CS4260: Analysis of single-cell transcriptomics of chronic myeloid leukemia

Akash, Hielke, Marthe, Matt

Contents

1 Supervised learning on responder status (Hielke) 1
1.1 Set-up . . . 1
1.2 Data wrangling and cleaning . . . 4
1.3 Create train and test patients . . . 4
1.4 Utilities . . . 4
1.5 Different supervised learning methods . . . 6
1.5.1 LDA . . . 6
1.5.1.1 Verify . . . 6
1.5.1.1.1 Visualize . . . 7
1.5.2 Random forest . . . 9
1.5.2.1 Verify . . . 9
1.5.2.1.1 Visualize . . . 10
1.5.3 Intermezzo: Most important genes . . . 12
1.5.4 LDA with less genes . . . 12
1.5.4.1 Verify . . . 13
1.5.4.1.1 Visualize . . . 13
1.5.5 QDA . . . 15
1.5.5.1 Verify . . . 15
1.5.5.1.1 Visualize . . . 16
1.5.6 SVM . . . 18
1.5.6.1 Verify . . . 18
1.5.6.1.1 Visualize . . . 19
1.5.7 Introducing regularization . . . 21
1.5.8 LASSO . . . 21
1.5.8.1 Verify . . . 21
1.5.8.1.1 Visualize . . . 22
1.5.9 Ridge . . . 24
1.5.9.1 Verify . . . 24
1.5.9.1.1 Visualize . . . 25

1 Supervised learning on responder status (Hielke)

Here, we applied numerous methods to see what supervised learning method would be best to distinguish between different responder statuses.

1.1 Set-up

Loading in the packages.

```r
# Data wrangling
library(tidyverse)  # collection of packages a.o: ggplot2, tibble, dplyr
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.0     v purrr   0.3.3
## v tibble  3.0.0     v dplyr   0.8.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(magrittr)  # piping
```

```
##
## Attaching package: 'magrittr'
```

```
## The following object is masked from 'package:purrr':
##
##     set_names
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

```r
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```r
library(janitor)  # data cleaning
```

```
##
## Attaching package: 'janitor'
```

```
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```r
# Plotting
library(ggplot2)
# library(gridExtra)
library(cowplot)  # grid plotting
```

```
##
## ************************************************************
```

```
## Note: As of version 1.0.0, cowplot does not change the
```

```
##   default ggplot2 theme anymore. To recover the previous
```

```
##   behavior, execute:
##   theme_set(theme_cowplot())
```

```
## ************************************************************
```

```r
# Machine learning
library(MASS)  # LDA, QDA
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin

library(glmnet)  # elasticnet, LASSO, Ridge

## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 3.0-2

library(e1071)  # SVM


# Other
library(readxl)  # Read in data from Excel
library(glue)  # String formatting

##
## Attaching package: 'glue'

## The following object is masked from 'package:dplyr':
##
##     collapse
```

We consider reproducible science to be important.

```
set.seed(46692)
```

Data loading.

```
# Get gene expression data (preprocessed)
gene_data <- readxl::read_excel("/home/hielke/repos/mlbio/correctedGeneDataSDMR.xlsx")
gene_data <- column_to_rownames(gene_data, var = "-0.99280765125085602")  # Yeah, this is weird.

# Get meta data_
meta <- read.csv("/home/hielke/repos/mlbio/Metadata.txt", sep="\t")
meta <- column_to_rownames(meta, var = "Cell")
```

1.2 Data wrangling and cleaning

Reform data and cleaning.

```
rownames(gene_data) %<>% make_clean_names(case = "none")
genes <- rownames(gene_data)
gene_data_t <- as_tibble(t(gene_data))
```

Sort the two dataframes so that they can be compared. (The index is the cell id.)

```
meta <- meta[sort(rownames(meta)), ]
gene_data_t <- gene_data_t[sort(rownames(gene_data_t)),]
```

1.3 Create train and test patients

```
good_pat <- as_vector(unique((dplyr::filter(meta, Responder_status == "good")$Patient_id)))
poor_pat <- as_vector(unique((dplyr::filter(meta, Responder_status == "poor")$Patient_id)))

# Randomness controlled by seed
good_pat_test <- base::sample(good_pat, 2)
poor_pat_test <- base::sample(poor_pat, 2)
pat_test <- fct_c(good_pat_test, poor_pat_test)

good_pat_train <- good_pat[!grepl(paste(pat_test, collapse="|"), good_pat)]
poor_pat_train <- poor_pat[!grepl(paste(pat_test, collapse="|"), poor_pat)]
pat_train <- fct_c(good_pat_train, poor_pat_train)

# This can be done since the frames are sorted on index.
gene_data_t$Patient_id <- meta$Patient_id
gene_data_t$Responder_status <- meta$Responder_status

# Remove "unknown" status (EDIT: Unexplainable bugs came from this.)
# meta_prog <- dplyr::filter(meta, Responder_status %in% c("good", "poor"))
# meta_prog$Responder_status %<>% factor

# gene_data_prog <- gene_data_t %>% filter(Responder_status %in% c("good", "poor"))
# gene_data_prog$Responder_status %<>% factor

# Create test/train set
gene_data_train <- gene_data_t %>% filter(Patient_id %in% pat_train)
gene_data_train$Responder_status %<>% factor
gene_data_test <- gene_data_t %>% filter(Patient_id %in% pat_test)
gene_data_test$Responder_status %<>% factor
```

1.4 Utilities

Here a collection of utilities is made that can be used with multiple supervised learning methods.

Create the formula that makes the link between all genes and the Reponder_status.

```
Responder_to_allgenes <- formula(paste("Responder_status~", paste(sprintf("`%s`", genes), collapse = "+
```

A function that summarizes the correctly classified cells.

NB: We are using tidyval here. So `predicted_labels` cannot be a string, but instead will be turned into a quosure upon evaluating the arguments. Here for more about that.

```
select_correct <- function(gene_data, predicted_labels) {
  gene_data %>%
```

```
    group_by(Patient_id) %>%
    mutate(cells=n(), correct=sum(Responder_status == !! enquo(predicted_labels))) %>%
    summarise_all(first) %>%
    # dplyr::select(!(one_of(genes))) %>%
    dplyr::select(one_of("Patient_id", "Responder_status", "cells", "correct"))
}
```

A function that can visualize the correctly annoted cells versus the wrongly annotated cells.

```
visualize_classes <- function(correct_classified, title) {

  # MANGLE

  correct_classified %<>% mutate(incorrect = cells - correct)

  split_correct_incorrect <- function(df, filter) {
    df %>%
      filter(Responder_status == filter) %>%
      dplyr::select(one_of("Patient_id", "correct", "incorrect")) %>%
      melt(id_vars="Patient_id")
  }

  correct_classified_good <- split_correct_incorrect(correct_classified, "good")
  correct_classified_poor <- split_correct_incorrect(correct_classified, "poor")


  # We have crated a correct and an incorrect column, but in the plot
  # we want to visualize "poor" and "good".
  # For _poor the order is already c("correct", "incorrect")   c("poor", "good")
  # But for _good we have to swap that.
  correct_classified_good$variable %<>% factor(c("incorrect", "correct"))

  # PLOT

  ylim_cells <- round(1.1 * max(c(correct_classified_good$value, correct_classified_poor$value)))

  # We are creating two plots and put them next to each other.
  # They share these layers.
  class_barplot_layers <- list(
    aes(x = Patient_id, y = value, group = variable, fill = variable),
    geom_col(position = "dodge"),
    labs(fill = "Classification"),
    xlab("Patient ID"),
    ylab("Amount of cells"),
    scale_fill_manual(values = c("red", "blue"), labels = c("poor", "good")),
    theme(legend.position = "none"),
    ylim(0, ylim_cells),
    theme(axis.text.x = element_text(angle = 90, hjust = 1))
  )

  # Here we create the elements, two plots, and a legend.
  g_good <- ggplot(correct_classified_good) +
    labs(title = "Outcome: good") +
    class_barplot_layers
```

```
  g_poor <- ggplot(correct_classified_poor) +
    labs(title = "Outcome: poor") +
    class_barplot_layers

  legend <- get_legend(
    g_poor + theme(legend.position = "right")
  )

  title <- ggdraw() + draw_label(title, fontface = "bold", x = 0, hjust = 0) +
    theme(plot.margin = ggplot2::margin(0, 0, 0, 7))

  plot_body <- plot_grid(g_good, g_poor, legend, align = "h", ncol = 3)

  plot_grid(title, plot_body, ncol = 1, rel_heights = c(.1, 1))
}
```

1.5 Different supervised learning methods

1.5.1 LDA

Use LDA to find the relationship that relates Responder_status to gene expression.

```
gene_data_lda <- lda(Responder_to_allgenes, data = gene_data_train, CV = FALSE)
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

1.5.1.1 Verify Work on train set

```
gene_data_train$Responder_status_lda <- predict(gene_data_lda)$class
```

```
correct_classified_train_lda <- select_correct(gene_data_train, Responder_status_lda)
correct_classified_train_lda
```

Patient_id	Responder_status	cells	correct
CML1266	poor	154	134
CML15	good	57	56
CML22	good	55	51
CML655	good	68	49
CML656	good	105	75
CML691	good	30	24
CML940	good	29	26
CML960	good	25	22
OX1071	poor	22	21
OX1083	poor	26	19
OX1249	poor	64	52
OX1302	poor	62	53
OX1407	poor	40	38
OX1570	good	43	35
OX1674	poor	78	69
OX1705	good	22	10
OX1902	poor	46	38
OX2038	poor	90	84
OX2343	poor	73	63
OX703	good	24	16
OX710	poor	42	42

Patient_id	Responder_status	cells	correct
OX714	good	77	63
OX750	poor	53	51
OX753	good	33	29
OX824	good	69	60

Work on test set

```
gene_data_test$Responder_status_lda <- predict(gene_data_lda, newdata = gene_data_test)$class
```

```
correct_classified_test_lda <- select_correct(gene_data_test, Responder_status_lda)
correct_classified_test_lda
```

Patient_id	Responder_status	cells	correct
OX1528	poor	30	22
OX2125	poor	43	31
OX664	good	43	13
OX967	good	58	24

1.5.1.1.1 Visualize Train set

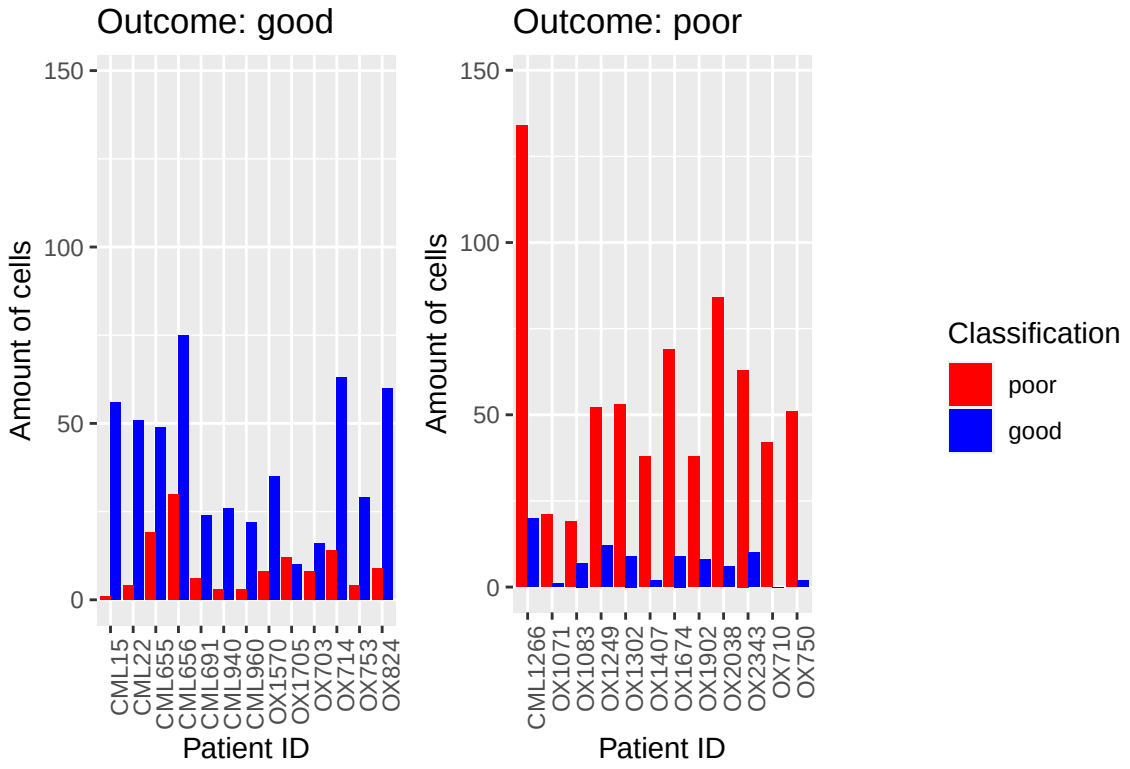
It is expected that the train set would perform reasonably well here.

```
visualize_classes(correct_classified_train_lda, "LDA: Train")
```

```
## Using Patient_id as id variables
## Using Patient_id as id variables
```

```
## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.
```

LDA: Train



Test set

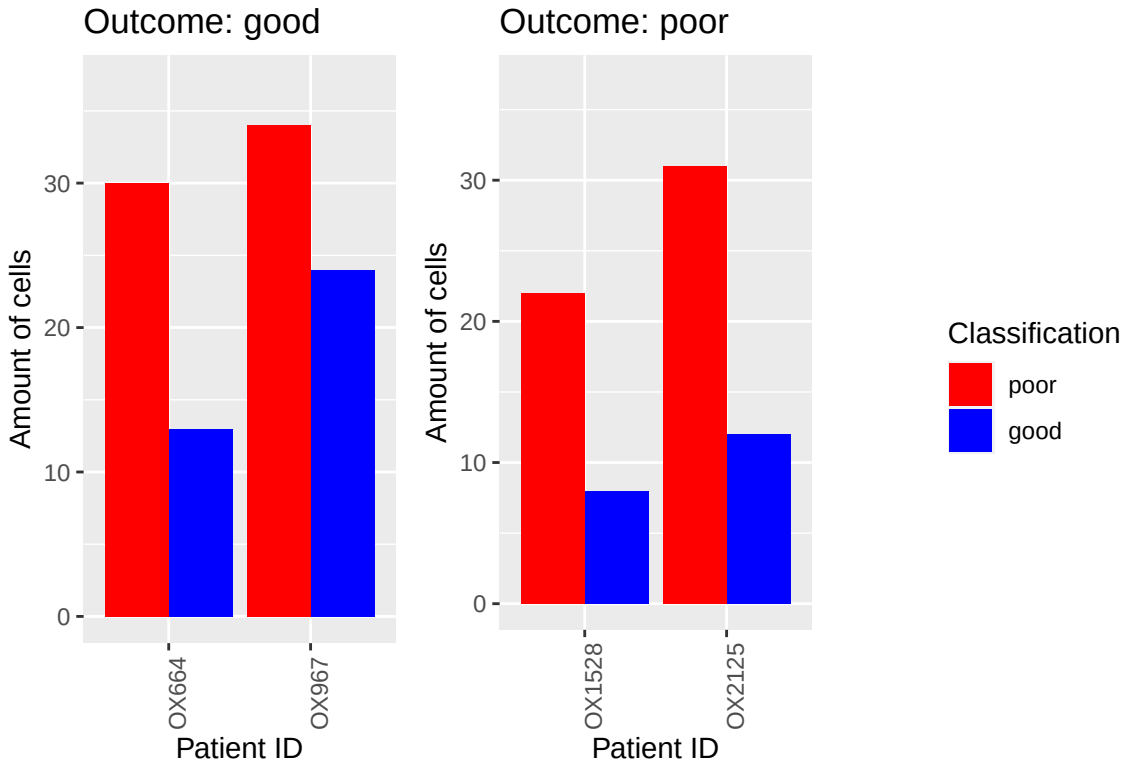
Here we can see how the LDA actually performs, and it does that very poorly.

```
visualize_classes(correct_classified_test_lda, "LDA: Test")
```

```
## Using Patient_id as id variables
## Using Patient_id as id variables
```

```
## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.
```

LDA: Test



1.5.2 Random forest

```
gene_data_rf <- randomForest(Responder_to_allgenes, data = gene_data_train)
```

1.5.2.1 Verify Work on train set

```
gene_data_train$Responder_status_rf <- predict(gene_data_rf)

correct_classified_train_rf <- select_correct(gene_data_train, Responder_status_rf)
correct_classified_train_rf
```

Patient_id	Responder_status	cells	correct
CML1266	poor	154	143
CML15	good	57	21
CML22	good	55	55
CML655	good	68	59
CML656	good	105	84
CML691	good	30	15
CML940	good	29	29
CML960	good	25	25
OX1071	poor	22	11
OX1083	poor	26	25
OX1249	poor	64	63
OX1302	poor	62	62
OX1407	poor	40	26
OX1570	good	43	0
OX1674	poor	78	76
OX1705	good	22	0

Patient_id	Responder_status	cells	correct
OX1902	poor	46	45
OX2038	poor	90	57
OX2343	poor	73	55
OX703	good	24	20
OX710	poor	42	15
OX714	good	77	52
OX750	poor	53	10
OX753	good	33	33
OX824	good	69	65

Work on test set

```
gene_data_test$Responder_status_rf <- predict(gene_data_rf, newdata = gene_data_test)
```

```
correct_classified_test_rf <- select_correct(gene_data_test, Responder_status_rf)
correct_classified_test_rf
```

Patient_id	Responder_status	cells	correct
OX1528	poor	30	13
OX2125	poor	43	8
OX664	good	43	11
OX967	good	58	23

1.5.2.1.1 Visualize Train set

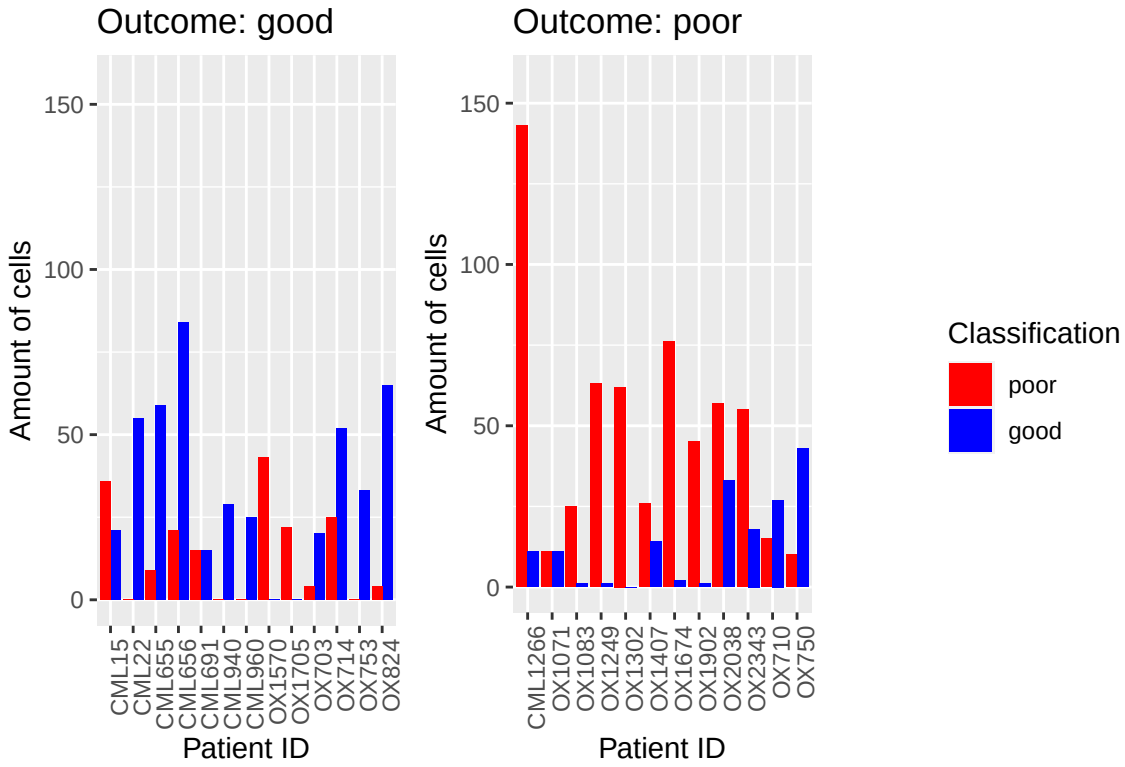
It is expected that the train set would perform reasonably well here. However, it seems that for not all patients it makes the true predictions.

```
visualize_classes(correct_classified_train_rf, "Random Forest: Train")
```

```
## Using Patient_id as id variables
## Using Patient_id as id variables
```

```
## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.
```

Random Forest: Train



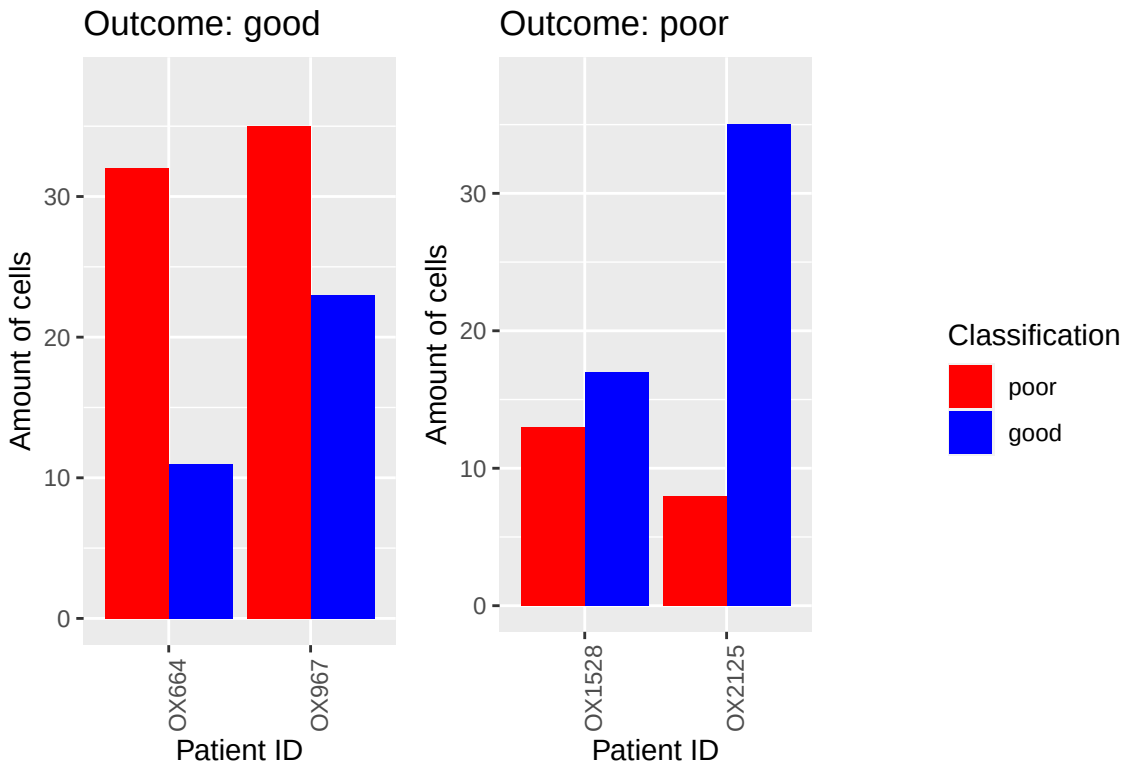
Test set

Here we can see how the Random Forest actually performs, and here it doesn't show anything good either.

```
visualize_classes(correct_classified_test_rf, "Random Forest: Test")
```

```
## Using Patient_id as id variables
## Using Patient_id as id variables
```

```
## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.
```



1.5.3 Intermezzo: Most important genes

The Random Forest does bring something extra to the table and that is that it can give a measure to how importance certain variables are. We can use this to make a stricter subset.

```
size_important_genes <- 500
important_genes <- (
  importance(gene_data_rf) %>%
    as_tibble(rownames = NA) %>%
    rownames_to_column("gene") %>%
    dplyr::arrange(MeanDecreaseGini) %>%
    top_n(size_important_genes)
)$gene
```

```
## Selecting by MeanDecreaseGini
```

Now we can create another formula for that new relationship

```
Responder_to_mostimportantgenes <- formula(
  paste("Responder_status~", paste(sprintf("`%s`", important_genes), collapse = "+"))
)
```

1.5.4 LDA with less genes

Now we can attempt to redo the LDA with less genes and see if we improve. We might be able to improve as we were doing pretty well on the train set, but not that well on the test, indicating we are overfitting a little bit.

```
gene_data_ldamore <- lda(Responder_to_mostimportantgenes, data = gene_data_train, CV = FALSE)
```

1.5.4.1 **Verify** Work on train set

```
gene_data_train$Responder_status_ldamore <- predict(gene_data_ldamore)$class
```

```
correct_classified_train_ldamore <- select_correct(gene_data_train, Responder_status_ldamore)
correct_classified_train_ldamore
```

Patient_id	Responder_status	cells	correct
CML1266	poor	154	136
CML15	good	57	49
CML22	good	55	50
CML655	good	68	56
CML656	good	105	92
CML691	good	30	24
CML940	good	29	26
CML960	good	25	24
OX1071	poor	22	21
OX1083	poor	26	24
OX1249	poor	64	56
OX1302	poor	62	51
OX1407	poor	40	30
OX1570	good	43	39
OX1674	poor	78	69
OX1705	good	22	15
OX1902	poor	46	41
OX2038	poor	90	86
OX2343	poor	73	66
OX703	good	24	13
OX710	poor	42	33
OX714	good	77	60
OX750	poor	53	41
OX753	good	33	27
OX824	good	69	55

Work on test set

```
gene_data_test$Responder_status_ldamore <- predict(gene_data_ldamore, newdata = gene_data_test)$class
```

```
correct_classified_test_ldamore <- select_correct(gene_data_test, Responder_status_ldamore)
correct_classified_test_ldamore
```

Patient_id	Responder_status	cells	correct
OX1528	poor	30	19
OX2125	poor	43	23
OX664	good	43	19
OX967	good	58	20

1.5.4.1.1 **Visualize** Train set

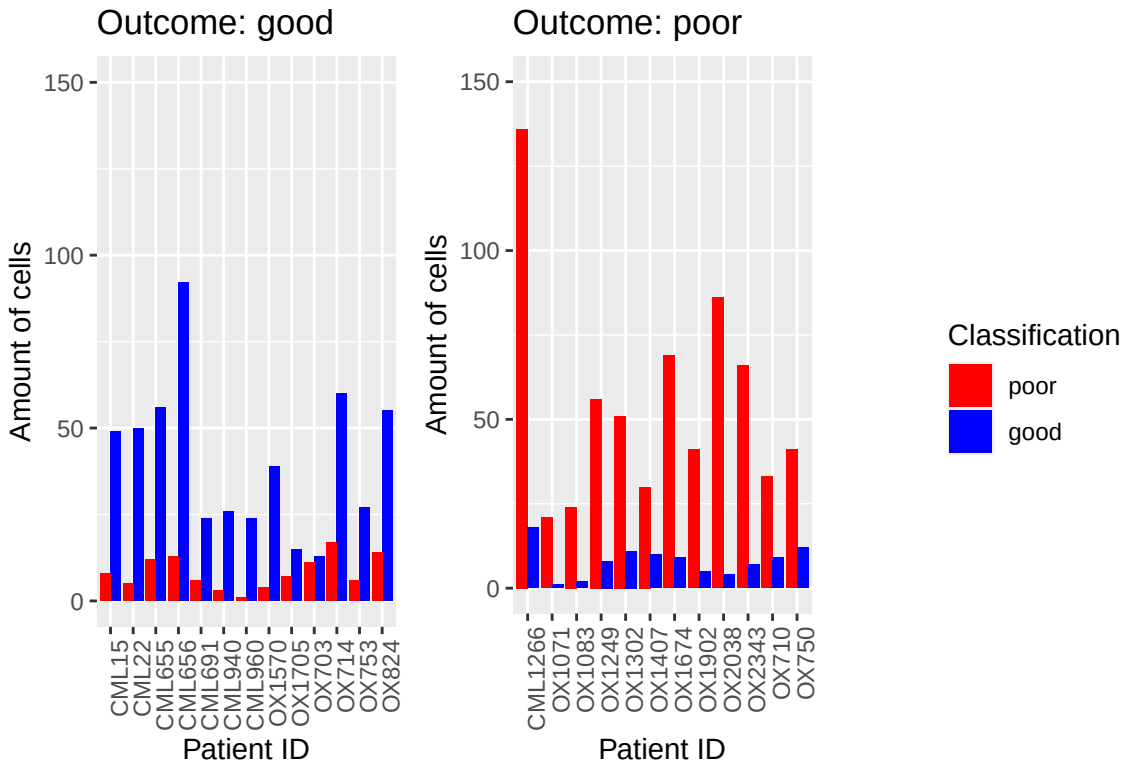
It is expected that the train set would perform reasonably well here.

```
visualize_classes(correct_classified_train_ldamore, glue("LDA ({size_important_genes}): Train"))
```

```
## Using Patient_id as id variables
## Using Patient_id as id variables
```

```
## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.
```

LDA (500): Train



Test set

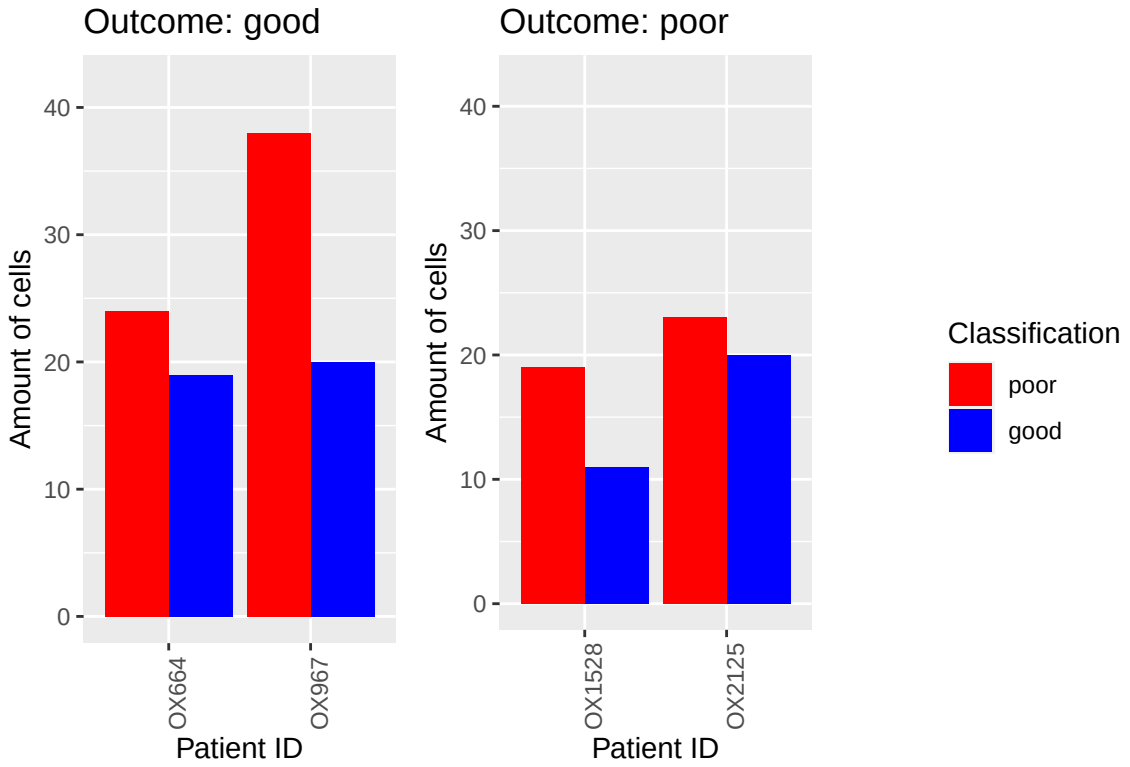
Here we can see how the ldamore actually performs, and it does that very poorly.

```
visualize_classes(correct_classified_test_ldamore, glue("LDA ({size_important_genes}): Test"))
```

```
## Using Patient_id as id variables
## Using Patient_id as id variables
```

```
## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.
```

LDA (500): Test



1.5.5 QDA

Instead of linear, we can also attempt quadratic.

NB: QDA can be picky, but it should work with the reduced amount of genes and the set seed.

```
gene_data_qda <- qda(Responder_to_mostimportantgenes, data = gene_data_train)
```

1.5.5.1 Verify Work on train set

```
gene_data_train$Responder_status_qda <- predict(gene_data_qda)$class
```

```
correct_classified_train_qda <- select_correct(gene_data_train, Responder_status_qda)
correct_classified_train_qda
```

Patient_id	Responder_status	cells	correct
CML1266	poor	154	154
CML15	good	57	57
CML22	good	55	55
CML655	good	68	68
CML656	good	105	105
CML691	good	30	30
CML940	good	29	29
CML960	good	25	25
OX1071	poor	22	22
OX1083	poor	26	26
OX1249	poor	64	64
OX1302	poor	62	62

Patient_id	Responder_status	cells	correct
OX1407	poor	40	40
OX1570	good	43	43
OX1674	poor	78	78
OX1705	good	22	21
OX1902	poor	46	46
OX2038	poor	90	90
OX2343	poor	73	73
OX703	good	24	24
OX710	poor	42	42
OX714	good	77	77
OX750	poor	53	53
OX753	good	33	33
OX824	good	69	69

Work on test set

```
gene_data_test$Responder_status_qda <- predict(gene_data_qda, newdata = gene_data_test)$class
```

```
correct_classified_test_qda <- select_correct(gene_data_test, Responder_status_qda)
correct_classified_test_qda
```

Patient_id	Responder_status	cells	correct
OX1528	poor	30	27
OX2125	poor	43	42
OX664	good	43	4
OX967	good	58	4

1.5.5.1.1 Visualize Train set

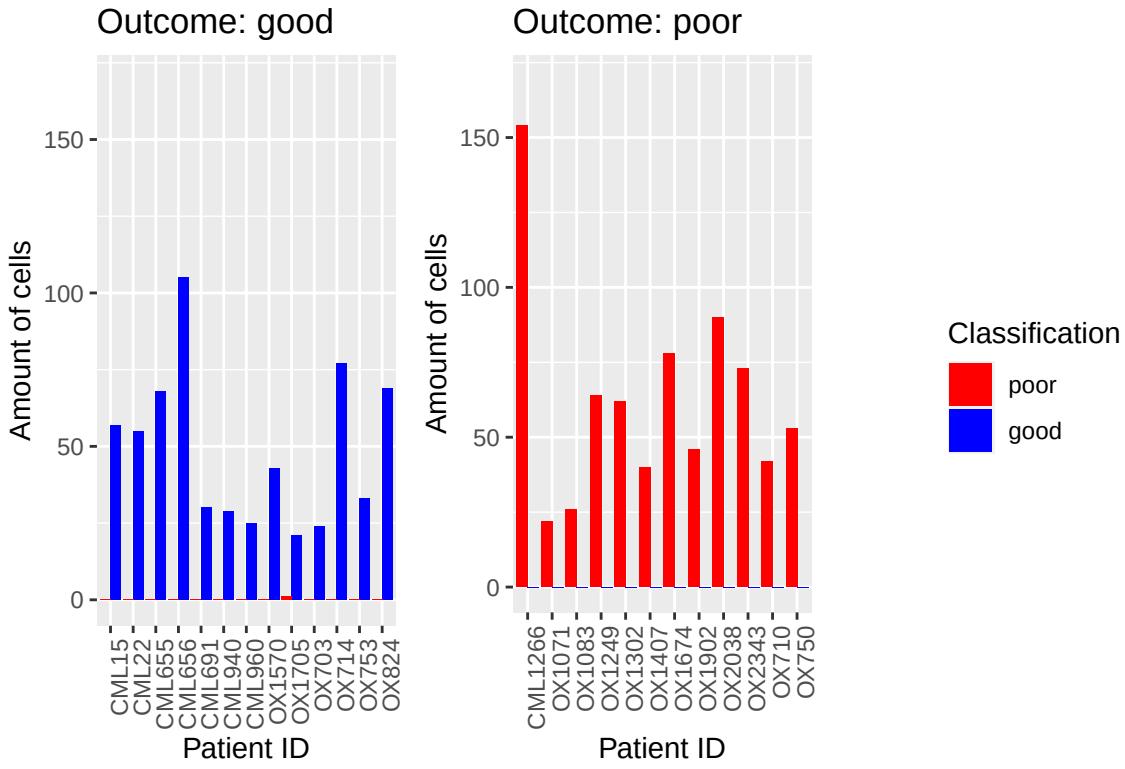
It is expected that the train set would perform reasonably well here.

```
visualize_classes(correct_classified_train_qda, "QDA: Train")
```

```
## Using Patient_id as id variables
## Using Patient_id as id variables
```

```
## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.
```

QDA: Train



Test set

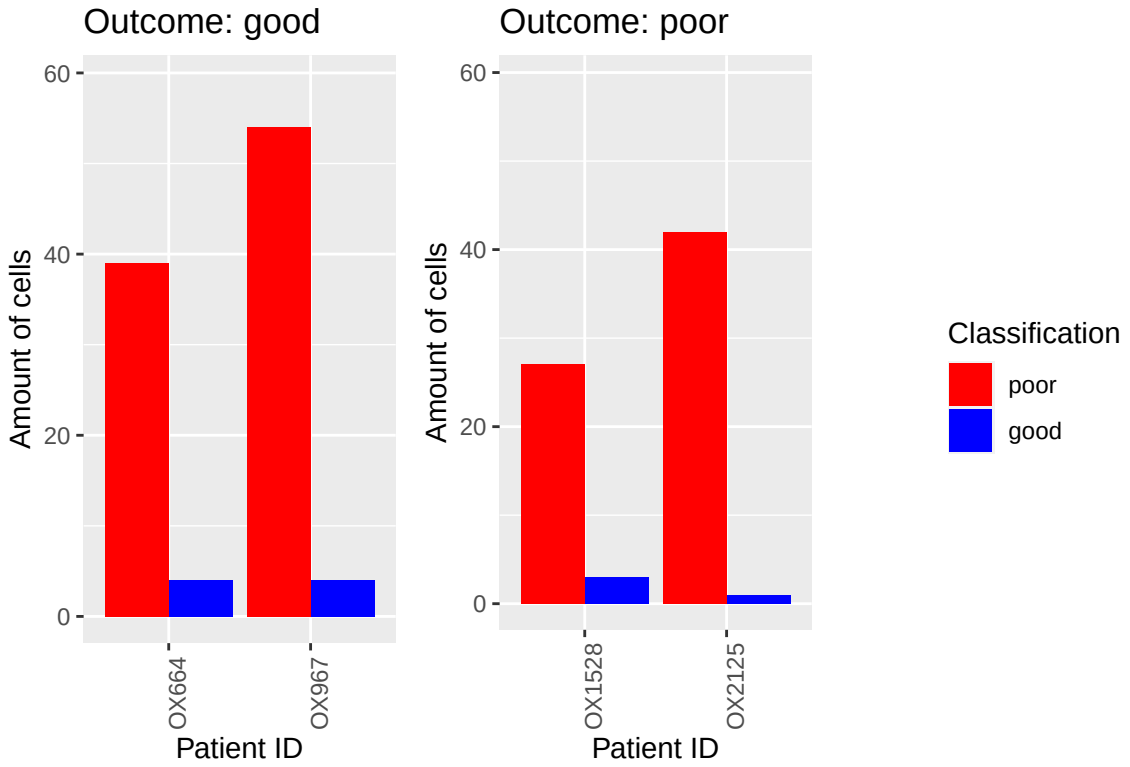
Here we can see how the QDA actually performs, and it does that very well indeed. Also important to note that the poerformance on poor is much more visible on both the train and the test data.

```
visualize_classes(correct_classified_test_qda, "QDA: Test")
```

```
## Using Patient_id as id variables
## Using Patient_id as id variables
```

```
## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.
```

QDA: Test



1.5.6 SVM

```
gene_data_svm <- svm(Responder_to_mostimportantgenes, data = gene_data_train)
```

1.5.6.1 Verify Work on train set

```
gene_data_train$Responder_status_svm <- predict(gene_data_svm)

correct_classified_train_svm <- select_correct(gene_data_train, Responder_status_svm)
correct_classified_train_svm
```

Patient_id	Responder_status	cells	correct
CML1266	poor	154	148
CML15	good	57	54
CML22	good	55	53
CML655	good	68	64
CML656	good	105	101
CML691	good	30	29
CML940	good	29	29
CML960	good	25	25
OX1071	poor	22	22
OX1083	poor	26	26
OX1249	poor	64	63
OX1302	poor	62	61
OX1407	poor	40	40
OX1570	good	43	41
OX1674	poor	78	77
OX1705	good	22	19

Patient_id	Responder_status	cells	correct
OX1902	poor	46	45
OX2038	poor	90	90
OX2343	poor	73	73
OX703	good	24	19
OX710	poor	42	42
OX714	good	77	69
OX750	poor	53	53
OX753	good	33	28
OX824	good	69	66

Work on test set

```
gene_data_test$Responder_status_svm <- predict(gene_data_svm, newdata = gene_data_test)
```

```
correct_classified_test_svm <- select_correct(gene_data_test, Responder_status_svm)
correct_classified_test_svm
```

Patient_id	Responder_status	cells	correct
OX1528	poor	30	24
OX2125	poor	43	32
OX664	good	43	14
OX967	good	58	16

1.5.6.1.1 Visualize Train set

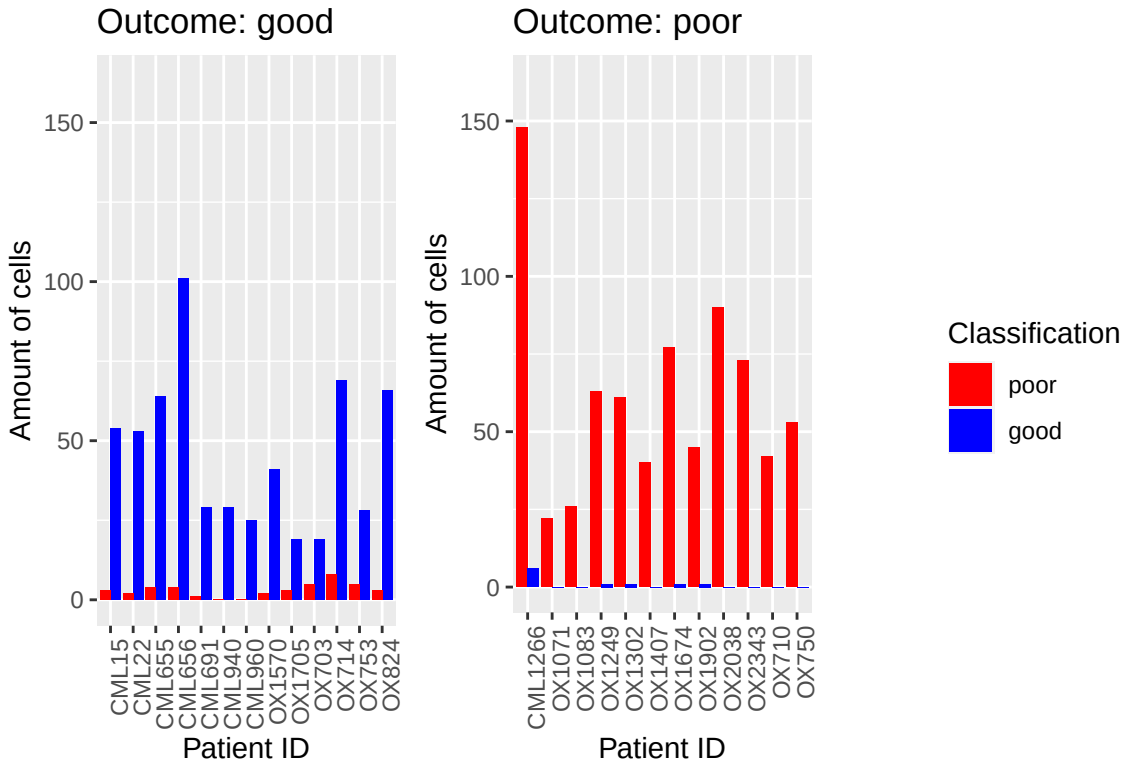
SVM is famous for its excellent performance. We can see that it performs indeed very well.

```
visualize_classes(correct_classified_train_svm, "SVM: Train")
```

```
## Using Patient_id as id variables
## Using Patient_id as id variables
```

```
## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.
```

SVM: Train



Test set

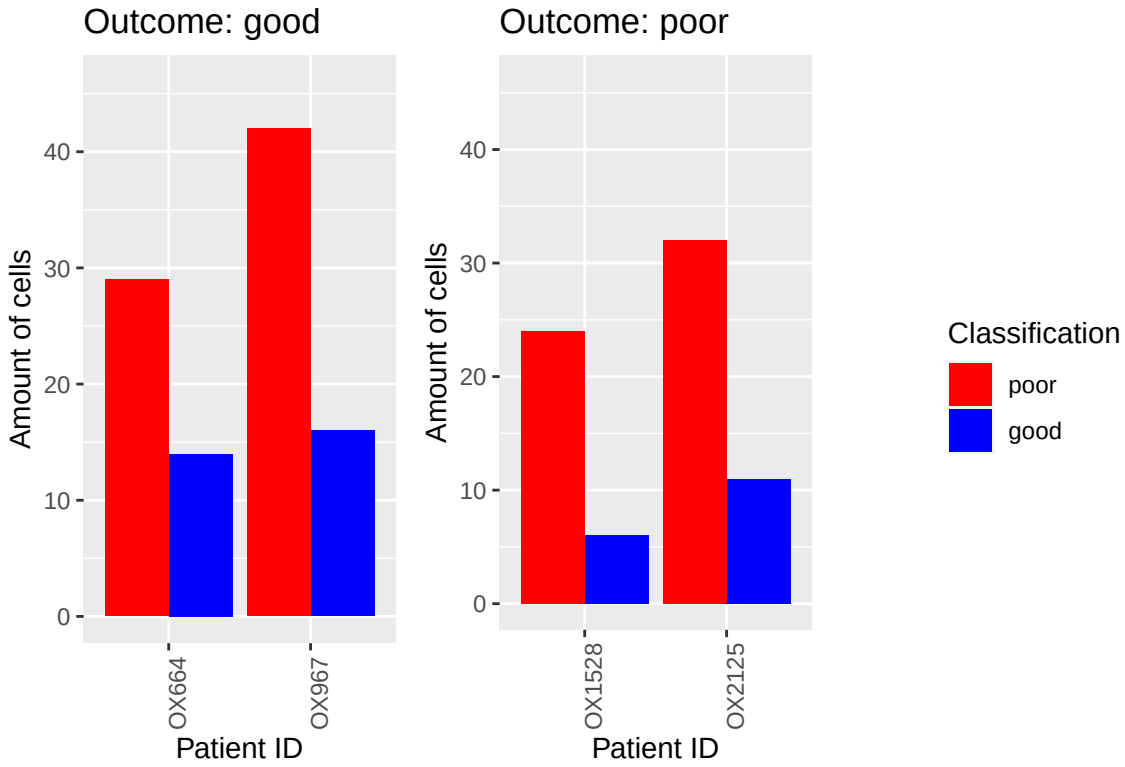
Here we can see how the SVM actually performs, and it does not reasonably well, especially for 'poor' outcome patients.

```
visualize_classes(correct_classified_test_svm, "SVM: Test")
```

```
## Using Patient_id as id variables
## Using Patient_id as id variables
```

```
## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.
```

SVM: Test



1.5.7 Introducing regularization

A way to improve on the classification is to introduce regularization. In essence with regularization you penalize the complexity of your model. As in general, less complex model are often more general models.

Here, we will be using glmnet, which implements the *elasticnet* regularization. The *elasticnet* regalarization is a mixed model of rigde and lasso regularization. When using *elasticnet*, one can change the paramater . If this paramater is equal to 1, this is in essence the lasso model. If this parameter is instead equal to 0, this is the ridge model.

With *glmnet* we can also quite easy use the logistics regression needed for this binary classification by using the `family = "binomial"` option.

1.5.8 LASSO

```
x <- model.matrix(Responder_to_allgenes, gene_data_train)[,-1]
y <- ifelse(gene_data_train$Responder_status == "good", 1, 0)

x_test <- model.matrix(Responder_to_allgenes, gene_data_test)[,-1]
```

Another parameter in this model that can be configured is the parameter. *glmnet* provides with an easy way to find the best parameter using cross validation.

```
cv_lasso <- cv.glmnet(x, y, alpha = 1, family = "binomial")
gene_data_lasso <- glmnet(x, y, alpha = 1, family = "binomial", lambda = cv_lasso$lambda.min)
```

1.5.8.1 Verify LASSO works slightly different from the previous classifiers as here we actually have probabilities instead of the classes. TODO: There might be a better idea to diagnose the patient combining the probabilities. Maybe just taking the mean and than take that number instead as the diagnosis.

```
gene_data_train$prob_train_lasso <- predict(gene_data_lasso, newx = x, type = "response")
gene_data_train$Responder_status_lasso <- ifelse(gene_data_train$prob_train_lasso >= .5, "good", "poor")

correct_classified_train_lasso <- select_correct(gene_data_train, Responder_status_lasso)
correct_classified_train_lasso
```

Patient_id	Responder_status	cells	correct
CML1266	poor	154	139
CML15	good	57	38
CML22	good	55	40
CML655	good	68	48
CML656	good	105	85
CML691	good	30	15
CML940	good	29	26
CML960	good	25	23
OX1071	poor	22	20
OX1083	poor	26	24
OX1249	poor	64	58
OX1302	poor	62	57
OX1407	poor	40	38
OX1570	good	43	26
OX1674	poor	78	74
OX1705	good	22	4
OX1902	poor	46	45
OX2038	poor	90	89
OX2343	poor	73	69
OX703	good	24	12
OX710	poor	42	38
OX714	good	77	39
OX750	poor	53	50
OX753	good	33	25
OX824	good	69	48

Work on test set

```
gene_data_test$prob_test_lasso <- predict(gene_data_lasso, newx = x_test, type = "response")
gene_data_test$Responder_status_lasso <- ifelse(gene_data_test$prob_test_lasso >= .5, "good", "poor")

correct_classified_test_lasso <- select_correct(gene_data_test, Responder_status_lasso)
correct_classified_test_lasso
```

Patient_id	Responder_status	cells	correct
OX1528	poor	30	23
OX2125	poor	43	32
OX664	good	43	12
OX967	good	58	20

1.5.8.1.1 Visualize Train set

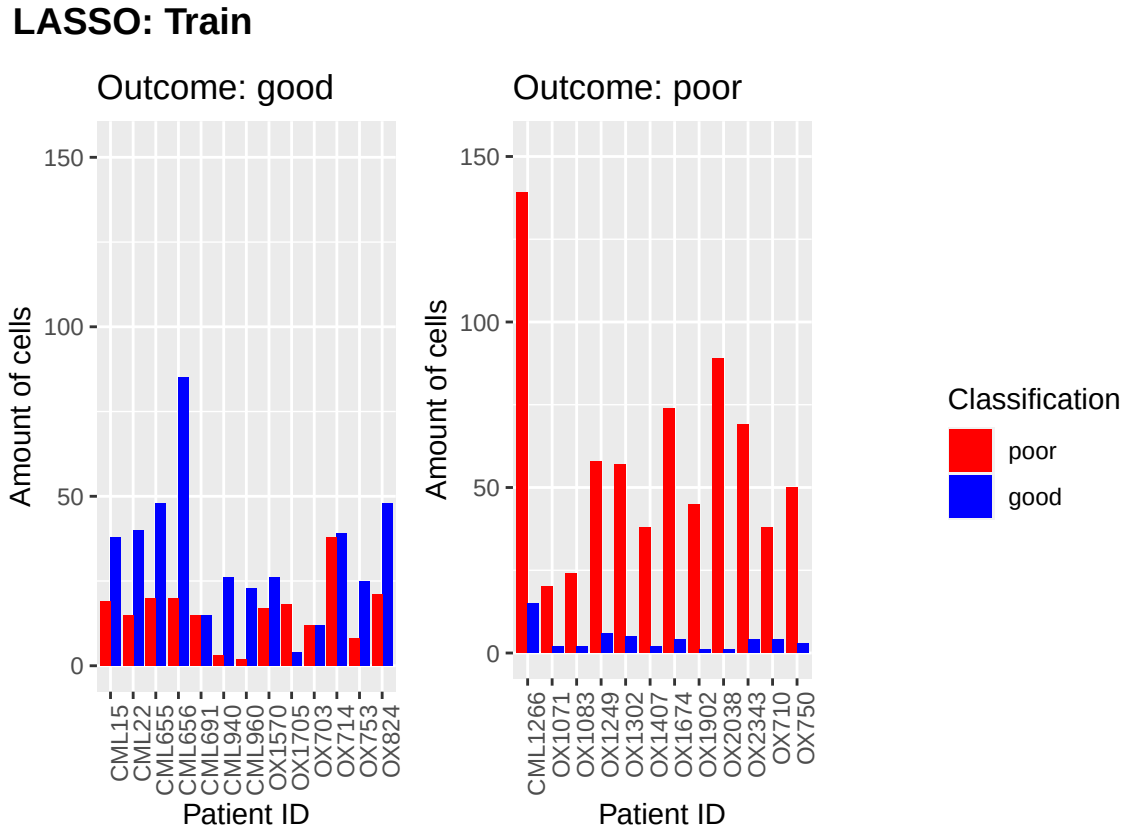
LASSO is famous for its excellent performance. We can see that it performs indeed very well.

```
visualize_classes(correct_classified_train_lasso, "LASSO: Train")
```

```
## Using Patient_id as id variables
```

```
## Using Patient_id as id variables
```

```
## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.
```



Test set

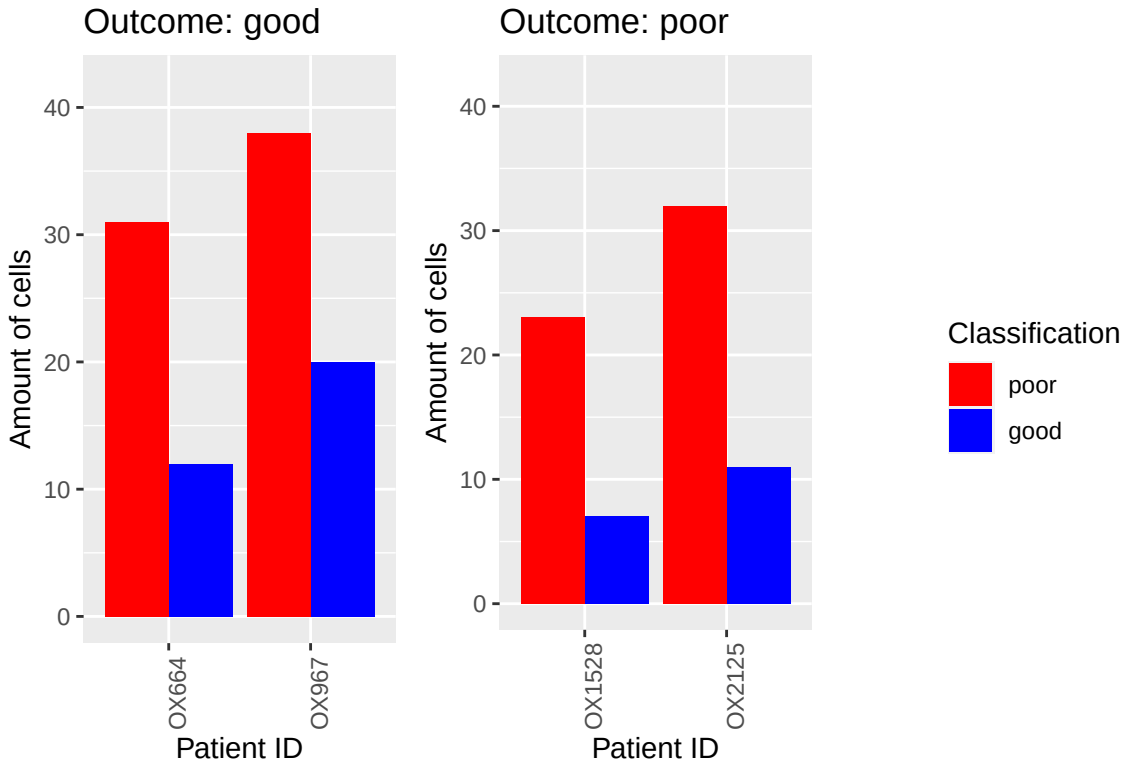
Here we can see how the LASSO actually performs, and it does not reasonably well, especially for 'poor' outcome patients.

```
visualize_classes(correct_classified_test_lasso, "LASSO: Test")
```

```
## Using Patient_id as id variables
## Using Patient_id as id variables
```

```
## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.
```

LASSO: Test



1.5.9 Ridge

```
cv_ridge <- cv.glmnet(x, y, alpha = 0, family = "binomial")
gene_data_ridge <- glmnet(x, y, alpha = 0, family = "binomial", lambda = cv_ridge$lambda.min)
```

1.5.9.1 Verify

```
gene_data_train$prob_train_ridge <- predict(gene_data_ridge, newx = x, type = "response")
gene_data_train$Responder_status_ridge <- ifelse(gene_data_train$prob_train_ridge >= .5, "good", "poor"
```

```
correct_classified_train_ridge <- select_correct(gene_data_train, Responder_status_ridge)
correct_classified_train_ridge
```

Patient_id	Responder_status	cells	correct
CML1266	poor	154	145
CML15	good	57	56
CML22	good	55	52
CML655	good	68	53
CML656	good	105	88
CML691	good	30	26
CML940	good	29	28
CML960	good	25	22
OX1071	poor	22	22
OX1083	poor	26	26
OX1249	poor	64	64
OX1302	poor	62	59
OX1407	poor	40	40

Patient_id	Responder_status	cells	correct
OX1570	good	43	37
OX1674	poor	78	77
OX1705	good	22	10
OX1902	poor	46	46
OX2038	poor	90	90
OX2343	poor	73	73
OX703	good	24	16
OX710	poor	42	42
OX714	good	77	58
OX750	poor	53	53
OX753	good	33	31
OX824	good	69	60

Work on test set

```
gene_data_test$prob_test_ridge <- predict(gene_data_ridge, newx = x_test, type = "response")
gene_data_test$Responder_status_ridge <- ifelse(gene_data_test$prob_test_ridge >= .5, "good", "poor")
```

```
correct_classified_test_ridge <- select_correct(gene_data_test, Responder_status_ridge)
correct_classified_test_ridge
```

Patient_id	Responder_status	cells	correct
OX1528	poor	30	26
OX2125	poor	43	39
OX664	good	43	3
OX967	good	58	9

1.5.9.1.1 Visualize Train set

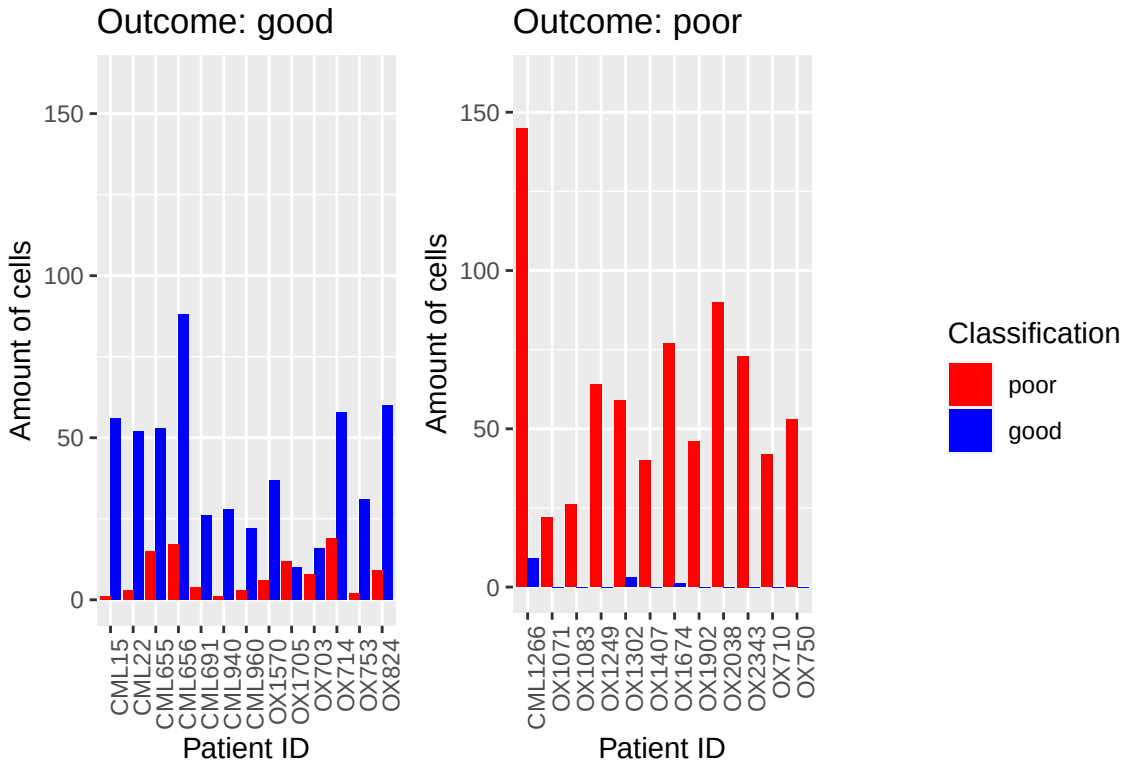
RIDGE is famous for its excellent performance. We can see that it performs indeed very well.

```
visualize_classes(correct_classified_train_ridge, "ridge: Train")
```

```
## Using Patient_id as id variables
## Using Patient_id as id variables
```

```
## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.
```

ridge: Train



Test set

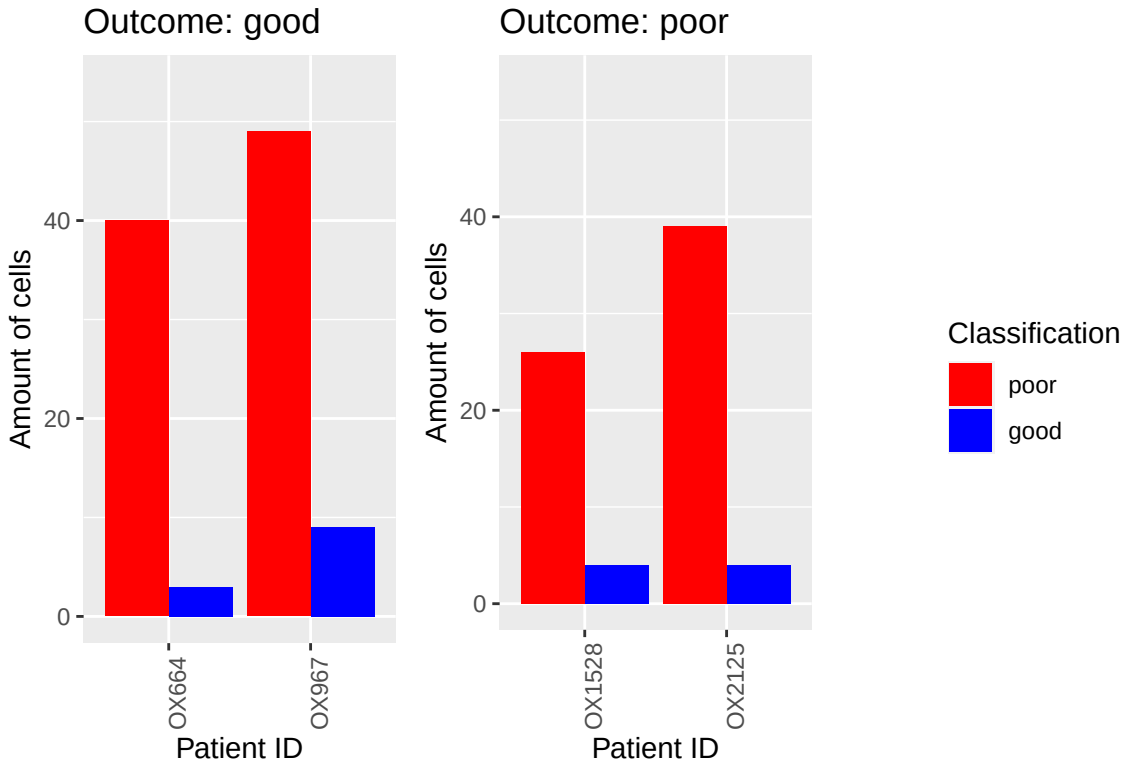
Here we can see how the ridge actually performs, and it does not reasonably well, only for 'poor' outcome patients it does work.

```
visualize_classes(correct_classified_test_ridge, "ridge: Test")
```

```
## Using Patient_id as id variables
## Using Patient_id as id variables
```

```
## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.
```

ridge: Test


Outcome: good
Outcome: poor
Amount of cells
40
20
0
OX664
OX967
OX1528
OX2125
Patient ID
Classification
poor
good