3.2

\[ g(x) = \begin{cases} 
1 & \log \frac{P(C_1|x)}{P(C_2|x)} > 0 \\
0 & \text{otherwise}
\end{cases} \]

3.3

Loss Function

\[ \lambda_{11} = \lambda_{22} = 0 \]
\[ \lambda_{12} = 10 \]
\[ \lambda_{21} = 1 \]

Risk

\[ R(\alpha_1|x) = 0 \cdot P(C_1|x) + 10 \cdot P(C_2|x) = 10 \cdot P(C_2|x) \]
\[ R(\alpha_2|x) = 1 \cdot P(C_1|x) + 0 \cdot P(C_2|x) = P(C_1|x) \]

Optimal Decision Rule

\[ \begin{cases} 
1 & P(C_1|x) > 10 \cdot P(C_2|x) \\
2 & \text{otherwise}
\end{cases} \]

4.2

The combined log likelihood is

\[ \mathcal{L}(p|\mathcal{X}) = \log \prod_{t=1}^{N} \prod_{i=1}^{K} p_i^{x_i^t} \]
\[ = \sum_{t=1}^{N} \sum_{i=1}^{K} x_i^t \log p_i \]

Multinomial densities can be treated as multiple instances of Bernoulli densities, so we have

\[ \mathcal{L}_i(p_i|\mathcal{X}) = \log \prod_{t=1}^{N} p_i^{x_i^t} (1 - p_i)^{1-x_i^t} \]
\[ = \sum_{t} x_i^t \log p_i + (N - \sum_{t} x_i^t) \log(1 - p_i) \]
\[ \frac{d\mathcal{L}_i}{dp_i} = \frac{\sum_{t} x_i^t}{p_i} - \frac{N - \sum_{t} x_i^t}{1 - p_i} \]

and setting \( \frac{d\mathcal{L}_i}{dp_i} = 0 \):
\[ 0 = \sum_{t} x_i^t - \frac{N - \sum_{t} x_i^t}{1 - p_i} \]
\[ 0 = (1 - p_i) \sum_{t} x_i^t - p_i(N - \sum_{t} x_i^t) \]
\[ 0 = (1 - p_i) \sum_{t} x_i^t - p_i(N - \sum_{t} x_i^t) \]
\[ 0 = \sum_{t} x_i^t - p_i \sum_{t} x_i^t - Np_i + p_i \sum_{t} x_i^t \]

\[ Np_i = \sum_{t} x_i^t \]
\[ p_i = \frac{\sum_{t} x_i^t}{N} \]

and so the MLE of \( p_i \) is
\[ \hat{p}_i = \frac{\sum_{t} x_i^t}{N} \]

which is equation 4.6.

### 4.3

Language = Ruby

```ruby
def random_gauss
    #
    # Returns a random number distributed normally with
    #   mean = 0 and standard deviation = 1
    #
    Math.sqrt(-2 * Math.log(rand)) * Math.cos(2*Math::PI*rand)
end
def normal_sample(mu, sigma, n)
    #
    # Generates a random sample of the given size with
    #   the given parameters
    #
    (1..n).map{|_|random_gauss*sigma + mu}
end
def stats(s)
    #
    # Computes the mean and standard deviation of the given sample,
    #   and returns them as a hash map
    #
    n = s.length.to_f
    mu=s.inject{|u,v|u+v}/n
    sigma=Math.sqrt(s.map{|x|(|x-mu)**2}.inject{|u,v|u+v}/n)
    {:mean=>mu, :std_dev=>sigma}
end
```
def bayes_stats(s, mu_mean, mu_std_dev)
    #
    # uses the supplied distribution parameters for the mean to compute
    # the mean and standard deviation of the sample
    #
    # returns a hashmap just like stats(s)
    #
    # Create some convenience variables to keep the formula clean
    n = s.length.to_f
    sample_stats = stats(s)
    sample_mu = sample_stats[:mean]
    sample_sigma = sample_stats[:std_dev]
    n_over_sample_sigma_squared = n/(sample_sigma**2)
    mu_inverse_variance = 1/(mu_std_dev**2)
    # Textbook formula 4.18
    estimated_mu =
        (n_over_sample_sigma_squared/
            (n_over_sample_sigma_squared + mu_inverse_variance))*sample_mu +
        (mu_inverse_variance/
            (n_over_sample_sigma_squared + mu_inverse_variance))*mu_mean
    new_std_dev = Math.sqrt(s.map{|x|(x-estimated_mu)**2}.inject{|u,v|u+v}/n)
    {
        mean=>estimated_mu, std_dev=>new_std_dev
    }
end

4.4

Starting with

\[
g_1(x) = \log P(C_1) - \log \sigma_1 - \frac{(x - \mu_1)^2}{2\sigma_1^2}
\]

and

\[
g_2(x) = \log P(C_2) - \log \sigma_2 - \frac{(x - \mu_2)^2}{2\sigma_2^2}
\]

we set them equal and solve for \(x\):

\[
\log P(C_1) - \log \sigma_1 - \frac{(x - \mu_1)^2}{2\sigma_1^2} = \log P(C_2) - \log \sigma_2 - \frac{(x - \mu_2)^2}{2\sigma_2^2}
\]

\[
\frac{(x - \mu_2)^2}{2\sigma_2^2} - \frac{(x - \mu_1)^2}{2\sigma_1^2} = \log \frac{C_2\sigma_1}{C_1\sigma_2}
\]

\[
\sigma_1^2(x - \mu_2)^2 - \sigma_2^2(x - \mu_1)^2 = 2\sigma_1^2\sigma_2^2 \log \frac{C_2\sigma_1}{C_1\sigma_2}
\]

\[
\sigma_1^2(x^2 - 2\mu_2 x + \mu_2^2) - \sigma_2^2(x^2 - 2\mu_1 x + \mu_1^2) = 2\sigma_1^2\sigma_2^2 \log \frac{C_2\sigma_1}{C_1\sigma_2}
\]

So we get a quadratic equation with
\[ a = (\sigma_1^2 - \sigma_2^2) \]
\[ b = 2(\sigma_2^2 \mu_1 - \sigma_1^2 \mu_2) \]
\[ c = \sigma_2^2 \mu_2^2 - \sigma_1^2 \mu_1^2 - 2\sigma_1^2 \sigma_2^2 \log \frac{C_2}{C_1} \]

and so the discriminant points are at

\[ x = \frac{-2(\sigma_2^2 \mu_1 - \sigma_1^2 \mu_2) \pm \sqrt{4(\sigma_2^2 \mu_1 - \sigma_1^2 \mu_2)^2 - 4(\sigma_1^2 - \sigma_2^2)(\sigma_1^2 \mu_2^2 - \sigma_2^2 \mu_1^2 - 2\sigma_1^2 \sigma_2^2 \log \frac{C_2}{C_1} \sigma_1 \sigma_2)}}{2(\sigma_1^2 - \sigma_2^2)} \]

which simplifies to

\[ x = \frac{(\sigma_2^2 \mu_2 - \sigma_1^2 \mu_1) \pm \sqrt{(\sigma_2^2 \mu_1 - \sigma_1^2 \mu_2)^2 - (\sigma_1^2 - \sigma_2^2)(\sigma_1^2 \mu_2^2 - \sigma_2^2 \mu_1^2 - 2\sigma_1^2 \sigma_2^2 \log \frac{C_2}{C_1} \sigma_1 \sigma_2)}}{(\sigma_1^2 - \sigma_2^2)} \]

\[ 4.6 \]

There are three possibilities when estimating discriminant points. The first is that variances are equal and priors are equal, and in this case the one discriminant point is simply the average of the two means, as given in the textbook on page 71. The second possibility is that variances are equal, but priors are not; the formula for this case will be derived below. The third possibility is that the variances are not equal, and the formula for this case was covered in question 4.4.

When variances are equal but priors are different:

\[
\log P(C_1) - \log \sigma - \frac{(x - \mu_1)^2}{2\sigma^2} = \log P(C_2) - \log \sigma - \frac{(x - \mu_2)^2}{2\sigma^2} \\
\frac{(x - \mu_1)^2 - (x - \mu_2)^2}{2\sigma^2} = \log \frac{P(C_1)}{P(C_2)} \\
2\mu_2 x - 2\mu_1 x + \mu_1^2 - \mu_2^2 = 2\sigma^2 \log \frac{P(C_1)}{P(C_2)} \\
x = \frac{2\sigma^2 \log \frac{P(C_1)}{P(C_2)} + \mu_2^2 - \mu_1^2}{2(\mu_2 - \mu_1)} \\
x = \frac{\mu_1 + \mu_2}{2} + \frac{\sigma^2 \log \frac{P(C_1)}{P(C_2)}}{\mu_2 - \mu_1}
\]

All three cases are covered by the following code:

```python
def set_priors(stats1, stats2):
    # convenience method to add prior values to a pair of statistics
    # If neither are set, set them both to 0.5
    # If one is set, set the other to (1 - p)
    stats1[:prior] = 0.5 unless stats1[:prior] or stats2[:prior]
    stats1[:prior] ||= 1 - stats2[:prior]
    stats2[:prior] ||= 1 - stats1[:prior]
end
```
def discriminant_points(stats1, stats2):
    # Special formulas if variances are equal
    if stats1[:std_dev] == stats2[:std_dev]:
        # Simply return average if no priors
        return [(stats1[:mean] + stats2[:mean]) / 2.0] unless (
            stats1[:prior] or stats2[:prior])
        set_priors(stats1, stats2)
        var = stats1[:std_dev]**2
        return (stats1[:mean] + stats2[:mean]) / 2.0 +
        var * Math.log(stats1[:prior] / stats2[:prior]) /
        (stats2[:mean] - stats1[:mean])
    end

set_priors(stats1, stats2)

# precompute variances
var1, var2 = [stats1, stats2].map{|s|s[:std_dev]**2}

# set quadratic formula variables and return solution
a = (var1 - var2)
b = 2 * (var2 * stats1[:mean] - var1 * stats2[:mean])
c = var1 * stats2[:mean] - var2 * stats1[:mean] -
    2 * var1 * var2 * Math.log(
        (stats2[:prior] * stats1[:std_dev])/
        (stats1[:prior] * stats2[:std_dev]))
[((-b + Math.sqrt(b**2 - 4*a*c))/(2*a),
    (-b - Math.sqrt(b**2 - 4*a*c))/(2*a))]
end

Using that code, we can write the following method that performs problem 4.6:

def four_point_six(n, mu_1, std_dev_1, mu_2, std_dev_2):
    # Using the given parameters, computes theoretic discriminant points,
    # generates a random sample (taking \n numbers from each distribution)
    # and estimates the discriminant points using parametric classification
    #
    theoretic = discriminant_points({:mean=>mu_1, :std_dev=>std_dev_1},
                                   {:mean=>mu_2, :std_dev=>std_dev_2})
sample_1 = stats(normal_sample(mu_1, std_dev_1, n))
sample_2 = stats(normal_sample(mu_2, std_dev_2, n))

    parametric = discriminant_points(sample_1, sample_2)

    puts "For two samples of (mean,std_dev) = [%.3f,%.3f],(%.3f,%.3f)]," %
    [mu_1, std_dev_1, mu_2, std_dev_2]
    puts " the theoretical discriminant points are \n        %.5f, %.5f" %
    theoretic
print " and the parametric estimates for two samples each of size "
puts "#{n} are\n  %.5f, %.5f\n" % parametric
end

Here is the result of running that code five times with random parameters:

5.times do
  four_point_six(100*(1+rand(50)), # n
                 rand*20-10, # mu_1
                 rand*10,   # std_dev_1
                 rand*20-10, # mu_2
                 rand*10)  # std_dev_2
end

# Result:
For two samples of (mean, std_dev) = [(3.789, 4.030), (-2.468, 7.235)],
  the theoretical discriminant points are -1.43173, 14.63542
  and the parametric estimates for two samples each of size 2800 are -1.45950, 14.11663

For two samples of (mean, std_dev) = [(-4.557, 8.529), (9.519, 2.355)],
  the theoretical discriminant points are 21.58227, -0.22115
  and the parametric estimates for two samples each of size 4000 are 21.76356, -0.21102

For two samples of (mean, std_dev) = [(-6.779, 3.257), (1.106, 3.379)],
  the theoretical discriminant points are -220.65970, 0.54861
  and the parametric estimates for two samples each of size 400 are 72.48602, 0.34669

For two samples of (mean, std_dev) = [(-2.068, 6.356), (-0.003, 8.362)],
  the theoretical discriminant points are -13.91411, 4.12457
  and the parametric estimates for two samples each of size 700 are -14.06240, 4.07968

For two samples of (mean, std_dev) = [(-6.270, 5.482), (-1.535, 0.471)],
  the theoretical discriminant points are 0.70119, -3.70171
  and the parametric estimates for two samples each of size 3300 are 0.69764, -3.70523

The values are fairly close with the exception of the third, where we can see that the variances were very close, and the sample sizes were low, which caused the minor discriminant point to jump from the negative side (-220) to the positive side (72), which means that the first sample had a higher estimated variance than the second sample (despite the opposite being true for the original parameters).
In practice, this effect is of little importance, since the probability of encountering a value that far from the two means is quite close to 0. The variances may as well be equal (with just one discriminant point).

7.4 Variables

A sample of vectors \( \{x^1 \cdots x^t \cdots x^N \} \) taken from components \( \{G_1 \cdots G_i \cdots G_k \} \) where each vector \( x^t \) has elements \( \{x^t_1 \cdots x^t_b \cdots x^t_m \} \). The (hidden) indicator variables for each \( x^t \) are \( \{z^t_1 \cdots z^t_k \} \). For the EM algorithm we use the prior probabilities for the class \( \pi_i = P(G_i) \), as well as the distribution of the components themselves:

\[
\Phi = \{ \phi_1 \cdots \phi_i \cdots \phi_k \}
\]

\[
\phi_i = \{ \phi_{i,1} \cdots \phi_{i,b} \cdots \phi_{i,m} \}
\]

where \( \phi_{i,b} \) is \( p(x_b = 1 | G_i) \).

Probability Density Function

For a Bernoulli mixture, we can express the probability density function of each mixture as a product of the probabilities of the individual variables:

\[
p_i(x^t) \equiv p(x^t | G_i) = \prod_{b=1}^{m} \phi_{i,b} x^t_b (1 - \phi_{i,b})^{1-x^t_b}
\]

Likelihood

The log likelihood is

\[
\mathcal{L}(\Phi | X, Z) = \sum_t \sum_i z^t_i [\log \pi_i + \log p_i(x^t | \Phi)]
\]

\[
= \sum_t \sum_i z^t_i [\log \pi_i + \log \prod_{b=1}^{m} \phi_{i,b}^{x^t_b} (1 - \phi_{i,b})^{1-x^t_b}]
\]

\[
= \sum_t \sum_i z^t_i [\log \pi_i + \sum_{b=1}^{m} [x^t_b \log \phi_{i,b} + (1 - x^t_b) \log (1 - \phi_{i,b})]]
\]

E-Step

For the E-step of the EM algorithm, we define

\[
Q(\Phi | \Phi^t) = E[\mathcal{L}(\Phi | X, Z) | X, \Phi^t]
\]

\[
= \sum_t \sum_i E[z^t_i | X, \Phi^t] [\log \pi_i + \sum_{b=1}^{m} [x^t_1 \log \phi_{i,b} + (1 - x^t_1) \log (1 - \phi_{i,b})]]
\]

In order to expand \( E[z^t_i | X, \Phi^t] \), we need to define

\[
E[z^t_i | X, \Phi^t] \equiv h^t_i \equiv P(G_i | x^t, \Phi^t) = \frac{p(x^t | G_i, \Phi^t) P(G_i)}{\sum_j p(x^t | G_j, \Phi^t) P(G_j)}
\]

\[
= \frac{\pi_i \prod_{b=1}^{m} \phi_{i,b}^{x^t_b} (1 - \phi_{i,b})^{1-x^t_b}}{\sum_j \pi_j \prod_{b=1}^{m} \phi_{j,b}^{x^t_b} (1 - \phi_{j,b})^{1-x^t_b}}
\]
**M-Step** In the M-step, we maximize $Q$ to find $\Phi^{t+1}$ as well as the new values for $\pi_t$ (starting from equations 7.11 and 7.12 in the textbook):

$$\pi_t = \frac{\sum_t h_t^t}{N}$$

$$= \sum_t \frac{\pi_t \prod_b \phi_{i,b}^t (1 - \phi_{i,b})^{1-x_b^t}}{N \sum_j \pi_j \prod_b \phi_{j,b}^t (1 - \phi_{j,b})^{1-x_b^t}}$$

we also need to solve for

$$\nabla_{\Phi} \sum_t \sum_i h_t^i \log p_i(x_i | \Phi) = 0$$

$$\nabla_{\Phi} \sum_t \sum_i h_t^i \sum_b [x_b^i \log \phi_{i,b} + (1 - x_b^i) \log (1 - \phi_{i,b})] = 0$$

$$\nabla_{\Phi} \sum_t \sum_i \sum_b h_t^i [x_b^i \log \phi_{i,b} + (1 - x_b^i) \log (1 - \phi_{i,b})] = 0$$

Once the summations have been rotated, we can isolate and maximize each individual $\phi_{i,b}$:

$$\nabla_{\phi_{i,b}} \sum_t h_t^i [x_b^i \log \phi_{i,b} + (1 - x_b^i) \log (1 - \phi_{i,b})] = 0$$

$$\nabla_{\phi_{i,b}} \sum_t [h_t^i x_b^i \log \phi_{i,b}] + \left( \sum_t h_t^i - \sum_t h_t^i x_b^i \right) \log (1 - \phi_{i,b}) = 0$$

$$\sum_{\phi_{i,b}} h_t^i x_b^i - \sum_{1 - \phi_{i,b}} h_t^i - \sum_t h_t^i x_b^i = 0$$

$$\sum_{\phi_{i,b}} h_t^i x_b^i = \sum_{1 - \phi_{i,b}} h_t^i - \sum_t h_t^i x_b^i$$

$$\sum_t h_t^i x_b^i - \phi_{i,b} \sum_t h_t^i x_b^i = \phi_{i,b} \sum_t h_t^i - \phi_{i,b} \sum_t h_t^i x_b^i$$

$$\phi_{i,b} = \sum_t h_t^i x_b^i \sum_t h_t^i$$

**Weka**

I used the "Census Income" dataset from [http://archive.ics.uci.edu/ml/datasets/Adult](http://archive.ics.uci.edu/ml/datasets/Adult) The sample is taken from the population of working adults, where the attributes are various demographic attributes (such as age, sex, education, country of origin, etc.) and the classes are based on income - one class for those who make more than $50,000/year, and one for those who make less.

Using the Naive Bayes classifier on Weka, the output is the following:
Time taken to build model: 0.28 seconds

=== Evaluation on test split ===
=== Summary ===
Correctly Classified Instances 9273 83.7594 %
Incorrectly Classified Instances 1798 16.2406 %
Kappa statistic 0.5029
Mean absolute error 0.1687
Root mean squared error 0.3658
Relative absolute error 46.3148 %
Root relative squared error 86.2523 %
Total Number of Instances 11071

=== Detailed Accuracy By Class ===

<table>
<thead>
<tr>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>ROC Area</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.934</td>
<td>0.477</td>
<td>0.864</td>
<td>0.934</td>
<td>0.898</td>
<td>0.897</td>
<td>&lt;=50K</td>
</tr>
<tr>
<td>0.523</td>
<td>0.066</td>
<td>0.709</td>
<td>0.523</td>
<td>0.602</td>
<td>0.897</td>
<td>&gt;50K</td>
</tr>
</tbody>
</table>
Weighted Avg. 0.838 0.38 0.828 0.838 0.828 0.897

=== Confusion Matrix ===

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>Classified as</th>
</tr>
</thead>
<tbody>
<tr>
<td>7913</td>
<td>557</td>
<td>a = &lt;=50K</td>
</tr>
<tr>
<td>1241</td>
<td>1360</td>
<td>b = &gt;50K</td>
</tr>
</tbody>
</table>