Code for slide deck on cross-validation and penalised regression

Benjamin Skov Kaas-Hansen

11/02/2020

Contents

Note

- I've used quite some functions from `dplyr` (e.g., `%>%`, `mutate` and `select`) and `tidyr` (`gather`). If you don't understand what's going on in a "chain" (operations linked by `%>%`), try to the chain sequentially (first only line 1, then lines 1 and 2, then lines 1-3, etc.) and see what happens.

Setup

```
packages <- c("plyr", "tidyr", "broom", "boot", "glmnet", "selectiveInference", "MASS", "tidyverse")
for (p in packages)
    library(p, character.only = TRUE)
knitr::opts_chunk$set(fig.align = "center")

theme_set(theme_minimal()+
            theme(axis.title = element_text(size = 11),
                  axis.text = element_text(size = 10),
                  strip.text = element_text(size = 11)))

# Little helper to get the glmnet coefficients in nice tidy format
pretty_coefs <- function(coefs) { # coefs: the output from coef(fit_object, s = [value])
    enframe(coefs[, 1], "predictor", "coefficient") %>%
        filter(coefficient != 0) %>%
        arrange(desc(abs(coefficient)))
}
```

Cross-validation Pima

```
data(PimaIndiansDiabetes2, package = "mlbench")
glimpse(PimaIndiansDiabetes2)
```

```
## Rows: 768
## Columns: 9
## $ pregnant <dbl> 6, 1, 8, 1, 0, 5, 3, 10, 2, 8, 4, 10, 10, 1, 5, 7, 0, 7, 1...
## $ glucose  <dbl> 148, 85, 183, 89, 137, 116, 78, 115, 197, 125, 110, 168, 1...
## $ pressure <dbl> 72, 66, 64, 66, 40, 74, 50, NA, 70, 96, 92, 74, 80, 60, 72...
## $ triceps  <dbl> 35, 29, NA, 23, 35, NA, 32, NA, 45, NA, NA, NA, NA, 23, 19...
## $ insulin  <dbl> NA, NA, NA, 94, 168, NA, 88, NA, 543, NA, NA, NA, NA, 846,...
## $ mass     <dbl> 33.6, 26.6, 23.3, 28.1, 43.1, 25.6, 31.0, 35.3, 30.5, NA, ...
## $ pedigree <dbl> 0.627, 0.351, 0.672, 0.167, 2.288, 0.201, 0.248, 0.134, 0....
## $ age      <dbl> 50, 31, 32, 21, 33, 30, 26, 29, 53, 54, 30, 34, 57, 59, 51...
## $ diabetes <fct> pos, neg, pos, neg, pos, neg, pos, neg, pos, pos, neg, pos...
```

```
summary(PimaIndiansDiabetes2)
```

```
##     pregnant         glucose         pressure         triceps
##  Min.   : 0.000   Min.   : 44.0   Min.   : 24.00   Min.   : 7.00
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 64.00   1st Qu.:22.00
##  Median : 3.000   Median :117.0   Median : 72.00   Median :29.00
##  Mean   : 3.845   Mean   :121.7   Mean   : 72.41   Mean   :29.15
##  3rd Qu.: 6.000   3rd Qu.:141.0   3rd Qu.: 80.00   3rd Qu.:36.00
##  Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00
##                   NA's   :5       NA's   :35       NA's   :227
##     insulin           mass          pedigree           age        diabetes
##  Min.   : 14.00   Min.   :18.20   Min.   :0.0780   Min.   :21.00   neg:500
##  1st Qu.: 76.25   1st Qu.:27.50   1st Qu.:0.2437   1st Qu.:24.00   pos:268
##  Median :125.00   Median :32.30   Median :0.3725   Median :29.00
##  Mean   :155.55   Mean   :32.46   Mean   :0.4719   Mean   :33.24
##  3rd Qu.:190.00   3rd Qu.:36.60   3rd Qu.:0.6262   3rd Qu.:41.00
##  Max.   :846.00   Max.   :67.10   Max.   :2.4200   Max.   :81.00
##  NA's   :374      NA's   :11
```

Homegrown

Very simplistic implementation but it illustrates the logic.

```
set.seed(42)
pima <- na.exclude(PimaIndiansDiabetes2) %>%
    mutate(cv_fold = sample(1:10, n(), replace = TRUE))
table(pima$cv_fold) # fairly equal distribution
```

```
##
##  1  2  3  4  5  6  7  8  9 10
## 44 40 29 39 39 47 29 26 46 53
```

```
err_cv <- c()
for (i in unique(pima$cv_fold)) {
    train <- filter(pima, cv_fold != i)
    mod <- glm(diabetes ~ age + mass + insulin + pregnant, data = train, family = binomial)

    val <- filter(pima, cv_fold == i)
    y_pred <- predict(mod, newdata = val, type = "response")
    y_true <- as.numeric(val$diabetes) - 1 # bring binary factor to 0/1 scale
    err <- mean(abs(y_true - y_pred) > 0.5)
    err_cv <- c(err_cv, err)
}
err_cv
```

```
##  [1] 0.2500000 0.2564103 0.2608696 0.2641509 0.3333333 0.2500000 0.3076923
##  [8] 0.3103448 0.3448276 0.2978723
```

```
mean(err_cv)
```

```
## [1] 0.2875501
```

Using packages

The approach you generally want to do (no need to re-invent the wheel).

```
binary_pred_cost <- function(y_true, y_pred) {
    mean(abs(y_true - y_pred) > 0.5)
}

pima_glm <- glm(diabetes ~ age + mass + insulin + pregnant, data = pima, family = binomial)

pima_loo <- cv.glm(pima, pima_glm, cost = binary_pred_cost)
pima_loo$delta # 1st is raw estimate, 2nd is bias-corrected
```

```
## [1] 0.2857143 0.2844908
```

```
pima_cv1 <- cv.glm(pima, pima_glm, cost = binary_pred_cost, K = 10)
pima_cv1$delta # 1st is raw estimate, 2nd is bias-corrected
```

```
## [1] 0.2806122 0.2801112
```

The `modelr`-package has some powerful functionalities for CV, LOOCV and bootstrapping. It's more involved but offers some powerful and sophisticated functionalities.

Lasso regression example: biopsies from breast cancer patients

Lasso regression

Look at the data (in Danish: vi skal tegne, før vi må regne)

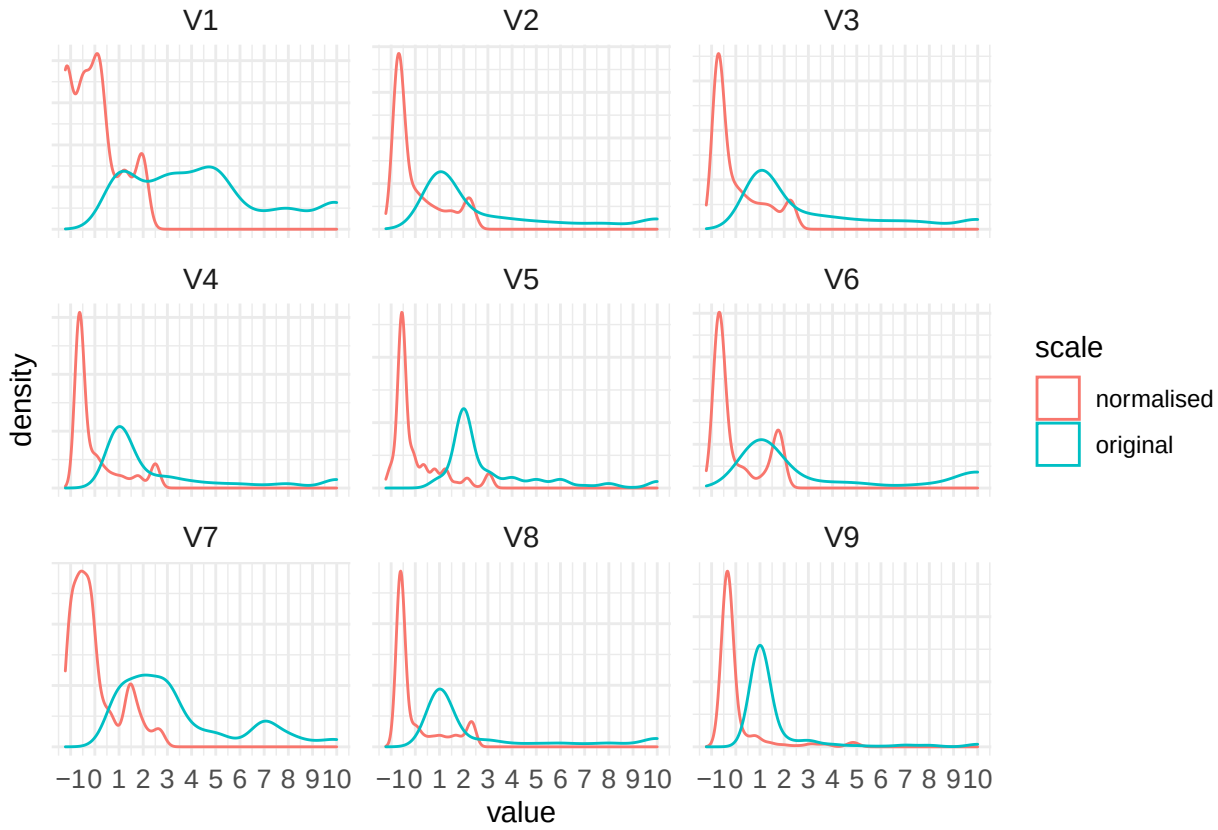
```
data(biopsy)
summary(biopsy) # NA's in V6; mean varies across variables but anyway somewhere around 2 and 4
```

```
##       ID                 V1              V2              V3        
##  Length:699         Min.   : 1.000   Min.   : 1.000   Min.   : 1.000  
##  Class :character   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000  
##  Mode  :character   Median : 4.000   Median : 1.000   Median : 1.000  
##                     Mean   : 4.418   Mean   : 3.134   Mean   : 3.207  
##                     3rd Qu.: 6.000   3rd Qu.: 5.000   3rd Qu.: 5.000  
##                     Max.   :10.000   Max.   :10.000   Max.   :10.000  
##                                                                       
##        V4              V5              V6              V7        
##  Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000  
##  1st Qu.: 1.000   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 2.000  
##  Median : 1.000   Median : 2.000   Median : 1.000   Median : 3.000  
##  Mean   : 2.807   Mean   : 3.216   Mean   : 3.545   Mean   : 3.438  
##  3rd Qu.: 4.000   3rd Qu.: 4.000   3rd Qu.: 6.000   3rd Qu.: 5.000  
##  Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000  
##                                    NA's   :16                       
##        V8              V9              class    
##  Min.   : 1.000   Min.   : 1.000   benign   :458  
```

```
##  1st Qu.: 1.000   1st Qu.: 1.000   malignant:241
##  Median : 1.000   Median : 1.000
##  Mean   : 2.867   Mean   : 1.589
##  3rd Qu.: 4.000   3rd Qu.: 1.000
##  Max.   :10.000   Max.   :10.000
##
```

```
biopsy_complete <- na.exclude(biopsy) # remove rows with any missing value
biopsy_predictors <- biopsy_complete %>%
    select(-ID, -class) %>%
    scale() # note attributes "remember" normlisation factors; useful for transforming test set
```

```
bind_rows(gather(as_tibble(biopsy_predictors), var, value) %>%
            mutate(scale = "normalised"),
          gather(select(biopsy_complete, -ID, -class), var, value) %>%
            mutate(scale = "original")) %>%
    ggplot(aes(x = value, colour = scale)) +
        geom_density(position = "identity") +
        scale_x_continuous(breaks = -2:10) +
        facet_wrap(~ var, scales = "free_y") +
        theme(axis.text.y = element_blank())
```

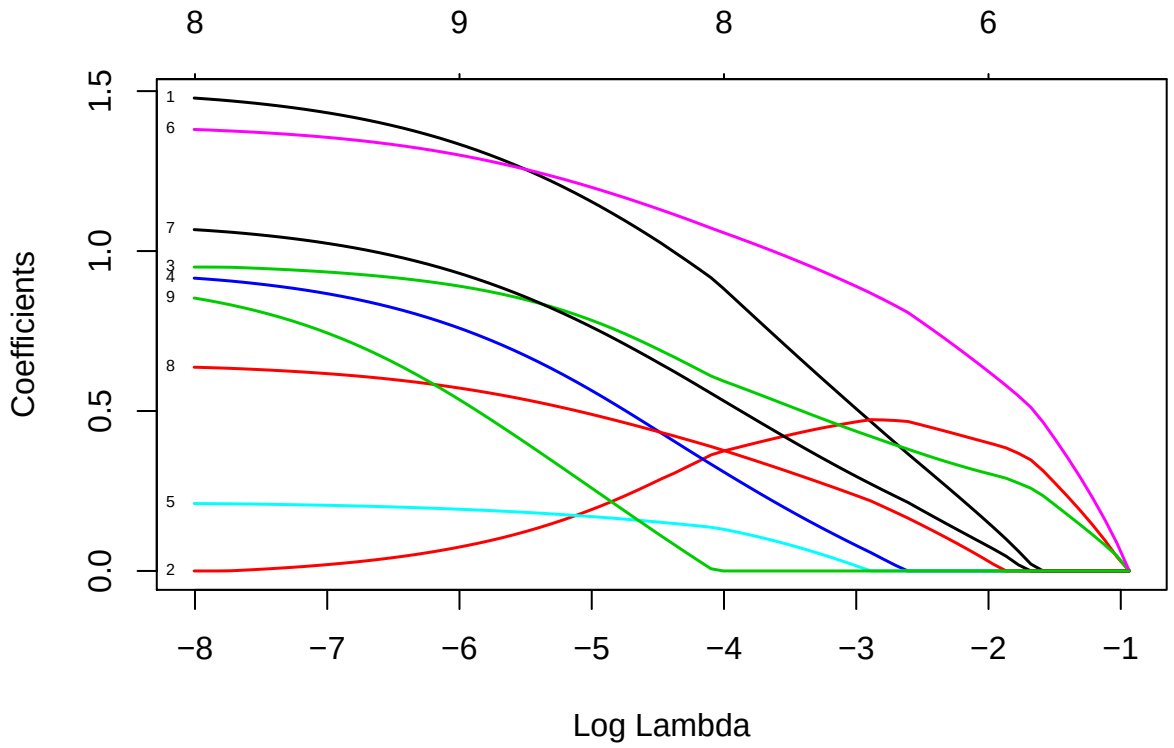


Fit model

```
lasso_logreg <- glmnet(biopsy_predictors, biopsy_complete$class, family = "binomial")
```

Coefficient profile plot (built-in: ugly but easy). Below I've made a custom fit function using ggplot2 in case anyone's interested

```
plot(lasso_logreg, xvar = "lambda", label = TRUE, lwd = 1.5)
```



And let's take a look at the non-zero coefficients (so the ones selected by the lasso regression). I've a custom function (`pretty_coefs`, see top of document) that returns these coefficients in a nice format. In general, if you find yourself doing the same (or almost the same) thing more than once, it's usually a good idea to pack that *thing* into a function. This gives shorter code, which is easier to read, maintain and debug.

```
pretty_coefs(coef(lasso_logreg, s = exp(-1.5)))
```

```
## # A tibble: 4 x 2
##   predictor   coefficient
##   <chr>             <dbl>
## 1 (Intercept)      -0.686
## 2 V6                0.414
## 3 V2                0.276
## 4 V3                0.207
```

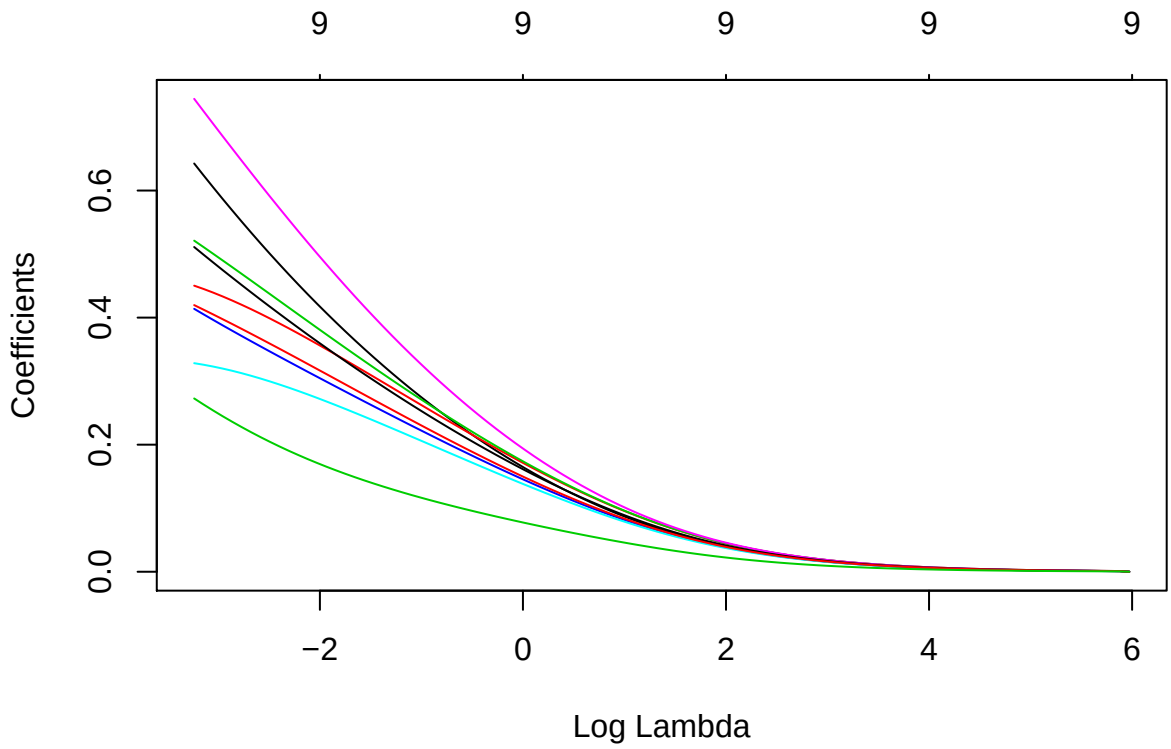
Alternative way to find the non-zero coefficients (this is basically what I've packed into the `pretty_coefs` function.)

```
lasso_logreg_coefs <- coef(lasso_logreg, s = exp(-1.5))
lasso_logreg_coefs[which(lasso_logreg_coefs != 0), 1]
```

```
## (Intercept)          V2          V3          V6
##  -0.6857851   0.2759895   0.2072426   0.4141450
```

Ridge and elastic net models

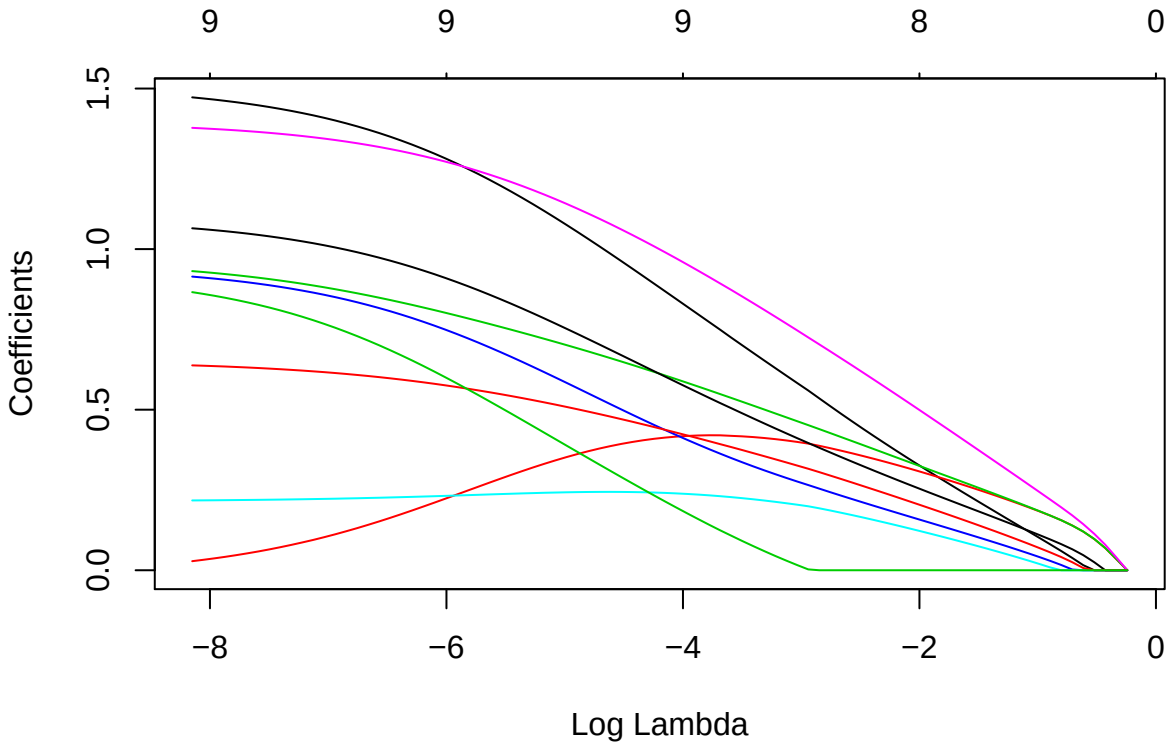
```
ridge_logreg <- update(lasso_logreg, alpha = 0)
plot(ridge_logreg, xvar = "lambda")
```




```
pretty_coefs(coef(ridge_logreg, s = exp(2))) # all shrunk but no real ranking
```

```
## # A tibble: 10 x 2
##    predictor   coefficient
##    <chr>             <dbl>
##  1 (Intercept)     -0.630
##  2 V6               0.0457
##  3 V3               0.0448
##  4 V2               0.0447
##  5 V7               0.0414
##  6 V1               0.0395
##  7 V8               0.0391
##  8 V4               0.0383
##  9 V5               0.0373
## 10 V9               0.0224
```

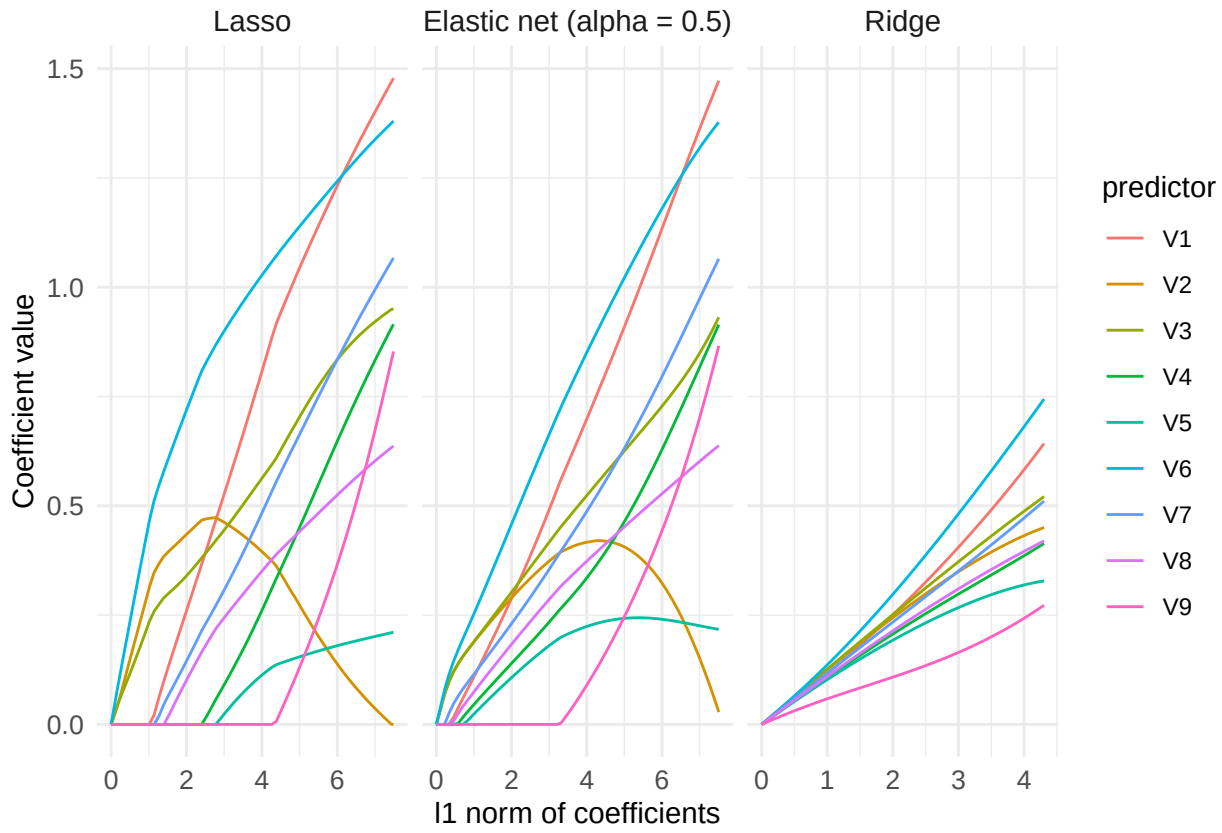
```
elastic_logreg <- update(lasso_logreg, alpha = 0.5)
plot(elastic_logreg, xvar = "lambda")
```




```
pretty_coefs(coef(elastic_logreg, s = exp(-1.2))) # all shrunk AND "ranking" (different coefficient size
```

```
## # A tibble: 9 x 2
##   predictor   coefficient
##   <chr>             <dbl>
## 1 (Intercept)     -0.719
## 2 V6               0.298
## 3 V3               0.215
## 4 V2               0.213
## 5 V1               0.148
## 6 V7               0.141
## 7 V8               0.0996
## 8 V4               0.0675
## 9 V5               0.0435
```

```
ldply(list(lasso = lasso_logreg, ridge = ridge_logreg, elastic = elastic_logreg),
      function(.) data.frame(as.matrix(t(.$beta)), x = apply(abs(.$beta), 2, sum)),
      .id = "mod") %>%
    gather(predictor, value, -x, -mod) %>%
    mutate(mod = factor(mod, levels = c("lasso", "elastic", "ridge"),
                        labels = c("Lasso", "Elastic net (alpha = 0.5)", "Ridge"))) %>%
    ggplot(aes(x, value, colour = predictor)) +
        geom_line() +
        labs(x = "l1 norm of coefficients", y = "Coefficient value") +
        facet_wrap(~ mod, scales = "free_x")
```



Over-fitting biopsy

It's a little data set but let's try and do a 80%/20% split into training and test sets.

```
set.seed(42) # reproducible stochastic code
train_idx <- runif(nrow(biopsy_complete)) <= 0.8 # not exactly indices
biopsy_train <- filter(biopsy_complete, train_idx)
biopsy_test <- filter(biopsy_complete, !train_idx)
all_equal(biopsy_complete, bind_rows(biopsy_train, biopsy_test)) # sanity check
```

```
## [1] TRUE
```

```
# Alternative with actual indices (mostly a matter of taste)
set.seed(42)
train_idx <- which(runif(nrow(biopsy_complete)) <= 0.8)
biopsy_train <- slice(biopsy_complete, train_idx)
biopsy_test <- slice(biopsy_complete, -train_idx)
all_equal(biopsy_complete, bind_rows(biopsy_train, biopsy_test)) # sanity check
```

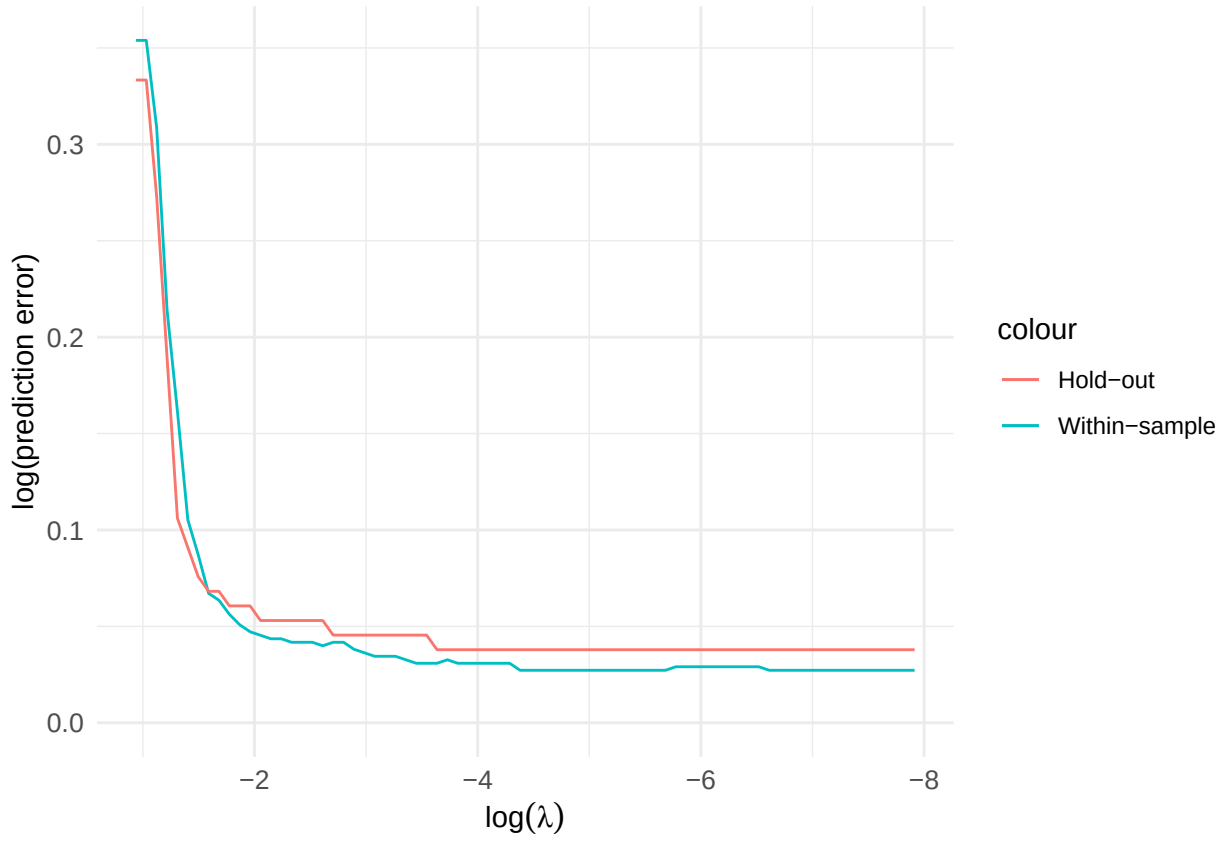
```
## [1] TRUE
```

```
# Normalise predictors and put them in matrix format
predictors_train <- biopsy_train %>%
    select(-ID, -class) %>%
    scale()
```

```r
# Use normalisation factors from training predictors to scale the test predictors
predictors_test <- biopsy_test %>%
    select(-ID, -class) %>%
    scale(center = attr(predictors_train, "scaled:center"),
          scale = attr(predictors_train, "scaled:scale"))

# Train model
lasso_biopsy_train <- glmnet(predictors_train, biopsy_train$class, family = "binomial")

# Prediction error in train and test sets
D_train <- predict(lasso_biopsy_train, predictors_train, type = "class") %>%
    apply(2, function(.) mean(. != biopsy_train$class))

D_test <- predict(lasso_biopsy_train, predictors_test, type = "class") %>%
    apply(2, function(.) mean(. != biopsy_test$class))

tibble(D_test = D_test[which.min(D_train)],
       D_train = D_train[which.min(D_train)],
       D_diff_abs = scales::percent(D_test - D_train),
       D_diff_rel = scales::percent((D_test - D_train) / D_train, big.mark = ","))
```

```
## # A tibble: 1 x 4
##   D_test D_train D_diff_abs D_diff_rel
##    <dbl>   <dbl> <chr>      <chr>
## 1 0.0379  0.0272 1%         39%
```

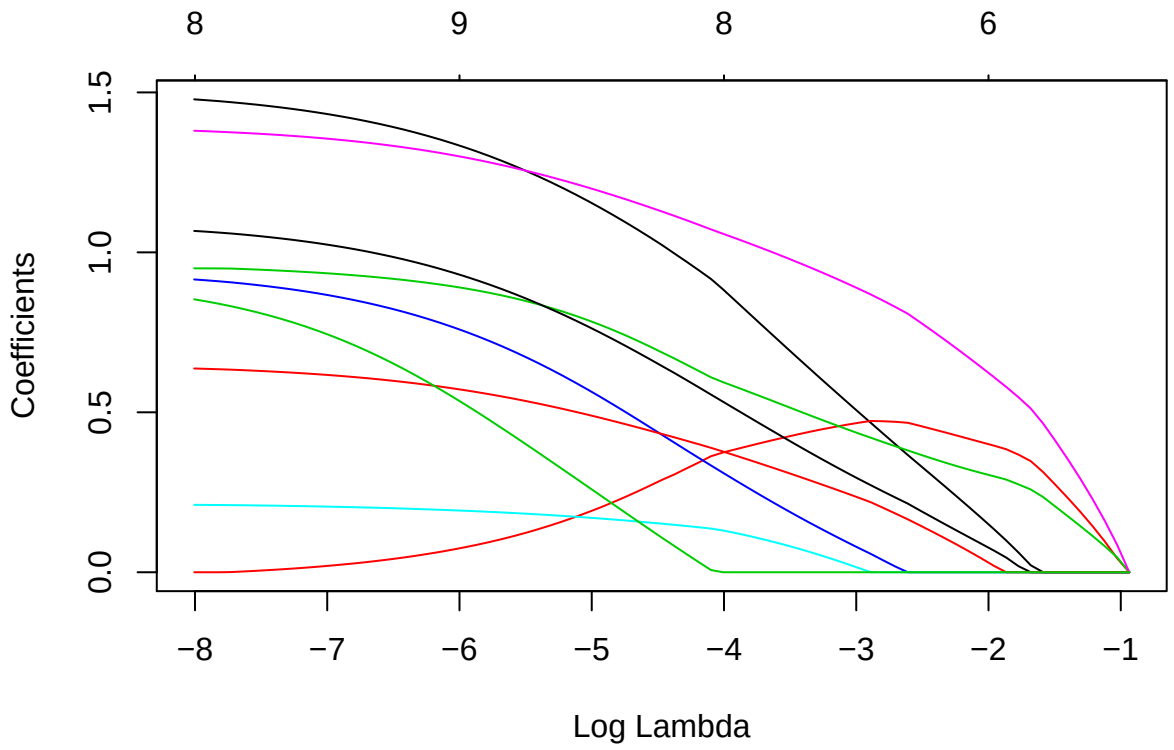
```r
ggplot() + # could define the (common) x-axis variable already here, but simpler to do it for each geom
    coord_cartesian(ylim = c(0, NA)) +
    geom_line(aes(x = log(lasso_biopsy_train$lambda), y = D_train, colour = "Within-sample")) +
    geom_line(aes(x = log(lasso_biopsy_train$lambda), y = D_test, colour = "Hold-out")) +
    labs(y = "log(prediction error)", x = expression(log(lambda))) +
    scale_x_reverse()
```



Delassoing (NB! This is an active area of research, so don't rely too much one this)

The purpose of selective inference is to try and obtain reliable confidence intervals (based on correct p values) for associations that have already been selected using lasso regression. The `coef` function requires more arguments than normally because it needs to interpolate between the grid values of $lambda$ for which is was trained. Also note that you need to divide the `lambda` values with the number of observations (see `Examples` in `?fixedLassoInf`).

```
plot(lasso_logreg, xvar = "lambda")
```



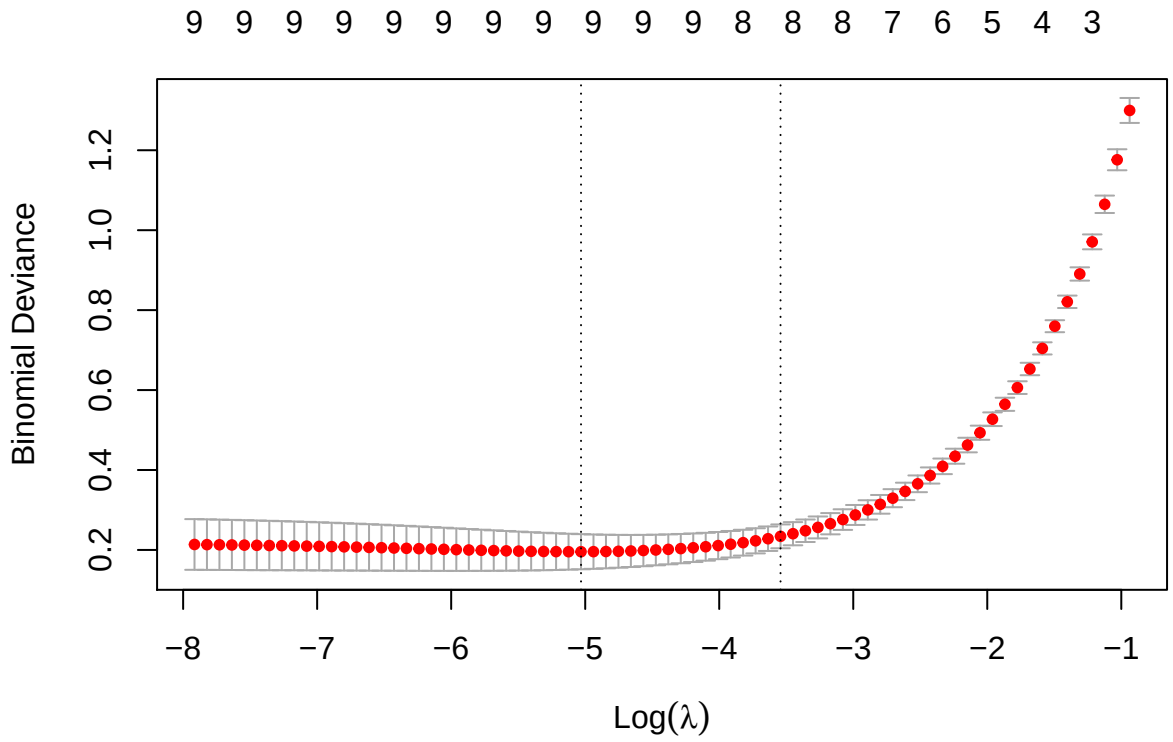

```
lambda <- exp(-2)
beta <- coef(lasso_logreg, x = biopsy_predictors, y = biopsy_complete$class,
             s = lambda/nrow(biopsy_complete), exact = TRUE)
delasso_fit <- fixedLassoInf(biopsy_predictors, as.numeric(biopsy_complete$class) - 1, beta,
                             lambda, "binomial", alpha = 0.05)
delasso_fit
```

```
##
## Call:
## fixedLassoInf(x = biopsy_predictors, y = as.numeric(biopsy_complete$class) -
##     1, beta = beta, lambda = lambda, family = "binomial", alpha = 0.05)
##
## Testing results at lambda = 0.135, with alpha = 0.050
##
##  Var  Coef Z-score P-value LowConfPt UpConfPt LowTailArea UpTailArea
##    1 1.507   3.821   0.000     0.720    2.282       0.024      0.025
##    3 0.951   1.844   0.065    -0.331    2.067       0.025      0.024
##    4 0.945   2.746   0.006     0.221    1.625       0.024      0.025
##    5 0.214   0.621   0.536    -1.897    0.859       0.025      0.024
##    6 1.396   4.119   0.000     0.723    2.064       0.024      0.024
##    7 1.093   2.656   0.008     0.223    1.902       0.025      0.025
##    8 0.649   1.918   0.056    -0.183    1.336       0.024      0.024
##    9 0.924   1.654   0.105    -0.610    2.024       0.025      0.024
##
## Note: coefficients shown are full regression coefficients
```

Cross-validation

Use cross-validation to find best λ value. We found quite clear over-fitting above. Let's try to remedy this with cross-validation. We see that we need a fairly high penalty before we any real predictor selection, and that hurts the out-of-sample prediction performance.

```
# Use the training set only to fit the CV model
lasso_logreg_cv <- cv.glmnet(predictors_train, biopsy_train$class, family = "binomial", nfolds = 10)
plot(lasso_logreg_cv) # note that the y axis does NOT start at 0, which can be misleading
```

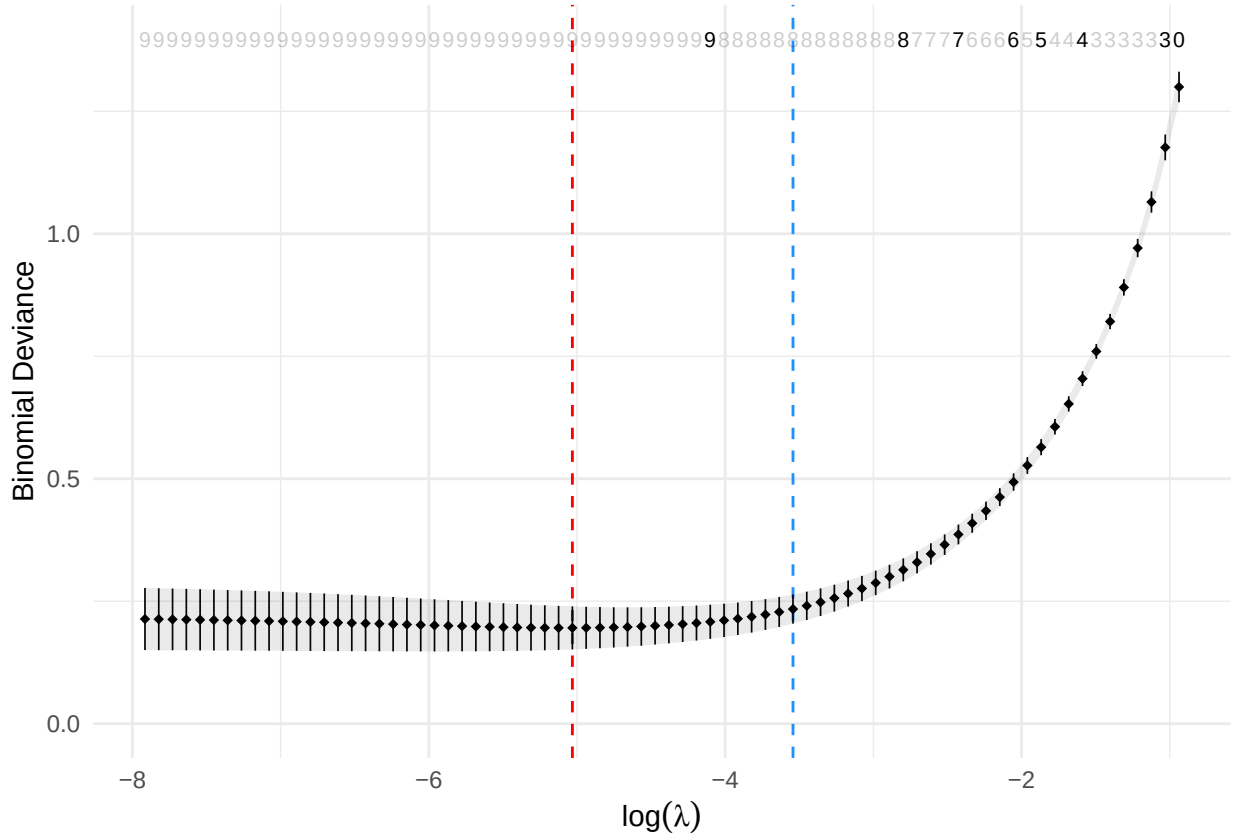



```
with(lasso_logreg_cv, data.frame(lambda.min, lambda.1se))
```

```
##    lambda.min lambda.1se
## 1 0.006531203 0.02893729
```

```
# With ggplot2 (more control and prettier) -- there are two helper functions to tune the appearance of
# the labels showing the number of predictors (you can choose which to use in the geom_text line)
fade_text <- function(x, alpha = 0.2) {
    ifelse(paste(x) == lag(paste(x), default = ""), alpha, 1)
}
every_n <- function(x, n = 5) {
    seq_along(x) %% n == 0
}
with(lasso_logreg_cv, tibble(lambda, cvm, cvup, cvlo, nzero)) %>%
    ggplot(aes(x = log(lambda))) +
        geom_vline(xintercept = log(lasso_logreg_cv$lambda.min), linetype = 2, size = 0.5, colour = "red
        geom_vline(xintercept = log(lasso_logreg_cv$lambda.1se), linetype = 2, size = 0.5, colour = "dod
        geom_linerange(aes(ymin = cvlo, ymax = cvup), size = 0.3) +
        geom_ribbon(aes(ymin = cvlo, ymax = cvup), alpha = 0.1) +
```

```
  geom_point(aes(y = cvm), shape = 18, size = 1.5) +
  geom_text(aes(y = max(cvup) * 1.05, label = nzero, alpha = fade_text(nzero)), size = 8 / ggplot
  scale_alpha_identity() +
  coord_cartesian(ylim = c(0, NA)) + # force y axis start at 0 and end where the data do
  labs(x = expression(log(lambda)), y = lasso_logreg_cv$name) +
  theme_minimal()
```



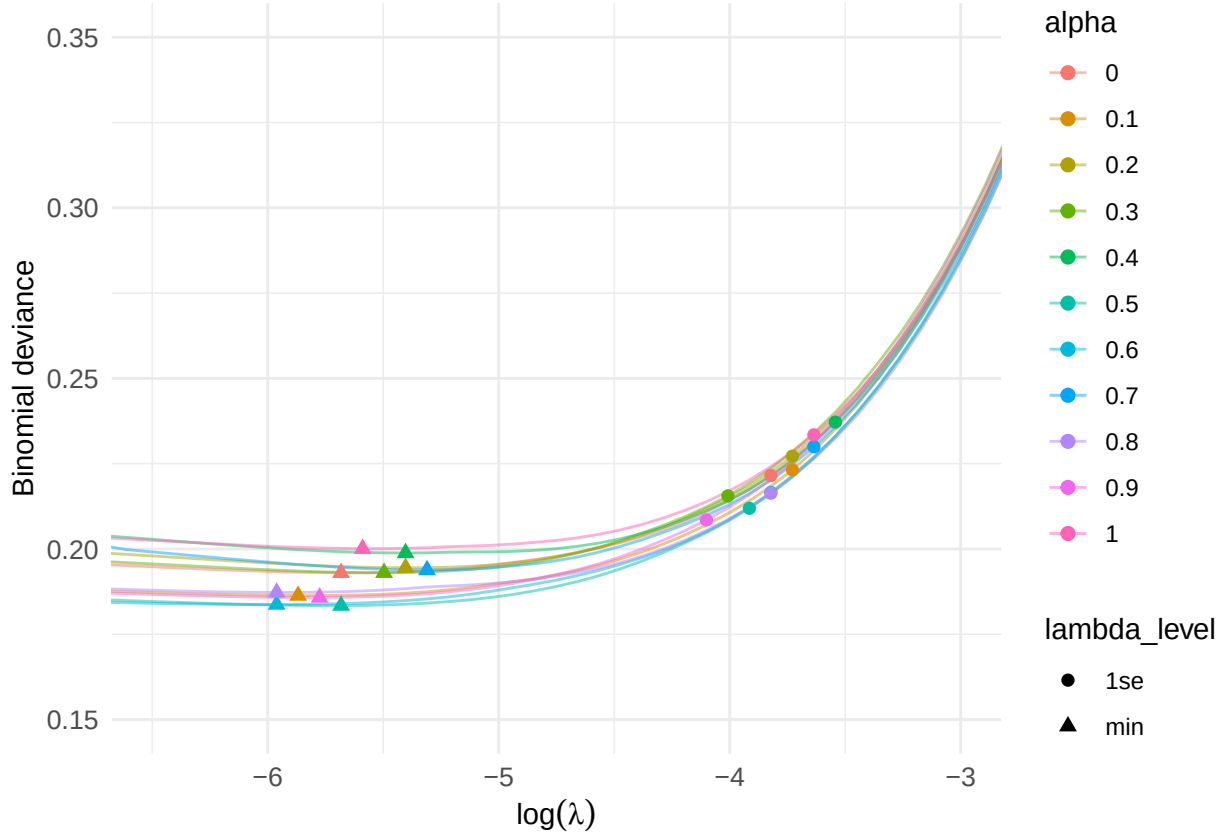
Evaluate performance of the CV model in the test set

```
train_pred_min <- predict(lasso_logreg_cv, predictors_train, s = "lambda.min", type = "class")
train_pred_1se <- predict(lasso_logreg_cv, predictors_train, s = "lambda.1se", type = "class")

test_pred_min <- predict(lasso_logreg_cv, predictors_test, s = "lambda.min", type = "class")
test_pred_1se <- predict(lasso_logreg_cv, predictors_test, s = "lambda.1se", type = "class")

data.frame(train_min = mean(biopsy_train$class == train_pred_min),
           test_min = mean(biopsy_test$class == test_pred_min),
           train_1se = mean(biopsy_train$class == train_pred_1se),
           test_1se = mean(biopsy_test$class == test_pred_1se))
```

```
##   train_min  test_min train_1se  test_1se
## 1 0.9727768 0.9621212  0.969147 0.9545455
```

Cross-validation to pick best combination of λ and α

```
alpha_lambda_cv_res <- expand.grid(lambda = exp(seq(-8, -1, length.out = 100)), # pretty much what's us
                                   alpha = seq(0, 1, 0.1)) %>%
    dlply("alpha", function(d) cv.glmnet(predictors_train, biopsy_train$class, family = "binomial", nfo
    llply(function(fit) mutate(tidy(fit),
                               lambda_level = case_when(lambda == fit$lambda.min ~ "min", lambda == fit
                               pred_err_mean = fit$cvm,
                               pred_err_up = fit$cvup,
                               pred_err_lo = fit$cvlo)) %>%
    bind_rows(.id = "alpha")

# Overlain plot of the best and "second best" (within 1 standard error of the best) predictions
ggplot(alpha_lambda_cv_res, aes(x = log(lambda), y = pred_err_mean, colour = alpha)) +
    coord_cartesian(xlim = c(-6.5, -3), ylim = c(0.15, 0.35)) +
    geom_line(alpha = 0.5) +
    geom_point(aes(shape = lambda_level), ~ filter(., !is.na(lambda_level)), size = 2) +
    labs(x = expression(log(lambda)), y = "Binomial deviance")
```




```
# And the best combination of alpha and lambda
alpha_lambda_cv_res %>%
    slice(which.min(pred_err_mean))
```

```
## # A tibble: 1 x 11
##   alpha  lambda estimate std.error conf.low conf.high nzero lambda_level
##   <chr>   <dbl>    <dbl>     <dbl>    <dbl>     <dbl> <int> <chr>
```

```
## 1 0.5   0.00341    0.183    0.0321    0.151     0.215     9 min
## # ... with 3 more variables: pred_err_mean <dbl>, pred_err_up <dbl>,
## #   pred_err_lo <dbl>
```

Exercise: cross-validation

```
library(MASS)
data(biopsy)
biopsy_complete <- na.exclude(biopsy)
summary(biopsy_complete)
```

```
##       ID                  V1              V2              V3
##  Length:683         Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  Class :character   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000
##  Mode  :character   Median : 4.000   Median : 1.000   Median : 1.000
##                     Mean   : 4.442   Mean   : 3.151   Mean   : 3.215
##                     3rd Qu.: 6.000   3rd Qu.: 5.000   3rd Qu.: 5.000
##                     Max.   :10.000   Max.   :10.000   Max.   :10.000
##        V4              V5              V6              V7
##  Min.   : 1.00   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.: 1.00   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 2.000
##  Median : 1.00   Median : 2.000   Median : 1.000   Median : 3.000
##  Mean   : 2.83   Mean   : 3.234   Mean   : 3.545   Mean   : 3.445
##  3rd Qu.: 4.00   3rd Qu.: 4.000   3rd Qu.: 6.000   3rd Qu.: 5.000
##  Max.   :10.00   Max.   :10.000   Max.   :10.000   Max.   :10.000
##        V8              V9              class
##  Min.   : 1.00   Min.   : 1.000   benign   :444
##  1st Qu.: 1.00   1st Qu.: 1.000   malignant:239
##  Median : 1.00   Median : 1.000
##  Mean   : 2.87   Mean   : 1.603
##  3rd Qu.: 4.00   3rd Qu.: 1.000
##  Max.   :10.00   Max.   :10.000
```

```
predictors <- biopsy_complete %>%
    select(-ID, -class)
pca_fit <- prcomp(predictors, scale = TRUE)
df_pca <- data.frame(pca_fit$x[, 1:4], outcome = biopsy_complete$class)
glm_fit <- glm(outcome ~ PC1 + PC2 + PC3 + PC4, data = df_pca, family = binomial)
summary(glm_fit)
```

```
##
## Call:
## glm(formula = outcome ~ PC1 + PC2 + PC3 + PC4, family = binomial,
##     data = df_pca)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.1791  -0.1304  -0.0619   0.0228   2.4799
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.0739     0.3035  -3.539 0.000402 ***
```

```
## PC1          -2.4140     0.2556  -9.445  < 2e-16 ***
## PC2          -0.1592     0.5050  -0.315 0.752540
## PC3           0.7191     0.3273   2.197 0.028032 *
## PC4          -0.9151     0.3691  -2.479 0.013159 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 884.35  on 682  degrees of freedom
## Residual deviance: 106.12  on 678  degrees of freedom
## AIC: 116.12
##
## Number of Fisher Scoring iterations: 8
```

```
tidy(glm_fit)
```

```
## # A tibble: 5 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)   -1.07      0.303    -3.54  4.02e- 4
## 2 PC1           -2.41      0.256    -9.44  3.56e-21
## 3 PC2           -0.159     0.505    -0.315 7.53e- 1
## 4 PC3            0.719     0.327     2.20  2.80e- 2
## 5 PC4           -0.915     0.369    -2.48  1.32e- 2
```

1. LOO-CV error rate

```
glm_fit_loocv <- cv.glm(df_pca, glm_fit)
glm_fit_loocv$delta
```

```
## [1] 0.02301890 0.02301788
```

2. Use proper cost function

```
glm_fit_loocv2 <- cv.glm(df_pca, glm_fit, cost = function(r, pi = 0) mean(abs(r-pi) > 0.5))
glm_fit_loocv2$delta
```

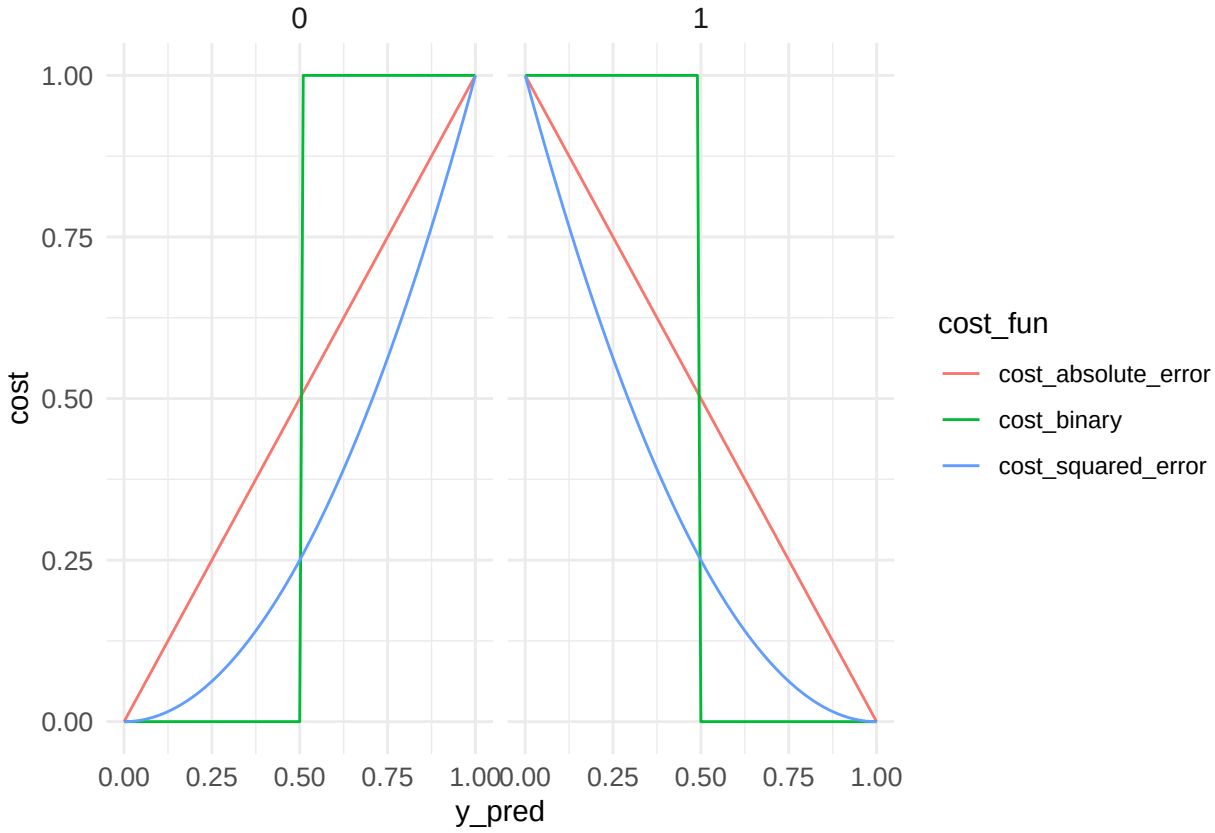
```
## [1] 0.02781845 0.02785918
```

3. Difference between error rates, and their interpretation

We make a plot to talk about how a prediction yields different different costs depending on the cost function.

```
expand.grid(y_obs = 0:1,
            y_pred = 0:100 / 100) %>%
    mutate(cost_squared_error = (y_pred - y_obs)^2,
           cost_binary = (abs(y_pred - y_obs) > 0.5) * 1,
           cost_absolute_error = abs(y_pred - y_obs)) %>%
    pivot_longer(starts_with("cost_"), names_to = "cost_fun", values_to = "cost") %>%
```

```
ggplot(aes(y_pred, cost, colour = cost_fun)) +
    geom_line() +
    facet_wrap(~ y_obs)
```



4. 10-fold CV

```
# The error rates change quite a bit, which makes sense because 10-fold CV has a lot fewer folds than LO
glm_fit_10cv <- update(glm_fit_loocv, K = 10)
glm_fit_10cv$delta
```

```
## [1] 0.02255579 0.02250644
```

```
# glm_fit_10cv2 <- cv.glm(df_pca, glm_fit, cost = function(r, pi = 0) mean(abs(r-pi) > 0.5), K = 10)
glm_fit_10cv2 <- update(glm_fit_loocv2, K = 10)
glm_fit_10cv2$delta
```

```
## [1] 0.02635432 0.02679163
```

```
ldply(list("LOO" = glm_fit_loocv, "LOO 2" = glm_fit_loocv2, "10-fold" = glm_fit_10cv, "10-fold 2" = glm_
      with, delta) %>%
    setNames(c("model", "error_rate", "corrected_error_rate"))
```

```
##       model error_rate corrected_error_rate
## 1       LOO 0.02301890           0.02301788
## 2     LOO 2 0.02781845           0.02785918
## 3   10-fold 0.02255579           0.02250644
## 4 10-fold 2 0.02635432           0.02679163
```

```
knitr::opts_chunk$set(include = FALSE)
```

Exercise: penalised regression

1. + 2. Lasso regression

3. Why does it normally make sense to normalise predictors

If predictors *not* on the same scale, those with large values will be discarded unduly from the model because they contribute a lot to the l1 norm.

4. Using CV to get reasonable estimate of λ

5. Obtain coefficients for "best" λ

6. Re-fit with correct family

Quite some difference between the sets of predictors kept:

7. As ridge regression

8. Get idea about sparse solution using ridge results?

The closer to zero the coefficients are, they less important they are to the prediction (thanks to normalisation of predictors), so one could set a threshold and only keep predictors with abs(coefficient) above that.

9. Elastic net ($\alpha = 0.5$)

Ideally, one should do CV over α as well (see the example in the code from the lecture).

10. Delasso the results

11. Selective inference (doesn't run)

Again, this is an active area of research so the package is not very stable and the results are probably not be relied upon for now. Included here to illustrate it can be done. Also, the `fixedLassoInf` function returns an error due to singularity (at least as of 3 November 2020).