Lab 2: Classification EDA, KNN, and logistic regression

PSTAT131-231

Week 2

Objectives

- Explore visualization options for categorical data
- Fit KNN classifiers and use cross-validation for selection of k
- Fit logistic regression models and visualize model outputs

Exploratory analysis for categorical data

We'll work with the `gss_cats` data from the `forcats` package to illustrate some possibilities for informative exploratory plots in the classification setting. This data comprise responses to a subset of questions on the General Social Survey in the years 2000 – 2014. We'll look at three years of data.

```
# 2007 data
gss <- filter(gss_cat, year %in% c(2006, 2010, 2014))
```

Working with factors in R

Aside from `year` (year of survey) and `tvhours` (hours spent watching TV), the remaining variables are *factors* – categorical variables with named levels. Check the structure of the `partyid` variable for political party affiliation:

```
# structure of a factor
str(gss$partyid)
```

```
##  Factor w/ 10 levels "No answer","Don't know",..: 10 6 10 10 10 9 7 10 10 9 ...
```

```
# names of levels
levels(gss$partyid)
```

```
##  [1] "No answer"          "Don't know"         "Other party"
##  [4] "Strong republican"  "Not str republican" "Ind,near rep"
##  [7] "Independent"        "Ind,near dem"       "Not str democrat"
## [10] "Strong democrat"
```

The actual values are integers that code for the factor levels. Coercing a factor to a numeric variable yields simply the codings:

```
# coerce to double
as.numeric(gss$partyid) %>% head()
```

```
## [1] 10  6 10 10 10  9
```

It's easy to tabulate the data by factor levels using `count`:

```
# tabulate party affiliations
gss %>% count(partyid) %>% mutate(prop = n/nrow(gss))
```

Let's imagine we're interested in predicting party affiliation based on other variables. We can combine the factor levels into independent, democrat, republican, and other:

```
# store original levels
old_party_levels <- levels(gss$partyid)

# combine factor levels
gss <- gss %>%
  mutate(partyid = fct_collapse(partyid,
                               'oth' = old_party_levels[1:3],
                               'rep' = old_party_levels[4:5],
                               'ind' = old_party_levels[6:8],
                               'dem' = old_party_levels[9:10]))

# tabulate and display
gss %>% count(partyid) %>% mutate(prop = n/nrow(gss)) %>% pander()
```

Your turn (1) The `relig` variable lists 16 distinct religious categories. In order of representation in the data, these are:

```
# tabulate religions
gss %>% count(relig) %>%
  arrange(desc(n))
```

Recode `relig` with one level for each of the five most represented groups and an 'all other' level. Be sure to change the `eval = F` code option to `eval = T` when you complete this part so that your result appears in your knitted document. (Hint: for more efficient code, try `fct_lump_n`.)

```
# recode religion variable
old_relig_levels <- levels(fct_lump_n(gss$relig, 5))


gss <- gss %>% mutate(relig = fct_recode(relig,
                               'Protestant' = old_relig_levels[5],
                               'Catholic' = old_relig_levels[4],
                               'None' = old_relig_levels[2],
                               'Christian' = old_relig_levels[1],
                               'Jewish' = old_relig_levels[3],
                               'all other' = old_relig_levels[6]))
```

Now check the reported income variable `rincome`. Notice that this factor has an inherent *ordering*, aside from some levels representing missing data of various types.

```
# last level is 'out of order'
levels(gss$rincome)
```

Let's reorder the factor so that the 'not applicable' level appears together with 'no answer', 'don't know', and 'refused'.

```
# reorder reported income
gss <- gss %>%
  mutate(rincome = fct_relevel(rincome, 'Not applicable'))
levels(gss$rincome)
```

Now let's group those categories together.

```
# group non-ordered categories together
adj_rincome_levels <- levels(gss$rincome)
```

```r
gss <- gss %>%
  mutate(rincome = fct_other(rincome, drop = adj_rincome_levels[1:4]))
```

Tidying

For exploratory analysis, we'll consider political party affiliation as our variable of interest and investiage its relationship to marital status, age, race, reported income, and religion (after the reorderings above).

```r
# select variables of interest
gss <- gss %>% select(-tvhours, -denom)
```

As you saw above, there is a nonresponse category for reported income. There is also one for marital status. We'll filter out those observations before going ahead, but first let's check to see if there's a pattern to nonresponse. There aren't that many observations for which the respondent gave 'No answer' to marital status.

```r
# tabulate marital statuses
gss %>% count(marital) %>% pander()
```

But there are quite a few nonresponses for reported income.

```r
# tabulate rincome
gss %>% count(rincome) %>% pander()
```

Your turn (2) Make a summary table that helps you investigate whether there's any apparent pattern to nonresponse in the variable of interest. Are respondents with certain party affiliations not reporting their income on the survey more often than others? Avoid examining raw counts, as the numbers of respondents of each party affiliation are not equal. (Hint: consider finding the proportion of respondents of each party affiliation that didn't report their income.)

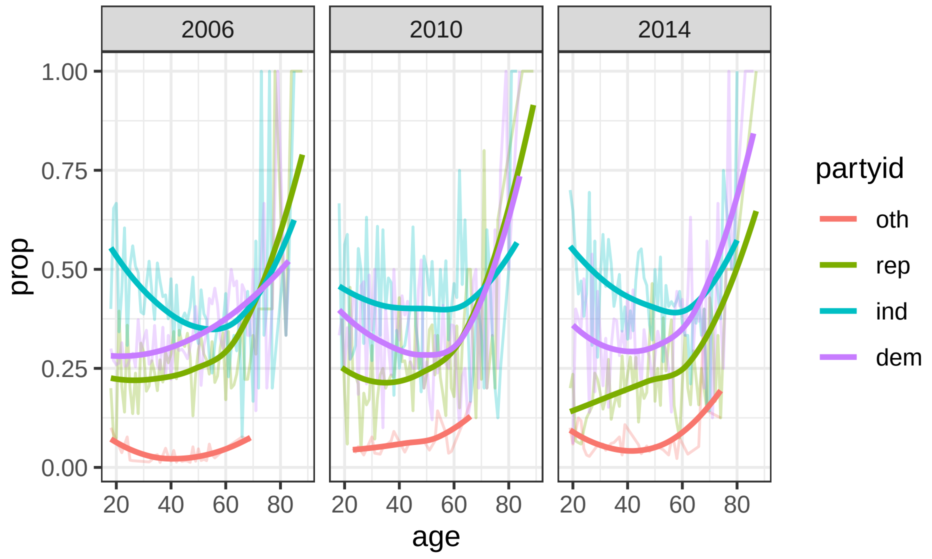
```r
# tabulate income nonresponse by party
gss_income <- gss %>%
  filter(rincome == "Other") %>%
  select(partyid)

gss_income %>%
  count(partyid) %>%
  mutate(prop= n/sum(n)) %>%
  pander()
```

Your turn (3) Now remove the observations with nonresponse for income or marital status with a `filter` command, and remove missing values with `na.omit`. Be sure to change the `eval = F` code option to `eval = T` when you complete this part so that your result appears in your knitted document. (*Hint*: use your solution from HW1.)

```r
# drop missing values
gss <- gss %>%
  filter(rincome != "Other" & marital != 'No answer') %>% na.omit()
head(gss)
```

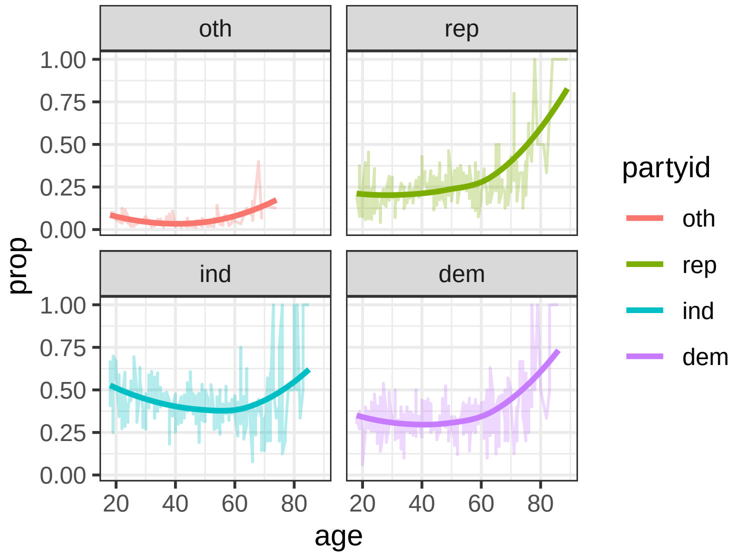
Visualizing factors

Let's first consider how to visualize the relationship between the party affiliation (a factor) and age (a continuous variable). The following plot shows the proportion of respondents of each political party against age by year. Does it look like there's a relationship between age and party affiliation? If so, does the relationship look different between different years?

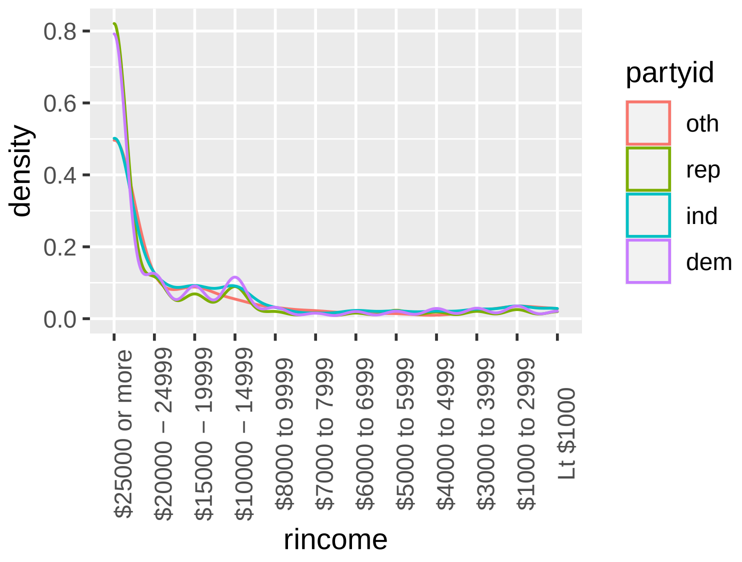


Why is there so much variation in the proportions? Check the sample sizes by running the code chain up to the `mutate` command before the result is piped into `ggplot`.

Your turn (4) Modify the plot above so that no distinction is made between years: plot the proportion of respondents in each party against age.

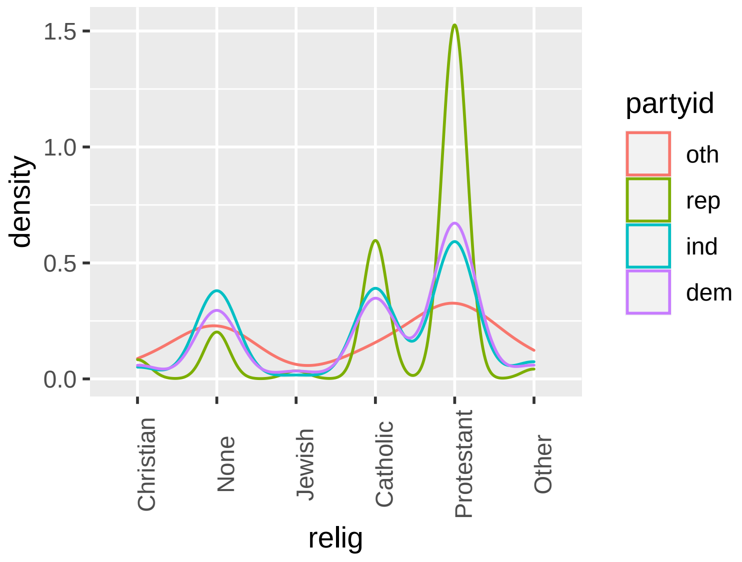


Now let's make an exploratory plot to visualize the relationship between party affiliation and reported income. Does it look like income distribution varies between respondents of different party affiliations?



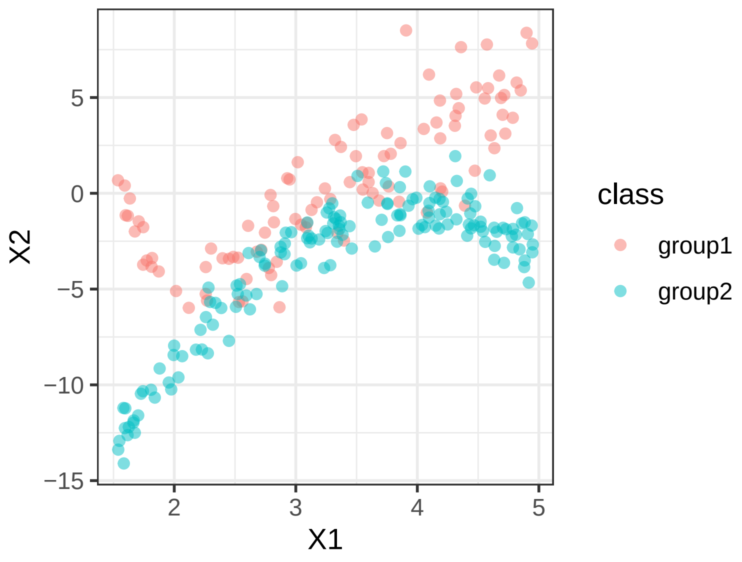
Your turn (5) Choose one of the other variables – marital status, race, or religion – and construct a plot to explore its relationship with party affiliation.

```
# plot codes go here
gss %>% ggplot(aes(x = relig)) +
  geom_density(aes(color = partyid, group = partyid)) +
  theme(axis.text.x = element_text(angle = 90))
```



KNN and Logistic regression

Here you will fit KNN and logistic regression classifiers to simulated data from lecture and compare their performance. The simulated data is shown below.



KNN

To fit the KNN classifier using `knn()`, the covariates need to be stored as a matrix and the class labels as a factor.

```r
# features
x_mx <- sim_data %>%
  select(X1, X2) %>%
  as.matrix()

# response
y <- sim_data %>% pull(class)
```

Let's start with $k = 10$.

```r
# 10-nearest neighbors
y_hat <- knn(train = x_mx, test = x_mx, cl = y, k = 10)

# misclassifications
table(y, y_hat)
```

To assess the misclassification error by leave-one-out cross validation, use `knn.cv()`:

```r
# leave-one-out CV predictions
y_hat_cv <- knn.cv(train = x_mx, cl = y, k = 10)
```

Your turn (6) Follow the example above and compute the misclassification errors using `y_hat_cv`.

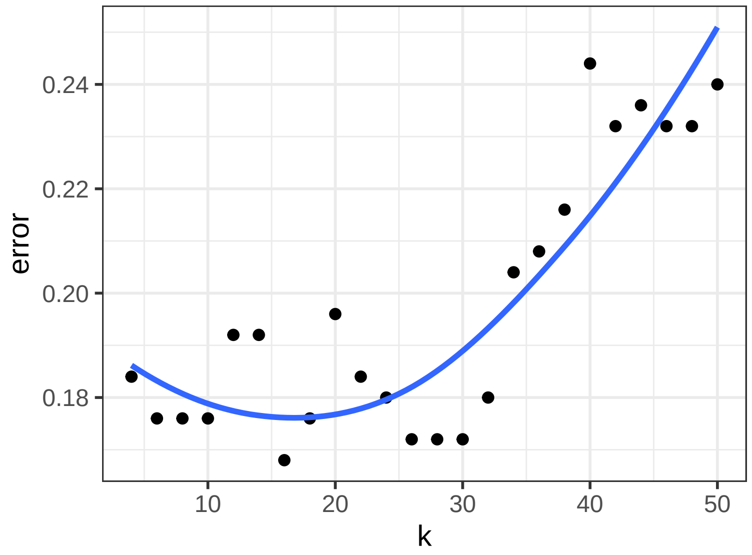
```r
# compute loocv misclassification error
table(y, y_hat_cv)
```

To select k, we can repeat this process for several values.

```r
# leave one out cv with `knn.cv()`
cv_out <- tibble(k = seq_range(5:50, n = 20, pretty = T)) %>%
  mutate(loocv_preds = map(k, ~ knn.cv(x_mx, y, .x)),
         class = map(k, ~ y)) %>%
  mutate(misclass = map2(loocv_preds, class,
                         ~ as.numeric(.x) - as.numeric(.y))) %>%
  mutate(error = map(misclass, ~ mean(abs(.x))))

# error rates for each k
cv_errors <- cv_out %>%
  select(k, error) %>%
  unnest(everything())
```

Your turn (7) Plot the cross validation estimates of misclassification error against k. Which k is best? Re-train the classifier using your selected k and cross-tabulate predicted class with true class. Be sure to change the `eval = F` code option to `eval = T` when you complete this part so that your result appears in your knitted document.

```
# plot errors against k
cv_errors %>%
ggplot(aes(x = k, y = error)) +
  geom_point() +
  geom_smooth(se = F, span = 1.5) +
  theme_bw()
```

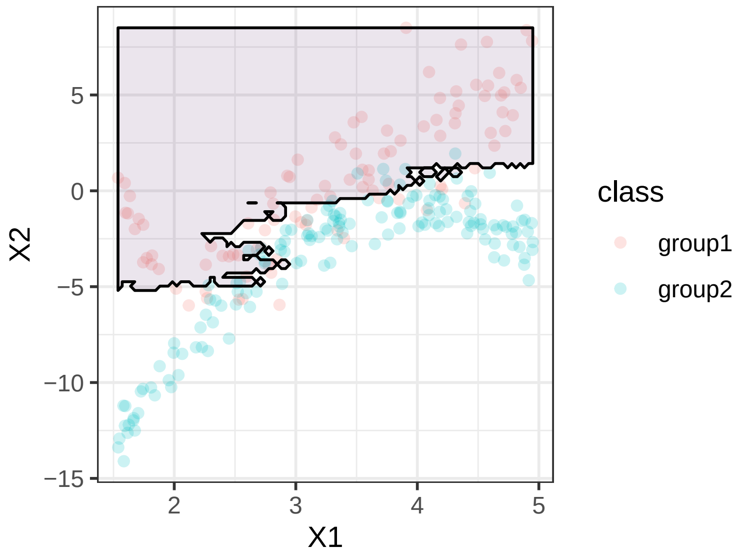


```
# select k
best_k <- cv_errors$k[which.min(cv_errors$error)]

# re-train
y_hat_knn <- knn(train = x_mx, test = x_mx, cl = y, k = best_k)

# cross-tabulate
errors_knn <- table(y, y_hat_knn)
```

To visualize the decision boundary, make sure you've stored your selection as `best_k` and run the code chunk below (only shown in .Rmd file). Be sure to change the `eval = F` code option to `eval = T` when you complete this part so that your result appears in your knitted document.

```
# generate grid
pred_df <- sim_data %>%
  data_grid(X1 = seq_range(X1, n = 100),
            X2 = seq_range(X2, n = 100))
```

```
# compute predictions
preds_knn <- knn(train = x_mx,
                 test = as.matrix(pred_df),
                 cl = y, k = best_k)
pred_df <- pred_df %>% bind_cols(pred_knn = as.numeric(preds_knn))

# visualize
sim_data %>%
  ggplot(aes(x = X1, y = X2)) +
  geom_point(aes(color = class), alpha = 0.2) +
  geom_contour_filled(aes(z = pred_knn),
                      data = pred_df, alpha = 0.1,
                      bins = 1, color = 'black',
                      show.legend = F) +
  theme_bw()
```



Logistic regression

To fit a logistic regression model to the data, use `glm()` with `family = 'binomial'`:

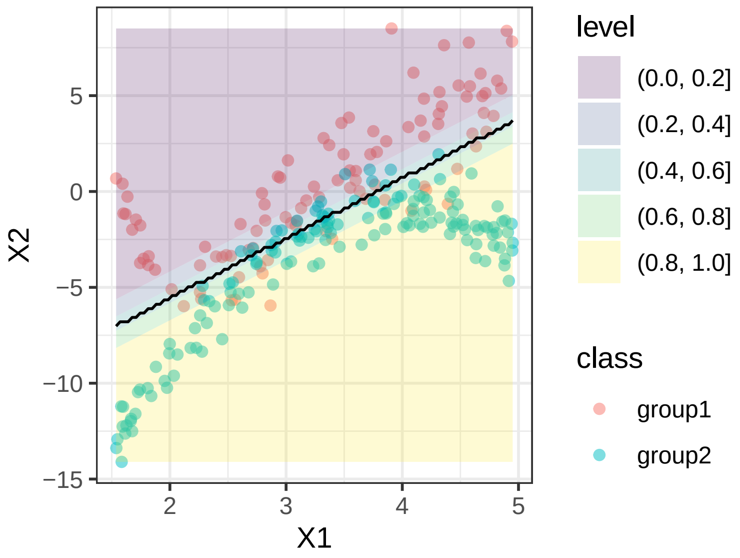
```
# fit logistic regression linear in x1 and x2
fit_glm <- glm(class ~ ., family = 'binomial', data = sim_data)
```

To obtain estimated classes, use `predict()` to obtain estimated probabilities and threshhold them to obtain a class label:

```
# compute estimated probabilities
p_hat_glm <- predict(fit_glm, sim_data, type = 'response')

# bayes classifier
y_hat_glm <- factor(p_hat_glm > 0.5, labels = c('No', 'Yes'))

# errors
error_glm <- table(y = y, y_hat_glm)
```

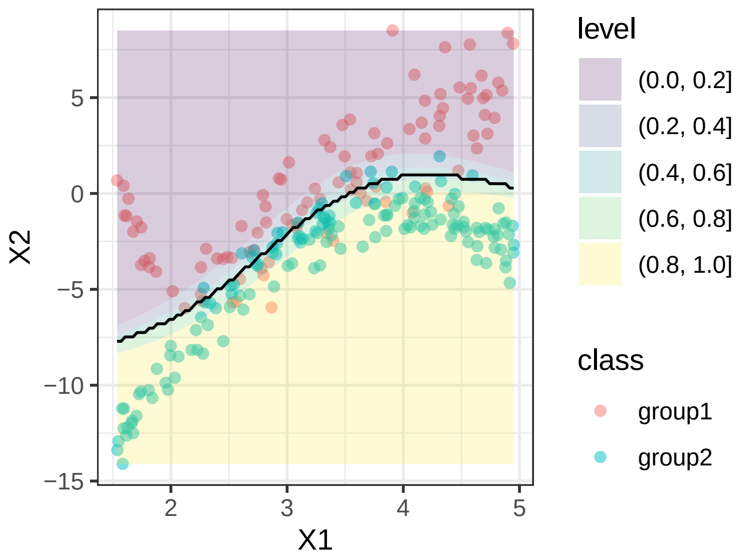
To visualize the decision boundary, run the code chunk below this line in the .Rmd file. Be sure to change the `eval = F` code option to `eval = T` when you complete this part so that your result appears in your knitted document.

```
preds_glm <- predict(fit_glm, pred_df, type = 'response')
pred_df <- pred_df %>%
  bind_cols(p_hat_glm = preds_glm,
            y_hat_glm = factor(preds_glm > 0.5,
                               labels = levels(sim_data$class)))

sim_data %>%
  ggplot(aes(x = X1, y = X2)) +
  geom_point(aes(color = class), alpha = 0.5) +
  geom_contour_filled(aes(z = p_hat_glm),
                      data = pred_df,
                      alpha = 0.2, bins = 5) +
  geom_contour(aes(z = as.numeric(y_hat_glm)),
               data = pred_df,
               bins = 1, show.legend = F,
               color = 'black') +
  theme_bw()
```



Notice that the decision boundary is linear, which doesn't capture the apparent boundary in the data scatter.

Your turn (8) Modify the logistic regression model to fit a polynomial in X1 with interactions with X2: use `class ~ poly(X1, 2)*X2`. (The asterisk in a formula specifies both first-order and product (interaction) terms, *e.g.* `X1*X2` means $\beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2$.) Plot the decision boundary and compute misclassification errors. Be sure to change the `eval = F` code option to `eval = T` when you complete this part so that your result appears in your knitted document.



level
(0.0, 0.2]
(0.2, 0.4]
(0.4, 0.6]
(0.6, 0.8]
(0.8, 1.0]
class
group1
group2
X2
5
0
−5
−10
−15
X1
2
3
4
5