errors. Try different values of the regularization parameter `lambda` to see if you can improve the test performance and eventually, plot the training and test performance over a range of "interesting" values.

h. As e., but now based on all raw inputs. How do you interpret the results?

i. If you *had* to choose a single model out of the ones that you have tried until now to classify new data that the model has never seen before (that is, not the test data that you used to validate your models on, but another "random" test set), which model would you prefer? Do you think you have enough information to reliably make such a choice? If not, what other experiments would you suggest, given the available data?

ROC Curves

j. Build an ROC curve for the best logistic regression model that you have found. Do this both for the training and for the test instances. What specificity can be obtained if we require a sensitivity of at least 0.9? Explain in your own words what this statement means.

Hints for Exercise 6

a. You can make use of the MATLAB routines `cov` for computing the covariance matrix (make sure that you get a matrix of size $n_{\text{inp}} \times n_{\text{ninp}}$ out of it), `eig` for computing both the eigenvectors and eigenvalues of the covariance matrix, `diag` for turning the matrix with eigenvalues into a vector of eigenvalues, and `sort` for sorting them.

b. `hold on` holds the current plot so that subsequent graphing commands are added instead of overwriting the previous ones.

c. Also for later purposes it may help to build a function `simu_log.m` with a similar semantics as `simulin.m` that you used in Exercise 5. That is, `simu_log` takes as inputs the matrix of inputs `x`, the vector of weights `w`, and the threshold `t`, and returns the vector of class labels `ypred`. You can then compare `ypred` with the labels `y` according to the data set, e.g., using something like `length(find(ypred == y))` (the number of correctly classified instances) or just `sumsqr(ypred-y)` (the number of misclassified instances).

d. To emphasize the decision boundary graphically, you can use `h = plot(...)` and then `set(h,'LineWidth',2)`. `h` is called a handle to a graphics object. With `get(h)` you can view its properties, that can be changed with `set(h,'Property','Value')` where `'Property'` stands for the particular property you want to change. Just `set(h,'Property')` gives you the possible values.

e. Just write a for-loop running over different numbers of nearest neighbors.

f. For building a grid that covers the "interested area" (for example, given by the axes that result from the plot you made under (b)) you can use `meshgrid`. It gives you a *matrix* of x and y coordinates. Something like `xx = x(:)';` and `yy = yy(:)';` concatenates the matrix into a (long) vector. Then put them together using `xxx = [xx;yy];` to make an $2 \times n_{\text{grid}}$ matrix of inputs. This is the proper format for `knnclass` that then yields an $1 \times n_{\text{grid}}$ vector of predicted classes to be visualized.

g. As you will notice, you will have to choose rather low values of `lambda`. Furthermore, you probably want to change `lambda` over different orders of magnitudes. You can use `semilogx` instead of `plot` for a logarithmic scaling on the x-axis.

j. ROC curves are discussed in the syllabus, section 6.3.3. Based on the information given there you should be able to work this out.