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

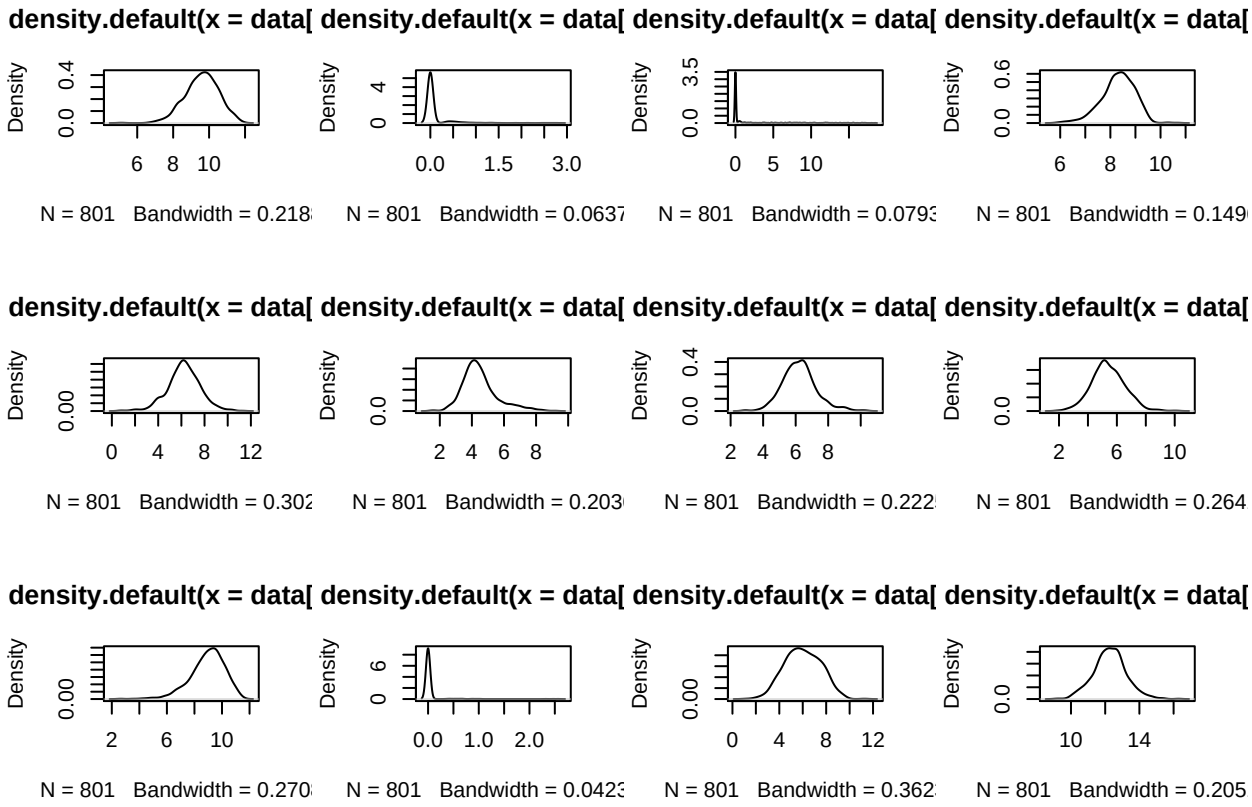
Number of observations: 801
Number of predictors: 20532

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

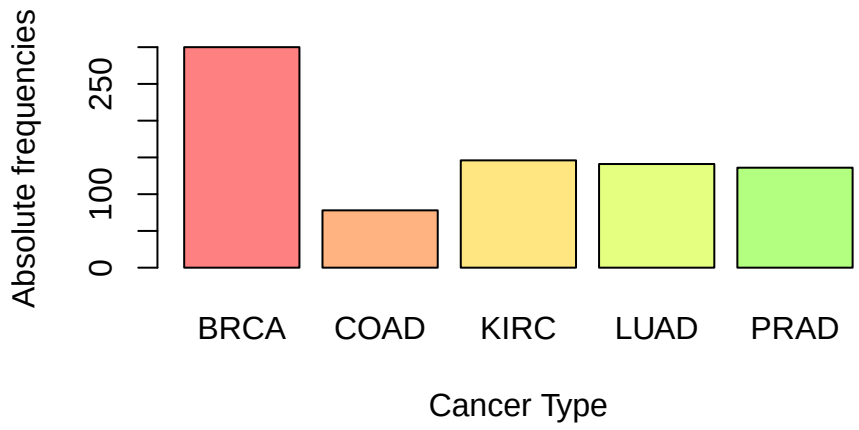
- lung adenocarcinoma (LUAD)
- breast carcinoma (BRCA)
- kidney renal clear-cell carcinoma (KIRC)
- colon adenocarcinoma (COAD)
- prostate adenocarcinoma (PRAD)

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

kde of 12 predictors picked at random:



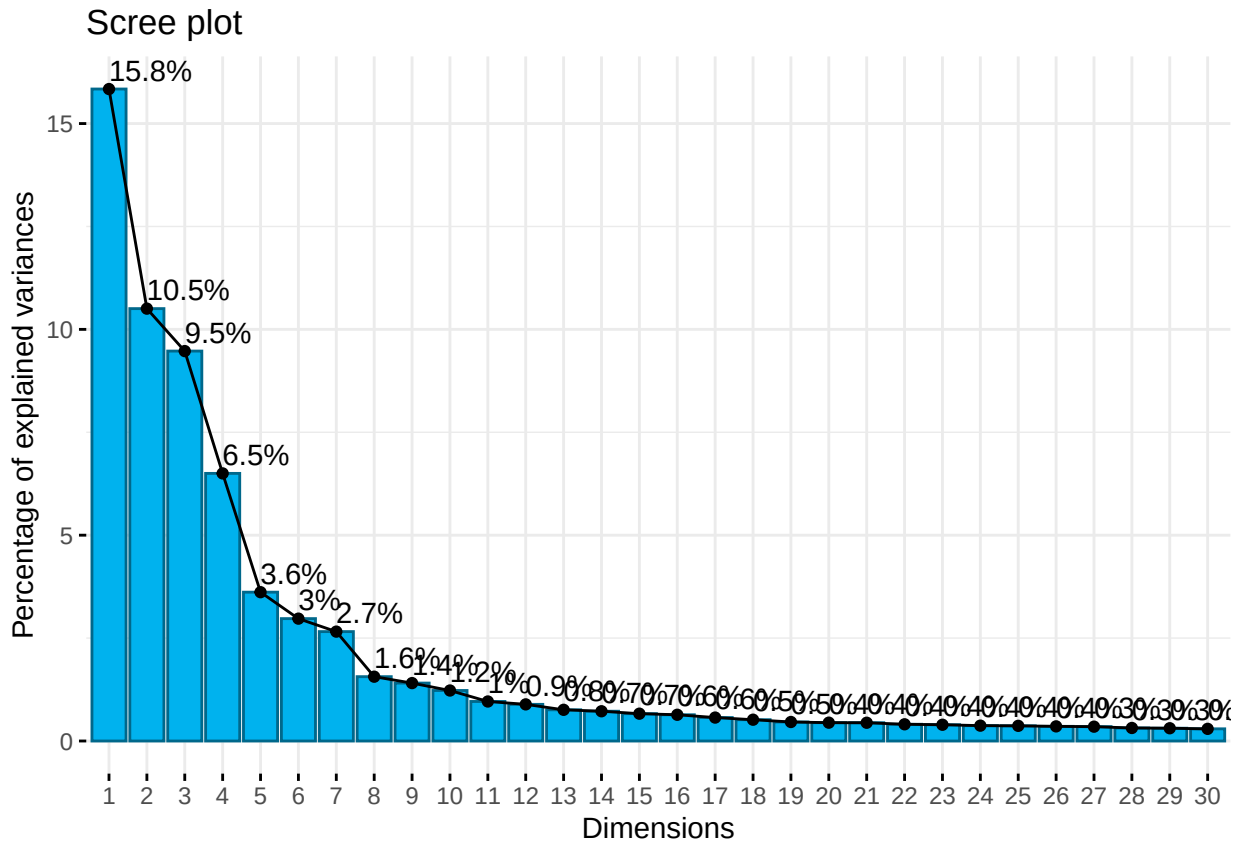
Response absolute frequencies:



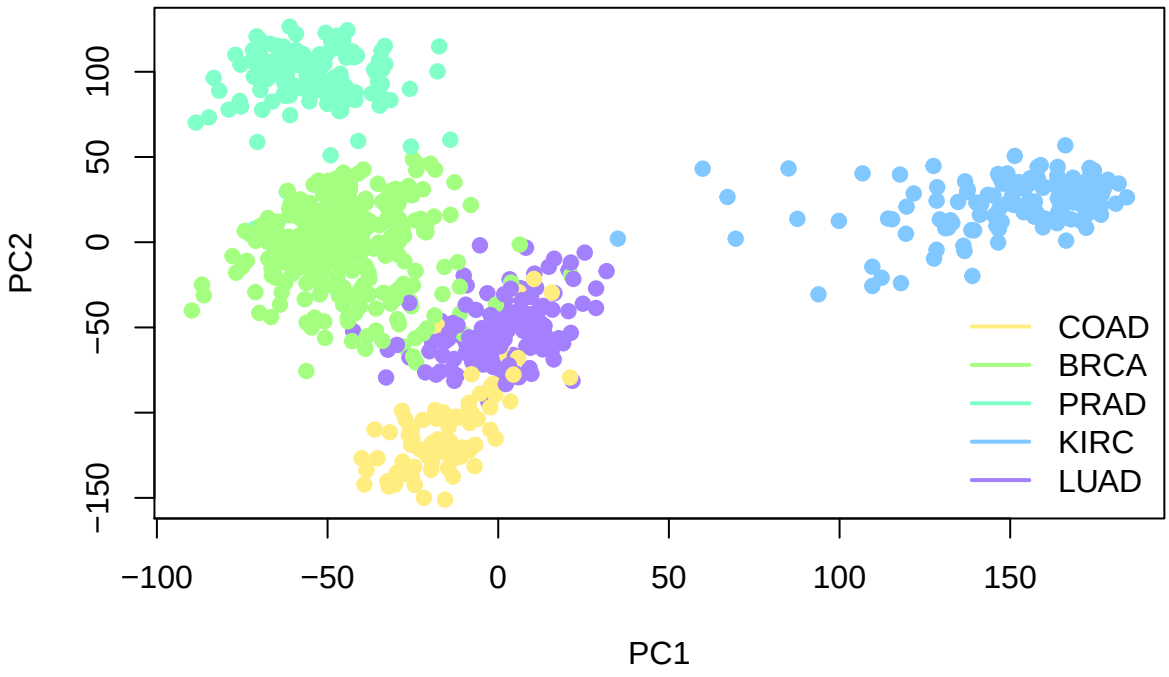
55% of the variability in the data is explained by 10 PCs:

```
## 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.

500 observations are used to fit the model, 300 to measure the accuracy.

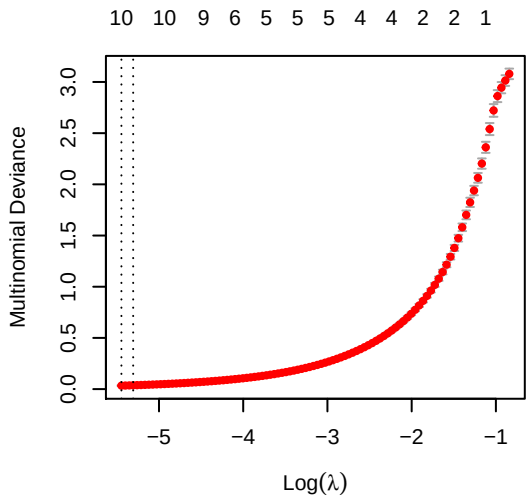
```
training_index = sample(1:nrow(data), 500)

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

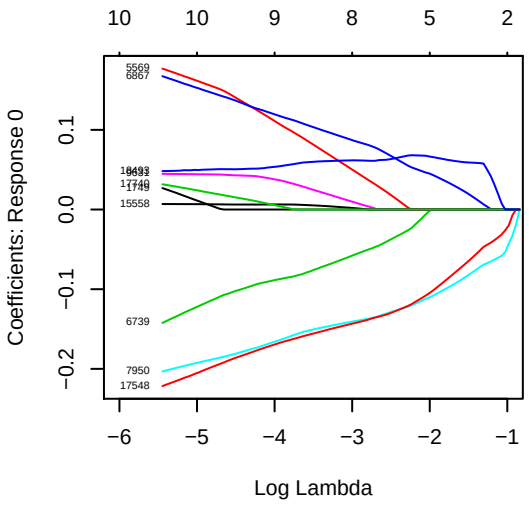
```
# no need to scale the data, glmnet does it by default
library(glmnet)
```

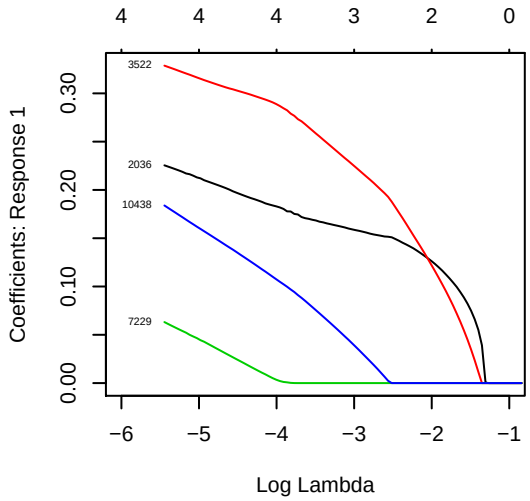
```
## Loading required package: Matrix
```

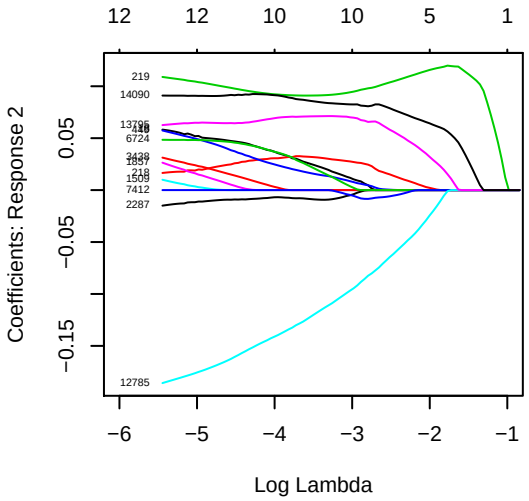
```
## Loaded glmnet 3.0-2
```

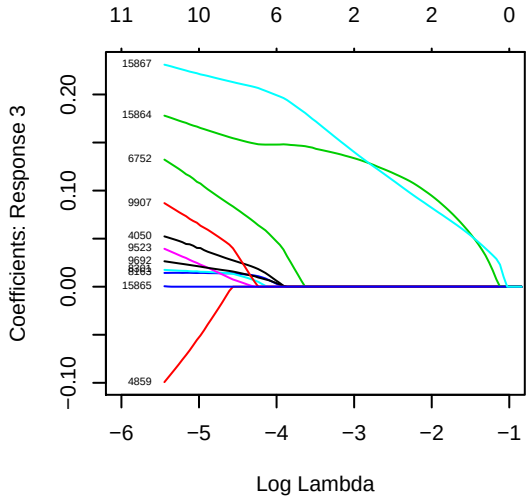
```
cvfit = cv.glmnet(x_lasso_train, y_lasso_train, alpha = 1, family = "multinomial")
coeffs = coef(cvfit, s = "lambda.min")

par(mfrow = c(3,2))
plot(cvfit)
# 0 BRCA
# 1 COAD
# 2 KIRC
# 3 LUAD
# 4 PRAD

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))
```

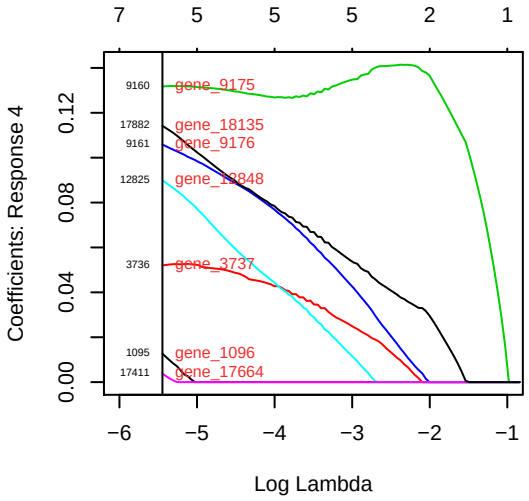

10 10 9 6 5 5 5 4 4 2 2 1
Multinomial Deviance
0.0 0.5 1.0 1.5 2.0 2.5 3.0
−5 −4 −3 −2 −1
Log(λ)


10 10 9 8 5 2
Coefficients: Response 0
0.1 0.0 −0.1 −0.2
5569
6867
15558
6739
7950
17548
−6 −5 −4 −3 −2 −1
Log Lambda


4 4 4 3 2 0
Coefficients: Response 1
0.30 0.20 0.10 0.00
3522
2036
10438
7229
−6 −5 −4 −3 −2 −1
Log Lambda


12 12 10 10 5 1
Coefficients: Response 2
0.05 −0.05 −0.15
219
14090
6724
7412
2287
12785
−6 −5 −4 −3 −2 −1
Log Lambda


11 10 6 2 2 0
Coefficients: Response 3
0.20 0.10 0.00 −0.10
15867
15864
6752
9907
4050
9523
15865
4859
−6 −5 −4 −3 −2 −1
Log Lambda


7 5 5 5 2 1
Coefficients: Response 4
0.12 0.08 0.04 0.00
9160
gene_9175
17882
gene_18135
9161
gene_9176
12825
3736
gene_3737
1095
gene_1096
17411
gene_17664
−6 −5 −4 −3 −2 −1
Log Lambda

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 very close to 1. The categories are very well "separated" from each other 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] 0.9900332
```

The mean output (using 0 for misclassifications):

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

```
## [1] 0.978587
```

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_1750"  "gene_5578"  "gene_6748"  "gene_6876"  "gene_7964"
##  [6] "gene_9652"  "gene_15589" "gene_17801" "gene_17993" "gene_18746"
## [11] "gene_2037"  "gene_3523"  "gene_7238"  "gene_10460" "gene_18"
## [16] "gene_219"   "gene_220"   "gene_450"   "gene_1510"  "gene_1858"
## [21] "gene_2288"  "gene_3439"  "gene_6733"  "gene_12808" "gene_13818"
## [26] "gene_14114" "gene_4051"  "gene_4867"  "gene_6761"  "gene_8178"
## [31] "gene_8316"  "gene_9544"  "gene_9713"  "gene_9928"  "gene_15895"
## [36] "gene_15896" "gene_15898" "gene_1096"  "gene_3737"  "gene_9175"
## [41] "gene_9176"  "gene_12848" "gene_17664" "gene_18135"
```

Lasso coefficients heatmap:

```
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)!
```

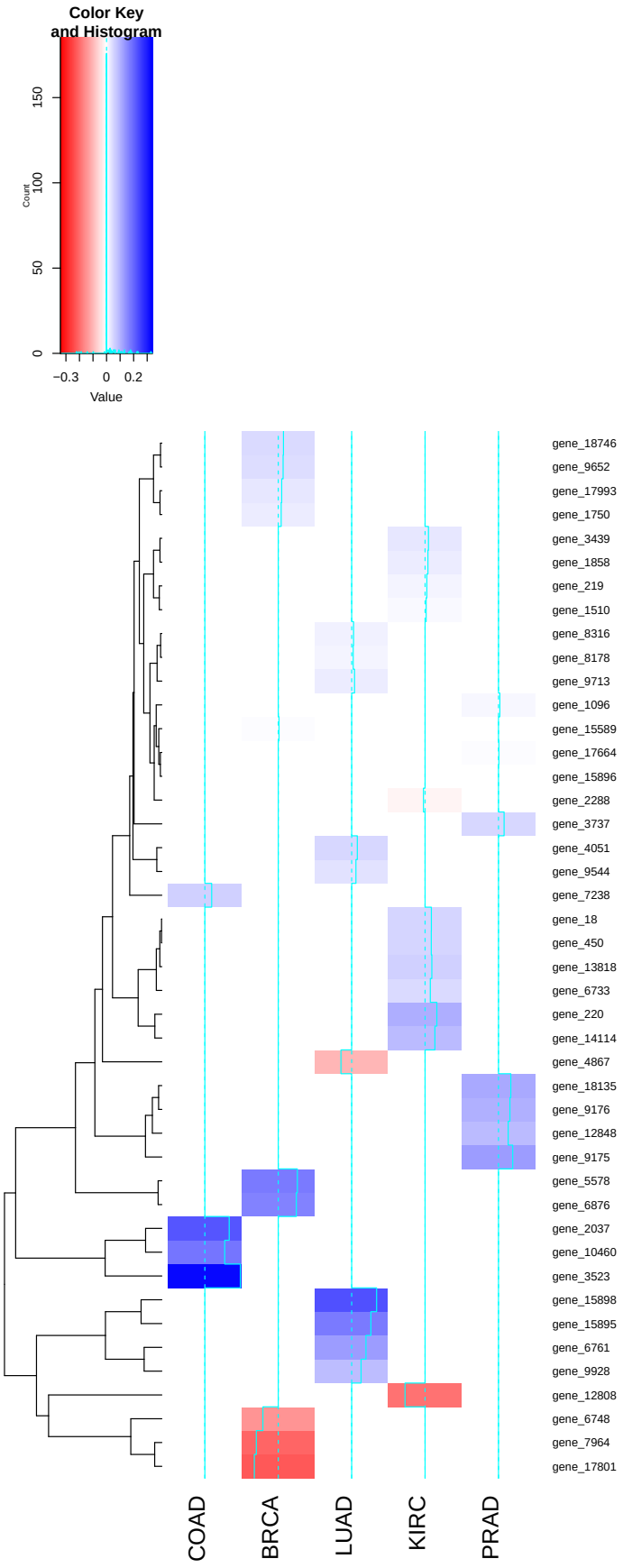
```
## unikn.guide() opens user guides.
```

```
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)
```


Color Key
and Histogram
Count
0
50
100
150
−0.3
0
0.2
Value
gene_18746
gene_9652
gene_17993
gene_1750
gene_3439
gene_1858
gene_219
gene_1510
gene_8316
gene_8178
gene_9713
gene_1096
gene_15589
gene_17664
gene_15896
gene_2288
gene_3737
gene_4051
gene_9544
gene_7238
gene_18
gene_450
gene_13818
gene_6733
gene_220
gene_14114
gene_4867
gene_18135
gene_9176
gene_12848
gene_9175
gene_5578
gene_6876
gene_2037
gene_10460
gene_3523
gene_15898
gene_15895
gene_6761
gene_9928
gene_12808
gene_6748
gene_7964
gene_17801
COAD
BRCA
LUAD
KIRC
PRAD

Fitting again the model but only including the significant predictors returned by the previous fit, plus all the 2-way interactions between the significant predictors:

```
f = as.formula(y ~ .*.)
y = y_lasso_train
x = model.matrix(f, data[training_index,sig_index])[,-1] #first column is the intersect - it's removed
cvfit_inter = cv.glmnet(x, y, alpha=1, family="multinomial")
coeffs_inter = coef(cvfit_inter, s = "lambda.min")

sig_index_inter = Reduce(union,c( coeffs_inter[["0"]]@i,
                                  coeffs_inter[["1"]]@i,
                                  coeffs_inter[["2"]]@i,
                                  coeffs_inter[["3"]]@i,
                                  coeffs_inter[["4"]]@i))

colnames(x[,sig_index_inter])
```

```
##  [1] "gene_6748"             "gene_17801"            "gene_1750:gene_6876"
##  [4] "gene_1750:gene_18746"  "gene_5578:gene_6876"   "gene_5578:gene_9652"
##  [7] "gene_6748:gene_17801"  "gene_6876:gene_9652"   "gene_6876:gene_18746"
## [10] "gene_6876:gene_6761"   "gene_7964:gene_6761"   "gene_17993:gene_2288"
## [13] "gene_18746:gene_4867"  "gene_2037"             "gene_6748:gene_3523"
## [16] "gene_7964:gene_10460"  "gene_2037:gene_12808"  "gene_3523:gene_7238"
## [19] "gene_7238:gene_10460"  "gene_10460:gene_8316"  "gene_2288"
## [22] "gene_12808"            "gene_6876:gene_1858"   "gene_6876:gene_14114"
## [25] "gene_7964:gene_18"     "gene_220:gene_6761"    "gene_220:gene_8316"
## [28] "gene_450:gene_8316"    "gene_1510:gene_4867"   "gene_1858:gene_3439"
## [31] "gene_1858:gene_6733"   "gene_1858:gene_13818"  "gene_2288:gene_6761"
## [34] "gene_6733:gene_13818"  "gene_1750:gene_15895"  "gene_12808:gene_15896"
## [37] "gene_12808:gene_15898" "gene_4051:gene_8316"   "gene_4051:gene_9713"
## [40] "gene_6761:gene_15895"  "gene_6761:gene_15898"  "gene_8178:gene_9713"
## [43] "gene_8316:gene_9928"   "gene_8316:gene_15898"  "gene_9175"
## [46] "gene_9176"             "gene_12848"            "gene_6748:gene_9175"
## [49] "gene_6748:gene_9176"   "gene_18746:gene_18135" "gene_1096:gene_9176"
## [52] "gene_9175:gene_9176"   "gene_9176:gene_17664"
```

The performance is similar - very high:

```
f_test <- as.formula(y_lasso_test ~ .*.)
x_test <- model.matrix(f_test, data[-training_index,sig_index])[,-1] #first column is the intersect - i

test_inter = predict(cvfit_inter, newx = x_test, s = cvfit_inter$lambda.min, type="response")
pred_inter = max.col(as.data.frame(test_inter))-1

mean(y_lasso_test == pred_inter)
```

```
## [1] 0.9933555
```

```
mean(apply(test_inter[,,1],1,max) * as.integer(as.integer(y_lasso_test) == pred_inter))
```

```
## [1] 0.9809774
```

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(data[training_index,], 2, mean)
sd = apply(data[training_index,], 2, sd)
x_train = as.matrix(scale(data[training_index,], center = mean, scale = sd))
x_test = as.matrix(scale(data[-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(data)[training_index,])
x_test = as.matrix(scale(data)[-training_index,])

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

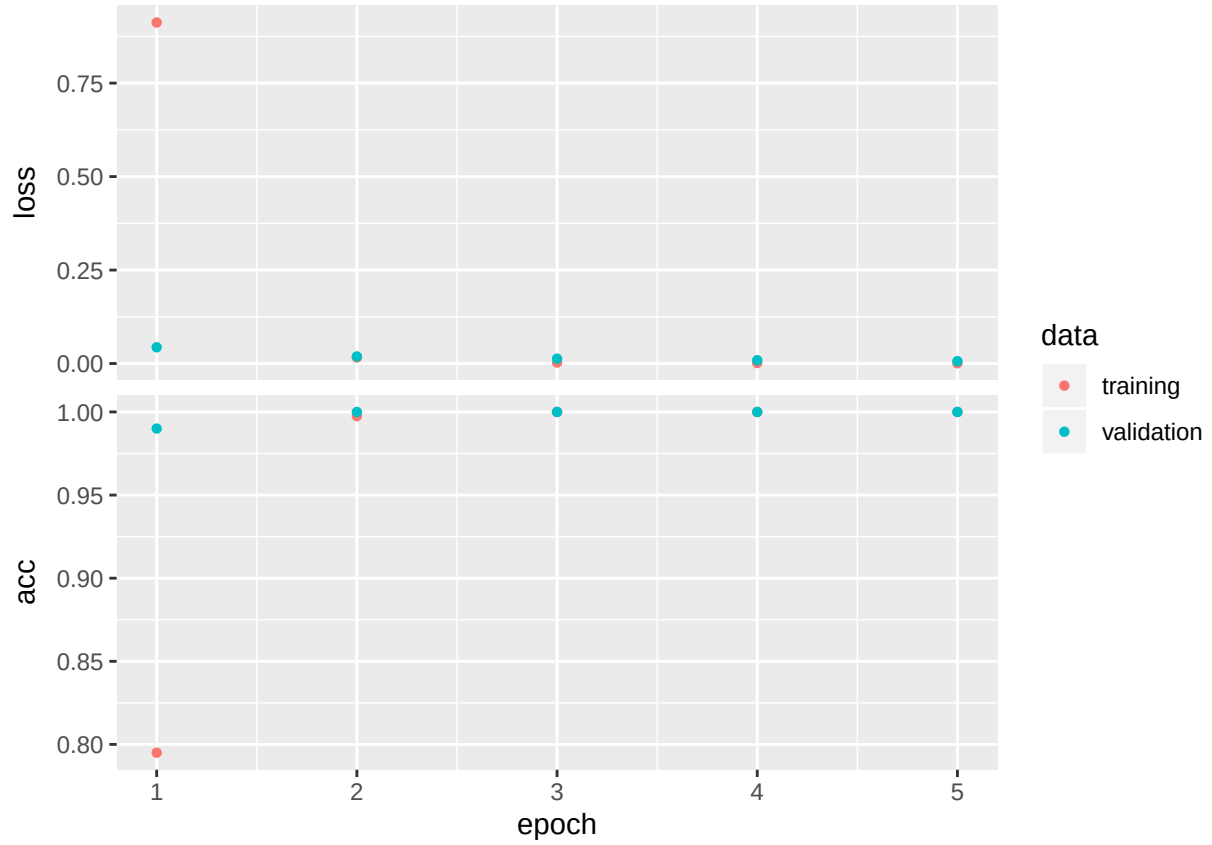
The model is trained in a matter of few seconds and yields an accurucy very close to 100%. No need for regularization or any other technique to help with the reduction of overfitting.

```
library(keras)

gmodel <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = ncol(x_train)) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 5, activation = "softmax")

#gmodel

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

validation_index = sample(1:nrow(x_train), 100)

set.seed(1)

ghistory <- gmodel %>% fit(
  x_train[-validation_index,],
  y_train[-validation_index,],
  epochs = 5,
  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] 0.02042744
##
## $acc
## [1] 0.9933555
```

```
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()
```

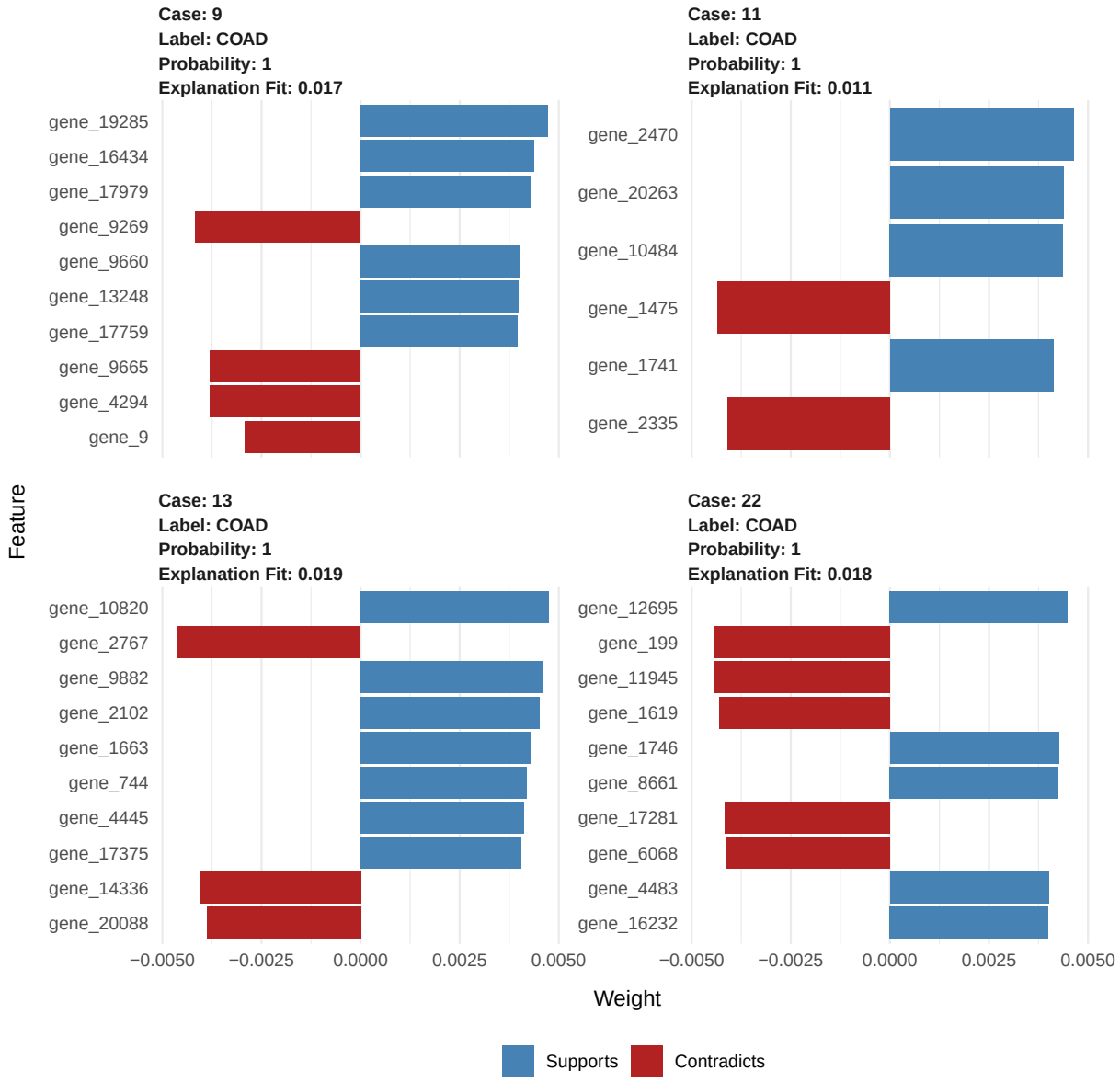
```
## # A tibble: 500 x 5
##         BRCA          COAD      KIRC          LUAD     PRAD
##        <dbl>         <dbl>     <dbl>         <dbl>    <dbl>
##  1 10.00e- 1 0.00000130     1.87e-10 0.000374      1.69e-15
##  2  6.76e-11 0.0000135      5.34e-12 1.000         2.57e- 5
##  3  4.07e- 7 0.000316      10.00e- 1 0.0000795     1.45e- 6
##  4  1.18e-10 0.000000114    4.55e-19 1.000         2.93e-12
##  5 10.00e- 1 0.000000153    4.59e-10 0.000000372   3.18e-17
##  6  7.01e- 8 0.000167      10.00e- 1 0.00000292    2.03e- 7
##  7 10.00e- 1 0.00000000142  3.29e- 9 0.000000417   1.91e-17
##  8  7.55e-14 0.00000456     3.65e-16 1.000         1.96e- 4
##  9  7.01e-17 1              4.46e-15 0.00000000118 7.11e-10
## 10  9.50e- 6 0.0000824     10.00e- 1 0.0000662     3.37e- 7
## # ... with 490 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)
```

4 data points of category COAD are analysed - COAD contains the highest coefficient in Lasso and the fewer number of selected predictors.

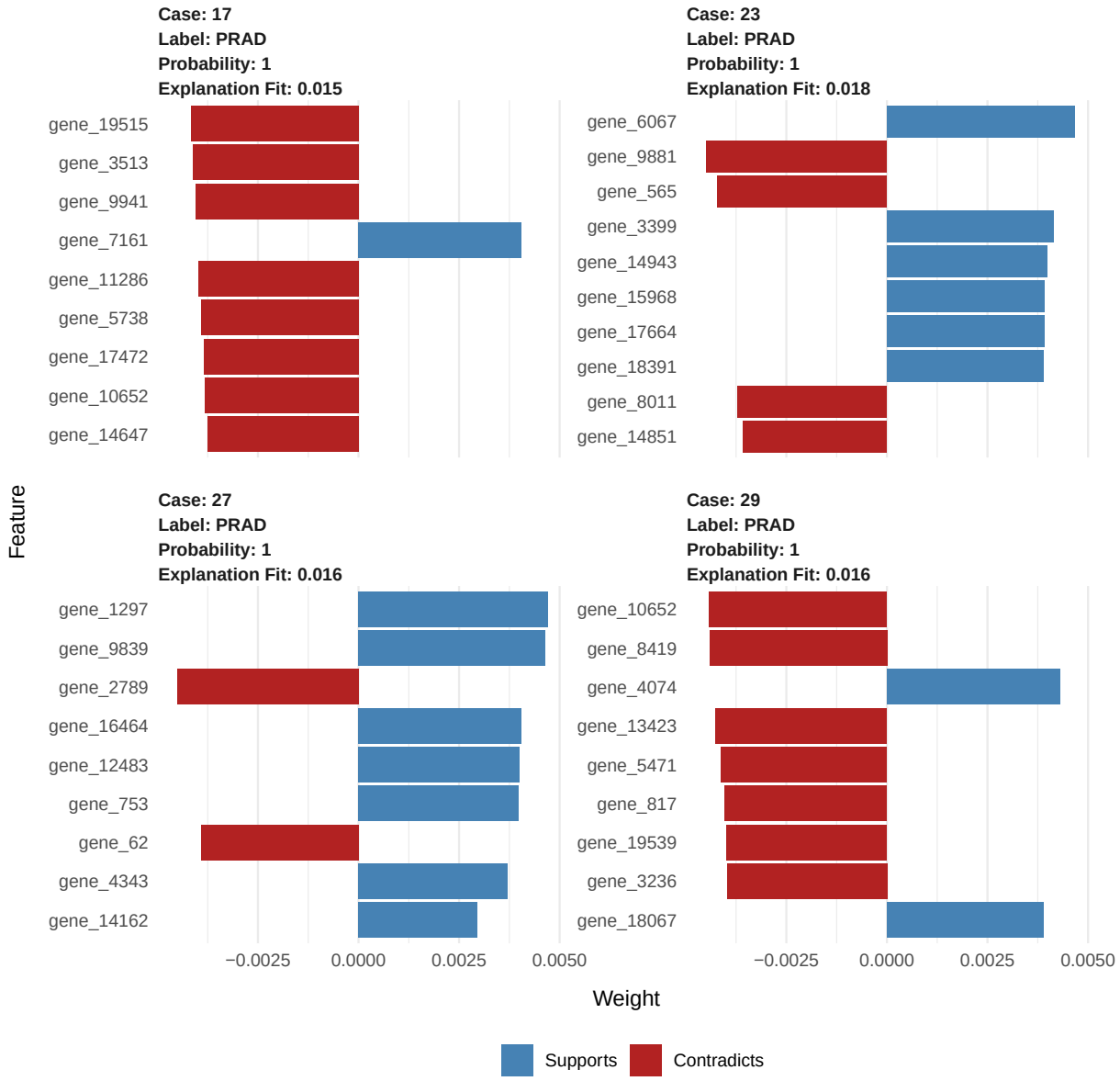
```
set.seed(1)
datapoints_index = which(labels[training_index] == "COAD")[1:4]

explanation_COAD <- lime::explain(
  x = as.data.frame(x_train)[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_COAD)
```



The same for 4 data points of category PRAD:

```
set.seed(1)
datapoints_index = which(labels[training_index] == "PRAD")[1:4]

explanation_PRAD <- lime::explain(
  x = as.data.frame(x_train)[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)
```



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.

No common genes are found between Lasso selected predictors and LIME explanation predictors (tried several times):

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

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

```
lime_feat = as.vector(explanation_PRAD[["feature"]])
lasso_feat = colnames(x_lasso_test[,sig_index])
intersect(lasso_feat,lime_feat)
```

```
## [1] "gene_17664"
```

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".