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

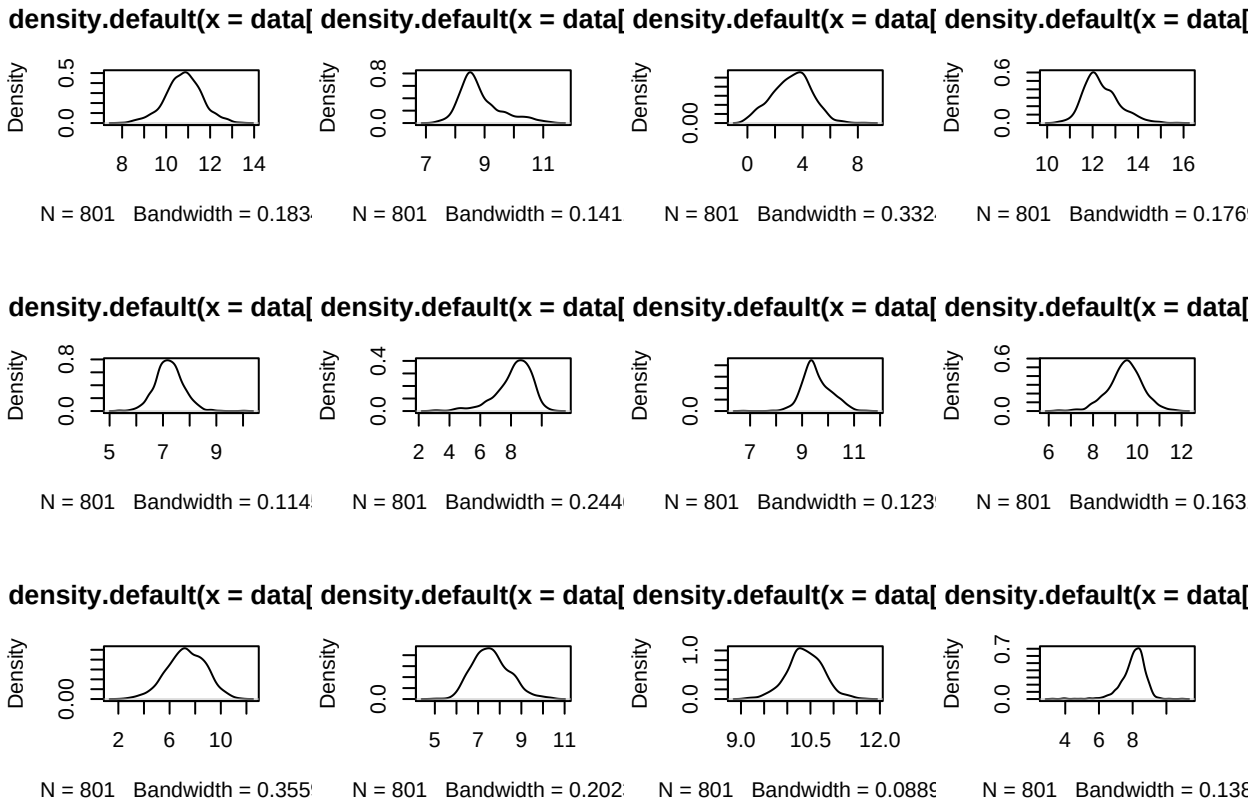
Number of observations: 801
Number of predictors: 20532

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

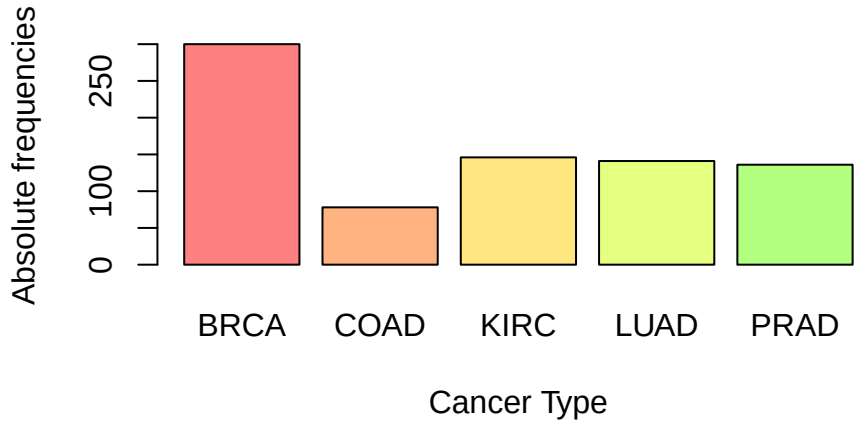
- lung adenocarcinoma (LUAD)
- breast carcinoma (BRCA)
- kidney renal clear-cell carcinoma (KIRC)
- colon adenocarcinoma (COAD)
- prostate adenocarcinoma (PRAD)

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

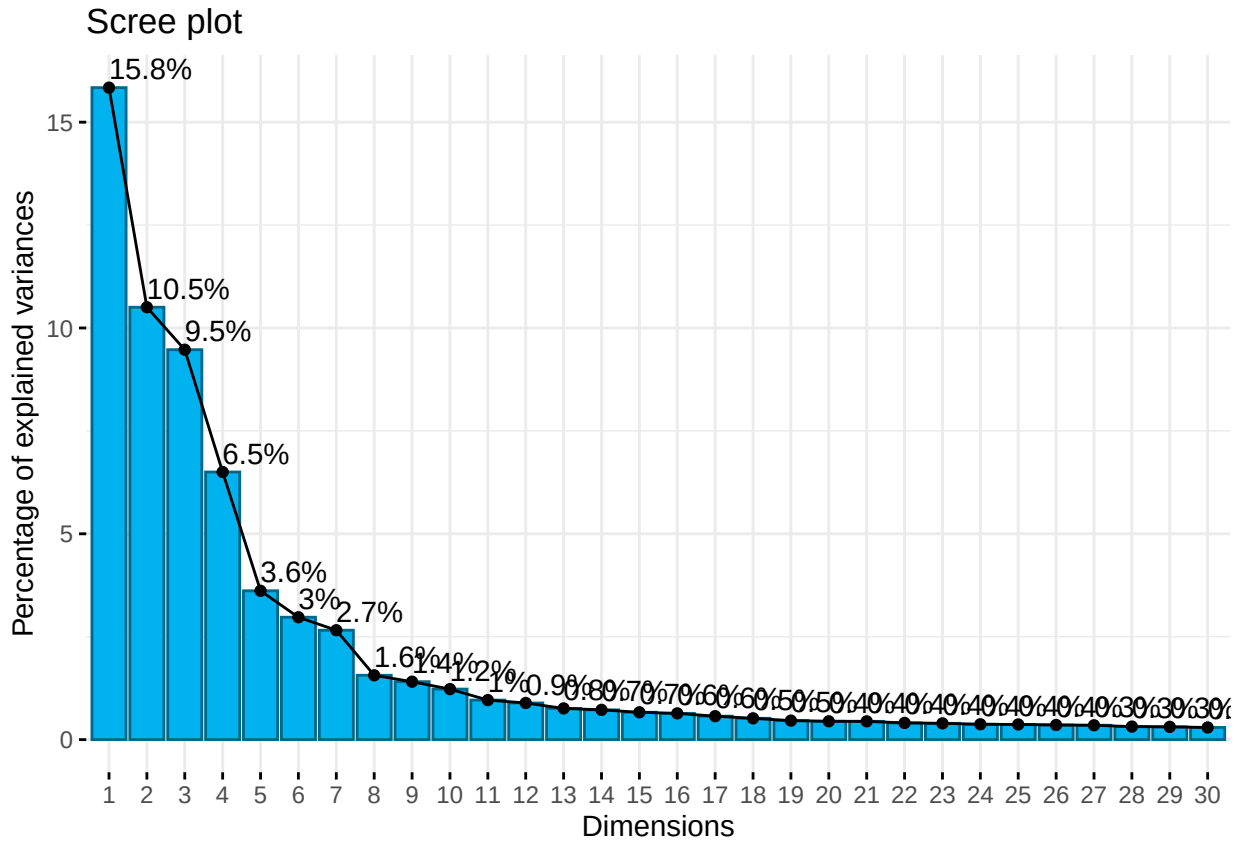
kde of 12 predictors picked at random:



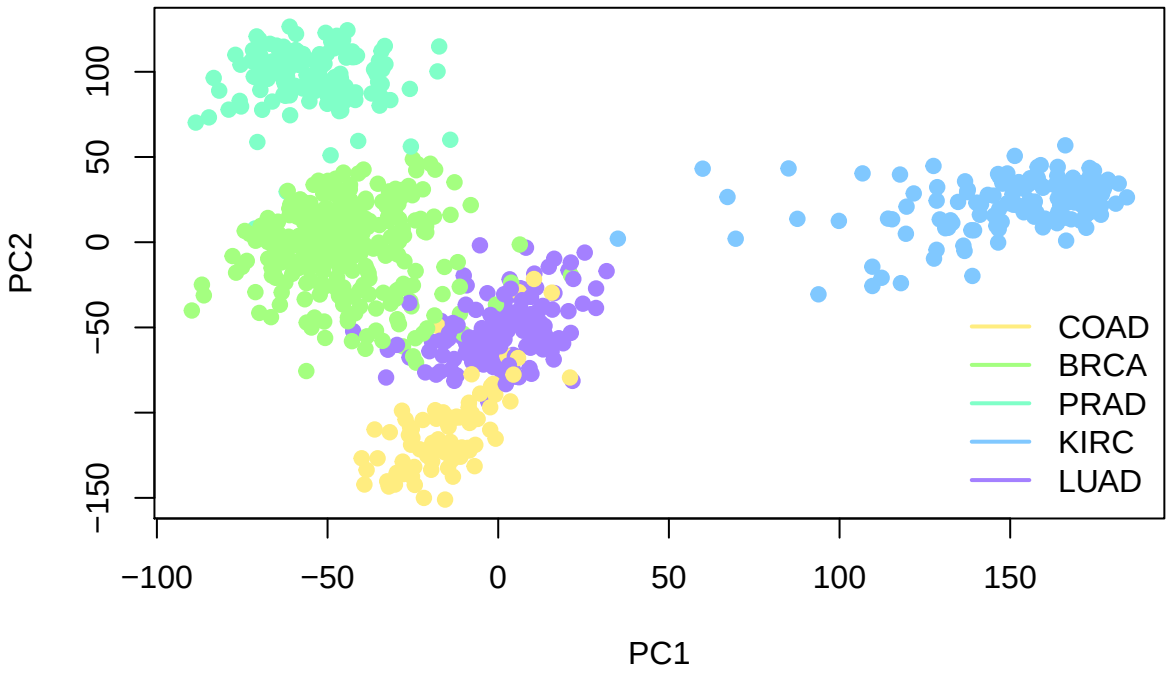
Response absolute frequencies:



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



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



pred_range = apply(data, 2, range) plot(density(pred_range[2,] - pred_range[1,]), main="range width")

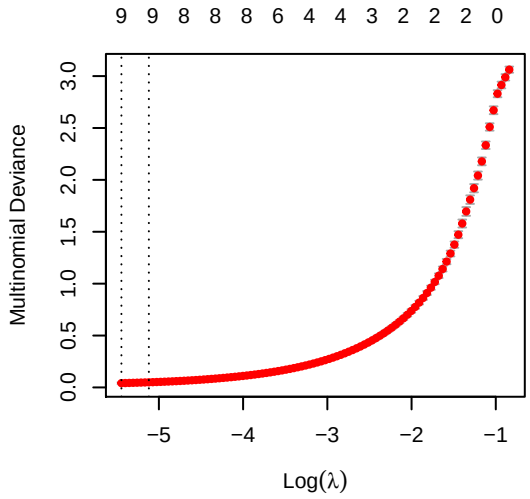
Lasso

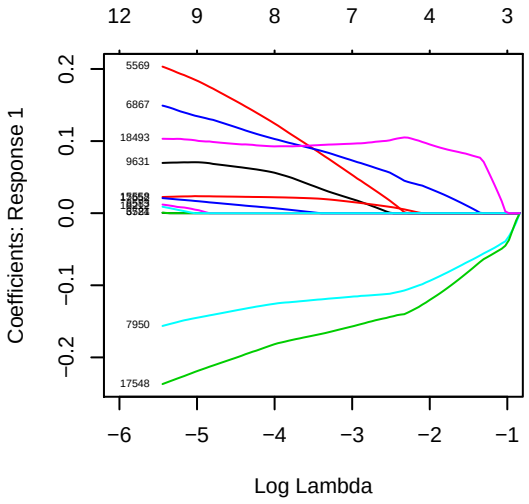
Since p » n, the estimation of the coefficients in a linear model will suffer from high variance. A lasso model is fit in order to reduce the output error (introducing some bias but largely decreasing variance). Lasso also performs variable selection which helps with interpretability. The aim is to reduce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

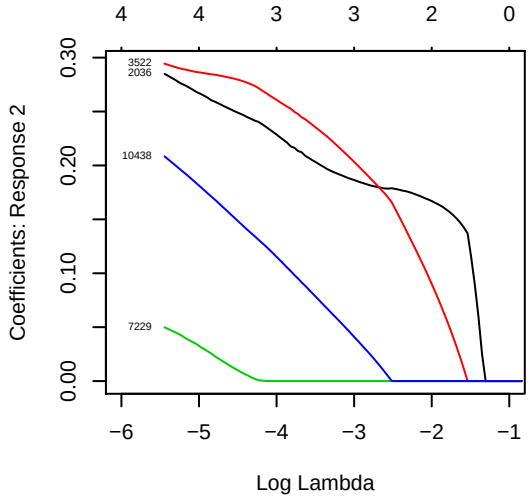
500 observations are used to fit the model, 300 to measure the accuracy.

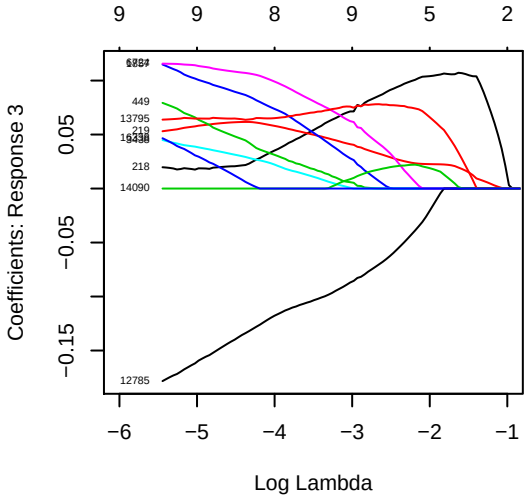
```
training_index = sample(1:nrow(data), 500)

x_lasso_train = as.matrix(data[training_index,])
y_lasso_train = as.numeric(labels[training_index])
x_lasso_test = as.matrix(data[-training_index,])
y_lasso_test = as.numeric(labels[-training_index])

set.seed(1)
```

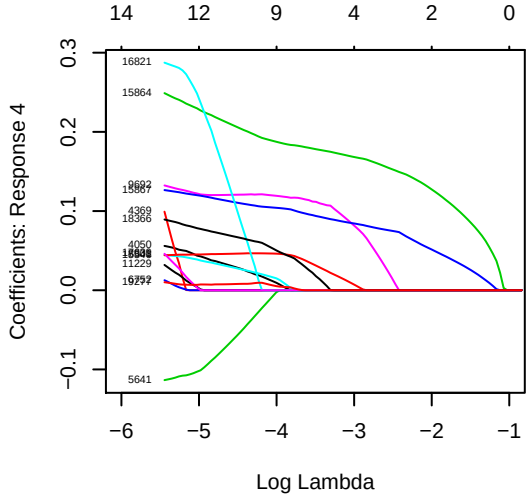
```
# no need to scale the data, glmnet does it by default
cvfit = cv.glmnet(x_lasso_train, y_lasso_train, alpha = 1, family = "multinomial")
coeffs = coef(cvfit, s = "lambda.min")

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$lambda.min), coeffs[["5"]]@x[-1], labels=colnames(x_lasso_test[,coeffs[["5"]]@i[-1]]), po
abline(v = log(cvfit$lambda.min))
```

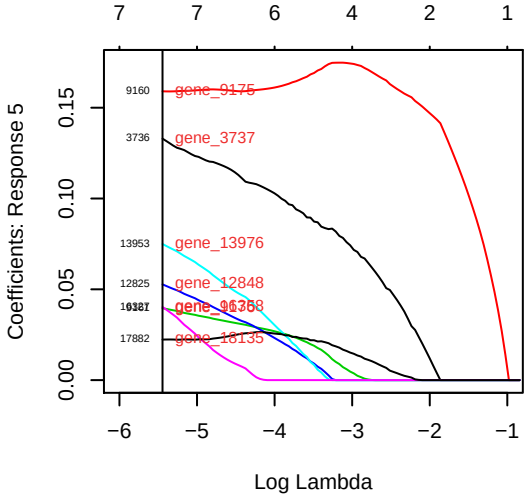

9 9 8 8 8 6 4 4 3 2 2 2 0
Multinomial Deviance
0.0 0.5 1.0 1.5 2.0 2.5 3.0
−5 −4 −3 −2 −1
Log(λ)


12 9 8 7 4 3
Coefficients: Response 1
0.2 0.1 0.0 −0.1 −0.2
5569
6867
18493
9631
7950
17548
−6 −5 −4 −3 −2 −1
Log Lambda


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


9 9 8 9 5 2
Coefficients: Response 3
0.05 −0.05 −0.15
449
13795
218
14090
12785
−6 −5 −4 −3 −2 −1
Log Lambda


14 12 9 4 2 0
Coefficients: Response 4
0.3 0.2 0.1 0.0 −0.1
16821
15864
18366
11229
5641
−6 −5 −4 −3 −2 −1
Log Lambda


7 7 6 4 2 1
Coefficients: Response 5
0.15 0.10 0.05 0.00
9160
3736
13953
12825
17882
gene_9175
gene_3737
gene_13976
gene_12848
gene_18135
−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 5, 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))

mean(y_lasso_test == pred)
```

```
## [1] 0.9966777
```

The mean output (using 0 for misclassifications):

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

```
## [1] 0.9845862
```

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:

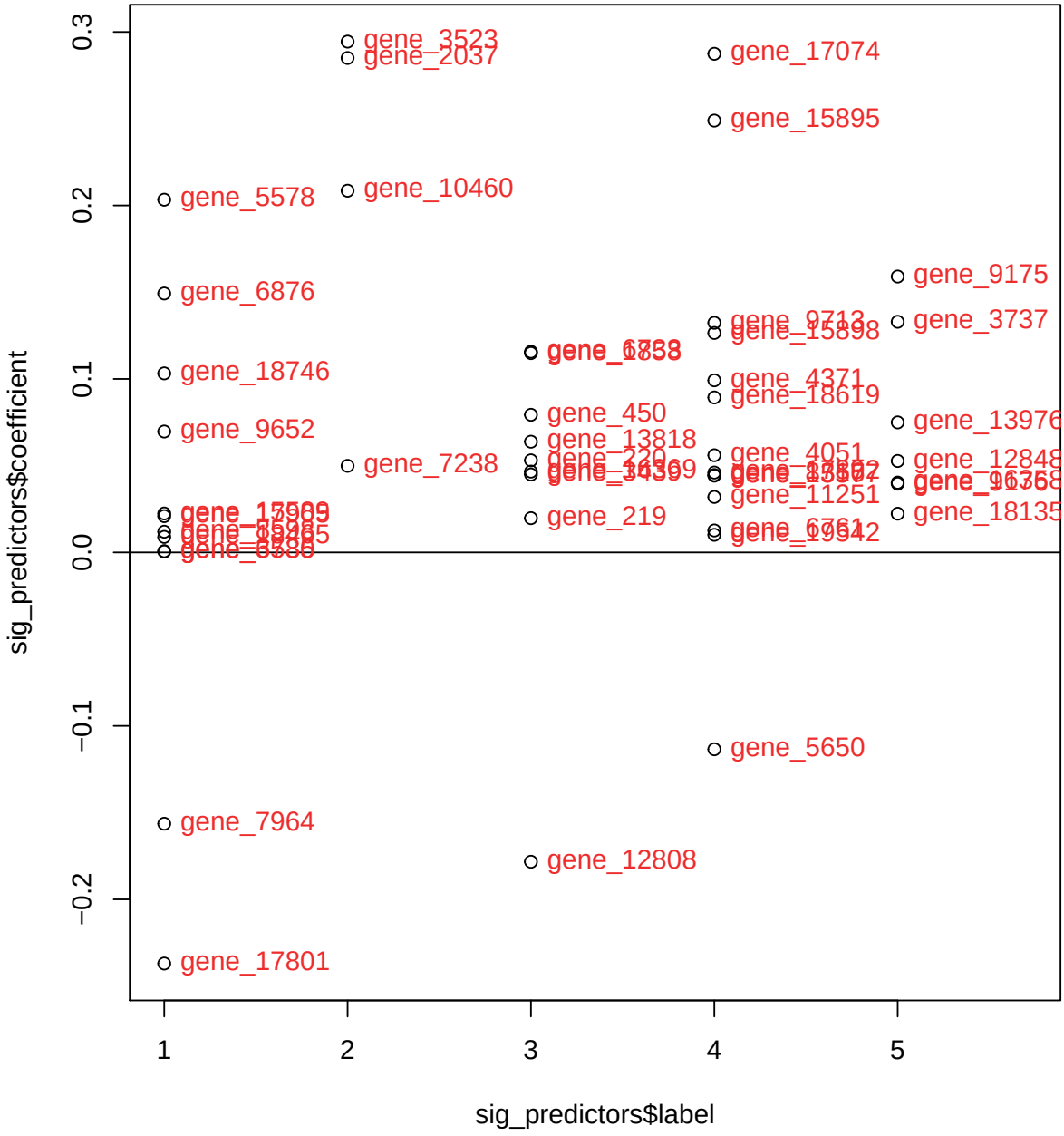
```
sig_index = Reduce(union,c(coeffs[["1"]]@i,coeffs[["2"]]@i,coeffs[["3"]]@i,coeffs[["4"]]@i,coeffs[["5"]]
# coeffs is a list of dgCMatrix
# @i are indices of non-zero values in the matrix (first one corresponds to the y-intercept)
# @x are the coefficients corresponding to the indices

colnames(x_lasso_test[,sig_index])
```

```
##  [1] "gene_3785"  "gene_5578"  "gene_6530"  "gene_6876"  "gene_7964"
##  [6] "gene_8598"  "gene_9652"  "gene_15589" "gene_17801" "gene_17905"
## [11] "gene_18465" "gene_18746" "gene_2037"  "gene_3523"  "gene_7238"
## [16] "gene_10460" "gene_219"   "gene_220"   "gene_450"   "gene_1858"
## [21] "gene_3439"  "gene_6733"  "gene_12808" "gene_13818" "gene_16369"
## [26] "gene_4051"  "gene_4371"  "gene_5650"  "gene_6761"  "gene_8316"
## [31] "gene_9713"  "gene_11251" "gene_15577" "gene_15895" "gene_15898"
## [36] "gene_17074" "gene_17892" "gene_18619" "gene_19542" "gene_3737"
## [41] "gene_9175"  "gene_9176"  "gene_12848" "gene_13976" "gene_16358"
## [46] "gene_18135"
```

Most of the coefficients have positive values:

```
sig_predictors = NULL

for (label in 1:5){
  for (index in 2:length(coeffs[[label]]@i))  {
    predictor = colnames(data)[coeffs[[label]]@i[index]]
    sigpre = data.frame(label=label, coefficient=coeffs[[label]]@x[index], predictor=predictor)
    sig_predictors = rbind(sig_predictors, sigpre)
  }
```

```
}

plot(sig_predictors$label, sig_predictors$coefficient, xlim = c(1,5.7))
abline(h=0)
text(sig_predictors$label, sig_predictors$coefficient, labels=sig_predictors$predictor, pos=4, col="fir
```

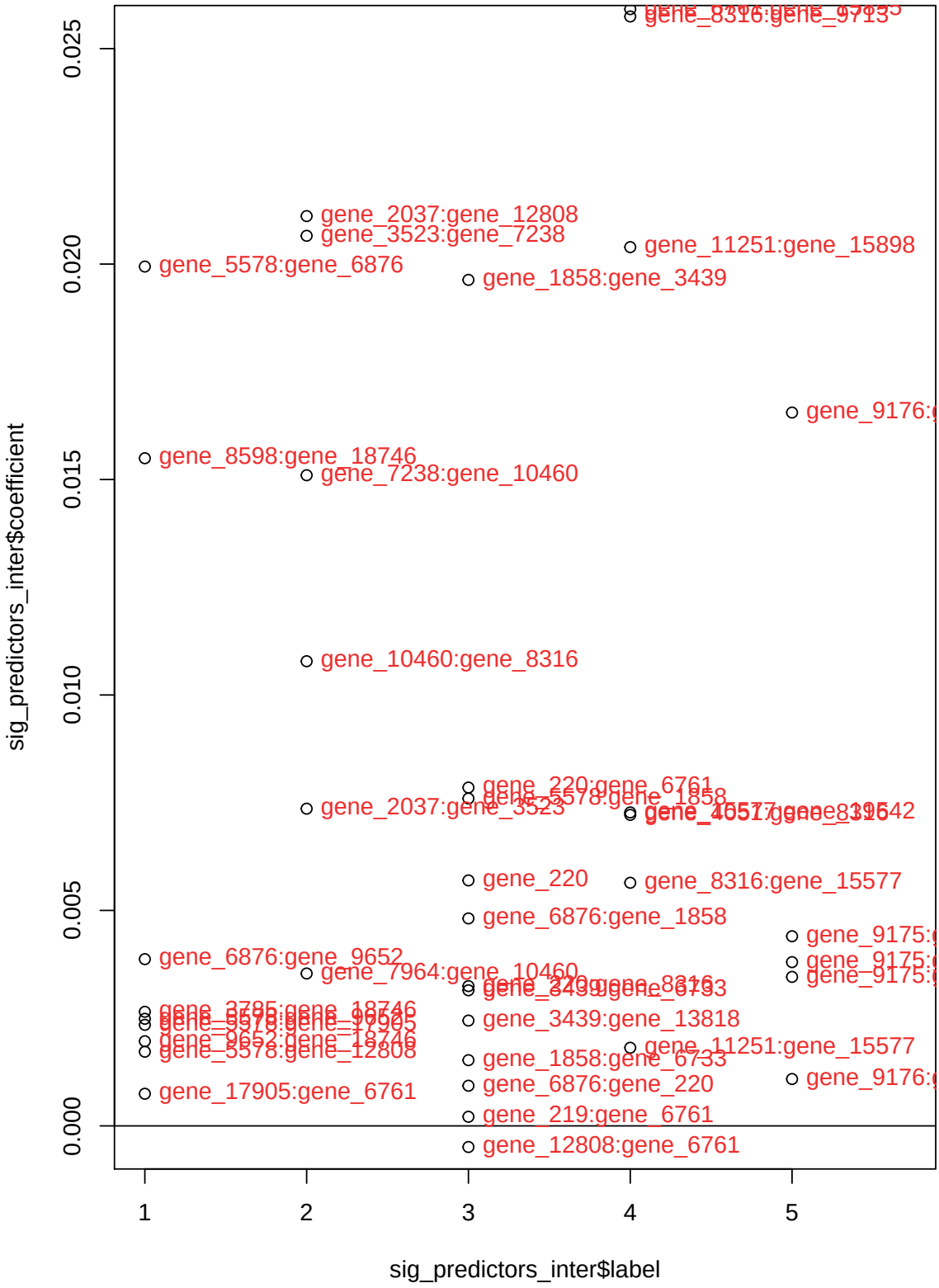


Fitting again the model but only including the significant predictors returned by the previous fit, plus all the 2-way interactions between the significant predictors:

```
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[["1"]]@i,
                                  coeffs_inter[["2"]]@i,
                                  coeffs_inter[["3"]]@i,
                                  coeffs_inter[["4"]]@i,
                                  coeffs_inter[["5"]]@i))

colnames(x[,sig_index_inter])
```

```
##  [1] "gene_17801"            "gene_3785:gene_18746"  "gene_5578:gene_6876"
##  [4] "gene_5578:gene_9652"   "gene_5578:gene_17905"  "gene_5578:gene_12808"
##  [7] "gene_6876:gene_9652"   "gene_7964:gene_6761"   "gene_7964:gene_11251"
## [10] "gene_8598:gene_18746"  "gene_9652:gene_18746"  "gene_17801:gene_6761"
## [13] "gene_17905:gene_6761"  "gene_2037"             "gene_7964:gene_10460"
## [16] "gene_2037:gene_3523"   "gene_2037:gene_12808"  "gene_3523:gene_7238"
## [19] "gene_7238:gene_10460"  "gene_10460:gene_8316"  "gene_219"
## [22] "gene_220"              "gene_12808"            "gene_5578:gene_1858"
## [25] "gene_6876:gene_220"    "gene_6876:gene_1858"   "gene_219:gene_6761"
## [28] "gene_220:gene_6761"    "gene_220:gene_8316"    "gene_1858:gene_3439"
## [31] "gene_1858:gene_6733"   "gene_3439:gene_6733"   "gene_3439:gene_13818"
## [34] "gene_12808:gene_6761"  "gene_4051:gene_8316"   "gene_6761:gene_15895"
## [37] "gene_8316:gene_9713"   "gene_8316:gene_15577"  "gene_11251:gene_15577"
## [40] "gene_11251:gene_15898" "gene_15577:gene_19542" "gene_9175:gene_9176"
## [43] "gene_9175:gene_13976"  "gene_9175:gene_16358"  "gene_9176:gene_13976"
## [46] "gene_9176:gene_16358"
```

```
data_sig_inter = as.data.frame(x)

sig_predictors_inter = NULL
for (label in 1:5){
  for (index in 2:length(coeffs_inter[[label]]@i))  {
    predictor = colnames(data_sig_inter)[coeffs_inter[[label]]@i[index]]
    sigpre = data.frame(label=label, coefficient=coeffs_inter[[label]]@x[index], predictor=predictor)
    sig_predictors_inter = rbind(sig_predictors_inter, sigpre)
  }
}
plot(sig_predictors_inter$label, sig_predictors_inter$coefficient, xlim = c(1,5.7), ylim = c(0,.025))
abline(h=0)
text(sig_predictors_inter$label, sig_predictors_inter$coefficient, labels=sig_predictors_inter$predicto
```


sig_predictors_inter$coefficient
0.000
0.005
0.010
0.015
0.020
0.025
1
2
3
4
5
sig_predictors_inter$label
gene_2037:gene_12808
gene_3523:gene_7238
gene_5578:gene_6876
gene_1858:gene_3439
gene_11251:gene_15898
gene_9176:
gene_8598:gene_18746
gene_7238:gene_10460
gene_10460:gene_8316
gene_220:gene_6761
gene_2037:gene_3523
gene_220
gene_8316:gene_15577
gene_6876:gene_1858
gene_9175:
gene_6876:gene_9652
gene_7964:gene_10460
gene_3439:gene_13818
gene_11251:gene_15577
gene_1858:gene_6733
gene_9176:
gene_17905:gene_6761
gene_6876:gene_220
gene_219:gene_6761
gene_12808:gene_6761

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))

mean(y_lasso_test == pred_inter)
```

```
## [1] 0.9966777
```

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

```
## [1] 0.9838473
```

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

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 = 6, activation = "softmax")

#gmodel

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

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

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] 0.05119609
##
## $accuracy
## [1] 0.986711
```

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

LIME

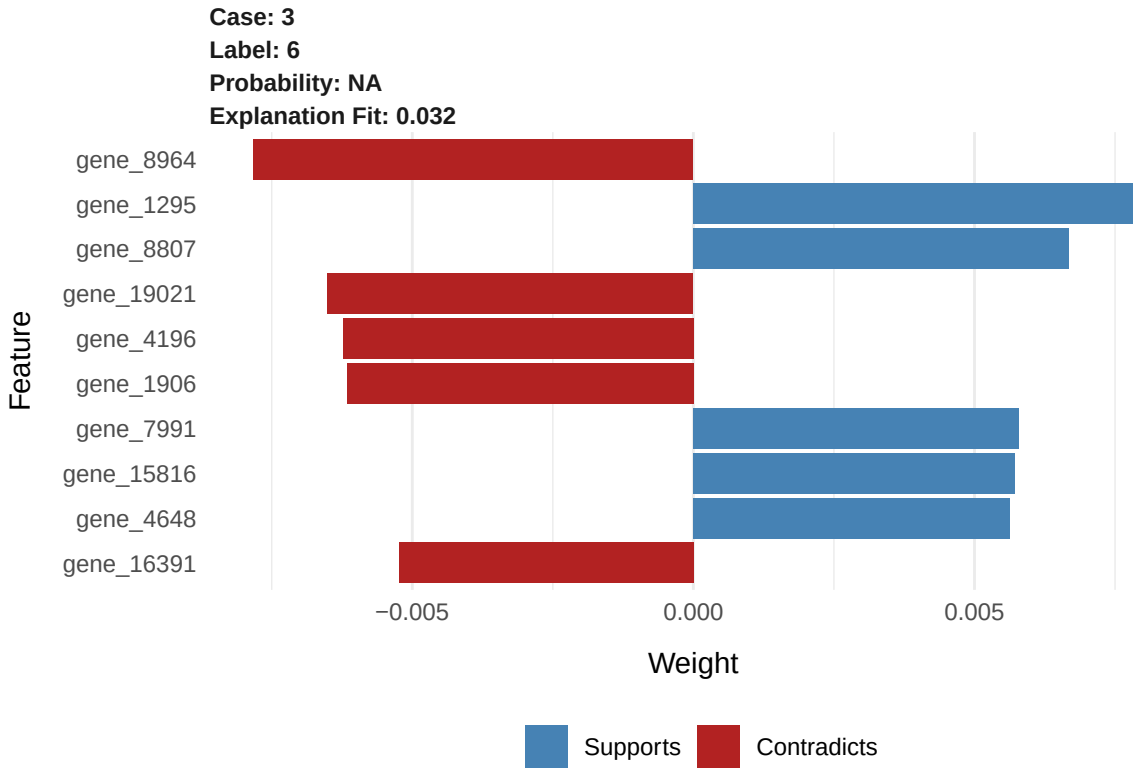
The Lasso model provides a global sense of the influence of the predictors. However if the data structure is complex it might not explain well the interpretation of particular predictions.

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.

```
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(no_class = pred[,1], label1 = pred[,2], label2 = pred[,3], label3 = pred[,4], label4 = pred
}

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

```
## # A tibble: 500 x 6
##    no_class    label1   label2   label3    label4   label5
##       <dbl>     <dbl>    <dbl>    <dbl>     <dbl>    <dbl>
##  1 6.47e-10  5.45e- 8 5.00e-15 4.44e-11  5.22e-11 1.00e+ 0
##  2 1.62e-14 10.00e- 1 4.70e- 9 3.46e- 7  1.77e- 4 1.28e- 6
##  3 1.40e-13  5.67e- 5 4.99e-11 4.65e-12 10.00e- 1 9.05e-16
##  4 2.30e-16 10.00e- 1 1.72e-11 9.49e- 8  4.68e- 4 1.88e- 9
##  5 6.54e-17 10.00e- 1 2.72e- 9 7.59e- 8  5.16e- 6 1.80e- 6
##  6 6.92e-14 10.00e- 1 6.56e- 5 1.02e- 4  7.01e- 5 1.08e- 6
##  7 2.50e-13  9.97e- 1 1.29e- 7 2.87e- 6  3.24e- 3 8.84e- 8
##  8 6.76e-12  1.56e-11 1.24e-19 1.14e-16  4.79e-14 1.00e+ 0
##  9 6.53e-13  3.00e-20 7.98e-22 1.00e+ 0  4.55e-19 5.00e-19
## 10 8.48e- 7  1.12e- 5 8.36e- 9 5.13e-11 10.00e- 1 1.20e-11
## # ... with 490 more rows
```

```r
# 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)

explanation <- lime::explain(
  x = as.data.frame(x_train)[3,],
  explainer = explainer,
  #n_permutations = 5000,
  #dist_fun = "euclidean", #?dist()
  #kernel_width = 0.75,
  feature_select = "lasso_path",
  n_features = 10,
  labels = 6,
)

#very los r-squared: too many variables? plot pc1-pc2
#not necessarily bad, high variance

#difference of values with global lasso, because this is local

plot_features(explanation)
```



R-squared ("explanation fit") is very low. This doesn't necesarily mean that the local model doesn't fit well, the variance in the data is high as seen in the PCA plot, 10 PCs are required for just 55% of explained variance.

Shapley values

KernelSHAP