gene expression cancer RNA-Seq

20/02/2020

Data description

Source: Samuele Fiorini, samuele.fiorini@dibris.unige.it, University of Genoa, redistributed under Creative Commons license.
Download: https://www.kaggle.com/murats/gene-expression-cancer-rnaseq

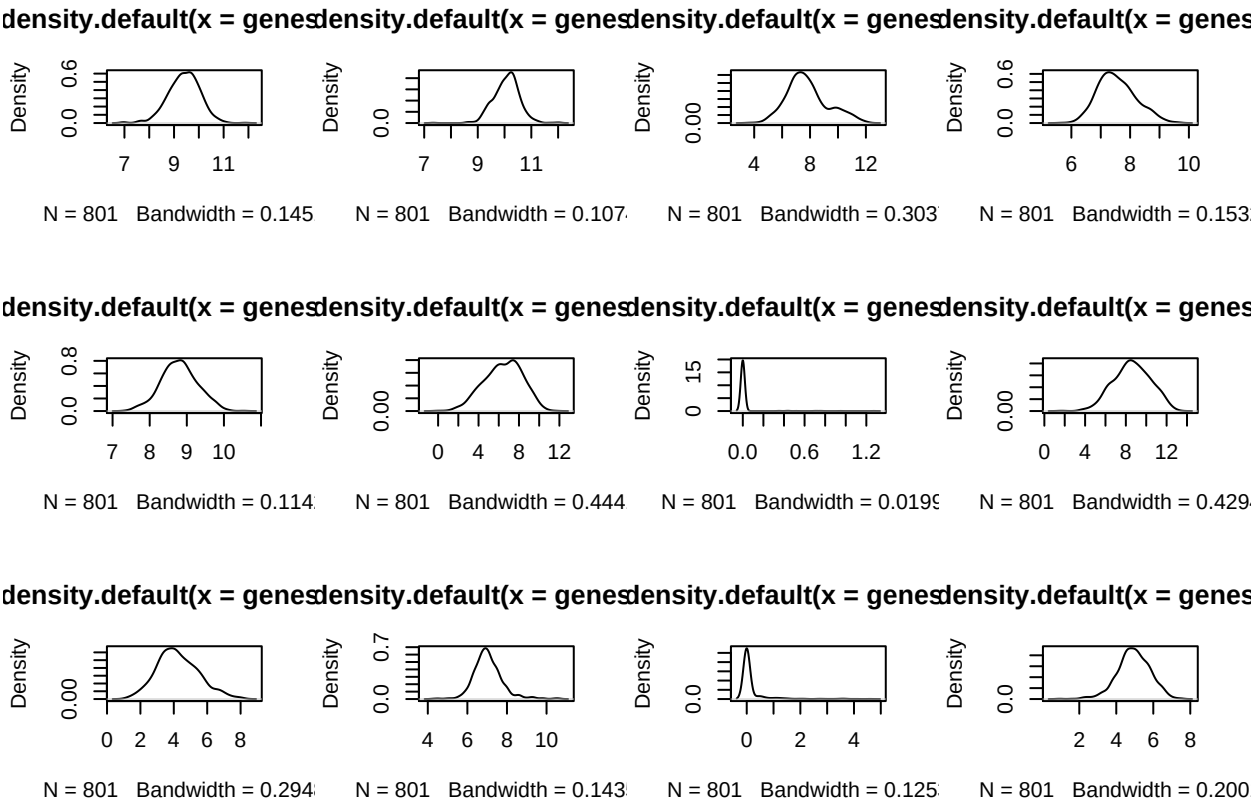
Number of observations: 801
Number of predictors: 20532

The observations represent patients with primary tumors occurring in different parts of the body, covering 12 tumor types (the response categorical variable) including:

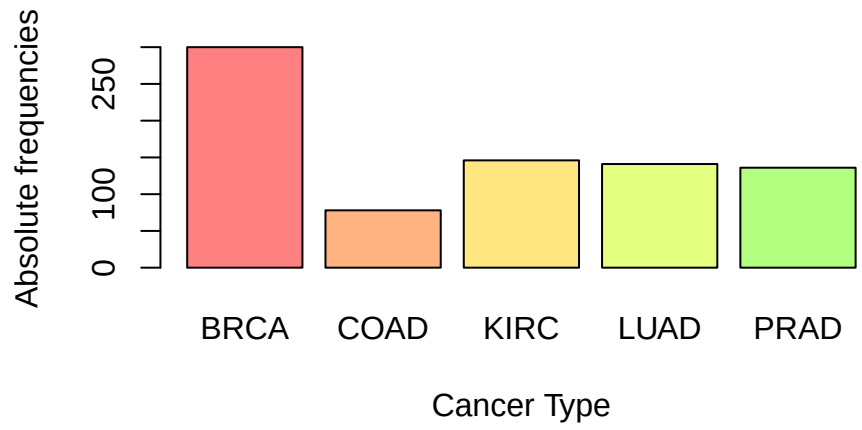
- lung adenocarcinoma (LUAD)
- breast carcinoma (BRCA)
- kidney renal clear-cell carcinoma (KIRC)
- colon adenocarcinoma (COAD)
- prostate adenocarcinoma (PRAD)

All the predictors are continuous variables representing RNA-Seq gene expression levels measured by a sequencing platform.

kde of 12 predictors picked at random. They look normal for the most part:



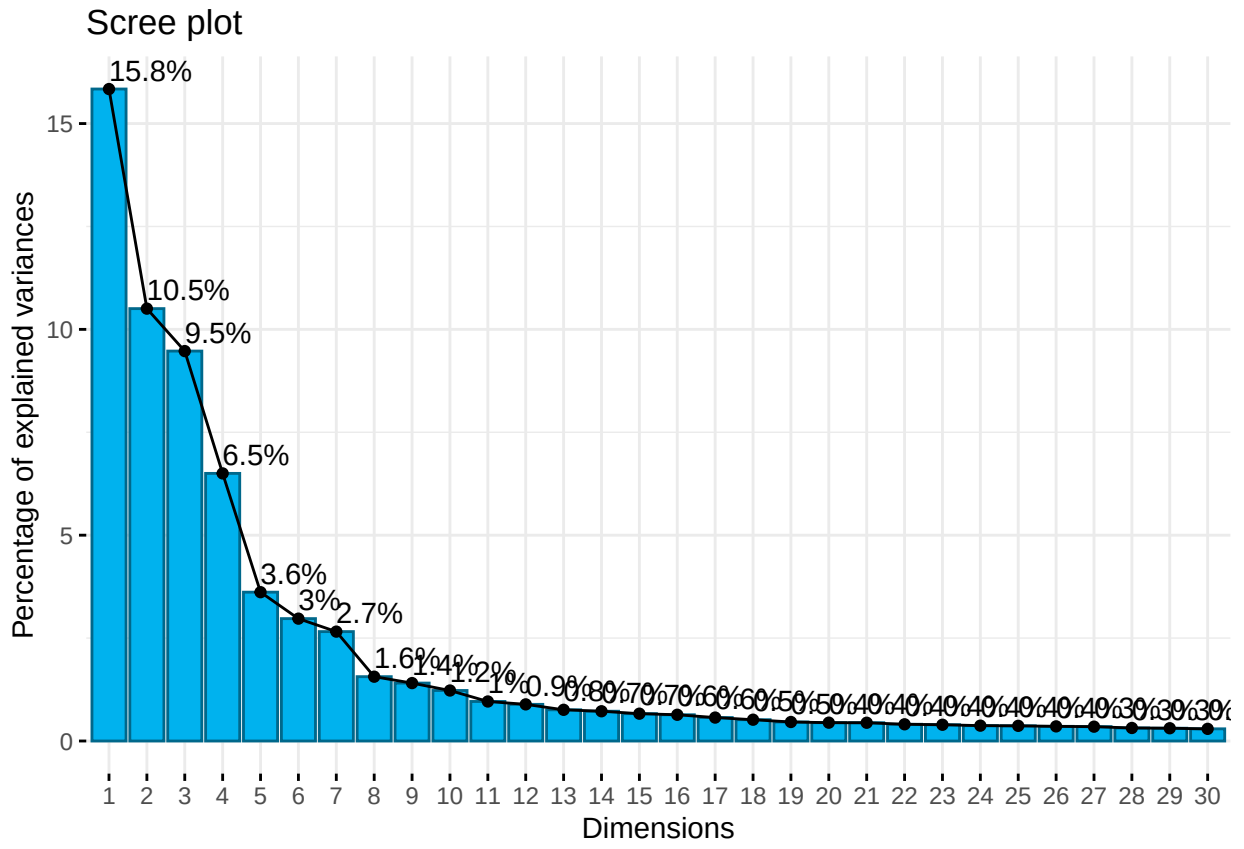
Response absolute frequencies:



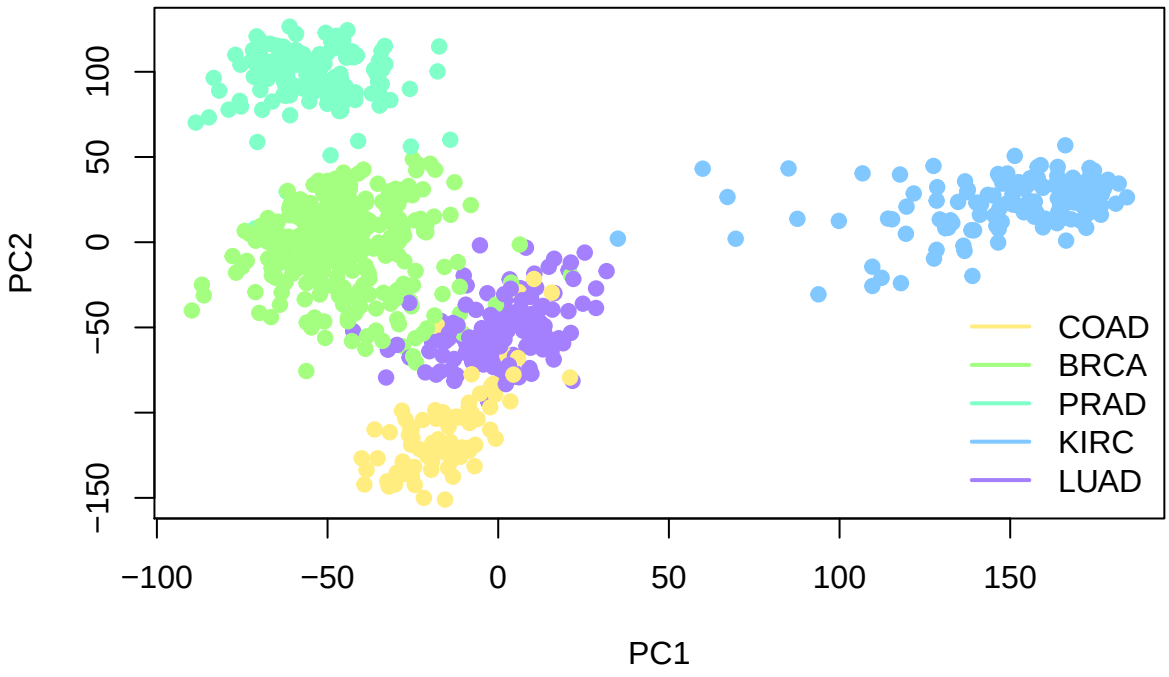
55% of the variability in the data is explained by 10 PCs:

```
## Loading required package: ggplot2
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```



First 2 PCs look enough to classify the types of cancer:



Lasso

Since p » n, the estimation of the coefficients in a linear model will suffer from high variance. A lasso model is fit in order to reduce the output error (introducing some bias but largely decreasing variance). Lasso also performs variable selection which helps with interpretability. The aim is to reduce the number of predictors from more than 20,000 to just a bunch.

80% observations are used to fit the model, the rest to measure the accuracy.

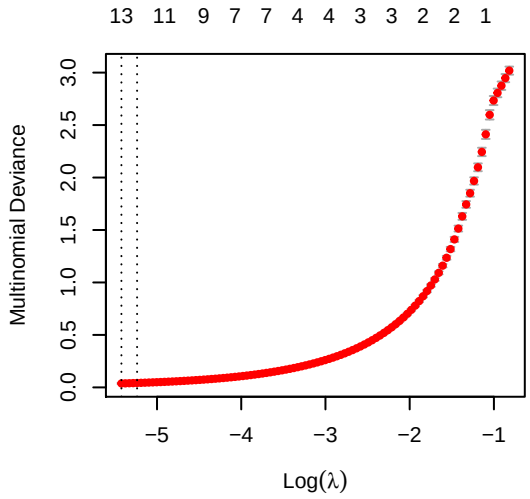
```
training_index = sample(1:nrow(genes), round(nrow(genes) * 0.8))

x_lasso_train = as.matrix(genes[training_index,])
y_lasso_train = as.numeric(labels[training_index])-1  # categories start from 0 (as expected by keras)
x_lasso_test = as.matrix(genes[-training_index,])
y_lasso_test = as.numeric(labels[-training_index])-1  # categories start from 0 (as expected by keras)
```

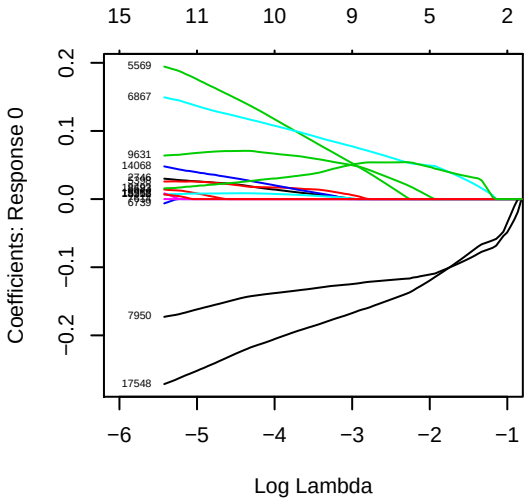
```
# no need to scale the data, glmnet does it by default
library(glmnet)
```

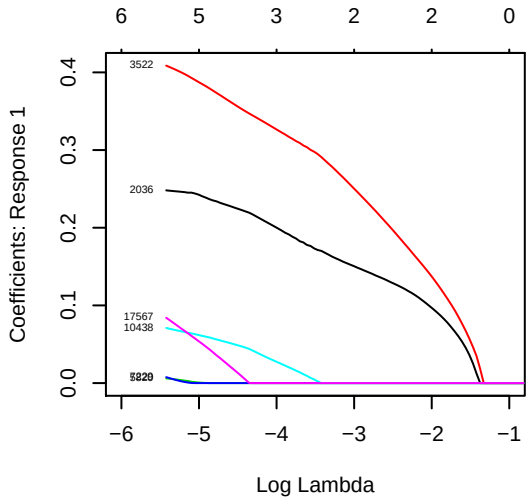
```
## Loading required package: Matrix
```

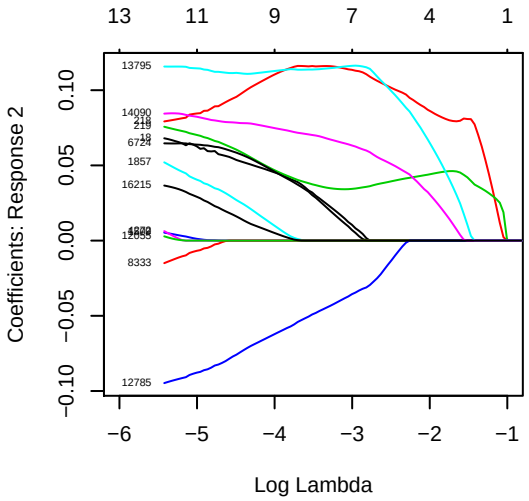
```
## Loaded glmnet 3.0-2
```

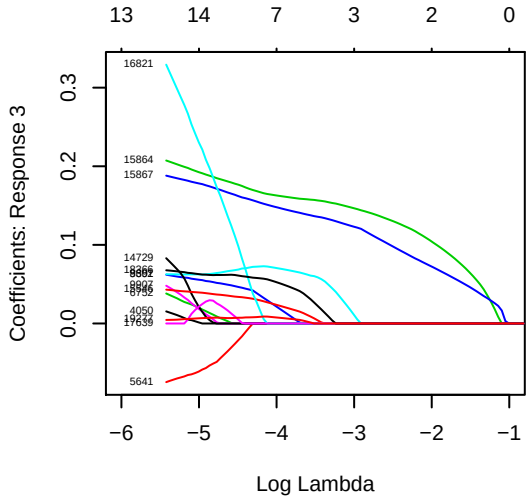
```
cvfit = cv.glmnet(x_lasso_train, y_lasso_train, alpha = 1, family = "multinomial")
coeffs = coef(cvfit, s = "lambda.min")

par(mfrow = c(3,2))
plot(cvfit)

fit = glmnet(x_lasso_train, y_lasso_train, alpha = 1, family = "multinomial")
plot(fit, xvar="lambda", label = TRUE, xlim=c(-6,-1))

#TODO: find a way to apply the text to all the 5 plots returned by plot.glmnet
text(log(cvfit$lambda.min), coeffs[["4"]]@x[-1], labels=colnames(x_lasso_test[,coeffs[["4"]]@i[-1]]), p
abline(v = log(cvfit$lambda.min))
```

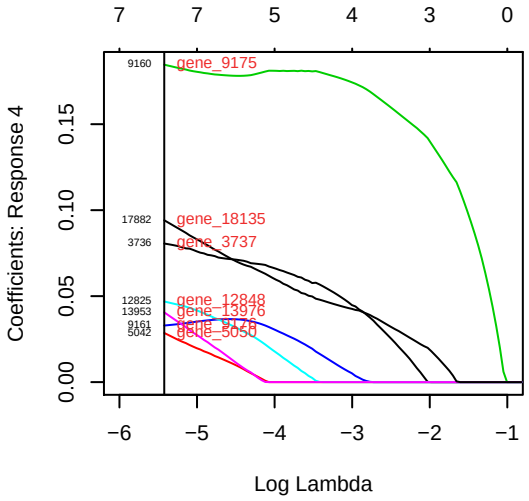

13 11 9 7 7 4 4 3 3 2 2 1
Multinomial Deviance
0.0 0.5 1.0 1.5 2.0 2.5 3.0
−5 −4 −3 −2 −1
Log(λ)


15 11 10 9 5 2
Coefficients: Response 0
−0.2 −0.1 0.0 0.1 0.2
5569
6867
9631
14068
7950
17548
−6 −5 −4 −3 −2 −1
Log Lambda


6 5 3 2 2 0
Coefficients: Response 1
0.0 0.1 0.2 0.3 0.4
3522
2036
17567
10438
−6 −5 −4 −3 −2 −1
Log Lambda


13 11 9 7 4 1
Coefficients: Response 2
−0.10 −0.05 0.00 0.05 0.10
13795
1857
16215
8333
12785
−6 −5 −4 −3 −2 −1
Log Lambda


13 14 7 3 2 0
Coefficients: Response 3
0.0 0.1 0.2 0.3
16821
15864
15867
14729
5641
−6 −5 −4 −3 −2 −1
Log Lambda


7 7 5 4 3 0
Coefficients: Response 4
0.00 0.05 0.10 0.15
9160
gene_9175
17882
gene_18135
3736
gene_3737
12825
gene_12848
13953
gene_13976
gene_8050
−6 −5 −4 −3 −2 −1
Log Lambda

0 BRCA

1 COAD

2 KIRC

3 LUAD

4 PRAD

The plots show the coefficient values for the selected predictors for each category. The higher the absolute value, the higher the influence (globally) in the output.
Note that the numbers in the plot correspond to column indices. In the last plot, for category 4 (PRAD), the name of the predictors are displayed in red too.

The accuracy of the test data prediction is close to 1. The categories barely overlap in the input space as seen in the PC1 vs PC2 plot which only accounts for 25% of the variability of the data, leading to the high accuracy.

```
test = predict(cvfit, newx = x_lasso_test, s = cvfit$lambda.min, type="response")
pred = max.col(as.data.frame(test))-1

mean(y_lasso_test == pred)
```

```
## [1] 0.9875
```

The mean output (using 0 for misclassifications):

```
label_output_prob = apply(test[,,1],1,max) * as.integer(as.integer(y_lasso_test) == pred)
mean(label_output_prob)
```

```
## [1] 0.9762241
```

Lasso shrinks less significant coefficients to 0 as λ increases (as in a linear optimization problem with constraint vertices in the predictor axes). With the optimal λ computed numerically.

The remaining significant predictors are:

```
sig_index = Reduce(union,c(coeffs[["0"]]@i,coeffs[["1"]]@i,coeffs[["2"]]@i,coeffs[["3"]]@i,coeffs[["4"]]
# coeffs is a list of dgCMatrix
# @i are indices of non-zero values in the matrix (first one corresponds to the y-intercept)
# @x are the coefficients corresponding to the indices

colnames(x_lasso_train[,sig_index])
```

```
##  [1] "gene_2747"  "gene_5407"  "gene_5578"  "gene_6748"  "gene_6876"
##  [6] "gene_7623"  "gene_7964"  "gene_8598"  "gene_9652"  "gene_14092"
## [11] "gene_15589" "gene_15999" "gene_17801" "gene_18465" "gene_18746"
## [16] "gene_2037"  "gene_3523"  "gene_5829"  "gene_7238"  "gene_10460"
## [21] "gene_17820" "gene_18"    "gene_219"   "gene_220"   "gene_1510"
## [26] "gene_1858"  "gene_4273"  "gene_6733"  "gene_8348"  "gene_12078"
## [31] "gene_12808" "gene_13818" "gene_14114" "gene_16246" "gene_4051"
## [36] "gene_5650"  "gene_6761"  "gene_8316"  "gene_9713"  "gene_9928"
## [41] "gene_14756" "gene_15577" "gene_15895" "gene_15898" "gene_17074"
```

```
## [46] "gene_18619" "gene_19542" "gene_3737"  "gene_5050"  "gene_9175"
## [51] "gene_9176"  "gene_12848" "gene_13976" "gene_18135"
```

Lasso coefficients heatmaps:

```
sparse = as.matrix(coeffs[["0"]])
sparse = cbind(sparse,as.matrix(coeffs[["1"]]))
sparse = cbind(sparse,as.matrix(coeffs[["2"]]))
sparse = cbind(sparse,as.matrix(coeffs[["3"]]))
sparse = cbind(sparse,as.matrix(coeffs[["4"]]))
colnames(sparse) = c("BRCA","COAD","KIRC","LUAD","PRAD")

sparse = sparse[-1,] #removing intercept row
coeff_matrix = sparse[rowSums(sparse) != 0,] #removing rows with all coeffs set to 0

col_breaks = c(seq(-0.34,-0.0001,length=100),  # for red
               seq(+0.0001,+0.34,length=100))  # for blue

library(unikn)
```

```
## Welcome to unikn (v0.2.0)!
```

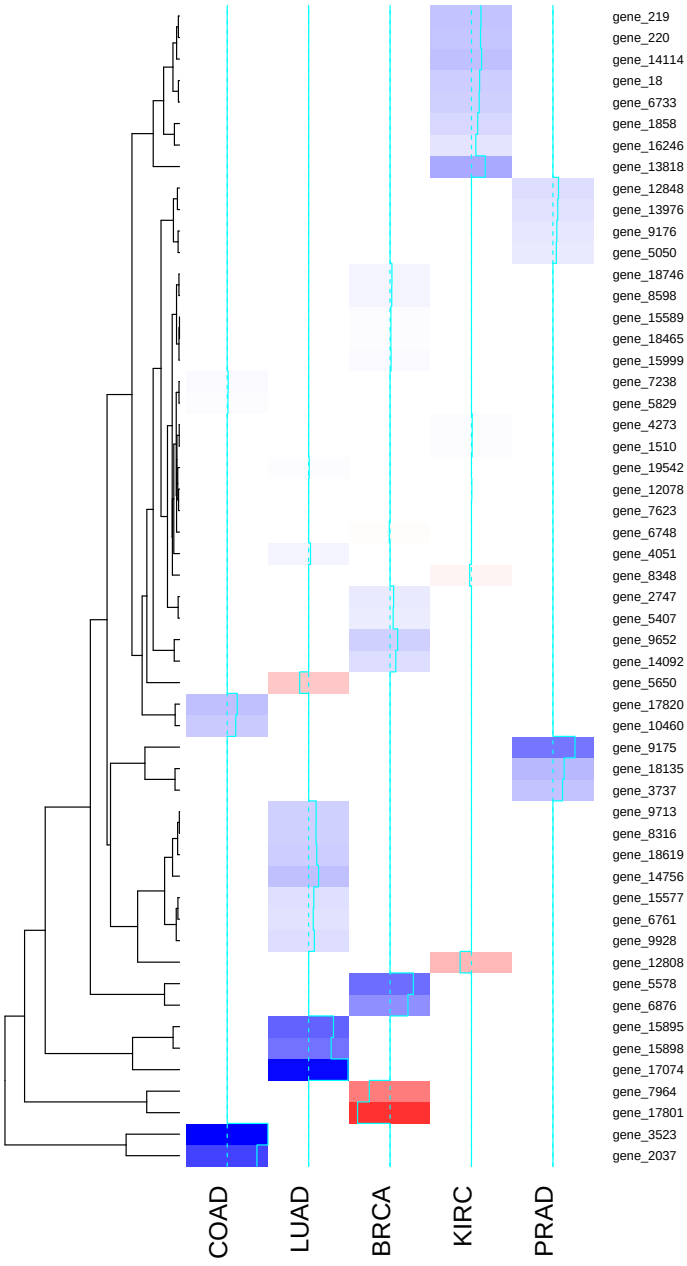
```
## unikn.guide() opens user guides.
```

```
library(gplots)
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
my_palette <- c(colorRampPalette(c("red","white"))(n = 99), "white", colorRampPalette(c("white","blue"))
heatmap.2(coeff_matrix, col= my_palette, breaks=col_breaks, dendrogram="row", symkey=FALSE, key=FALSE)
```

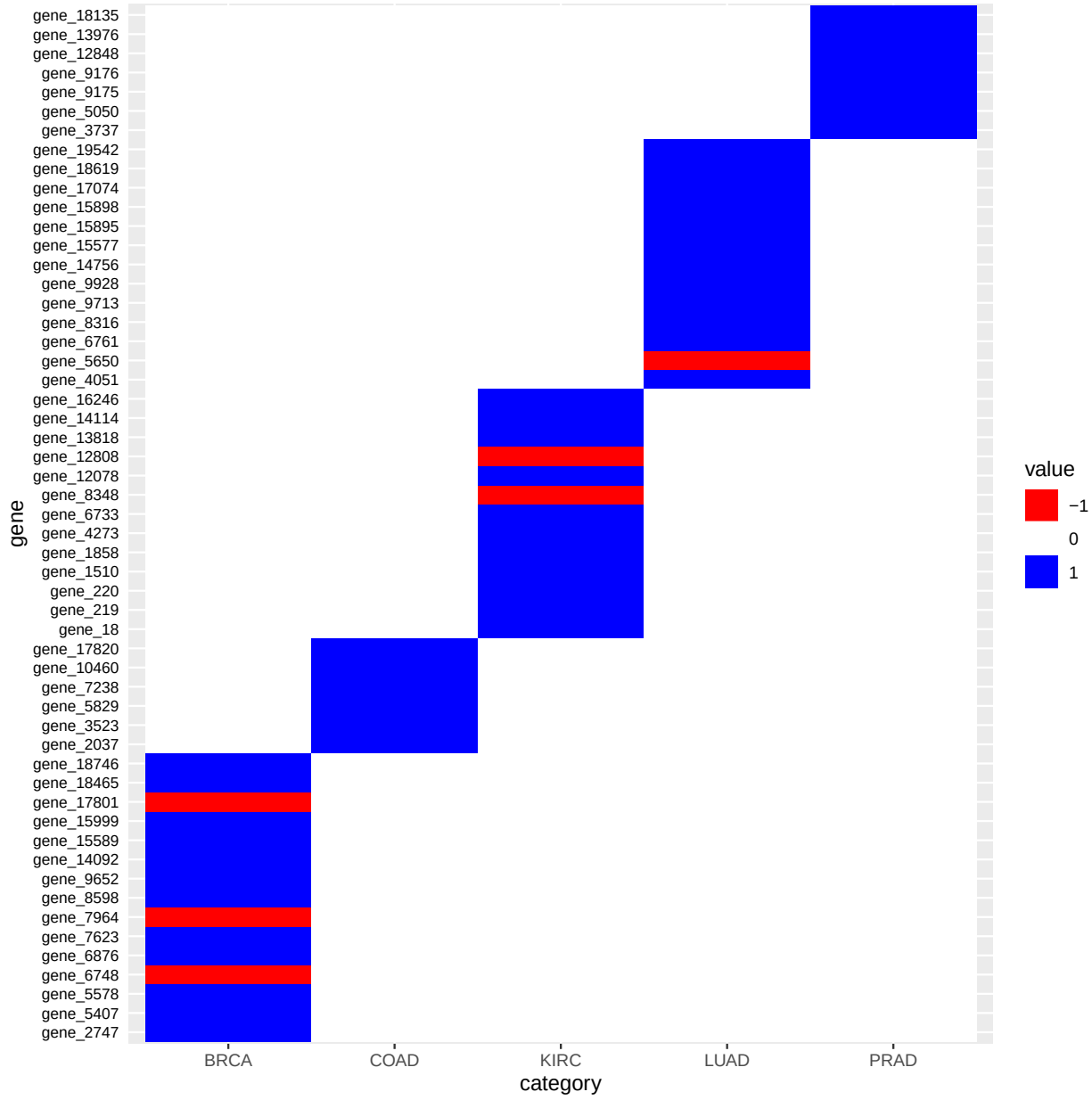

gene_219
gene_220
gene_14114
gene_18
gene_6733
gene_1858
gene_16246
gene_13818
gene_12848
gene_13976
gene_9176
gene_5050
gene_18746
gene_8598
gene_15589
gene_18465
gene_15999
gene_7238
gene_5829
gene_4273
gene_1510
gene_19542
gene_12078
gene_7623
gene_6748
gene_4051
gene_8348
gene_2747
gene_5407
gene_9652
gene_14092
gene_5650
gene_17820
gene_10460
gene_9175
gene_18135
gene_3737
gene_9713
gene_8316
gene_18619
gene_14756
gene_15577
gene_6761
gene_9928
gene_12808
gene_5578
gene_6876
gene_15895
gene_15898
gene_17074
gene_7964
gene_17801
gene_3523
gene_2037
COAD
LUAD
BRCA
KIRC
PRAD

```r
for(i in 1:nrow(coeff_matrix)){
  for(j in 1:ncol(coeff_matrix)){
    coeff_matrix[i,j] <- ifelse(coeff_matrix[i,j] < 0, -1,
                                ifelse(coeff_matrix[i,j] > 0, 1, 0))
  }
}

# Melt de hat_betas_fd + factores
library(reshape)
```

```
##
## Attaching package: 'reshape'
```

```
## The following object is masked from 'package:Matrix':
##
##     expand
```

```r
to_plot <- coeff_matrix
colnames(to_plot)  <- colnames(sparse)
aux <- c()
for(j in 1:ncol(coeff_matrix)){ # "Truco ordenación"
  aux <- c(aux, which(to_plot[,j]!=0))
}
aux <- rownames(to_plot)[aux]
aux <- unique(aux)
to_plot2 <- melt(to_plot)
colnames(to_plot2) <- c("gene","category","value")
to_plot2$gene <- factor(to_plot2$gene, levels = aux)
to_plot2$category <- factor(to_plot2$category)
to_plot2$value <- factor(to_plot2$value, levels = c(-1, 0, 1))

# Heatmap
library(ggplot2)
ggplot(to_plot2, aes(x = category, y = gene, fill = value)) +
  geom_tile() +
  scale_fill_manual(values = c("red", "white", "blue")) +
  theme(axis.text.y = element_text(color="black", size=8),
        axis.ticks.y.left = element_blank())
```


gene_18135
gene_13976
gene_12848
gene_9176
gene_9175
gene_5050
gene_3737
gene_19542
gene_18619
gene_17074
gene_15898
gene_15895
gene_15577
gene_14756
gene_9928
gene_9713
gene_8316
gene_6761
gene_5650
gene_4051
gene_16246
gene_14114
gene_13818
gene_12808
gene_12078
gene_8348
gene_6733
gene_4273
gene_1858
gene_1510
gene_220
gene_219
gene_18
gene_17820
gene_10460
gene_7238
gene_5829
gene_3523
gene_2037
gene_18746
gene_18465
gene_17801
gene_15999
gene_15589
gene_14092
gene_9652
gene_8598
gene_7964
gene_7623
gene_6876
gene_6748
gene_5578
gene_5407
gene_2747
gene
BRCA
COAD
KIRC
LUAD
PRAD
category
value
−1
0
1

Densely connected network

The problem of classifying patients into types of cancer from a large number of predictors is similar to the classic example of classifying short newswires into topics (reuters 1986 dataset). Both are multiclass classifications from a very large number of predictors (in the reuters example, each predictor represents the presence or absence of a particular word - tens of thousands of usual words are included).

There is no need to preprocess the data, the rows in our dataset are ready to fed the model as input vectors.

```
library(keras)

# This scaling is advised...
mean = apply(genes[training_index,], 2, mean)
sd = apply(genes[training_index,], 2, sd)
x_train = as.matrix(scale(genes[training_index,], center = mean, scale = sd))
x_test = as.matrix(scale(genes[-training_index,], center = mean, scale = sd))
# ...but this way the model diverges (loss=NA in the first iteration)
# TODO: why? #https://stackoverflow.com/questions/40050397/deep-learning-nan-loss-reasons

# scaling the union of training data and test data instead..
x_train = as.matrix(scale(genes)[training_index,])
x_test = as.matrix(scale(genes)[-training_index,])

y_train = to_categorical(as.numeric(labels[training_index])-1)
y_test = to_categorical(as.numeric(labels[-training_index])-1)
```

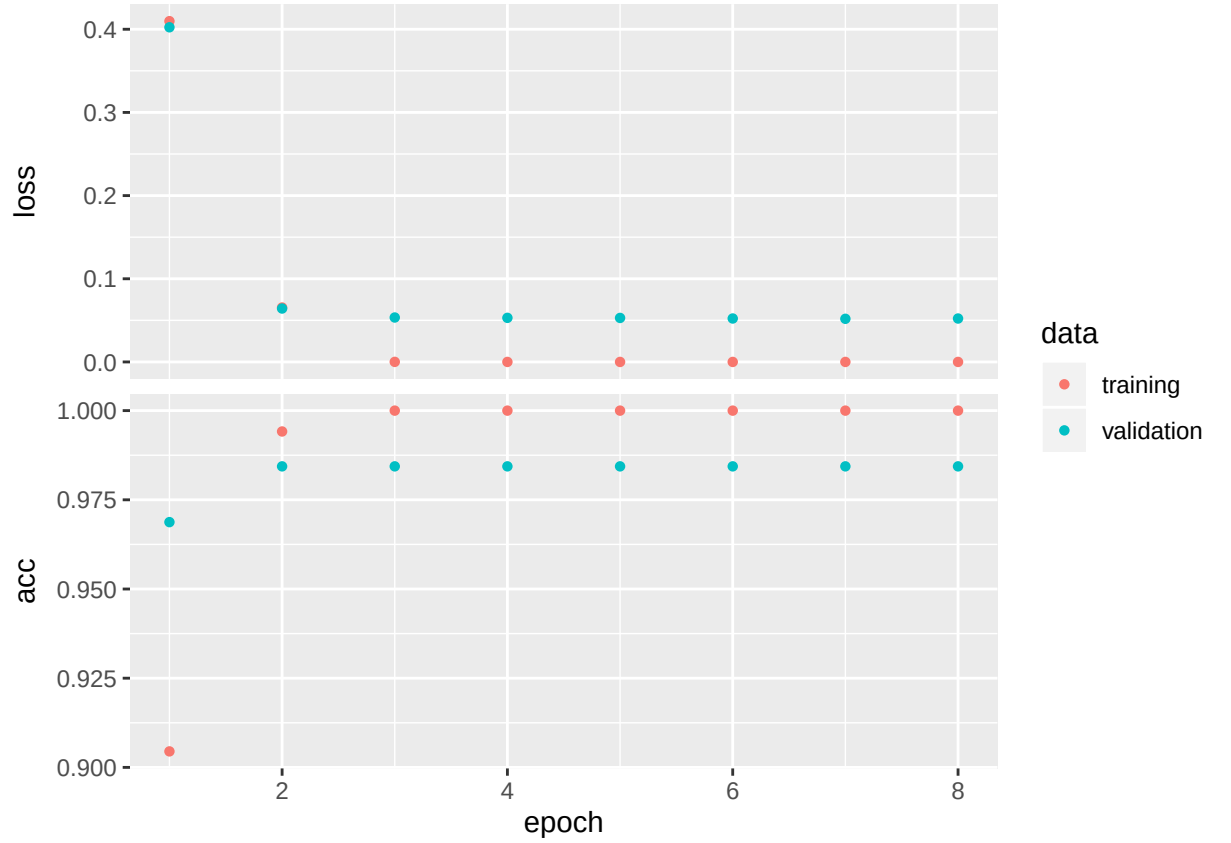
The model is trained in a matter of few seconds and yields an accuracy very close to 100%. No need for regularization or any other technique to help with the reduction of overfitting.

```
library(keras)

gmodel <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu", input_shape = ncol(x_train)) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 5, activation = "softmax")

#gmodel

set.seed(123)
gmodel %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",
  metrics = c("accuracy")
)

validation_index = sample(1:nrow(x_train), nrow(x_train)*0.2)

ghistory <- gmodel %>% fit(
  x_train[-validation_index,],
  y_train[-validation_index,],
  epochs = 8,
  batch_size = 32,
  validation_data = list(x_train[validation_index,],  y_train[validation_index,])
)

#print(ghistory)
```

```
(results <- gmodel %>% evaluate(x_test, y_test))
```

```
## $loss
## [1] 9.829161e-05
##
## $acc
## [1] 1
```

```
plot(ghistory)
```




```
output = gmodel %>% predict(x_test)
```

LIME

The Lasso model provides a global sense of the influence of the predictors. However if the data structure is complex it might not explain well the interpretation of particular predictions.

LIME (Local interpretable model-agnostic explanations) is a model-agnostic interpretability model that aims to explain better individual predictions by assuming that the data structure is linear around particular inputs. This technique simulates data around the input values by permuting predictor variables so there is enough data to fit a linear model localy.

glmnet is not supported by the LIME library (lime::?model_type).
Since LIME is model-agnostic and the keras model has almost 100% accuracy we'll use it for interpretation of the predictions.

```
library(lime)

#class(gmodel())

#?model_type

# Setup of lime::model_type()
model_type.keras.engine.sequential.Sequential <- function(x, ...) {"classification"}

# Setup of lime::predict_model()
predict_model.keras.engine.sequential.Sequential <- function (x, newdata, type, ...) {
  pred <- predict(object = x, x = as.matrix(newdata))
  data.frame(BRCA = pred[,1], COAD = pred[,2], KIRC = pred[,3], LUAD = pred[,4], PRAD = pred[,5])
}

predict_model (x = gmodel,
               newdata = as.data.frame(x_train),
               type    = 'raw') %>%
tibble::as_tibble()
```

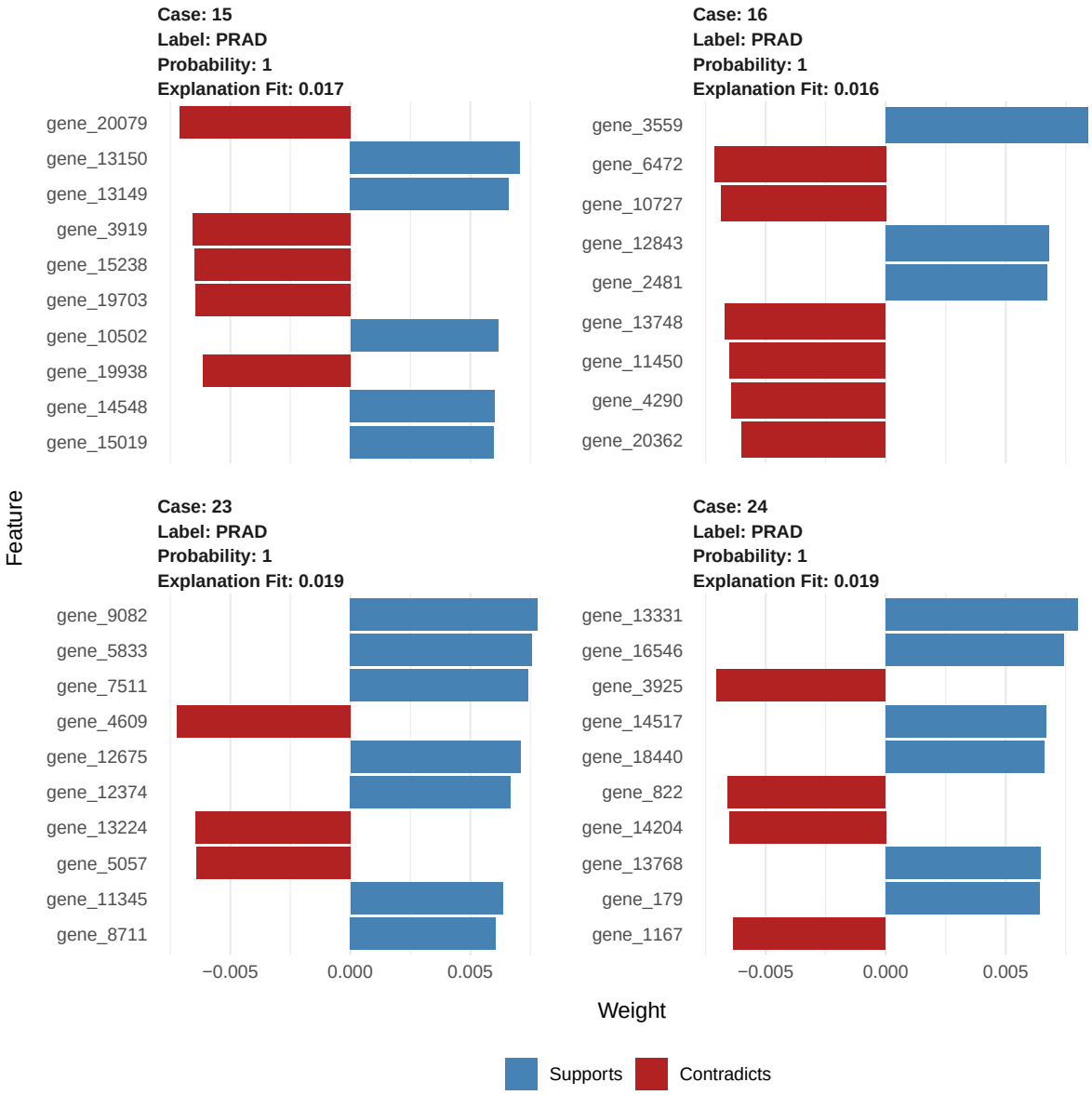
```
## # A tibble: 641 x 5
##        BRCA     COAD     KIRC     LUAD     PRAD
##       <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
##  1 1.98e- 9 1.25e- 9 2.34e-17 1.00e+ 0 5.99e-14
##  2 3.68e-18 2.85e-18 6.70e-22 1.00e+ 0 5.44e-22
##  3 1.00e+ 0 6.08e-28 5.87e-28 2.72e-24 7.88e-29
##  4 4.02e-10 1.63e-18 1.00e+ 0 6.25e-17 2.40e-13
##  5 1.15e-15 1.00e+ 0 2.06e-26 2.47e-15 3.19e-12
##  6 4.71e-15 1.00e+ 0 1.95e-27 1.46e-16 1.11e-14
##  7 1.04e-14 3.31e-18 2.25e-16 2.62e-16 1.00e+ 0
##  8 2.46e- 8 1.25e-16 1.00e+ 0 2.93e-12 8.86e-15
##  9 1.00e+ 0 7.17e-19 3.44e-15 4.43e-14 2.32e-16
## 10 1.85e-14 7.40e-17 6.24e-15 1.00e+ 0 2.12e-13
## # ... with 631 more rows
```

```
# will be used to create hte local model
explainer <- lime(
  x = as.data.frame(x_train),
  model = gmodel,
  bin_continuous = FALSE
)

#class(explainer)
```

```
#summary(explainer)
```

The predictions of data points of category PRAD are analysed:

```
set.seed(1)
datapoints_index = which(labels[-training_index] == "PRAD")[1:4]
#datapoints_index = sort(output[,5],index.return=T, decreasing = TRUE)$ix[1:2]

explanation_PRAD <- lime::explain(
  x = as.data.frame(x_test)[datapoints_index,],
  explainer = explainer,
  n_permutations = 10000,
  #dist_fun = "euclidean", #?dist()
  #kernel_width = 0.75,
  feature_select = "lasso_path",
  n_features = 10,
  n_labels = 1
)

plot_features(explanation_PRAD)
```


Case: 15
Label: PRAD
Probability: 1
Explanation Fit: 0.017
gene_20079
gene_13150
gene_13149
gene_3919
gene_15238
gene_19703
gene_10502
gene_19938
gene_14548
gene_15019
Case: 16
Label: PRAD
Probability: 1
Explanation Fit: 0.016
gene_3559
gene_6472
gene_10727
gene_12843
gene_2481
gene_13748
gene_11450
gene_4290
gene_20362
Case: 23
Label: PRAD
Probability: 1
Explanation Fit: 0.019
gene_9082
gene_5833
gene_7511
gene_4609
gene_12675
gene_12374
gene_13224
gene_5057
gene_11345
gene_8711
Case: 24
Label: PRAD
Probability: 1
Explanation Fit: 0.019
gene_13331
gene_16546
gene_3925
gene_14517
gene_18440
gene_822
gene_14204
gene_13768
gene_179
gene_1167
−0.005
0.000
0.005
Feature
Weight
Supports
Contradicts

Observations:

Each time the LIME model is run, the selected coefficients explaining the outputs are different (unless a seed is set before execution). The reason for this could come from:

- Not enough number of permutations for the large number of predictors we have (>20000), leading to the "simulated" data to be very different each time for the same data point. I tried increasing *n_permutations* from 5000 (default) to 25000 but the results keep changing (and not enough RAM - 32 GB - to increase the value more than that; takes very long too).
- High variance and low correlation between predictors as seen in the PCA analysis. R^2 of the local models ("explanation fit" in the LIME plots) is very low for all the data points analysed.

Different data points for the same category are explained by different predictors too.

Common genes between Lasso selected predictors and LIME explanation predictors:

```
# 0 BRCA
# 1 COAD
# 2 KIRC
# 3 LUAD
# 4 PRAD

# no lasso selected predictor among all the predictors returned by LIME
lime_feat = as.vector(explanation_PRAD[["feature"]])
lasso_feat = colnames(x_lasso_test[,coeffs[["4"]]@i])
intersect(lasso_feat,lime_feat)
```

```
## character(0)
```

Reasons for this could be:

- A poor LIME model as described above.
- The more influential genes explaining individual predictions are different from the genes selected by Lasso. Having high variance but non-overlapping categories could lead to this? As a large variety of predictors are able to classify well all the categories, so small differences in the input will change which are the more influential predictors, Lasso only reflecting on "aggregate".