Chapter 12 Examples

Brooke Anderson

5/7/2020

```
library(tidyverse)
```

Diabetes example

```
diabetes <- read_csv("data/diabetes.csv")

diabetes %>%
  slice(1:3)
```

```
## # A tibble: 3 x 7
##      id relwt glufast glutest steady insulin group
##   <dbl> <dbl>   <dbl>   <dbl>  <dbl>   <dbl> <dbl>
## 1     1  0.81      80     356    124      55     3
## 2     3  0.94     105     319    143     105     3
## 3     5  1         90     323    240     143     3
```

```
diabetes %>%
  dim()
```

```
## [1] 144   7
```

This dataset has observations on 144 people, all adults who were not obese. They've grouped these people into three groups (`group`): those who are "normal" (non-diabetic), those with chemical diabetes, and those with overt diabetes. Based on the paper they reference, for these subjects:

> "The variables used were age, relative weight, fasting plasma glucose, area under the plasma glucose curve for the three hour oral glucose tolerance test, area under the plasma insulin curve for the OGTT, and the steady state plasma glucose response."

Based on this, I ny guesses for the columns in the dataframe are:

- `id`: Patient unique identified
- `relwt`: Relative weight
- `glufast`: Fasting plasma glucose
- `glutest`: Area under the plasma gluocse curve for the three hour oral glucose tolerance test
- `steady`: The steady-state plasma glucose response
- `insulin`: Area under the plasma insulin curve for the OGTT
- `group`: Which of the three groups the subject belonged to (normal, chemical diabetes, or overt diabetes)

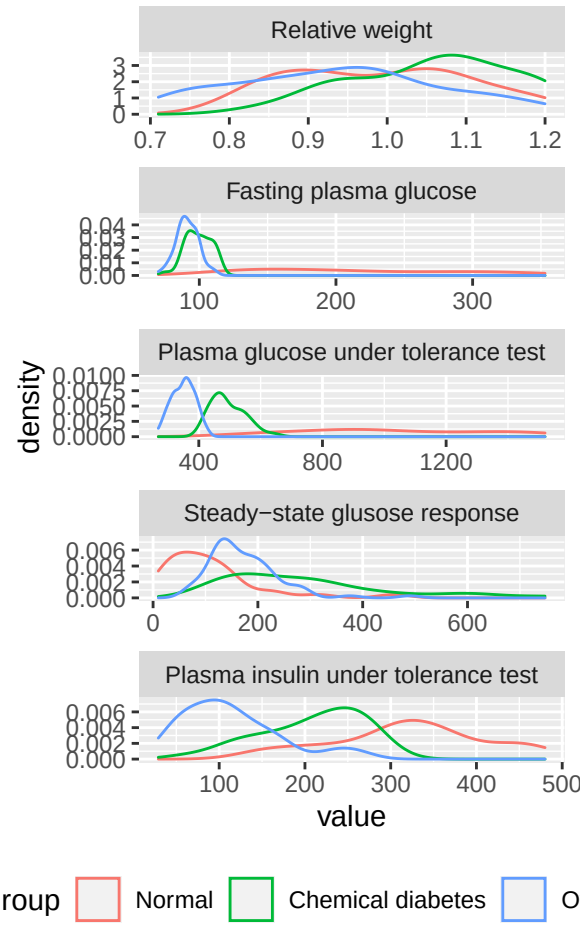
One question here is how each of the measured variables is associated with the person's group status. Here's some code that does an adaptation of Figure 12.3 in the book:

```
diabetes <- diabetes %>%
  # Convert `group` to a factor and change the level names to be clearer
  # (what they actually represent)
  mutate(group = as_factor(group),
```

```
        group = fct_recode(group,
                           "Normal" = "1",
                           "Chemical diabetes" = "2",
                           "Overt diabetes" = "3"))
# Find out how many subjects are in each group
diabetes %>%
  group_by(group) %>%
  count()
```

```
## # A tibble: 3 x 2
## # Groups:   group [3]
##   group                n
##   <fct>            <int>
## 1 Normal              32
## 2 Chemical diabetes   36
## 3 Overt diabetes      76
```

```
diabetes %>%
  # Reshape so we can plot everything with faceting
  pivot_longer(relwt:insulin) %>%
  # Convert `name` (the new column created when you reshaped the data)
  # to a factor and replace with what they really are (for better labels in
  # the graph)
  mutate(name = as_factor(name),
         name = fct_recode(name,
                           "Relative weight" = "relwt",
                           "Fasting plasma glucose" = "glufast",
                           "Plasma glucose under tolerance test" = "glutest",
                           "Steady-state glusose response" = "steady",
                           "Plasma insulin under tolerance test" = "insulin")) %>%
  ggplot(aes(x = value, color = group)) +
  geom_density() +
  facet_wrap(~ name, scales = "free", ncol = 1) +
  theme(legend.position = "bottom")
```



For this example dataset, I think we'll be trying to see if we can build a model to predict a person's group status (normal, chemical diabetes, or overt diabetes) based on these measurements.

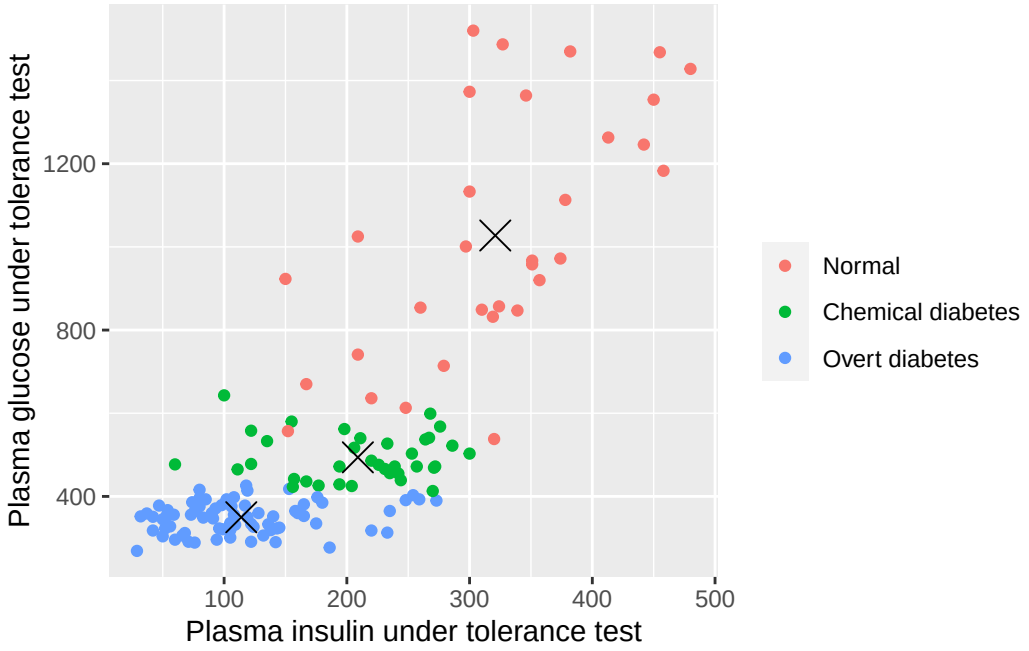
They start by trying to predict `group` based on two measurements, `insulin` and `glutest`. Here's a scatterplot of those variables, with group status shown by color:

```
# Calculate the means of each variable by group
# We'll add these as "X"s to show the group mean in the plot
group_means <- diabetes %>%
  group_by(group) %>%
  summarize_all(mean)
group_means
```

```
## # A tibble: 3 x 7
##   group               id relwt glufast glutest steady insulin
##   <fct>            <dbl> <dbl>   <dbl>   <dbl>  <dbl>   <dbl>
## 1 Normal           128.  0.992   214.    1027.   109.    321.
## 2 Chemical diabetes  91.8 1.06     99.3    494.   288     209.
## 3 Overt diabetes     39.8 0.937    91.2    350.   173.    114
```

```
# Since we're plotting from two dataframes, wait and add the "data"
# when you add each geom.
ggplot() +
  geom_point(data = diabetes,
             aes(x = insulin, y = glutest, color = group)) +
  geom_point(data = group_means,
```

```
            aes(x = insulin, y = glutest),
            shape = 4, size = 5) +
  labs(x = "Plasma insulin under tolerance test",
       y = "Plasma glucose under tolerance test",
       color = "")
```



Try using LDA to build a predictive model based on these two measurements in the data. Fit the model using `lda`:

```
library(MASS)
diabetes_lda <- lda(group ~ insulin +  glutest, data = diabetes)
```

This has a special class, `lda`. Often, the output from fitting models in R have special S4 (or S3) classes, which are essentially fancy lists.

```
diabetes_lda %>%
  class()
```

```
## [1] "lda"
```

These will often have interesting `print` methods, which will show some results from fitting the model:

```
# The `print` method runs by default when you run just the object's name
diabetes_lda
```

```
## Call:
## lda(group ~ insulin + glutest, data = diabetes)
##
## Prior probabilities of groups:
##            Normal Chemical diabetes    Overt diabetes
##         0.2222222         0.2500000         0.5277778
##
## Group means:
##                     insulin   glutest
## Normal            320.9375 1027.3750
## Chemical diabetes 208.9722  493.9444
```

```
## Overt diabetes    114.0000  349.9737
##
## Coefficients of linear discriminants:
##                   LD1         LD2
## insulin -0.004463900 -0.01591192
## glutest -0.005784238  0.00480830
##
## Proportion of trace:
##    LD1    LD2
## 0.9677 0.0323
```

As with any object in the "list" family, you might be able to find out what's in the object using `names` and `str`:

```
diabetes_lda %>%
  names()
```

```
##  [1] "prior"   "counts"  "means"   "scaling" "lev"     "svd"     "N"
##  [8] "call"    "terms"   "xlevels"
```

```
diabetes_lda %>%
  str()
```

```
## List of 10
##  $ prior  : Named num [1:3] 0.222 0.25 0.528
##   ..- attr(*, "names")= chr [1:3] "Normal" "Chemical diabetes" "Overt diabetes"
##  $ counts : Named int [1:3] 32 36 76
##   ..- attr(*, "names")= chr [1:3] "Normal" "Chemical diabetes" "Overt diabetes"
##  $ means  : num [1:3, 1:2] 321 209 114 1027 494 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:3] "Normal" "Chemical diabetes" "Overt diabetes"
##   .. ..$ : chr [1:2] "insulin" "glutest"
##  $ scaling: num [1:2, 1:2] -0.00446 -0.00578 -0.01591 0.00481
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2] "insulin" "glutest"
##   .. ..$ : chr [1:2] "LD1" "LD2"
##  $ lev    : chr [1:3] "Normal" "Chemical diabetes" "Overt diabetes"
##  $ svd    : num [1:2] 16.26 2.97
##  $ N      : int 144
##  $ call   : language lda(formula = group ~ insulin + glutest, data = diabetes)
##  $ terms  :Classes 'terms', 'formula'  language group ~ insulin + glutest
##   .. ..- attr(*, "variables")= language list(group, insulin, glutest)
##   .. ..- attr(*, "factors")= int [1:3, 1:2] 0 1 0 0 0 1
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:3] "group" "insulin" "glutest"
##   .. .. .. ..$ : chr [1:2] "insulin" "glutest"
##   .. ..- attr(*, "term.labels")= chr [1:2] "insulin" "glutest"
##   .. ..- attr(*, "order")= int [1:2] 1 1
##   .. ..- attr(*, "intercept")= int 1
##   .. ..- attr(*, "response")= int 1
##   .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. ..- attr(*, "predvars")= language list(group, insulin, glutest)
##   .. ..- attr(*, "dataClasses")= Named chr [1:3] "factor" "numeric" "numeric"
##   .. .. ..- attr(*, "names")= chr [1:3] "group" "insulin" "glutest"
##  $ xlevels: Named list()
##  - attr(*, "class")= chr "lda"
```

You can use this model object to predict new class levels. You can do this by either for the original data or with new data that includes columns with `insulin` and `glutest`. For example, here's a case of using this model to predict the subjects' groups from the original data (of course, we know what the real values are!) based on each subject's measures of `insulin` and `glutest` and on this model we just built. By default, it predicts on the data used to fit the model, so we don't need to input that:

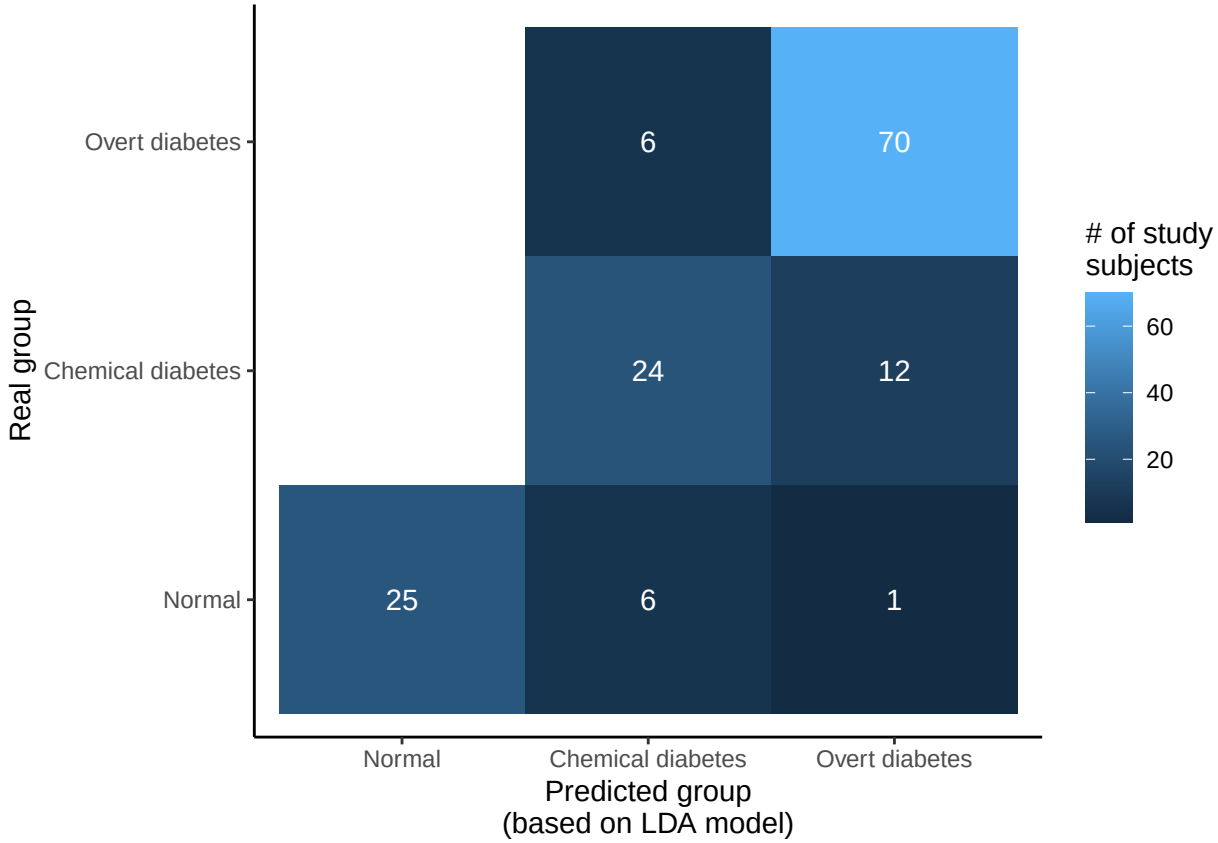
```
diabetes_lda %>%
  predict() %>%
  # The` predict` object includes several elements. Right now, we just want to get the
  # classes
  pluck("class")
```

```
##   [1] Overt diabetes    Overt diabetes    Overt diabetes    Overt diabetes
##   [5] Overt diabetes    Overt diabetes    Overt diabetes    Overt diabetes
##   [9] Overt diabetes    Overt diabetes    Overt diabetes    Overt diabetes
##  [13] Overt diabetes    Overt diabetes    Overt diabetes    Overt diabetes
##  [17] Overt diabetes    Overt diabetes    Overt diabetes    Overt diabetes
##  [21] Overt diabetes    Overt diabetes    Overt diabetes    Overt diabetes
##  [25] Overt diabetes    Chemical diabetes Overt diabetes    Overt diabetes
##  [29] Overt diabetes    Overt diabetes    Overt diabetes    Overt diabetes
##  [33] Chemical diabetes Overt diabetes    Chemical diabetes Chemical diabetes
##  [37] Overt diabetes    Chemical diabetes Overt diabetes    Overt diabetes
##  [41] Overt diabetes    Chemical diabetes Overt diabetes    Chemical diabetes
##  [45] Chemical diabetes Chemical diabetes Chemical diabetes Chemical diabetes
##  [49] Chemical diabetes Chemical diabetes Chemical diabetes Chemical diabetes
##  [53] Overt diabetes    Overt diabetes    Chemical diabetes Overt diabetes
##  [57] Normal            Chemical diabetes Normal            Normal
##  [61] Normal            Normal            Normal            Normal
##  [65] Normal            Chemical diabetes Normal            Chemical diabetes
##  [69] Chemical diabetes Normal            Normal            Normal
##  [73] Overt diabetes    Overt diabetes    Overt diabetes    Overt diabetes
##  [77] Overt diabetes    Overt diabetes    Overt diabetes    Overt diabetes
##  [81] Overt diabetes    Overt diabetes    Overt diabetes    Overt diabetes
##  [85] Chemical diabetes Overt diabetes    Overt diabetes    Overt diabetes
##  [89] Overt diabetes    Overt diabetes    Overt diabetes    Overt diabetes
##  [93] Overt diabetes    Overt diabetes    Overt diabetes    Overt diabetes
##  [97] Overt diabetes    Overt diabetes    Overt diabetes    Overt diabetes
## [101] Overt diabetes    Overt diabetes    Chemical diabetes Overt diabetes
## [105] Overt diabetes    Chemical diabetes Overt diabetes    Overt diabetes
## [109] Overt diabetes    Overt diabetes    Overt diabetes    Overt diabetes
## [113] Chemical diabetes Overt diabetes    Overt diabetes    Chemical diabetes
## [117] Chemical diabetes Chemical diabetes Chemical diabetes Overt diabetes
## [121] Chemical diabetes Chemical diabetes Chemical diabetes Chemical diabetes
## [125] Chemical diabetes Chemical diabetes Overt diabetes    Overt diabetes
## [129] Normal            Normal            Normal            Normal
## [133] Normal            Chemical diabetes Normal            Normal
## [137] Normal            Normal            Overt diabetes    Chemical diabetes
## [141] Normal            Normal            Normal            Normal
## Levels: Normal Chemical diabetes Overt diabetes
```

We will often want to compare that to the *true* classes for each study subject. Here's one way of doing that:

```
diabetes_lda %>%
  predict() %>%
  pluck("class") %>%
  as_tibble() %>%
```

```
  rename(predicted_group = value) %>%
  # Add on a column with the real group labels (by default, the predictions will
  # be in the same order as the original data, so things should line up fine here)
  mutate(true_group = diabetes$group) %>%
  # Get the numbers in each combo of true and predicted group labels
  group_by(predicted_group, true_group) %>%
  count() %>%
  # Make a plot to compare the true and predicted groups
  ggplot(aes(x = predicted_group, y = true_group, fill = n, label = n)) +
  geom_tile() +
  geom_text(color = "white") +
  labs(x = "Predicted group\n(based on LDA model)",
       y = "Real group",
       fill = "# of study\nsubjects") +
  theme_classic()
```



To help visualize the predictions, they suggest that we make a regularly-spaced grid all across the range of the original data (so it will fill up the background of the original scatterplot). Then, we'll use the LDA model to predict the class at each of these grid points. This will let us see what the model would predict at each area of the original range of data.

There's a wonderful function called `expand_grid` that lets you create a dataframe with every possible combination of two or more sets of vectors. We can use that to create a grid across the original plot area (we can use `min` and `max` on each variable to figure out what that is).

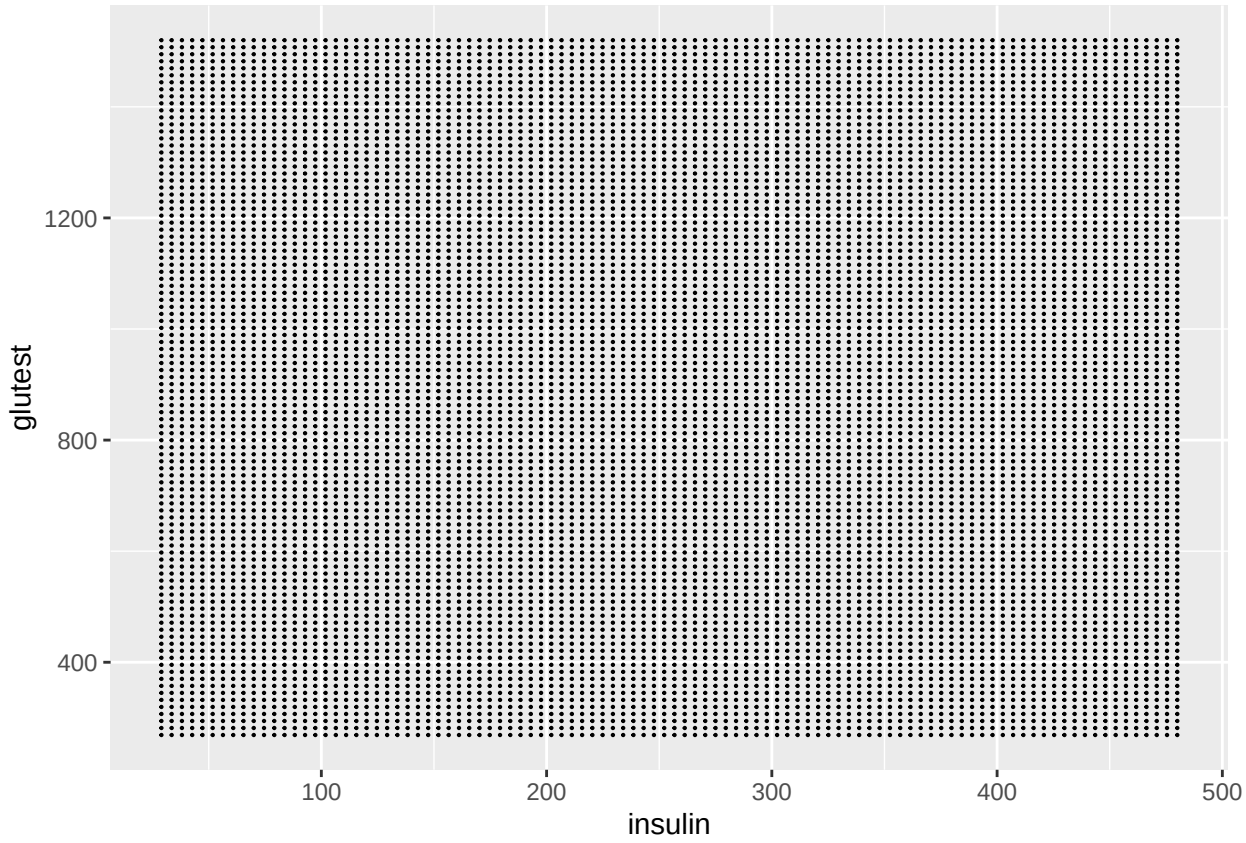
```
# Create a new dataframe with points to predict at. They'll be regularly-spaced
# points throughout the space of the original data
pred_grid <- expand_grid(insulin = seq(from = min(diabetes$insulin),
```

```
                          to = max(diabetes$insulin),
                          length.out = 100),
              glutest = seq(from = min(diabetes$glutest),
                          to = max(diabetes$glutest),
                          length.out = 100))

pred_grid %>%
  slice(1:5)
```

```
## # A tibble: 5 x 2
##   insulin glutest
##     <dbl>   <dbl>
## 1      29    269
## 2      29    282.
## 3      29    294.
## 4      29    307.
## 5      29    320.
```

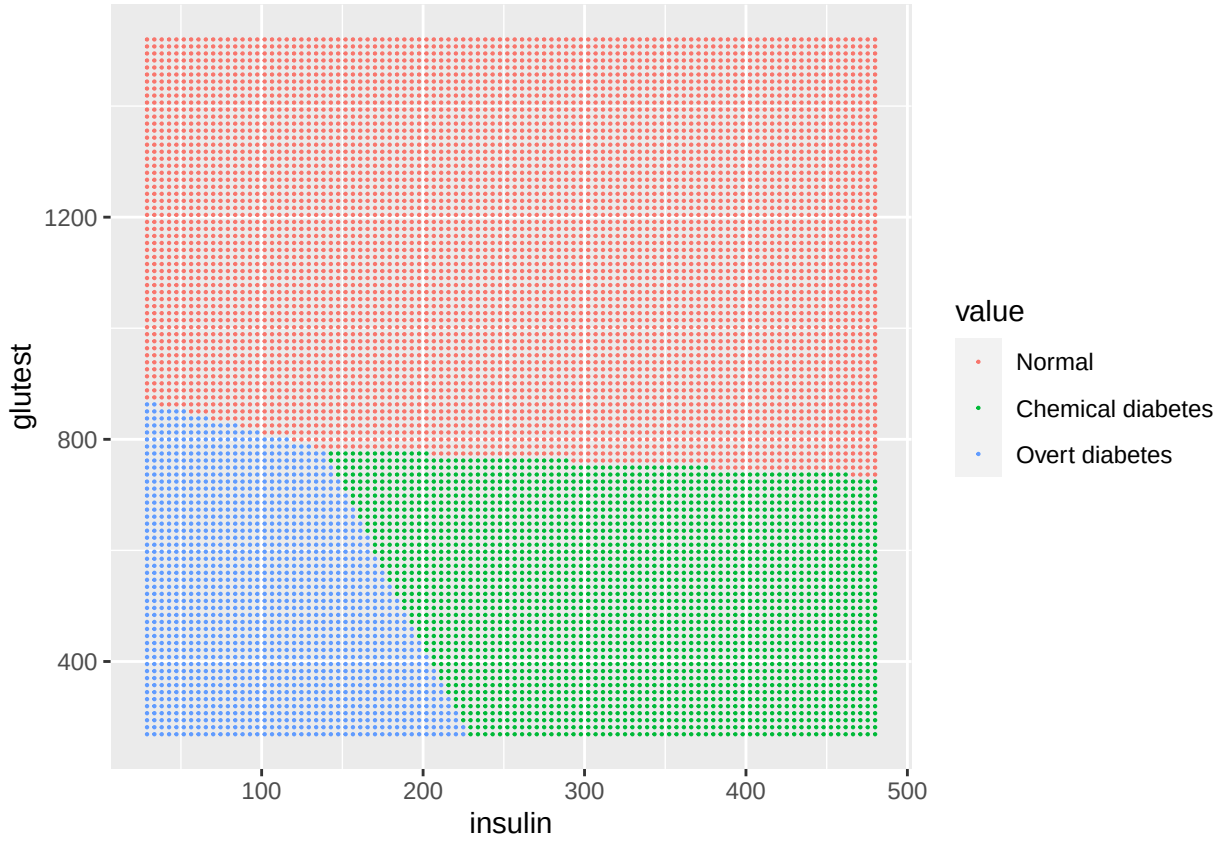
```
# This is just to show you that you do indeed get a regularly-spaced grid throughout
# the original graph area
ggplot(pred_grid, aes(x = insulin, y = glutest)) +
  geom_point(size = 0.1)
```



Now we want to add on our predictions for each of these grid points, based on the LDA model we built:

```
diabetes_lda %>%
  # Predict the groups for each point in the expanded grid
  predict(newdata = pred_grid) %>%
```

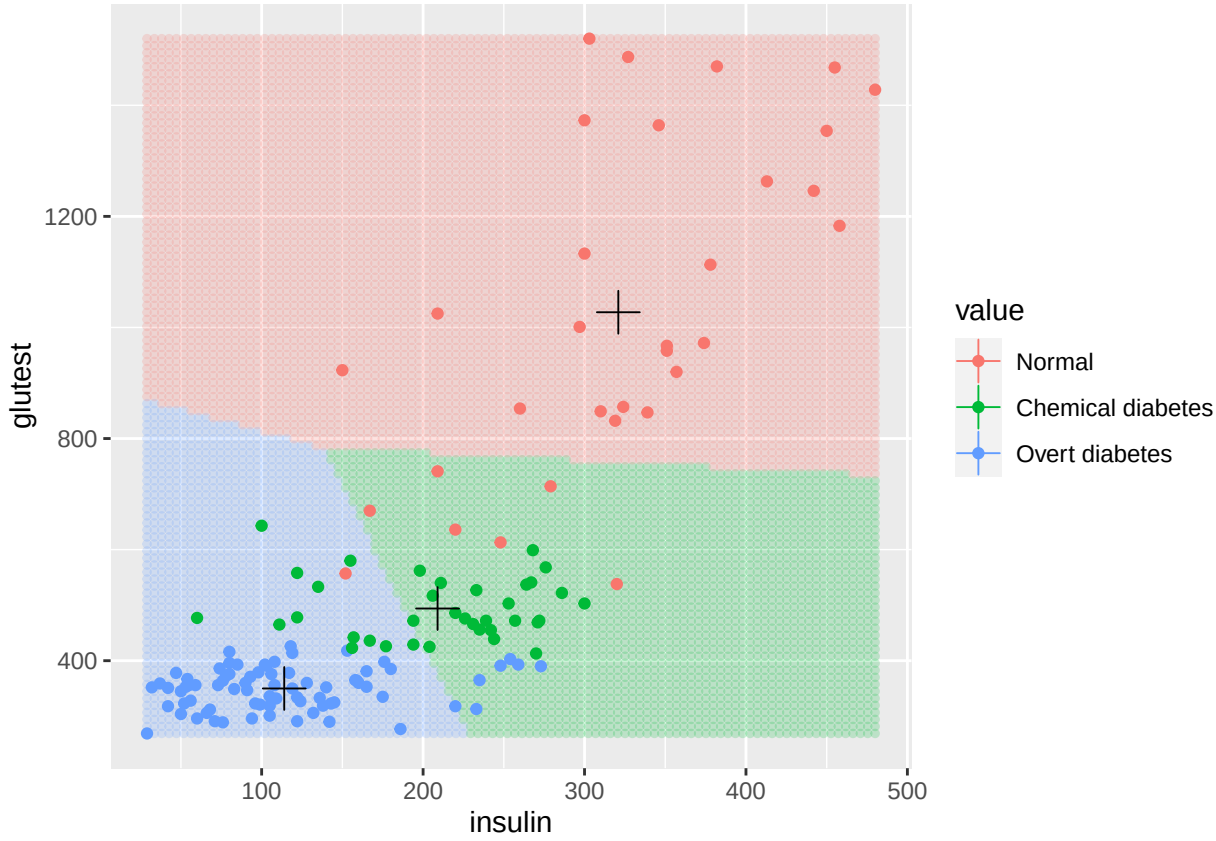
```
  pluck("class") %>%
  as_tibble() %>%
  # Add back in the values of `insulin` and `glutest` so we can plot
  # everything together
  mutate(insulin = pred_grid$insulin,
         glutest = pred_grid$glutest) %>%
  # Plot everything
  ggplot(aes(x = insulin, y = glutest, color = value)) +
  geom_point(size = 0.1)
```



Now you can add back in the original data:

```
diabetes_lda %>%
  # Predict the groups for each point in the expanded grid
  predict(newdata = pred_grid) %>%
  pluck("class") %>%
  as_tibble() %>%
  # Add back in the values of `insulin` and `glutest` so we can plot
  # everything together
  mutate(insulin = pred_grid$insulin,
         glutest = pred_grid$glutest) %>%
  # Plot the predictions for all the regions of the plot, based on the LDA model
  ggplot(aes(x = insulin, y = glutest, color = value)) +
  geom_point(size = 1, alpha = 0.2) +
  # Add in the observed data points
  geom_point(data = diabetes,
             aes(x = insulin, y = glutest, color = group)) +
```

```
  # Plot the points showing mean values of insulin and glutest for each group
  geom_point(data = group_means,
             aes(x = insulin, y = glutest, color = NULL),
             shape = 3, size = 5)
```



They discuss how this model is more likely, for observations between the group means for `insulin` and `glutest` for groups 2 and 3, to predict 3 (overt diabetes) rather than 2 (chemical diabetes). This is because the original data has a much higher proportion of observations in group 3 than group 2, which leads to the default prior weights for the LDA being skewed toward group 2.

```
# Calculate the total number of observations in each group, as well as
# the proportion of all observations in each group
diabetes %>%
  group_by(group) %>%
  count() %>%
  ungroup() %>%
  mutate(total = sum(n),
         prop = n / total)
```

```
## # A tibble: 3 x 4
##   group                 n total  prop
##   <fct>             <int> <int> <dbl>
## 1 Normal               32   144 0.222
## 2 Chemical diabetes    36   144 0.25
## 3 Overt diabetes       76   144 0.528
```

```
# Compare to the priors in the LDA model---you'll see they're just using
# the proportion of observations in the group in the data used to train the
```

```
# model
diabetes_lda %>%
  pluck("prior")
```

```
##            Normal Chemical diabetes    Overt diabetes
##         0.2222222         0.2500000         0.5277778
```

You can change re-fit an LDA model while forcing the prior weights for each group to be equal by adding weights for each observation. In essence, the aim is to "downweight" the class where we had more observations (overt obesity) and "upweight" the other so that the priors are even between the three groups:

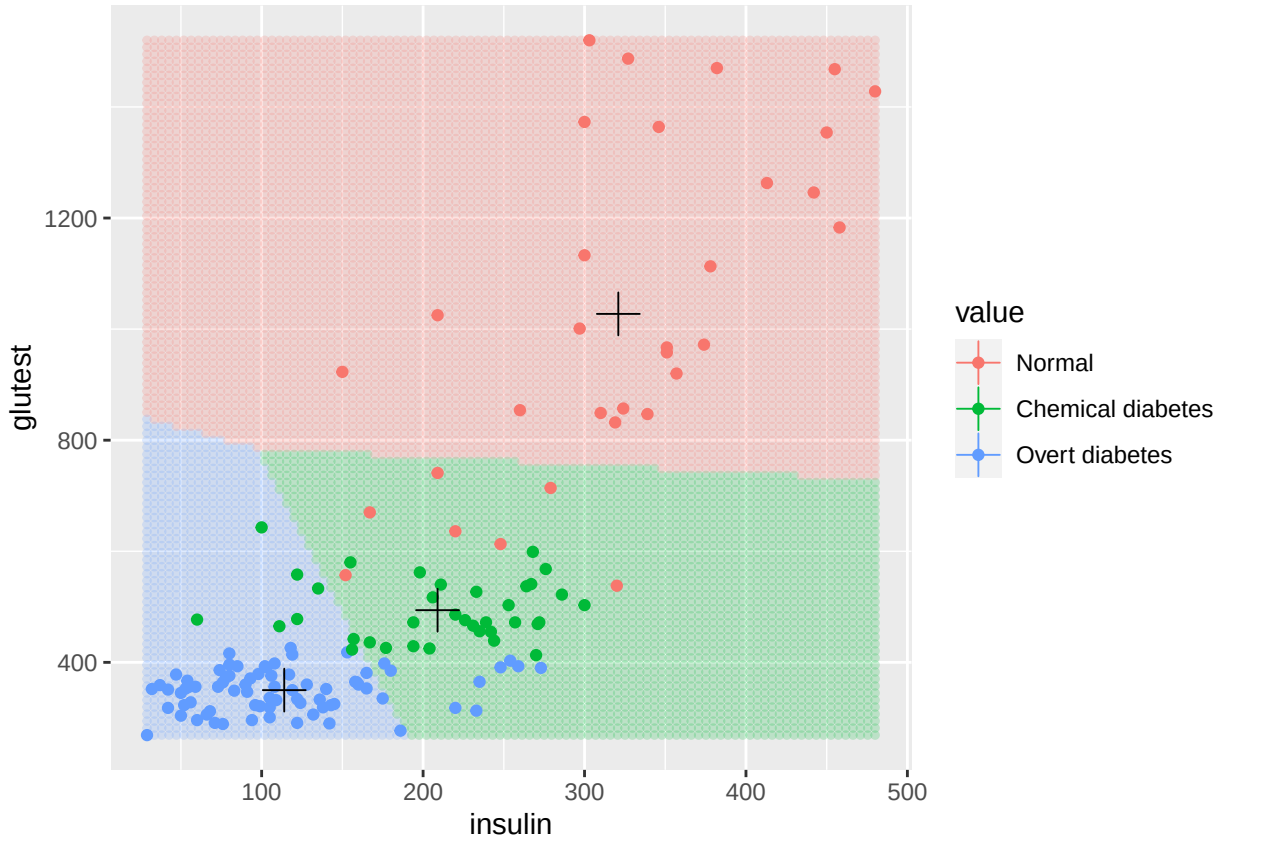
```
diabetes_up <- lda(group ~ insulin + glutest, data = diabetes,
                   prior = c(1/3, 1/3, 1/3))
diabetes_up
```

```
## Call:
## lda(group ~ insulin + glutest, data = diabetes, prior = c(1/3,
##     1/3, 1/3))
##
## Prior probabilities of groups:
##            Normal Chemical diabetes    Overt diabetes
##         0.3333333         0.3333333         0.3333333
##
## Group means:
##                     insulin   glutest
## Normal            320.9375 1027.3750
## Chemical diabetes 208.9722  493.9444
## Overt diabetes    114.0000  349.9737
##
## Coefficients of linear discriminants:
##                 LD1          LD2
## insulin -0.00376296 -0.016092098
## glutest -0.00598921  0.004550442
##
## Proportion of trace:
##   LD1   LD2
## 0.969 0.031
```

Create a plot with the predicted regions based on this new model, with a uniform prior:

```
diabetes_up %>%
  # Predict the groups for each point in the expanded grid
  predict(newdata = pred_grid) %>%
  pluck("class") %>%
  as_tibble() %>%
  # Add back in the values of `insulin` and `glutest` so we can plot
  # everything together
  mutate(insulin = pred_grid$insulin,
         glutest = pred_grid$glutest) %>%
  # Plot the predictions for all the regions of the plot, based on the LDA model
  ggplot(aes(x = insulin, y = glutest, color = value)) +
  geom_point(size = 1, alpha = 0.2) +
  # Add in the observed data points
  geom_point(data = diabetes,
             aes(x = insulin, y = glutest, color = group)) +
  # Plot the points showing mean values of insulin and glutest for each group
```

```
  geom_point(data = group_means,
             aes(x = insulin, y = glutest, color = NULL),
             shape = 3, size = 5)
```



You might want to compare how well several models perform. One measure might be the accuracy of each model in classifying the data—what percent of the observations does the model assign the correct class? Here, we'll check that in the original data used to fit the model, but just note that that method of checking models can be prone to reward models that overfit to the training data.

```
# Fit your three models and put the results of each in the three slots in a list
list(two_var_unweighted = lda(group ~ insulin + glutest,
                                        data = diabetes),
     two_var_weighted = lda(group ~ insulin + glutest,
                            data = diabetes,
                            prior = c(1/3, 1/3, 1/3)),
     five_var_unweighted = lda(group ~ relwt + glufast + glutest +
                                 steady + insulin, data = diabetes)
     ) %>%
  # Use `map` to run `predict` for each of the model objects
  map(predict) %>%
  # Pluck out the first element of the output of predict for each (the class predictions)
  map(1) %>%
  # Convert this from a vector to a one-column dataframe
  map(as_tibble) %>%
  # Add in a column with the true values of each observation (from the original data)
  map(~ mutate(., true_class = diabetes$group)) %>%
  # Stick everything together into one big data frame
```

```
  bind_rows(.id = "model") %>%
  # Check for when the prediction is right (i.e., the same as the true group
  # based on the original data)
  mutate(right = value == true_class) %>%
  # Get summaries of what percent of the time the prediction is right for each
  # model (and what percent of the time it's wrong, since that's what they
  # calculated in the book)
  group_by(model) %>%
  summarize(perc_right = round(100 * mean(right), 1),
            perc_wrong = round(100 * mean(!right), 1)) %>%
  mutate(model = fct_recode(model,
                            "Unweighted, all 5 variables" = "five_var_unweighted",
                            "Unweighted, insulin and glucose test" = "two_var_unweighted",
                            "Weighted, insulin and glucose test" = "two_var_weighted")) %>%
  knitr::kable(col.names = c("Model",
                             "Percent predicted correctly",
                             "Percent predicted incorrectly"))
```

Model	Percent predicted correctly	Percent predicted incorrectly
Unweighted, all 5 variables	90.3	9.7
Unweighted, insulin and glucose test	82.6	17.4
Weighted, insulin and glucose test	83.3	16.7

Embryonic cell state example

They give another example, where they have some gene expression data from measurements made at several different embryonic days, and they want to see if they can build a model that uses expression levels for four of these genes to predict the development stage (embryonic day).

Load the data:

```
library("Hiiragi2013")
data("x")

x %>%
  class()
```

```
## [1] "ExpressionSet"
## attr(,"package")
## [1] "Biobase"
```

This data is in an `ExpressionSet` class. You can extract the expression levels with the `exprs` accessor method:

```
embryonicDay <- x %>%
  # Extract expression levels
  exprs() %>%
  # Transpose, so that samples are on rows and gene expression levels
  # in columns
  t() %>%
  # Put the rownames in a column, because we want to keep that info (it
  # helps identify each sample). You can do this while you convert to
  # a tibble.
  as_tibble(rownames = "sample_id") %>%
```

```
  # Keep just the measurements for the four genes that you care about
  dplyr::select(sample_id, `1426642_at`, `1418765_at`, `1418864_at`, `1416564_at`) %>%
  # Use regular expressions to pull out the embryonic day from the sample IDs
  mutate(embry_day = str_extract(sample_id, "E.+"),
         embry_day = str_remove(embry_day, "\\(.+"),
         embry_day = str_squish(embry_day),
         embry_day = as_factor(embry_day))

embryonicDay %>%
  dplyr::slice(1:3)
```

```
## # A tibble: 3 x 6
##   sample_id `1426642_at` `1418765_at` `1418864_at` `1416564_at` embry_day
##   <chr>            <dbl>        <dbl>        <dbl>        <dbl> <fct>
## 1 1 E3.25           6.61        12.0          3.16         3.49 E3.25
## 2 2 E3.25           7.39         9.23         3.52         3.47 E3.25
## 3 3 E3.25           5.68        11.2          3.47         3.84 E3.25
```

At this point, we have a much smaller dataframe, with data for each sample giving the sample ID, the expression levels of four genes, and the embryonic day. We're going to try to build a model that predicts embryonic day (`embry_day`) based on the four gene expression levels.

However, it sounds like we also want to use the probe IDs for these genes to get symbols and gene names for each of them. To do that, it sounds like we can first use the `annotation` function to figure out what annotation package to use for this data:

```
x %>%
  annotation()
```

```
## [1] "mouse4302"
```

Once we know that, we can load the right package and then get the annotation dataset:

```
library("mouse4302.db")
probe_annos <- mouse4302.db %>%
  AnnotationDbi::select(keys = embryonicDay %>% colnames() %>% str_subset("^[0-9]"),
                        columns = c("SYMBOL", "GENENAME"))

probe_annos
```

```
##      PROBEID SYMBOL                                              GENENAME
## 1 1426642_at    Fn1                                           fibronectin 1
## 2 1418765_at  Timd2 T cell immunoglobulin and mucin domain containing 2
## 3 1418864_at  Gata4                                 GATA binding protein 4
## 4 1416564_at   Sox7                  SRY (sex determining region Y)-box 7
```

Now we can merge this in with out original data to get some better column names:

```
embryonicDay %<>%
  # Lengthen the data so we can join with the better symbols and replace these
  # as the column names, then we'll pivot back out to this shape
  pivot_longer(-c(sample_id, embry_day)) %>%
  full_join(probe_annos %>% dplyr::select(PROBEID, SYMBOL),
            by = c("name" = "PROBEID")) %>%
  dplyr::select(-name) %>%
  pivot_wider(names_from = SYMBOL, values_from = value)

# Yah, better column names!
```

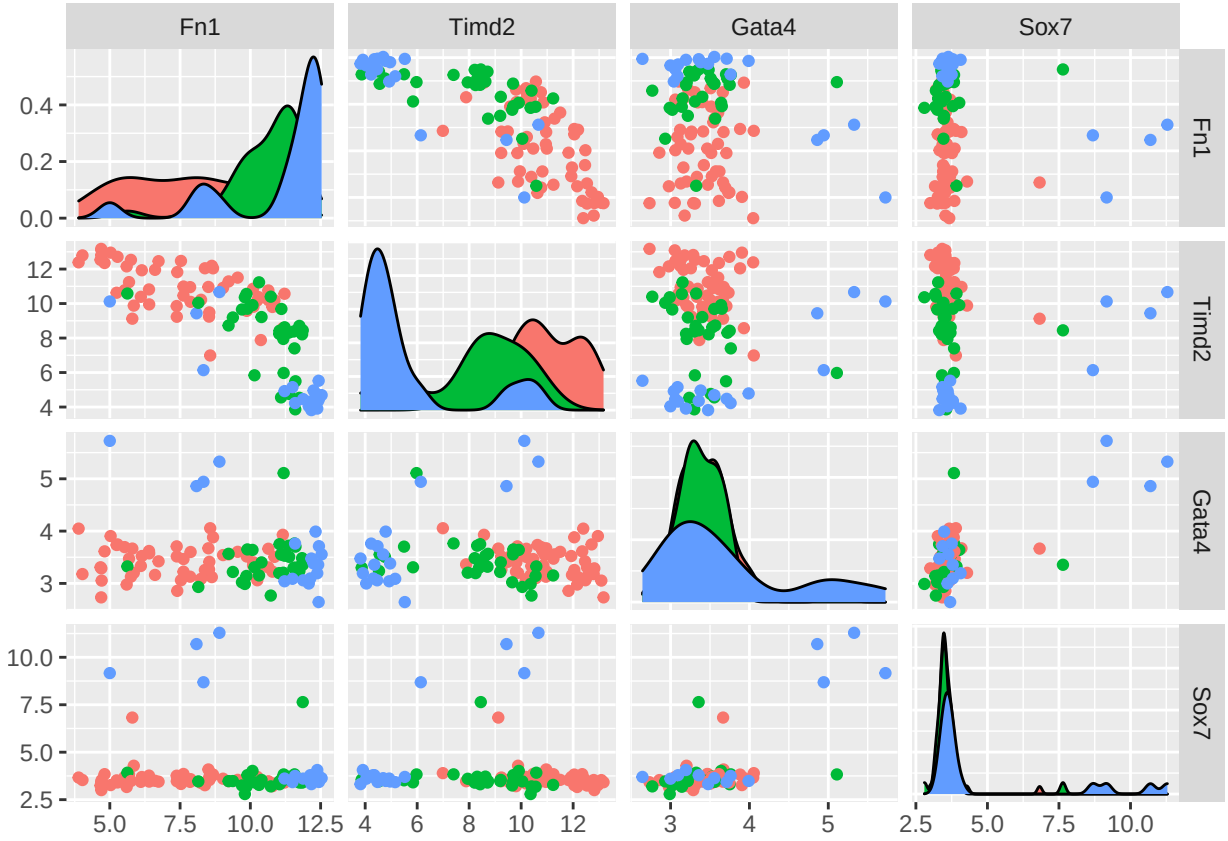
```
embryonicDay %>%
  dplyr::slice(1:3)
```

```
## # A tibble: 3 x 6
##   sample_id embry_day   Fn1 Timd2 Gata4  Sox7
##   <chr>     <fct>     <dbl> <dbl> <dbl> <dbl>
## 1 1 E3.25   E3.25      6.61 12.0   3.16  3.49
## 2 2 E3.25   E3.25      7.39  9.23  3.52  3.47
## 3 3 E3.25   E3.25      5.68 11.2   3.47  3.84
```

Investigate associations between these four gene expression levels and the embryonic day:

```
library(GGally)

embryonicDay %>%
  ggpairs(aes(color = embry_day),
          # Only show for the four gene expression columns
          columns = 3:6,
          upper = list(continuous = "points"))
```



Fit an LDA predictive model:

```
ec_lda <- lda(embry_day ~ Fn1 + Timd2 + Gata4 + Sox7,
              data = embryonicDay)
ec_lda
```

```
## Call:
## lda(embry_day ~ Fn1 + Timd2 + Gata4 + Sox7, data = embryonicDay)
##
```

```
## Prior probabilities of groups:
##     E3.25      E3.5      E4.5
## 0.5247525 0.2970297 0.1782178
##
## Group means:
##             Fn1     Timd2    Gata4     Sox7
## E3.25  7.594618 10.946221 3.414730 3.608829
## E3.5  10.580176  8.232908 3.423559 3.646317
## E4.5  11.092527  5.556256 3.733854 5.027741
##
## Coefficients of linear discriminants:
##               LD1        LD2
## Fn1   -0.09917286 -0.5625105
## Timd2  0.53482316 -0.3345209
## Gata4  0.37357898 -0.4520269
## Sox7  -0.60955649  0.3915777
##
## Proportion of trace:
##    LD1    LD2
## 0.9369 0.0631
```

Check the linear combinations LD1 and LD2 that serve as discriminating variables:

```
ec_lda %>%
  pluck("scaling") %>%
  round(1)
```

```
##        LD1  LD2
## Fn1   -0.1 -0.6
## Timd2  0.5 -0.3
## Gata4  0.4 -0.5
## Sox7  -0.6  0.4
```

Data for the exercise

The `ElemStatPackage` has been orphaned and isn't on CRAN anymore. However, it's up on GitHub, so I grabbed the data file you'll need from there. You can download it yourself at: https://github.com/cran/ElemStatLearn/blob/master/data/prostate.RData

```
load("data/prostate.RData")

prostate %>%
  head()
```

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -0.4307829
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20 -0.1625189
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636
## 6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0  0.7654678
##   train
## 1  TRUE
## 2  TRUE
## 3  TRUE
## 4  TRUE
```

```
## 5  TRUE
## 6  TRUE
```

Here's a description of the data, from the archived help files:

> "Data to examine the correlation between the level of prostate-specific antigen and a number of clinical measures in men who were about to receive a radical prostatectomy."

Here's what the variables mean:

- `lcavol`: log cancer volume
- `lweight`: log prostate weight
- `age`: in years
- `lbph`: log of the amount of benign prostatic hyperplasia
- `svi`: seminal vesicle invasion
- `lcp`: log of capsular penetration
- `gleason`: a numeric vector with the Gleason score
- `pgg45`: percent of Gleason score 4 or 5
- `lpsa`: response (the thing you are trying to predict), the level of prostate-specific antigen
- `train`: a logical vector, of whether the data was to be part of the training dataset (TRUE) or the testing one (FALSE)

So, you're trying to predict the values of `lpsa` based on the variables `lcavol` through `pgg45`.

In this exercise, they ask you to use the `glmnet` package:

```
# install.packages("glmnet")
library(glmnet)
```

For `glmnet`, I think you need to input a matrix of the predictors (`x`) and, in the same order, a vector with the outcomes (`y`). So, here's an example of building a model to predict `lpsa` from `prostate` based on all the variables between `lcavol` and `pgg45`:

```
mod_basic <- glmnet(x = prostate %>% dplyr::select(lcavol:pgg45) %>% as.matrix(),
                    y = prostate %>% pull(lpsa),
                    family = "gaussian")

mod_basic
```

```
##
## Call:  glmnet(x = prostate %>% dplyr::select(lcavol:pgg45) %>% as.matrix(),      y = prostate %>% pul
##
##    Df    %Dev  Lambda
## 1   0 0.00000 0.84340
## 2   1 0.09159 0.76850
## 3   1 0.16760 0.70020
## 4   1 0.23070 0.63800
## 5   1 0.28320 0.58130
## 6   1 0.32670 0.52970
## 7   1 0.36280 0.48260
## 8   1 0.39280 0.43980
## 9   2 0.42210 0.40070
## 10  2 0.44900 0.36510
## 11  3 0.48010 0.33270
## 12  3 0.50660 0.30310
## 13  3 0.52850 0.27620
## 14  3 0.54680 0.25160
## 15  3 0.56190 0.22930
```

```
## 16  3 0.57450 0.20890
## 17  3 0.58490 0.19040
## 18  3 0.59360 0.17350
## 19  3 0.60080 0.15800
## 20  3 0.60670 0.14400
## 21  4 0.61230 0.13120
## 22  5 0.61750 0.11960
## 23  5 0.62240 0.10890
## 24  5 0.62640 0.09926
## 25  5 0.62980 0.09044
## 26  5 0.63250 0.08240
## 27  5 0.63490 0.07508
## 28  5 0.63680 0.06841
## 29  6 0.63890 0.06234
## 30  6 0.64210 0.05680
## 31  6 0.64480 0.05175
## 32  6 0.64700 0.04715
## 33  6 0.64880 0.04297
## 34  6 0.65030 0.03915
## 35  7 0.65160 0.03567
## 36  7 0.65270 0.03250
## 37  7 0.65360 0.02961
## 38  7 0.65440 0.02698
## 39  7 0.65500 0.02459
## 40  7 0.65550 0.02240
## 41  8 0.65670 0.02041
## 42  8 0.65780 0.01860
## 43  8 0.65880 0.01695
## 44  8 0.65950 0.01544
## 45  8 0.66020 0.01407
## 46  8 0.66070 0.01282
## 47  8 0.66120 0.01168
## 48  8 0.66160 0.01064
## 49  8 0.66190 0.00970
## 50  8 0.66210 0.00884
## 51  8 0.66230 0.00805
## 52  8 0.66250 0.00734
## 53  8 0.66270 0.00668
## 54  8 0.66280 0.00609
## 55  8 0.66290 0.00555
## 56  8 0.66300 0.00506
## 57  8 0.66300 0.00461
## 58  8 0.66310 0.00420
## 59  8 0.66320 0.00382
## 60  8 0.66320 0.00348
## 61  8 0.66320 0.00318
## 62  8 0.66330 0.00289
## 63  8 0.66330 0.00264
## 64  8 0.66330 0.00240
## 65  8 0.66330 0.00219
## 66  8 0.66330 0.00199
## 67  8 0.66330 0.00182
## 68  8 0.66330 0.00166
## 69  8 0.66340 0.00151
```

```
## 70  8 0.66340 0.00138
```