20. Figure 4.4 of HTF gives a 2-dimensional plot of the "vowel training data" (available on the book's website at http://www-stat.stanford.edu/~tibs/ElemStatLearn/index.html or from http://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Vowel+Recognition+-++Deterding+Data%29). The ordered pairs of first 2 canonical variates are plotted to give a "best" reduced rank LDA picture of the data like that below (lacking the decision boundaries).

![Figure 1: Decision boundaries for the vowel training data, in the two-dimensional subspace spanned by the first two canonical variates, according to a) nearest mean and b) logistic regression.](image)

a) Use Section 10.1 of the course outline to reproduce Figure 4.4 of HTF (color-coded by group, with group means clearly indicated). Keep in mind that you may need to multiply one or both of your coordinates by $-1$ to get the exact picture.

Use the R function `lda` (MASS package) to obtain the group means and coefficients of linear discriminants for the vowel training data. Save the lda object by a command such as `LDA=lda(insert formula, data=vowel).`
R Code

vowel <- read.table("vowel.train", header = TRUE, sep = ",", quote = "")
da <- vowel[, -1]
color <- c("#60FFFF", "#B52000", "#FF99FF", "#20B500", "#FFCA00", "red", "green", "blue", "grey75", "#FFB2B2", "#928F52")

Y <- da[, 1]
X <- da[, -1]
K <- length(unique(Y))
N <- length(Y)
p <- ncol(X)
mu.k <- do.call("rbind", lapply(1:K, function(k) colMeans(X[Y == k,])))
mu.bar <- colMeans(mu.k)
mu.k.tmp <- matrix(rep(t(mu.k), N / K), nrow = ncol(mu.k))
Sigma <- (t(X) - mu.k.tmp) %*% t(t(X) - mu.k.tmp) / (N - K)
Sigma.eigen <- eigen(Sigma)
Sigma.inv.sqrt <- Sigma.eigen$vectors %*% diag(1/sqrt(Sigma.eigen$values)) %*% t(Sigma.eigen$vectors)
X.star <- t(Sigma.inv.sqrt %*% (t(X) - mu.bar))
mu.k.star <- t(Sigma.inv.sqrt %*% (t(mu.k) - mu.bar))

M <- mu.k.star
M.svd <- eigen(t(M) %*% M / K)

X.new <- X.star %*% M.svd$vectors
mu.k.new <- mu.k.star %*% M.svd$vectors

plot(-X.new[, 1], X.new[, 2], col = color[Y], pch = Y + 1, main = "Linear Discriminant Analysis",}

---
b) Reproduce a version of Figure 1a) above. You will need to plot the first two canonical coordinates as in a). Decision boundaries for this figure are determined by classifying to the nearest group mean. Do the classification for a fine grid of points covering the entire area of the plot. If you wish, you may plot the points of the grid, color coded according to their classification, instead of drawing in the black lines.

R Code

library(MASS)
LDA <- lda(y ~ ., data = da)
X.pred <- predict(LDA, newdata = X)
mu.k.pred <- predict(LDA, newdata = as.data.frame(LDA$means))
X.new <- X.pred$x
mu.k.new <- mu.k.pred$x
x.lim <- range(X.new[, 1])
c) Make a version of Figure 1b) above with decision boundaries now determined by using logistic regression as applied to the first two canonical variates. You will need to create a data frame with columns \( y \), canonical variate 1, and canonical variate 2. Use the \texttt{vglm} function (VGAM package) with \texttt{family=multinomial()} to do the logistic regression. Save the lda object by a command such as \texttt{LR=vglm(insert formula, family=multinomial(), data=data set)}. A set of observations can now be classified to groups by using the command \texttt{predict(LR, newdata, type="response")}, where \texttt{newdata} contains the observations to be classified. The outcome of the predict function will be a matrix of probabilities. Each row contains the probabilities that a corresponding observation belongs to each of the groups (and thus sums to 1). We classify to the group with maximum probability. As in b), do the classification for a fine grid of points covering the entire area of the plot. You may again plot the points of the grid, colorcoded according to their classification, instead of drawing in the black lines.

\begin{verbatim}
  y.lim <- range(X.new[, 2])
  x.grid <- 100
  y.grid <- 100
  grid <- cbind(seq(x.lim[1], x.lim[2], length = x.grid),
                rep(seq(y.lim[1], y.lim[2], length = y.grid), each = x.grid))
  get.class <- function(grid, mu){
    K <- nrow(mu)
    apply(grid, 1, function(x) which.min(dist(rbind(x, mu))[1:K]))
  }
  grid.class <- get.class(grid, mu.k.new[, 1:2])
  plot(X.new[, 1], X.new[, 2], col = color[Y], type = "n",
       main = "Classified to nearest mean",
       xlab = "Coordinate 1", ylab = "Coordinate 2")
  points(grid[, 1], grid[, 2], col = color[grid.class], pch = 19, cex = 0.3)
  points(X.new[, 1], X.new[, 2], pch = Y + 1, cex = 0.8)
  points(mu.k.new[, 1], mu.k.new[, 2], pch = 19, cex = 1.5)
\end{verbatim}
library(VGAM)
da.glm <- as.data.frame(cbind(Y, X.new[, 1], X.new[, 2]))
colnames(da.glm) <- c("y", "c.1", "c.2")
LR <- vglm(y ~ ., family = multinomial(), data = da.glm)

X.pred <- predict(LR, newdata = da.glm[, -1], type = "response")
mu.k.new <- do.call("rbind",
                   lapply(1:K, function(k) colMeans(da.glm[da.glm[, 1] == k, -1])))
mu.k.pred <- predict(LR, newdata = as.data.frame(mu.k.new), type = "response")
X.new.class <- apply(X.pred, 1, which.max)
mu.k.new.class <- apply(mu.k.pred, 1, which.max)

x.grid <- 100
y.grid <- 100
grid <- cbind(seq(x.lim[1], x.lim[2], length = x.grid),
               rep(seq(y.lim[1], y.lim[2], length = y.grid), each = x.grid))
grid <- as.data.frame(grid)
colnames(grid) <- c("c.1", "c.2")
grid.pred <- predict(LR, grid, type = "response")
grid.class <- apply(grid.pred, 1, which.max)

plot(X.new[, 1], X.new[, 2], col = color[Y], type = "n",
     main = "Classified by Logistic Regression",
     xlab = "Coordinate 1", ylab = "Coordinate 2")
points(grid[, 1], grid[, 2], col = color[grid.class], pch = 19, cex = 0.3)
points(X.new[, 1], X.new[, 2], pch = Y + 1, cex = 0.8)
points(mu.k.new[, 1], mu.k.new[, 2], pch = 19, cex = 1.5)

a) So that you can see and plot results, first use the 2 canonical variates employed in problem 20 and use `rpart` in R to find a classification tree with empirical error rate comparable to the reduced rank LDA classifier pictured in Figure 1a. Make a plot showing the partition of the region into pieces associated with the 11 different classes.

(5 pts) I use default settings in `lda()` and `rpart()`, and the LDA result has the error about 0.3163 and regression partitioning has the error about 0.2576. The controls in `rpart()` use a value of `cp` complexity parameter (default 0.1) to build a tree. Therefore, I prune the tree according the value of `cp` in the `prune()` function to fit in a similar error of LDA. The corresponding `cp` value is about 0.0167. The following plots show the tree result and final classifications.

```
vowel <- read.table("vowel.train", header = TRUE, sep = ",", quote = "")
da <- vowel[, -1]
Y <- da[, 1]
X <- da[, -1]
color <- c("#60FFFF", "#B52000", "#FF99FF", "#20B500", "#FFCA00", "red", "green", "blue", "grey75", "#FFB2B2", "#928F52")

library(MASS)
LDA <- lda(y ~ ., data = da)
X.pred <- predict(LDA, newdata = X)
X.1 <- X.pred$x[, 1]
X.2 <- X.pred$x[, 2]
$error.LDA <- mean(as.numeric(X.pred$class) != Y)

library(rpart)
```
b) Beginning with the original data set (rather than with the first 2 canonical variates) and use `rpart` in R to find a classification tree with empirical error rate comparable to the classifier in a). How much (if any) simpler/smaller is the tree here than in a)?

(5 pts) Again, the default settings in `rpart()` give me an error about 0.1629 based on original data set. So, I prune the tree according the value of `cp` in the `prune()` function to fit in a similar error of LDA. The corresponding `cp` value is about 0.0166. Comparing to the part a) and the problem 20, this gives a better fit and more similar to the LDA results. Note that the partition rules are more complex than that in the part a).
22. Consider the famous Wisconsin Breast Cancer data set at [http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29](http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29). Compare an appropriate SVM based on the original input variables to a classifier based on logistic regression with the same variables. (BTW, there is a nice paper "Support Vector Machines in R" by Karatzoglou, Meyer, and Hornik that appeared in *Journal of Statistical Software* that you can get to through ISU. And there is an R SVM tutorial at [http://planatscher.net/svmtut/svmtut.html](http://planatscher.net/svmtut/svmtut.html).

(10 pts) I take off patients with missing values "?" and perform SVM and logistic regression on the Breast Cancer data set. The results are shown in the following, and the error rates for misclassification are \((5 + 11)/682 \approx 0.0235\) for SVM and \((10 + 11)/682 \approx 0.0308\) for logistic regression.
> pred.glm <- predict(model.glm, newdata = da.glm[, -1], type = "response")
> pred.glm <- apply(pred.glm, 1, which.max)
> pred.glm[pred.glm == 2] <- 4
> pred.glm[pred.glm == 1] <- 2
> table(pred.glm, t(da.class))

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>433</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>228</td>
</tr>
</tbody>
</table>

23. (Izenman Problem 14.4.) Consider the following 10 two-dimensional points, the first five points, \((1, 4), (3.5, 6.5), (4.5, 7.5), (6, 6), (1.5, 1.5)\), belong to Class 1 and the second five points \((8, 6.5), (3, 4.5), (4.5, 4), (8, 1.5), (2.5, 0)\), belong to Class 2. Plot these points on a scatterplot using different symbols or colors to distinguish the two classes. Carry through by hand the AdaBoost algorithm on these points, showing the weights at each step of the process. (Presumably Izenman’s intention is that you use simple 2-node trees as your base classifiers.) Determine the final classifier and calculate its empirical misclassification rate.

The results may depend on how you design the classifiers and code the data. I perform a naive partition method on both coordinates for each iteration reweighted by \(\omega\)'s, and select the best one as a candidate. At the end of all iterations, I weight the candidates as the final AdaBoost results. Overall, a resonable classifier should provide an acceptable result.

(10 pts) The following table gives the detail of the first five iterations.

<table>
<thead>
<tr>
<th></th>
<th>(m = 1)</th>
<th>(m = 2)</th>
<th>(m = 3)</th>
<th>(m = 4)</th>
<th>(m = 5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\omega_{1m})</td>
<td>0.1000</td>
<td>0.1000</td>
<td>0.1000</td>
<td>0.6333</td>
<td>0.6333</td>
</tr>
<tr>
<td>(\omega_{2m})</td>
<td>0.1000</td>
<td>0.2333</td>
<td>0.2333</td>
<td>0.2333</td>
<td>1.0333</td>
</tr>
<tr>
<td>(\omega_{3m})</td>
<td>0.1000</td>
<td>0.2333</td>
<td>0.2333</td>
<td>0.2333</td>
<td>1.0333</td>
</tr>
<tr>
<td>(\omega_{4m})</td>
<td>0.1000</td>
<td>0.2333</td>
<td>0.2333</td>
<td>0.2333</td>
<td>1.0333</td>
</tr>
<tr>
<td>(\omega_{5m})</td>
<td>0.1000</td>
<td>0.1000</td>
<td>0.1000</td>
<td>0.6333</td>
<td>0.6333</td>
</tr>
<tr>
<td>(\omega_{6m})</td>
<td>0.1000</td>
<td>0.1000</td>
<td>0.1000</td>
<td>0.6333</td>
<td>0.6333</td>
</tr>
<tr>
<td>(\omega_{7m})</td>
<td>0.1000</td>
<td>0.1000</td>
<td>0.3667</td>
<td>0.3667</td>
<td>0.3667</td>
</tr>
<tr>
<td>(\omega_{8m})</td>
<td>0.1000</td>
<td>0.1000</td>
<td>0.3667</td>
<td>0.3667</td>
<td>0.3667</td>
</tr>
<tr>
<td>(\omega_{9m})</td>
<td>0.1000</td>
<td>0.1000</td>
<td>0.1000</td>
<td>0.1000</td>
<td>0.1000</td>
</tr>
<tr>
<td>(\omega_{10m})</td>
<td>0.1000</td>
<td>0.1000</td>
<td>0.3667</td>
<td>0.3667</td>
<td>0.3667</td>
</tr>
<tr>
<td>(\text{err})</td>
<td>0.3000</td>
<td>0.2143</td>
<td>0.1364</td>
<td>0.1842</td>
<td>0.1774</td>
</tr>
<tr>
<td>(\alpha_m)</td>
<td>0.8473</td>
<td>1.2993</td>
<td>1.8458</td>
<td>1.4881</td>
<td>1.5339</td>
</tr>
<tr>
<td>split variable</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>cutoff</td>
<td>2.0000</td>
<td>7.0000</td>
<td>5.2500</td>
<td>2.0000</td>
<td>7.0000</td>
</tr>
</tbody>
</table>
my.optim.tree <- function(omega){
  ret <- list(miss = Inf, f = NULL, split = NULL, cutoff = NULL)
  f <- rep(1, N)
  for(i in uniq.split[[1]]){
    f[X[, 1] < i] <- -1
    miss <- sum((X.class != f) * omega)
    if(miss < ret$miss) ret <- list(miss = miss, f = f, split = 1, cutoff = i)
  }
  f <- rep(-1, N)
  for(i in uniq.split[[2]]){
    f[X[, 2] < i] <- 1
    miss <- sum((X.class != f) * omega)
    if(miss < ret$miss) ret <- list(miss = miss, f = f, split = 2, cutoff = i)
  }
  ret
}

N <- 10
X <- matrix(c(c(1, 4), c(3.5, 6.5), c(4.5, 7.5), c(6, 6), c(1.5, 1.5),
              c(8, 6.5), c(3, 4.5), c(4.5, 4), c(8, 1.5), c(2.5, 0)), ncol = 2, byrow = TRUE)
X.class <- 1 - 2 * (rep(1:2, each = 5) == 1)
uniq.split <- lapply(1:2, function(i){ tmp <- sort(unique(X[, i]))
                                      diff(tmp) / 2 + tmp[-length(tmp)] })
M <- 5
omega <- rep(1/N, N)
ret <- list()
for(m in 1:M){
  ret[[m]] <- my.optim.tree(omega)
  ret[[m]]$omega <- omega
  ret[[m]]$bar.err.m <- ret[[m]]$miss / sum(ret[[m]]$omega)
  ret[[m]]$alpha <- log((1 - ret[[m]]$bar.err.m) / ret[[m]]$bar.err.m)
  omega <- omega * exp(ret[[m]]$alpha * (X.class != ret[[m]]$f))
}

### Summary
output <- NULL
for(m in 1:M){
  output <- cbind(output, c(ret[[m]]$omega, ret[[m]]$bar.err.m, ret[[m]]$alpha,
                           ret[[m]]$split, ret[[m]]$cutoff))
}
colnames(output) <- paste("m", 1:M, sep = "")
rownames(output) <- c(paste("w.", 1:10, sep = ""), "err", "alpha", "split", "cutoff")
output

(5 pts) The following plot is the final AdaBoost results for the first five iterations. The colored grid indicate the prediction of classification given the iteration $m$. The colored regions don’t change after $m = 3$ and match the true classifications.
R Code

```r
output.f <- do.call("rbind", lapply(ret, function(i){ i$alpha * i$f }))

### For new X.
xlim <- range(X[, 1])
ylim <- range(X[, 2])
X.new <- cbind(rep(seq(xlim[1], xlim[2], length = 20), 20),
               rep(seq(ylim[1], ylim[2], length = 20), rep(20, 20)))

my.classifier <- function(ret.m, X.new){
  f <- rep(1, nrow(X.new))
  if(ret.m$split == 1){
    f[X.new[, 1] < ret.m$cutoff] <- -1
  } else{
    f[X.new[, 2] > ret.m$cutoff] <- -1
  }
  ret.m$alpha * f
}
```
(4 pts) For the models $m = 1, 2$ and the training set $\mathcal{X} = \{x, y\}$, we have $\pi(m|\mathcal{T}) \propto \pi(m) \int f_m(\mathcal{T}|\theta_m)g_m(\theta_m)d\theta_m$ where $\theta_1 = \{a, b\}$, $\theta_2 = \{c\}$, $f_m = f_m(y|x)f_m(x)$ and $g_m$ are...
priors. Note that $\pi(m) = 0.5$ and $f_x$ can be ignored from Monte Carlo integrations. Therefore,

$$\pi(m | T) \propto \int f_m(y|x)g_m(\theta_m)d\theta_m \approx \sum_{i=1}^{N} f_m(y|x, \theta_m(i))$$

for large $N$ and summing to one constrain. The posterior probabilities for the two models are $\pi(1|T) \approx 0.9939$ and $\pi(2|T) \approx 0.0061$.

(4 pts) Let $\gamma_1(\theta_1) = 10 \cdot \Phi \left( \frac{3-a}{b} \right)$ and $\gamma_2(\theta_2) = 10 \cdot \Phi \left( \frac{3-a}{b} \right)$. Then,

$$E[\gamma_m(\theta_m)|m, T] = \frac{\int \gamma_m(\theta_m)f_m(y|x)g_m(\theta_m)d\theta_m}{\int f_m(y|x)g_m(\theta_m)d\theta_m} \approx \frac{\sum_{i=1}^{N} \gamma_m(\theta_m)f_m(y|x, \theta_m(i))}{\sum_{i=1}^{N} f_m(y|x, \theta_m(i))}$$

for large $N$. The posterior means of the quantities of interest in the two Bayes models are $E[\gamma_{1}(\theta_{1})|1, T] \approx 3.8422$ and $E[\gamma_{2}(\theta_{2})|2, T] \approx 6.921225$.

(2 pts) The predictor of $y$ for $x = 3$ is $\gamma_{m}(\theta_{m}, x) = E_{\theta}[Y|X = 3] = 10p(x = 3|\theta_{m})$ since $y|x$ has a Binomial distribution. The posterior will be

$$E[\gamma_{m}(\theta_{m}, x)|T] \approx \frac{\sum_{m=1}^{M} \pi(m | T) \sum_{i=1}^{N} 10p(x = 3|\theta_{m})f_m(y|x, \theta_{m}(i))}{\sum_{i=1}^{N} f_m(y|x, \theta_{m}(i))}$$

for large $N$. The overall predictors of $y$ for $x = 3$ is $E[\gamma_{m}(\theta_{m}, x)|T] \approx 3.8676$.

---

R Code

```r
x <- 1:5
y <- c(1, 4, 3, 5, 6)
set.seed(6021024)
N.simu <- 10000
a <- runif(N.simu, 0, 6)
b.inv <- rexp(N.simu)
c <- rexp(N.simu)
f.m <- matrix(0, nrow = N.simu, ncol = 2)
for(i in 1:N.simu){
f.m[i, 1] <- exp(sum(dbinom(y, 10, pnorm((x - a[i]) * b.inv[i]), log = TRUE)))
f.m[i, 2] <- exp(sum(dbinom(y, 10, 1 - 1 / ((c[i] + 1) * x), log = TRUE)))
}
(posterior.m <- colSums(f.m) / sum(f.m))
(posterior.1 <- sum(10 * pnorm((3 - a) * b.inv) * f.m[, 1]) / sum(f.m[, 1]))
(posterior.2 <- sum(10 * (1 - 1 / ((c + 1)) * f.m[, 2]) / sum(f.m[, 2])))
(posterior.1 * posterior.m[1] + posterior.2 * posterior.m[2])
```
25. Consider the $N = 50$ training set and predictors of Problem 2. Find (according to Section 9.2.2 of the class outline) $w_{\text{stack}}$ and hence identify a stacked predictor of $y$. How does $\text{err}$ for this stacked predictor compare to those of the two predictors being combined?

(10 pts) Following from the settings of the problems 1 and 2, the stacking method is to find optimal weights for both predictors obtained by the OLS ($\hat{f}_1$) and the nearest averages ($\hat{f}_2$). The optimal weights for two models are $-0.03949157$ and $1.01987057$. The empirical errors (sum of squared error) are given in the followings, and they are $\text{err}_1 \approx 1.6700$, $\text{err}_2 \approx 1.5040$, and $\text{err}_{\text{stack}} \approx 1.5039$ for the fitting methods 1, 2, and stacking. The stacking method is optimized under sum squared errors of predictions by methods 1 and 2, and gives a slightly small empirical error than both.

R Output

```r
> da <- read.table("data.hw1.Q2.set1.txt", sep = "\t", quote = "", header = TRUE)
> estimate.a.j <- function(da.org){
+   a.j <- rep(mean(da.org$y), 10)
+   for(j in 1:10){
+     id <- da.org$x > (j - 1) / 10 & da.org$x <= j / 10
+     if(any(id)){
+       a.j[j] <- mean(da.org$y[id])
+     }
+   }
+   a.j
+ }
> ret.f.1 <- lm(y ~ x, data = da)
> pred.f.1 <- predict(ret.f.1, da)
> ret.f.2 <- estimate.a.j(da)
> pred.f.2 <- ret.f.2[ceiling(da$x * 10)]
> argmin.stack <- function(omega){
+ }
> ret <- optim(c(0.5, 0.5), argmin.stack)
> ret$par
[1] -0.03949157 1.01987057
> (err.f.1 <- mean((da$y - pred.f.1)^2))
[1] 1.670021
> (err.f.2 <- mean((da$y - pred.f.2)^2))
[1] 1.504019
> (err.f.stack <- mean((da$y - pred.f.stack)^2))
[1] 1.503901
```
26. Get the famous \( p = 2 \) “Ripley dataset” (synth.tr) commonly used as a classification example from the author’s site http://www.stats.ox.ac.uk/pub/PRNN/. Using several different values of \( \lambda \) and constants \( c \), find \( g \in \mathcal{H}_K \) and \( \beta_0 \in \mathcal{R} \) minimizing

\[
\sum_{i=1}^{N} (1 - y_i (\beta_0 + g(x_i)))_+ + \lambda \|g\|_{\mathcal{H}_K}^2
\]

for the (Gaussian) kernel \( K(x, z) = \exp(-c \|x - z\|^2) \). Make contour plots for those functions \( g \), and, in particular, show the \( g(x) = 0 \) contour that separates \([-1.5, 1.0] \times [-2, 1.3]\) into the regions where a corresponding SVM classifies to classes -1 and 1.

(10 pts) For this problem, I have to maximize \( \mathbf{1}' \alpha - \frac{1}{2} \alpha' \left( \frac{1}{N} \mathbf{H} \right) \alpha \) subject to \( 0 \leq \alpha \leq \lambda \mathbf{1} \) and \( \alpha' \mathbf{y} = 0 \) for \( \mathbf{H}_{N \times N} = (y_i y_j K(x_i, x_j)) \). Let \( \alpha^{\text{opt}} \) be the solution, and set \( \beta_0(\alpha^{\text{opt}}) = \frac{1}{N^+} \sum_{i: \alpha_i > 0} y_i - \sum_j y_j K(x_i, x_j) \) where \( N^+ = \sum_{i=1}^{N} I(\alpha_i > 0) \). Then, \( \hat{g}(x) = \sum_{i=1}^{N} \alpha_i^{\text{opt}} y_i K(x, x_i) + \beta_0(\alpha^{\text{opt}}) \) and \( \hat{f}(x) = \text{sign}(\hat{g}(x)) \). The following gives the results for different \( \lambda \) and \( c \). Note that the lecture outline uses \( \lambda \frac{1}{2} \) in the Equation (104) to penalize the squared norm of the function \( g \) rather than \( \lambda \).

Elegant solution: Some classmates use \texttt{ipop()} in the library \texttt{kernlab} to solve this problem. This may be a better and quick way for solving kernel problems with constraints. The code is relatively short if you know how to plug in the arguments into \texttt{ipop()}.

Nasty trick: There are several algorithms can handle optimization with complicated constraints. In R, \texttt{constrOptim()} implements a barier algorithm (Lange 2001), but this function is not quite easy to manipulate for large number of constraints. In general, there is a nasty trick I play around with this kind of problems. When I plug in \texttt{optim()} an objective function, I add some checking points inside the objective function to fool \texttt{optim()}. For example, I usually use \texttt{if(...)...else...} to return a \( \pm \text{Inf} \) (dependent on minimizing or maximizing) to fool \texttt{optim()} if the new/next estimates (generated by the optimizing code) are out of any constraint. At the end, if the \texttt{optim()} reports convergence, then the problem may be solved, otherwise it is definitely NOT ok.

If the problem has some signals showing the monotonicity of objective function, then we can just add or substract (dependent on minimizing or maximizing) some arbitrarily values to the initial objective value. In the following, I use \texttt{lower} as the lowest (initial) objective value with substraction 1000 (choose any positive whatever you like) from it. Then, I input it as an argument to adjust the objective function. Can this trick fail? Yes, but for this problem, it converges.
R Code (Warning: this may have some bugs)
d <- read.table("synth.tr", sep = "", quote = "", header = TRUE)
d <- cbind(d$xs, d$ys)
y <- d$yc * 2 - 1
N <- length(y)

get.K <- function(d.i, d.j, c = 1){
  N.i <- nrow(d.i)
  N.j <- nrow(d.j)
  K <- matrix(0, nrow = N.i, ncol = N.j)
  for(i in 1:N.i){
    for(j in 1:N.j){
      K[i, j] <- exp(-c * sum((d.i[i,] - d.j[j,])^2))
    }
  }
  K
}
dual <- function(alpha.1, H, y, N, lambda, lower){
  alpha <- c(alpha.1, -sum(alpha.1 * y[-N]) / y[N])
  ret <- lower ### Here is the trick.
  if(all(alpha > 0) && all(alpha < lambda)){
    ret <- sum(alpha) - alpha %*% H %*% alpha / 2
  }
  -ret
}

estimate.alpha <- function(da.tr, y, N, lambda, c){
  lambda <- lambda / 2
  K <- get.K(da.tr, da.tr, c)
  H <- (y %o% y * K) / lambda^2
  init <- rep(lambda / 2, N)
  lower <- sum(init) - init %*% H %*% init / 2 - 1000 ### Lower bound of trick.
  ret <- optim(init[-N], dual, gr = NULL, H, y, N, lambda, lower, method = "BFGS")
  if(ret$convergence == 0){
    ret$alpha <- c(ret$par, -sum(ret$par * y[-N]) / y[N])
    id <- ret$alpha > 0
    ret$beta.0 <- mean(y[id] - colSums(ret$alpha[id] * y[id] * t(K)[id, id]))
  }
  ret
}

get.f.hat <- function(ret, y, K.grid){
  (colSums(ret$alpha * y * K.grid) + ret$beta.0 > 0) * 2 - 1
}

### Estimate alpha and beta.0
lambda <- c(100, 100, 100, 0.01, 0.01, 0.01)
c <- c(100, 10, 1, 100, 10, 1)
for(i in 1:6){
  eval(parse(text = paste("m.", i, " <- estimate.alpha(da.tr, y, N, lambda = ", lambda[i], ", c = ", c[i], ", sep = "\n"))\n}

### Establish grids for classification.
N.grid <- 25
x.grid <- seq(-1.5, 1.0, length = N.grid)
y.grid <- seq(-0.2, 1.3, length = N.grid)
da.grid <- cbind(rep(x.grid, N.grid), rep(y.grid, rep(N.grid, N.grid)))
K.grid.100 <- get.K(da.tr, da.grid, 100)
K.grid.10 <- get.K(da.tr, da.grid, 10)
K.grid.1 <- get.K(da.tr, da.grid, 1)
for(i in 1:6){
  eval(parse(text = paste("f.", i, " <- get.f.hat(m.", i, ", y, K.grid.", c[i], ", sep = "\n"))\n}

### Plot contour.
par(mfrow = c(2, 3))
for(i in 1:6){
  eval(parse(text = paste("f.hat <- f.", i, sep = "\n")))
  plot(da.grid[, 1], da.grid[, 2], col = f.hat + 3, pch = 19, cex = 0.3,
  xlab = "xs", ylab = "ys",
  main = substitute(paste(lambda == a, ",", c == b, sep = "\n"),
  list(a = lambda[i], b = c[i])))
  contour(x.grid, y.grid, matrix(f.hat, nrow = N.grid),
  nlevels = 1, labels = "", add = TRUE)
  points(da.tr[, 1], da.tr[, 2], pch = (y + 1) / 2 + 1)
}
Total: 80 pts (Q20: 15, Q21: 10, Q22: 10, Q23: 15, Q24: 10, Q25: 10, Q26: 10). Any reasonable solutions are acceptable.