

ICS 274 Homework 3

Probabilistic Learning: Theory and Algorithms, ICS 274, Winter 2009

Due Date: Thursday, February 19th, in class

Homework Problems

The homeworks are intended to help you work through the concepts we discuss in class in more detail. It is important that you try to work through the problems yourself. If you can't solve a problem, you can discuss it with another person, but the basic rule is that you can only discuss the problem verbally (no exchange or viewing of written notes, solutions, material, code, etc). Homeworks will consist of a mix of review problems related to what we are discussing in class and programming assignments in MATLAB so that you can actually try out some of the ideas we discuss in class with actual data and learning algorithms.

Please answer each of the following questions and submit your written solutions. In problems that ask for a proof you must submit a complete mathematical proof (i.e., each line must follow logically from the preceding one, no "hand-waving" allowed!).

Problem 1: (10 points)

Consider a 2 class, 1-dimensional Gaussian classification problem where the class probabilities are $p(c_1) = p(c_2) = 0.5$, and

$$p(x|c_i) \sim N(\mu_i, \sigma_i), \quad i = 1, 2$$

Let $\mu_1 = 0$, $\sigma_1 = 1$, $\mu_2 = \mu$, $\sigma_2 = \sigma$.

1. Derive a general expression for the location of the Bayes optimal decision boundary as a function of μ and σ .
2. Now let $\mu = 1$, $\sigma = 2$.
 - Determine the location of the optimal decision boundary
 - Sketch the densities $p(x|c_i)$, the posterior probabilities for $p(c_i|x)$, and the location of the optimal decision regions.
 - Estimate the Bayes error rate p_e^* for the problem.
3. Comment on the case where $\mu = 0$, σ is much greater than 1. Describe a practical example of a classification problem where such a situation might arise.

Problem 2: (10 points)

Consider a 2-class Gaussian problem, with d -dimensional feature vectors \underline{x} and class means $\underline{\mu}_i$, $i = 1, 2$. Assume that $\Sigma_1 = \Sigma_2 = \Sigma$, i.e., both classes have a common covariance matrix. Prove that under these assumptions that $g_i(\underline{x})$ is a linear discriminant function, i.e., that

$$g_i(\underline{x}) = \underline{w}_i^T \underline{x} + w_{i0}$$

and derive equations for the weight vector \underline{w}_i and scalar offset w_{i0} . Clearly show all steps in your derivation of the result.

Problem 3: (10 points)

Consider the situation in Problem 2, for the case of $d = 2$ and $p(c_1) = p(c_2)$, and with the mean of class 1 at $(1, 1)$ and the mean of the second class at $(4, 4)$ and with a covariance matrix defined so that $\sigma_{11} = 1$, $\sigma_{22} = 4$, and $\sigma_{12} = 1$. Sketch a picture in the 2-dimensional space (x_1, x_2) that shows (roughly) (a) probability contours for $p(\underline{x}|c_i)$, $i = 1, 2$, and (b) the location of the optimal decision boundary relative to the 2 class means. If you wish you can use MATLAB to generate the plots for you, but hand-drawn is also fine.

Problem 4: (10 points)

Redo problem 3 but now assume that $p(c_1) = 0.8, p(c_2) = 0.2$. Provide a 1-line interpretation of the results.

Problem 5: (10 points)

For problem 4 derive an equation for the optimal decision boundary between the classes explicitly in terms of x_1 and x_2 .

Problem 6: (10 points)

Consider a linear classifier defined by linear discriminant functions

$$g_i(\underline{x}) = \underline{w}_i^T \underline{x} + w_{i0} \quad i = 1, \dots, m.$$

Prove that the decision regions are **convex** by showing that if $\underline{x}_1 \in DR_i$ and $\underline{x}_2 \in DR_i$ then $\alpha \underline{x}_1 + (1 - \alpha) \underline{x}_2 \in DR_i$ if $0 \leq \alpha \leq 1$. The notation $\underline{x}_1 \in DR_i$ means that \underline{x}_1 lies within the decision region DR_i .

MATLAB Assignment (30 points)

For your MATLAB assignments please turn in (a) clearly documented printouts of your code and (b) any plots or write-ups requested below (all attached to your written problem solutions above).

You are to write a series of MATLAB functions that will estimate the parameters of a Gaussian density from data, evaluate and plot the resulting parameter estimates, build a 2-class Gaussian classifier, classify a test data set, measure the error and plot the resulting classifications in two-dimensions.

Part A

Your first function is called `gaussparams.m`: it takes a real-valued d -dimensional data set (in the form specified below) and generates maximum likelihood estimates of the parameters for the Gaussian model from this data. You are to write this function—hint: it need only be about 2 lines long (!) if you take advantage of the MATLAB mean and covariance functions.

```
function [params] = gaussparams(data);
%
% [params] = gaussparams(data);
%
% Function to calculate maximum likelihood estimates of Gaussian parameters
%
% INPUTS:
%     data:  n x d data matrix, n observations (rows), d variables (columns)
%
% OUTPUT:
%     params: a structure containing the following:
%     params.mu = d x 1 vector containing the maximum likelihood
%               estimate of the mean
%     params.covar = d x d matrix containing the maximum likelihood
%               estimate of the covariance matrix
%
% .... your code goes here .....
```

You should check that your function is working correctly by simulating data from a simple known Gaussian distribution (use `mvnrnd.m` from Homework 1), with a fairly large number of data points (say 1000) and checking that the resulting parameter estimates are fairly close to the true ones.

Part B

Next we will extend our MATLAB functions to build a two-class Gaussian classifier, and we will test how it works. First we build a more general function that builds on our `gaussparams.m` function from before. The function `gaussclassparams.m` takes in a labeled data set and produces maximum likelihood estimates of the covariance, mean, and class prior probability, for each of the m classes:

```
function [cparams] = gaussclassparams(data, labels);
```

```

%
% [cparams] = gaussparams(data);
%
% Function to calculate maximum likelihood estimates of Gaussian parameters
%
% INPUTS:
%     data:  n x d data matrix, n observations (rows), d variables (columns)
%     labels: n x 1 vector of class labels corresponding to "data",
%             where label values are assumed to go from 1 to m
%
% OUTPUT:
%     cparams:  an m x 1 array of structures, where cparams(k) is the kth such
%               such structure containing:
%     cparams(k).prior = estimated prior probability for class i
%     cparams(k).mu = d x 1 vector containing the maximum likelihood
%                   estimate of the mean for class i
%     cparams(k).covar = d x d matrix containing the maximum likelihood
%                   estimate of the covariance matrix for class i

... put some error checking here, e.g., to check labels and data
are the same size.....

m = max(unique(labels)); % find out how many labels there are: we
% should really also check here that all labels from 1 to m are present

.... loop over each of the m classes

index = labels==k % find the indices of the data points belong to class k
cdata = data(index,:) % extract the data points belonging to class k

..... now estimate the parameters for data in class k using your
..... gaussparams.m function

params = ... cparams(k).prior = ....
.....

```

Part C

Now we build a function to take a cparams structure and a data set (with labels), classify each data point, compare the classification with the true label, and calculate the resulting error rate:

```

function [error, predlabels] = gaussclassify(data, labels, cparams);
%
% [cparams] = gaussparams(data);
%

```

```

% Function to calculate maximum likelihood estimates of Gaussian parameters
%
% INPUTS:
%     data:  n x d data matrix, n observations (rows), d variables (columns)
%     labels: n x 1 vector of class labels corresponding to "data",
%             where label values are assumed to go from 1 to m
%     cparams: an m x 1 array of structures, where cparams(k) is the kth such
%             such structure containing:
%     cparams(k).prior = estimated prior probability for class i
%     cparams(k).mu = d x 1 vector containing the maximum likelihood
%                   estimate of the mean for class i
%     cparams(k).covar = d x d matrix containing the maximum likelihood
%                   estimate of the covariance matrix for class i
%
% OUTPUT:
%     error: the number of disagreements between labels and prelabels, divided by n
%     prelabels: n x 1 vector of *predicted* class labels corresponding to "data",
%             where label values are assumed to go from 1 to m

... put some error checking here, e.g., to check labels and data
are the same size.....

m = max(unique(labels)); % find out how many labels there are: we
% should really also check here that all labels from 1 to m are present

for i=1:n
    .... loop over each of the n observations in data: (one
could in fact "vectorize" this instead of looping but for our
purposes a "for loop" is fine:

for k=1:m
    .... an inner loop over each class k

density(i,k) = gdensity(data(i,:), cparams(k)); % calculate the conditional Gaussian
% density for data point i, given the parameters of class k: you
% might just want to write a simple function gdensity.m to do this

joint(i,k) = density(i,k)*cparams(k).prior % multiply by the prior: this
% joint probability is our discriminant function for class k

end % continue to loop over class values k

prelabel(k) = max....
%now take the maximum over the m class values of joint(i,:)

```

```
end % continue to loop over data points i

error = .... % calculate the overall error rate
```

Part D

Now perform the following experiment (where you can again automate the whole process using a suitably-defined MATLAB script function):

1. Simulate a training data set from 2 classes in two-dimensions, 10 points from each class. Class 1 is zero mean, identity covariance matrix. Class 2 has mean at (2,2) and identity covariance matrix. You will need to create a vector of class label data (of size 20 by 1) as well—you can write a function that automates all of this by taking as input the class parameters, and calls `mvnrnd.m` for each class.
2. Use your `gaussclassparams.m` function to estimate the class parameters on the training data: check that they are reasonable (i.e., that your function is working as you think it should). Plot the data and parameters in two-dimensions, using color or symbols to indicate which data points are from which class (you can modify your plotting function from Homework 1, using the “hold” command to overlay a second set of Gaussian points on the first).
3. Generate a separate (independent) test data set from the same two Gaussian classes as in step 1, but now with 5000 data points per class.
4. Repeat step 1 for 100 different training data sets, use `gaussclassparams.m` function to estimate the class parameters on each training data, and use each set of parameters and `gaussclassify.m` to classify the test data. Report the average accuracy as a percentage and the standard deviation of this percentage (across the 100 runs)
5. Report the accuracy of the Gaussian classifier on the test data, but where you use the test data for training (just do this once on the 5000 data points from part 1).

Repeat steps 3, 4, and 5 above with the same sample sizes except now in a 5-dimensional setting, where the first mean is at the origin, the second mean is at [2,2,2,2,2], and the covariance for each class is the identity matrix.

Comment on the differences between the results in 2 dimensions versus 5 dimensions.

Part E

(OPTIONAL): write a function to plot the decision boundary resulting from `gaussclassparams.m` in the 2 dimensional, 2 class case. Plot the decision boundary overlaid over the test data (and/or over the training data). Another option here is to create a fine grid on the 2d space, calculate the posterior probability $p(c_1|\underline{x})$ over the space, and then plot either as a gray-scale pixel image (quite neat to look at, you can use a function such as `imagesc.m`) or a contour plot (using `contour.m`).