CS 520 — Artificial Intelligence
Assignment One
Out: Feb 14, 2000
Due: March 1, 2000

Written Exercises

Problem 1. Consider the following decision rule for a two-category one-dimensional problem: decide ω_1 if $x > \theta$; otherwise decide ω_2.

(a) Show that the probability of error for this rule is given by

$$P(\text{error}) = P(\omega_1)\int_{-\infty}^{\theta} p(x|\omega_1)dx + P(\omega_2)\int_{\theta}^{\infty} p(x|\omega_2)dx$$

(b) By differentating, show that a necessary condition to minimize $P(\text{error})$ is that θ satisfy

$$p(\theta|\omega_1)P(\omega_1) = p(\theta|\omega_2)P(\omega_2)$$

(c) Does this equation define θ uniquely?

(d) Give an example where a value of θ satisfying the equation actually *maximizes* the probability of error.

Note: for a discussion of probability of error and classification see lecture notes for Jan 26.

Problem 2. Show that if our model is poor, the maximum likelihood classifier we derive is not the best—even among our poor model set—by exploring the following example. Suppose we have two equally probable categories (i.e., $P(\omega_1) = P(\omega_2) = 0.5$). Further, we know that $p(x|\omega_1) \sim N(0,1)$ but *assume* that $p(x|\omega_2) \sim N(\mu,1)$. (Thus, μ is the parameter we seek to estimate by the method of maximum likelihood.) Imagine however that the *true* underlying distribution is $p(x|\omega_2) \sim N(1,10^6)$.

(a) What is the value of our maximum likelihood estimate $\hat{\mu}$ in our poor model, given a large amount of data?

(b) What is the decision boundary arising from this maximum likelihood estimate in the poor model?

(c) Ignore for the moment the maximum likelihood approach, and use the methods described in class on Jan 31 to derive the Bayes optimal decision boundary given the *true* underlying distributions— $p(x|\omega_1) \sim N(0.1)$ and $p(x|\omega_2) \sim N(1,10^6)$. Be careful to include all portions of the decision boundary.

(d) Now consider again classifiers based on the (poor) model assumption $p(x|\omega_2) \sim N(\mu,1)$. Using your result immediately above, find a *new* value of μ that will give lower error than the maximum likelihood classifier.

Note: maximum likelihood estimation was the topic of the lecture on Feb 2, and MLE for Normal distributions is covered in the day's lecture notes.

Problem 3. Consider an extreme case of the general issue discussed in problem 2, one in which it is possible that the maximum likelihood solution leads to the *worst* possible classifier, i.e., one with an error that approaches 100% (in probability). Suppose our data in fact comes from two one-dimensional distributions of the forms

$$p(x|\omega_1) \sim [(1-k)\delta(x-1) + k\delta(x+X)] \quad \text{and}$$
$$p(x|\omega_2) \sim [(1-k)\delta(x+1) + k\delta(x-X)]$$

X is a positive real number; k lies in the range $0 \leq k < 0.5$ and represents the portion of the total probability mass concentrated at the point $\mp X$; $\delta(x)$ is the Dirac delta function—which has a nonzero only at the point $x = 0$ but nevertheless integrates to one.

Suppose our (poor) models are of the form $p(x|\omega_1) \sim N(\mu_1, \sigma_1^2)$ and $p(x|\omega_2) \sim N(\mu_2, \sigma_2^2)$ and we form a maximum likelihood classifier.

(a) Consider the symmetries in the problem and show that in the infinite data case the decision boundary will always be at $x = 0$, regardless of k and X.

(b) Recall that the maximum likelihood estimate of either mean, $\hat{\mu}_i$, is the mean of its distribution. For a fixed k, find the value of X such that the maximum likelihood estimates of the means "switch", i.e., where $\hat{\mu}_1 \geq \hat{\mu}_2$.

(c) Find a dependence $X(k)$ which will guarantee that the estimated mean $\hat{\mu}_1$ for $p(x|\omega_1)$ is less than zero. (By symmetry, this will also ensure that $\hat{\mu}_2 > 0$.)

(d) Given your $X(k)$ just derived, state the classification error in terms of k.

(e) Discuss how if our model is wrong (here, if it does not include the delta functions), the error can approach 100% (in probability). Does this surprising answer arise because we have found some local minimum in parameter space?

Computer Exercises

This problem asks you to implement two classifiers for multivariate, normally-distributed measurements—one based on the linear discriminant functions described in class on January 31; and another based on the nearest neighbor heuristic described in class on February 9.

To test and analyze your classifiers, I have provided a program called `gen_data`. It is available on `paul` as—

```
/grad/u1/mdstone/520/gen_data
```

If you prefer to work elsewhere, on another machine, you can obtain the source code from—

```
http://www.cs.rutgers.edu/˜mdstone/class/520/gen_data.c
```

and compile it yourself (just be sure to link against the math libraries: on unix, cc gen_data.c -lm).

The program works with multivariate distributions that satisfy the assumptions of linear classifiers: all the features are independent and across classes each feature has a variance of σ^2. A classification setup is then specified by

- the number of states of nature c
- the dimensionality of the feature spaces k
- the standard deviation parameter σ for the problem
- for each class ω_i, the prior probability $P(\omega_i)$ and the k-component mean parameter μ_i

`gen_data` expects to read off these parameters in order off the standard input. For example, take this sample input—

```
2 4 1.0
0.25 1   2   2   1
0.75 2 1.5 1.5 1.5
```

So it describes a binary classification problem, involving measurements with four components and unit variance. The first class, representatives of which appear a quarter of the time, is characterized by the prototype $\mu_1 = \langle 1, 2, 2, 1 \rangle$. The second class, representatives of which appear three-quarters of the time, is characterized by the prototype $\mu_2 = \langle 2, 1.5, 1.5, 1.5 \rangle$. `gen_data` expects a single number n on the command line; it then writes to `stdout`:

- A single line containing the number of classes, the number of features, and the number of samples to follow in the file; then
- A sequence of n measurements drawn from the classification problem specified on `stdin`, one per line thereafter, in the form:

  $$c x_1 x_2 (\ldots) x_k$$

  where c is the state of nature behind the measurement and x_1 through x_k provide space-delimited values for each component of the measurement.

If you save your classification problem as a file `dist` — for distribution — then you would typically run `gen_data` with something like

```
gen_data 200 < dist > test.data
```

By the way, the reason this program is provided here is that the code to sample from a Gaussian is weird and not particularly illuminating.

Program 1. Write a program `p1` as follows. Read in a datafile of samples in the format output by `gen_data` and output the maximum likelihood estimate of the parameters of the classification problem from which the samples were drawn; it may be convenient to accept the datafile on standard input and to output the parameter estimate in the format accepted by `gen_data`.

In estimating the variance computationally, it is useful to observe the equivalence

$$\sum_j (x_{ij} - \hat{\mu}_i)^2 = (\sum_j x_{ij}^2) - n\hat{\mu}_i^2$$

Program 2. Write a program p2 as follows. Read in a parameter estimate for the classification problem and a datafile of samples. Construct the Bayes-optimal linear decision algorithm for the classification problem and calculate the classifications assigned to each sample. Tabulate the results of the classification in a *confusion matrix* M. M_{ij} gives the number of test points of class i that were classified as belonging to class j. As the output of the run, p2 should report the confusion matrix and also the overall error rate for your classifier.

It may be convenient to accept the parameter estimates on stdin and to give the file name for the test data as a command-line argument; that way you can use commands to sequence the programs you write together such as

```
p1 < train.data | p2 test.data
```

Program 3. Write a program p3 as follows. p3 should input a training file of samples output for a classification problem; then it should read in a test file of samples output from the same classification distribution. It should classify each test point with the same label found with the nearest neighbor to the test point in training data.

Again, report results as a confusion matrix and an overall error rate; if you take the test samples on the command line and read the training data from standard input, it may give some consistency and some opportunities for running the classifier piped with other programs.

Analysis 1. Check your results in p1 informally by comparing the estimated parameters to the parameters that produced the training data. Do means and variances converge as you would expect, given more data?

Analysis 2. Design a classification problem in which you expect nearest neighbor and linear classifiers to have the same performance. Construct training and test data for your problem using gen_data. Exhibit the classification problem and the output for p1|p2 and p3. Explain the results; show in particular that the confusion matrices you produce are characteristic of the two algorithms.

Analysis 3. Design a classification problem in which you expect nearest neighbor to do distinctly worse than a linear classifiers. Construct training and test data for your problem using gen_data. Exhibit the classification problem and the output for p1|p2 and p3. Explain the results; show in particular that the confusion matrices you produce are characteristic of the two algorithms.