Interpretation with LIME of classification in high-dimensional tabular data

2020

Introduction

TODO

Data Description

The dataset was uploaded to Kaggle by/thanks to UCI Machine Learning Repository:
https://www.kaggle.com/murats/gene-expression-cancer-rnaseq

Source: Samuele Fiorini, samuele.fiorini@dibris.unige.it, University of Genoa, redistributed under Creative Commons license.

The data consists of 801 patients.

The 20532 independent variables are all RNA sequencing gene expression levels measured by a sequencing platform (Illumina HiSeq).

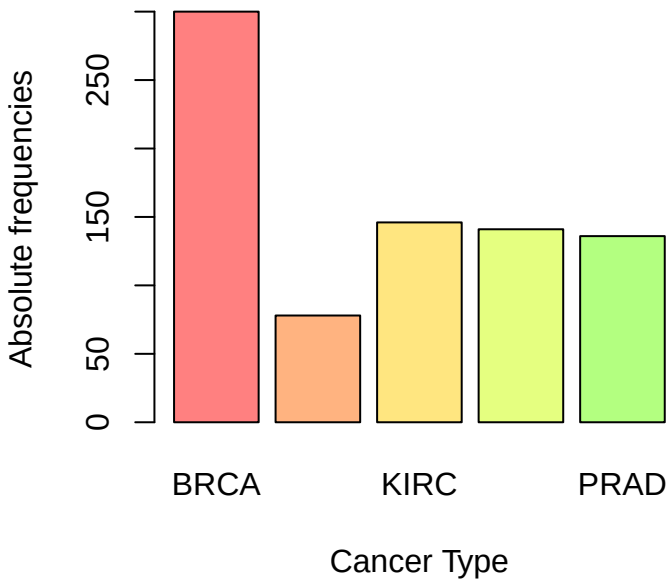
The dependent categorical variable represent primary tumors occurring in different parts of the body, covering 5 tumor types including:

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

A machine learning model will be trained to predict the type of tumor for new patients.
TODO: buscar un dataset similar pero que contenga tambien pacientes sanos.

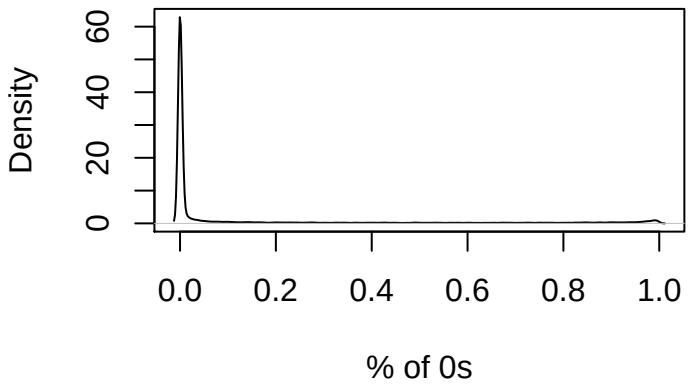
Data Analysis and Preprocessing

Absolute frequencies of the response:

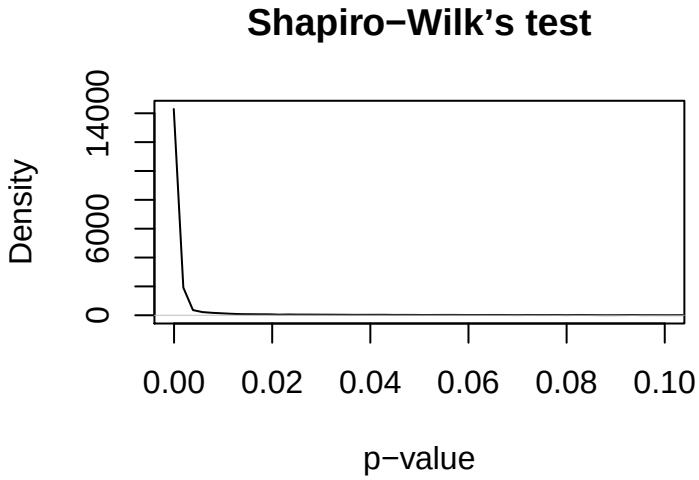


268 gene expression variables contain only 0s are removed.

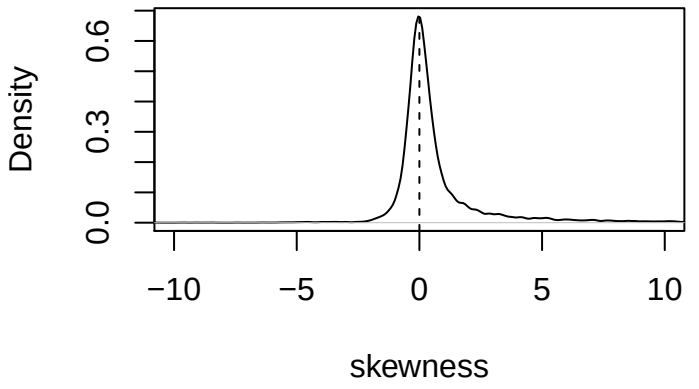
From the remaining 20264 gene expressions, 670 have at least 95% of 0s:

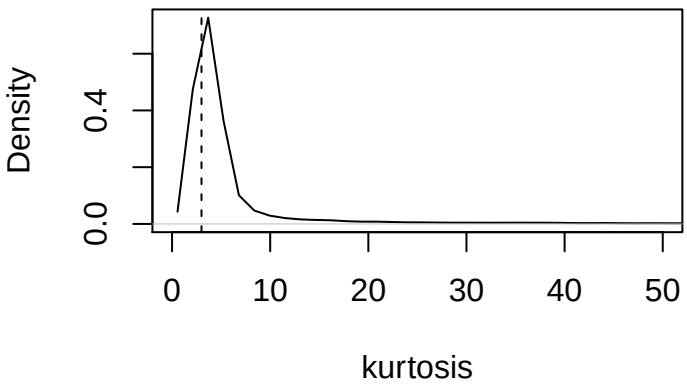


The predictors don't follow distributions close to normal. Nornality cannot be assumed:

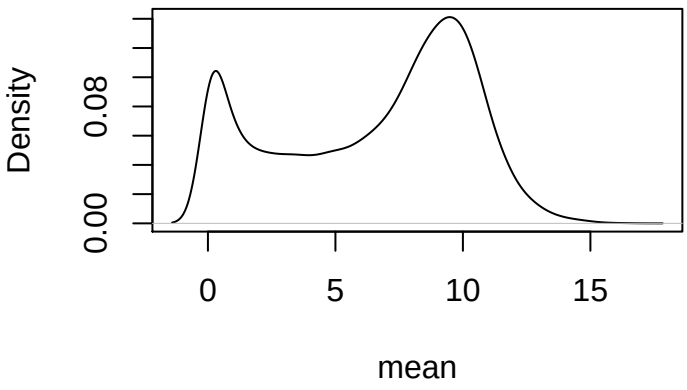


Symmetry and kurtosis:





The distribution of means suggests there are two categories of gene expression:

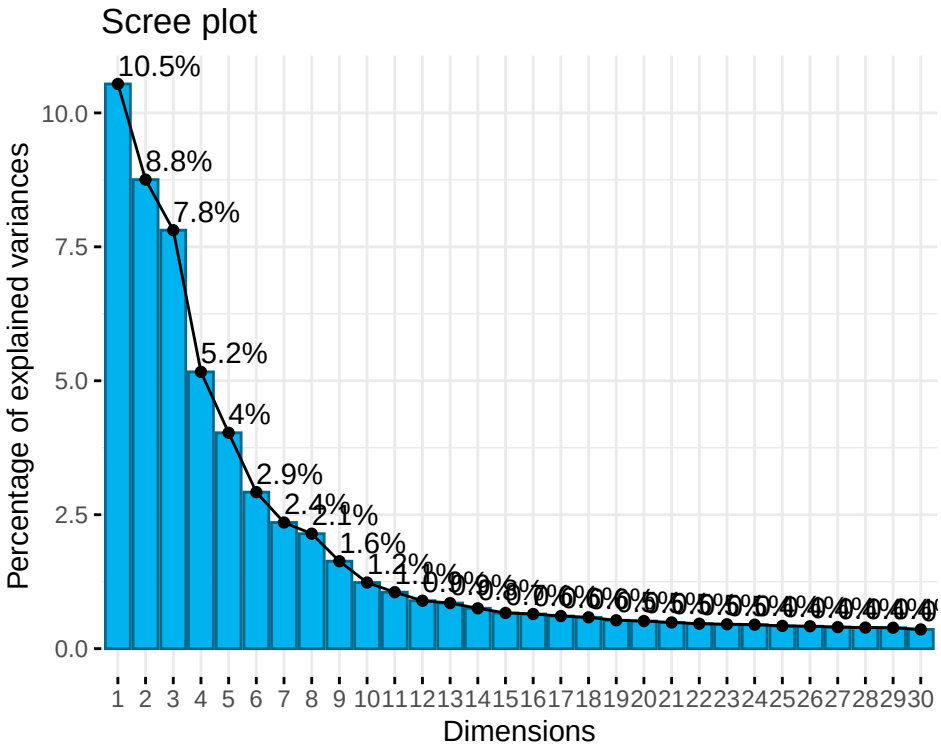


Outliers:

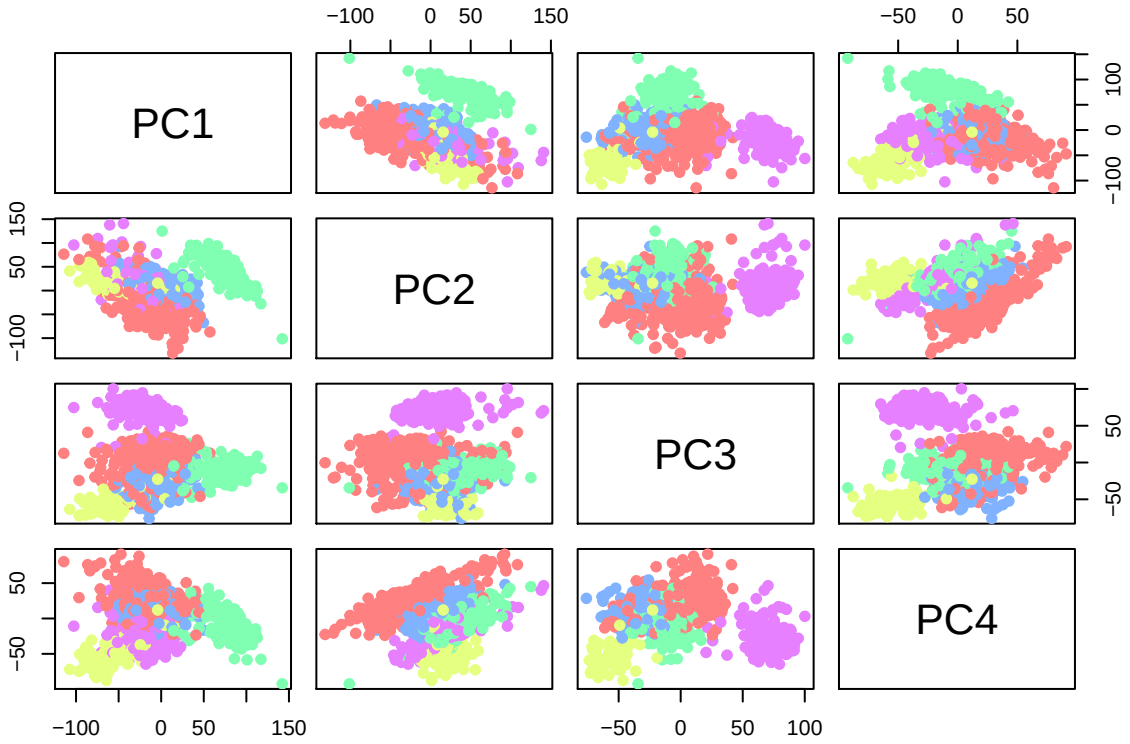
TODO

The data is scaled to help with the training of the prediction model.

At least 45% of the variability of the data is explained by the first 10 PCA components:

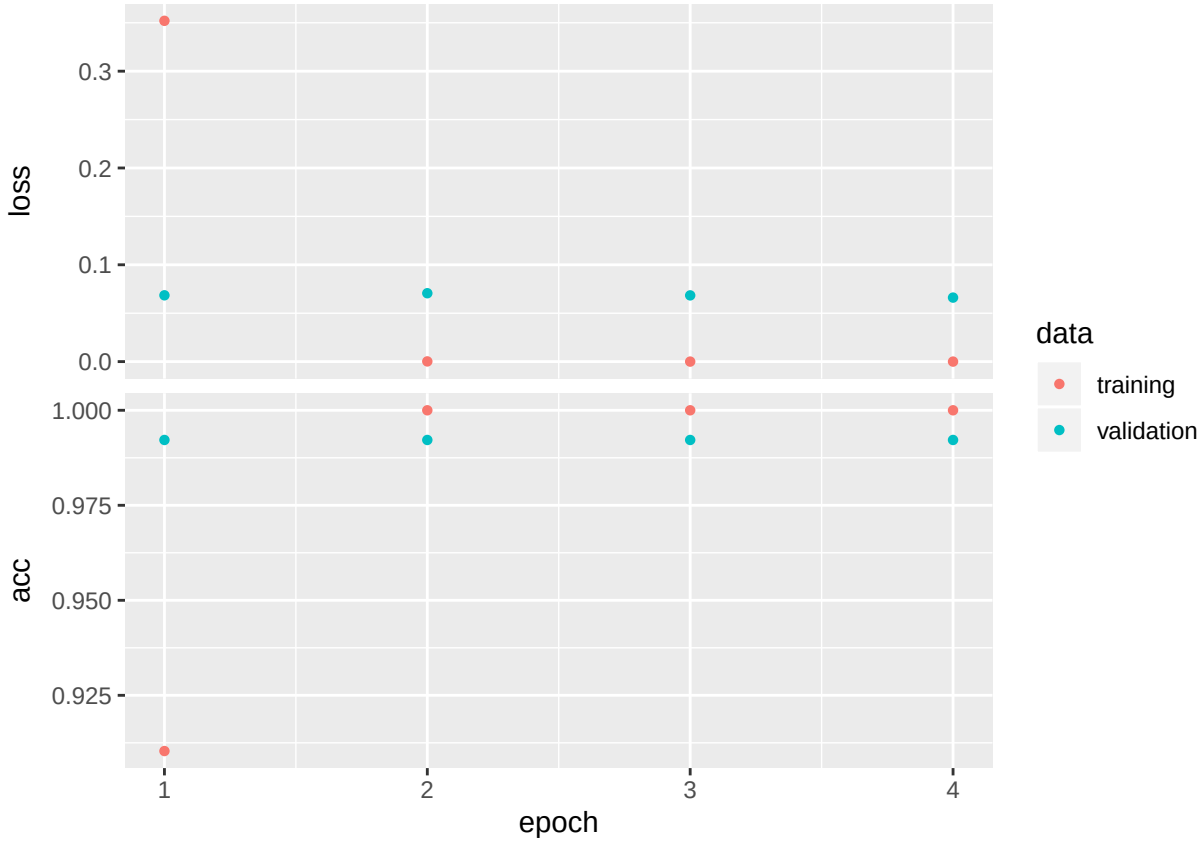


First 4 PCs look enough to classify the types of cancer despite only accounting for 32.27% of the variability of the data, which suggests high collinearity:


PC1
PC2
PC3
PC4
−100
0
50
150
−50
0
50
100
−100
0
50
150
−50
0
50
100
−100
0
50
150
−50
0
50
100

Fully connected network

A fully connected network is trained with 80% of the data:



The accuracy of the network is close to 1 (sometimes 1 depending on the random initialization of the network):

```
## $loss
## [1] 0.02142848
##
## $acc
## [1] 0.99375
```

Neural networks use to be power prediction tools but they come with a cost in terms of interpretability of the predictions. They contain a huge number of parameters making difficult the identification of features that are influetial in the response of the model.

Surrogate model

To tackle the interpretability issue of the prediction model (which we'll call *black box* from now on), a surrogate interpretable model is fitted in parallel. This surrogate model will help us to undestand the relationship between the features and the response at a global level (i.e. for no particular prediction).

Lasso will be the model of choice, as it performs feature selection efficiently for high-dimensional data and it's very easy to understand.

Because the purpose of this model is interpretation, we'll add a parameter on top of lasso to select the number of features we want to interpret the black box model. We'll fit several models with different lambdas and pick the one with the desired number of features. This is done for each category. With a fixed number of features we'll be able to fairly compare the surrogate model with other models with the same number of features as we'll later with LIME.

20 features are selected (about 0.05% of the total features).

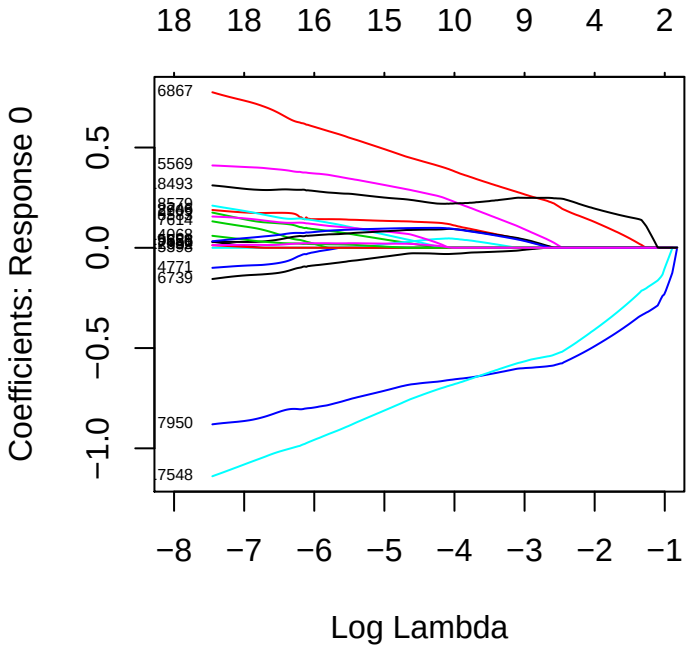
```
library(glmnet)
```

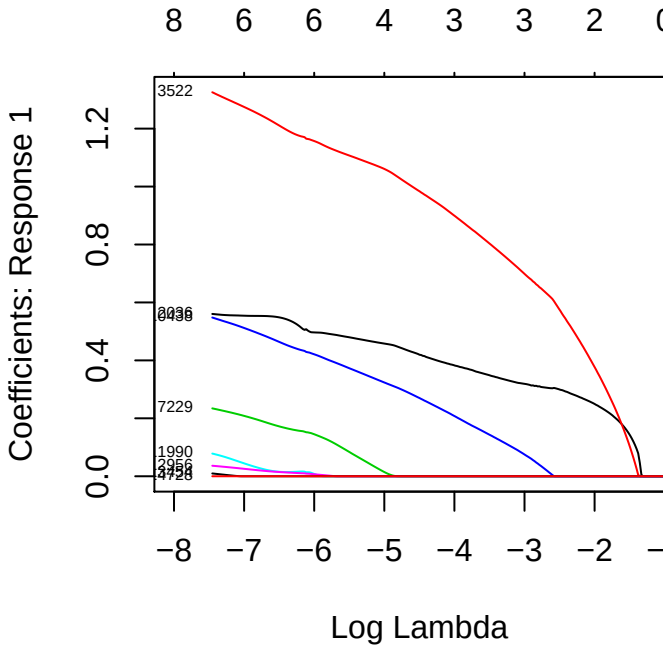
```
## Loading required package: Matrix
```

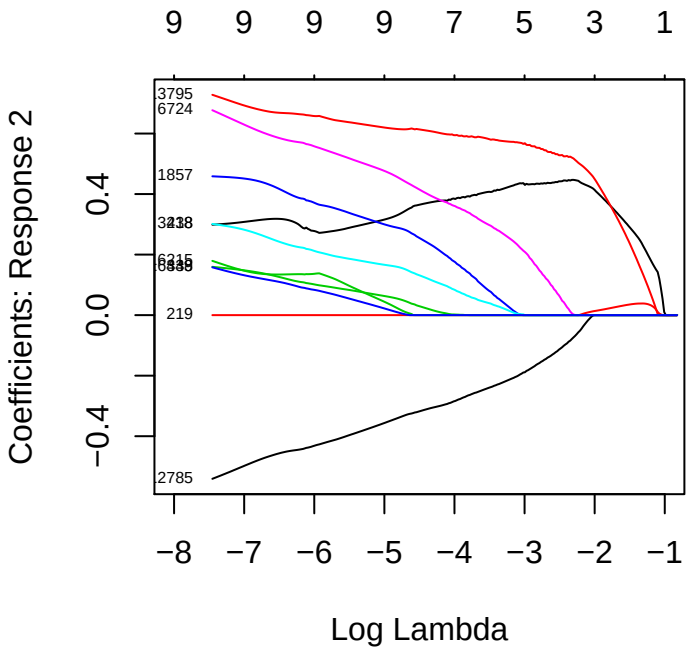
```
## Loaded glmnet 3.0-2
```

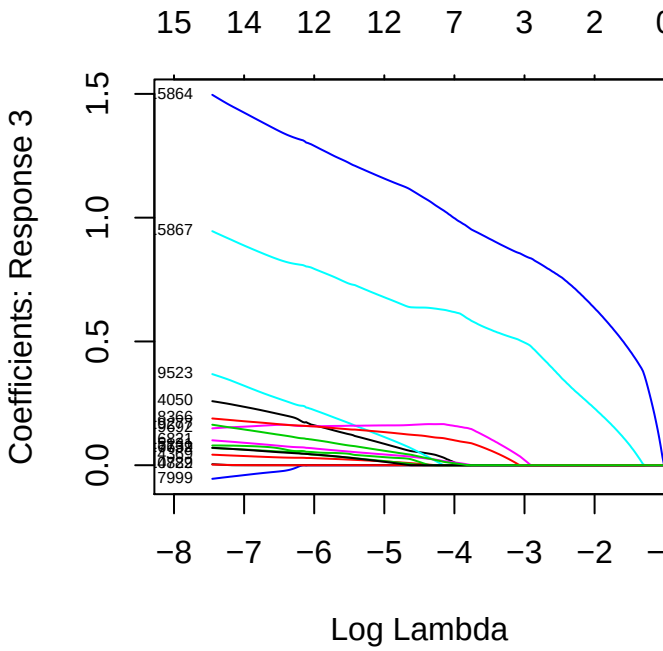
```
# this function looks for a lasso model with n_features selected for each category
# (if possible, it will depend on the lambda grid)
get_surrogate_features <- function(surrogate, n_features){

  surrogate_coeffs = coef(surrogate)

  coeffs = vector(mode = "list", length = 5)
  names(coeffs) = c("0","1","2","3","4")
  index = coeffs
  lambda = coeffs

  for (i in 0:4) {

    # code I picked up from the LIME package: https://github.com/thomasp85/lime/blob/49df0a131deee4919a
    lasso_sparse = surrogate_coeffs[[as.character(i)]]
    has_value <- apply(lasso_sparse[-1,], 2, function(x) x != 0)
    f_count <- apply(has_value, 2, sum)  # number of parameters for each lambda (columns in lasso_spars

    # In case that no model with correct n_feature size was found return features <= n_features
    lambda_index <- rev(which(f_count <= n_features))[1]

    # Selected features
    index[[as.character(i)]] = which(has_value[, lambda_index])
    coeffs[[as.character(i)]] = lasso_sparse[which(has_value[, lambda_index])+1,lambda_index]
    lambda[[as.character(i)]] = surrogate[["lambda"]][lambda_index]

    #TODO: this is from the lime package
    #fit <- glmnet(x_train[,index[[as.character(i)]]], cancer[training_index], alpha = 0, lambda = 2 /
    #r2 <- fit$dev.ratio
    #fff <- coef(fit)[-1, 1]
    #print(fff)
    #coeffs[[as.character(i)]] = fff#fit$beta@x
    #names(coeffs[[as.character(i)]]) = names(index[[as.character(i)]])
```

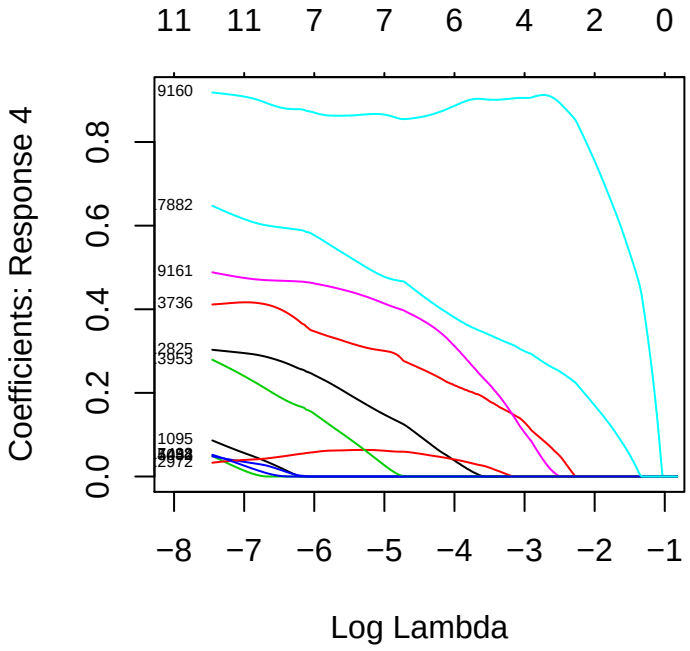
```
  }

  list(index = index, coeffs = coeffs, lambda = lambda)
}

n_features = 10
surrogate = glmnet(x_train, cancer[training_index], alpha = 1, family = "multinomial", nlambda=300, lam
surrogate_features = get_surrogate_features(surrogate, n_features)

plot(surrogate, xvar="lambda", label = TRUE, xlim=c(-8,-1))
```











The plots above show the coefficient values for the selected features for each category. The higher the absolute value, the higher the influence (globally) in the output.

Because the number of features is fixed, the selected value for lambda is not optimal for the fitting. Not a

problem since the accuracy of the Lasso model is close to 1 with the test data:
TODO: que pasa si el accuracy es mucho peor que el accuracy del black box model?

```
# Each category has a different lambda. Here we use the smallest one for all the categories.
min_lambda = min(as.numeric(surrogate_features$lambda))
surrogate_test = predict(surrogate, newx = x_test, s = min_lambda, type="response")
surrogate_pred = max.col(as.data.frame(surrogate_test))-1

mean(cancer[-training_index] == surrogate_pred)
```

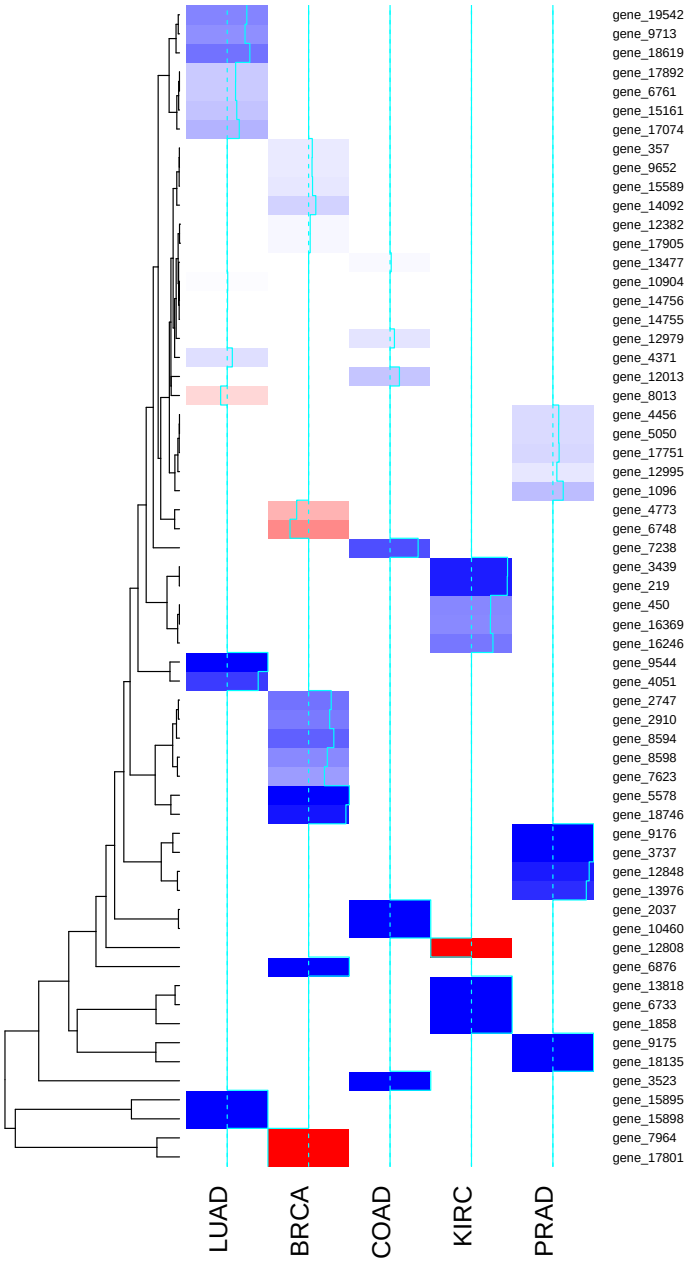
```
## [1] 0.9875
```

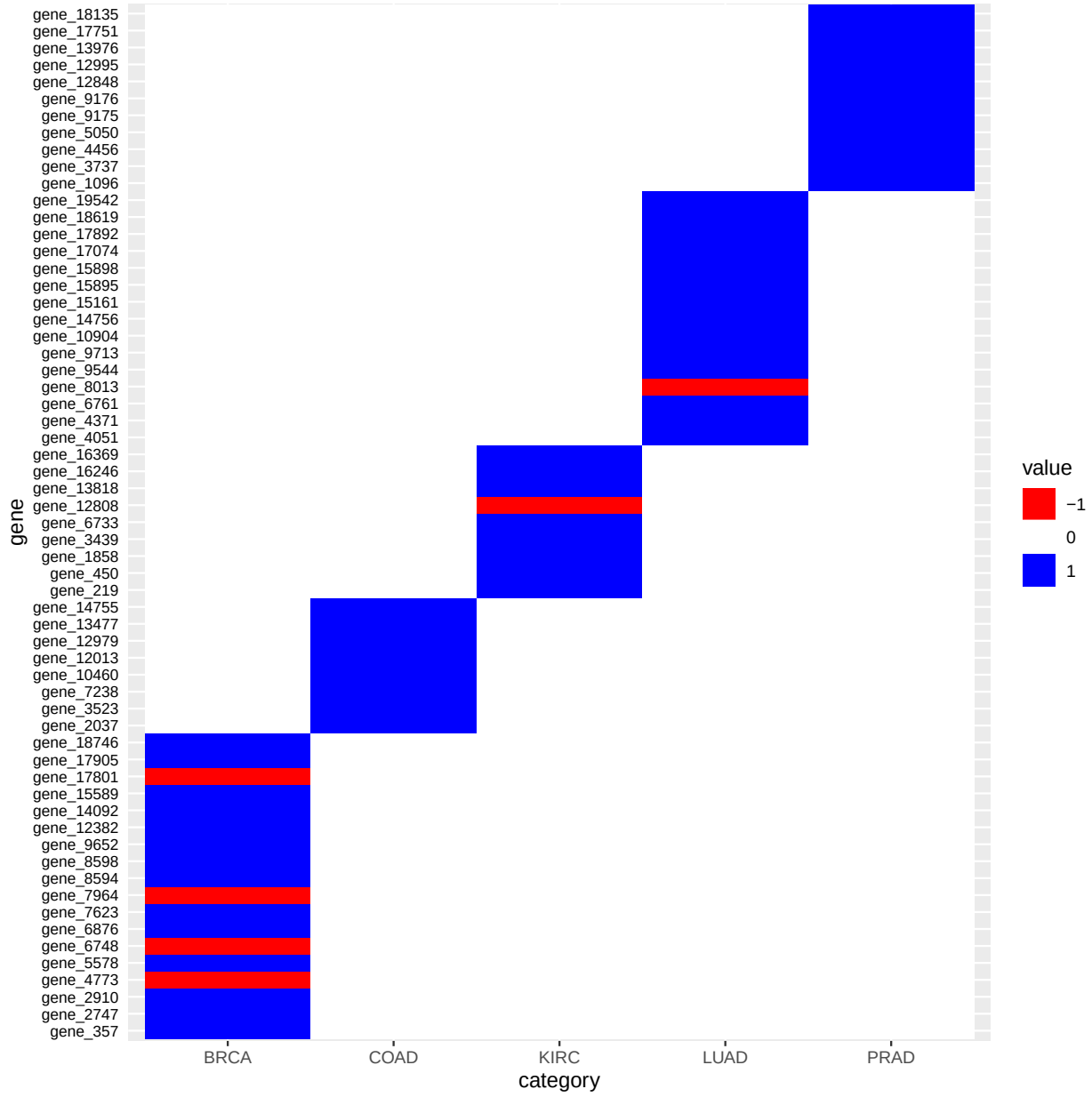
The mean output probability (using 0 for misclassifications):

```
label_output_prob = apply(surrogate_test[,,1],1,max) * as.integer(as.integer(cancer[-training_index]) ==
mean(label_output_prob)
```

```
## [1] 0.9860302
```

Lasso coefficients heatmaps:


gene_19542
gene_9713
gene_18619
gene_17892
gene_6761
gene_15161
gene_17074
gene_357
gene_9652
gene_15589
gene_14092
gene_12382
gene_17905
gene_13477
gene_10904
gene_14756
gene_14755
gene_12979
gene_4371
gene_12013
gene_8013
gene_4456
gene_5050
gene_17751
gene_12995
gene_1096
gene_4773
gene_6748
gene_7238
gene_3439
gene_219
gene_450
gene_16369
gene_16246
gene_9544
gene_4051
gene_2747
gene_2910
gene_8594
gene_8598
gene_7623
gene_5578
gene_18746
gene_9176
gene_3737
gene_12848
gene_13976
gene_2037
gene_10460
gene_12808
gene_6876
gene_13818
gene_6733
gene_1858
gene_9175
gene_18135
gene_3523
gene_15895
gene_15898
gene_7964
gene_17801
LUAD
BRCA
COAD
KIRC
PRAD



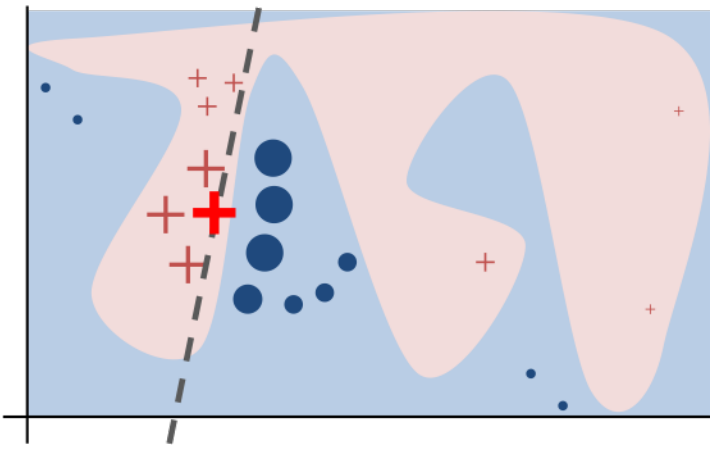
The variables selected by lasso give a global sense of the more influential predictors in driving the model, however, if the data is complex, in some regions of the input space the variables explaning the classifications could be completely different to the ones selected in the surrogate model.

LIME

Intuition

TODO

(image taken from https://github.com/marcotcr/lime)



Formulation

TODO

Algorithm

- In order to fit a local linear model around the data point which prediction we want to explain, we need enough data in the surroundings of that data point. In order to achieve that, kernel densities estimations are computed for each variable, then new simulated data points are introduced among the original data points by sampling from the kdes, increasing this way the "density" of the data that was used in the training of the black box. This is specially important with sparse data like our's where $p \gg n$, where data points are "far" away from each other making for a poor local model fit.

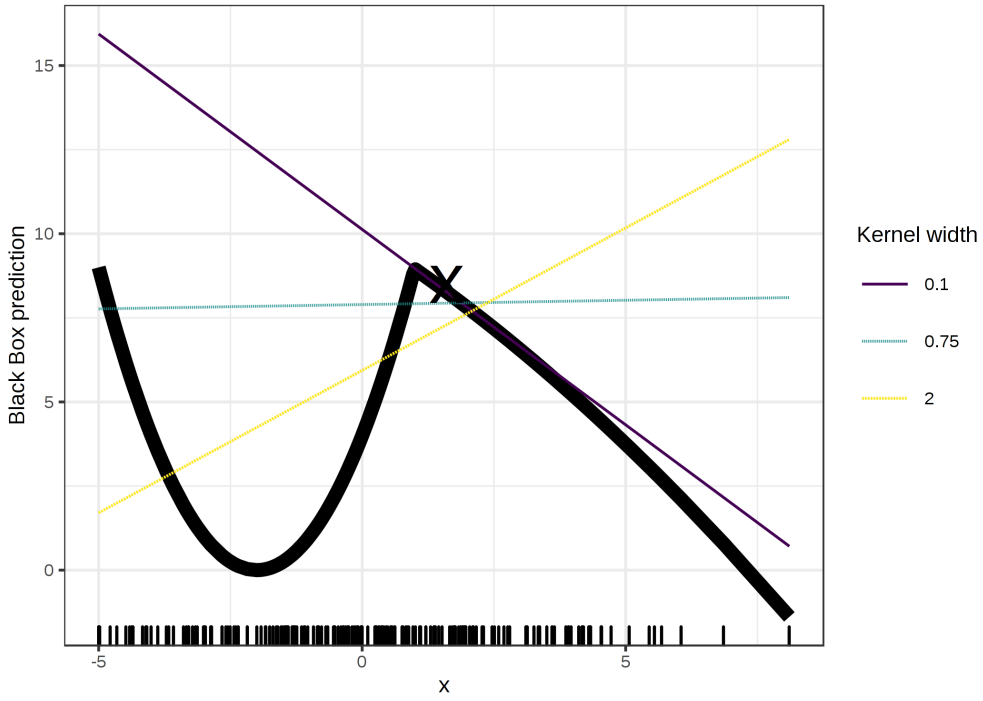
The parameter to tune in this step is the number of simulated data points (**n_permutations**).
The risk if the parameter is too low, is the high variance of the simulated data each the same prediction is interpreted. If each time the same data point is explained the simulated data is different the variance in the interpretation will be high making the interpretations inconsistent and untrustworthy.

TODO: no veo inconveniente en incrementar este parametro al maximo hasta que las limitaciones en CPU/memoria lo permitan.

TODO: Los estimated kernel densities de cada variable no tienen en cuenta las relaciones entre las variables lo cual puede generar data points "irreales". Idea -> aplicar dimensionality reduction y usar multivariate kernel density estimation para generar un data set mas real.

- Of data points simulated by sampling from the features kdes, we are specially interested in those in the surroundings to the data point of interest, since we are fitting a local model. So we need to give more weight to data near the instance of interest and this is done with a smoothing exponential kernel. The width of the kernel is a parameter of the LIME model (**kernel_width**) and probably the more tricky one as shown in the following example of a 1-dimensional dataset:

(image taken from https://christophm.github.io/interpretable-ml-book/lime.html)



In this example changes in the kernel width leads to drastic changes in the local linear fit for the instance of interest (the cross in the plot). Adding more dimensions would increase the sensitivity of the width parameter even more.

In the *lime* package the default value is $0.75\sqrt{p}$. The more number of dimensions for the same number of observations the more space is the data space, that's why the kernel width is increased depending on p.
The appropriate value seems to depend on the surroundings of the data point being explained so there isn't a clear rule of thumb to follow for this parameter.

Too small values could lead to insufficient data to fit the local model and too much variance in different interpretations for the same dat point to explain.
Too high values could lead to loosing "locality", hence the included data becoming less linear and the explanation of the local model less accurate.

TODO: Buscar algun criterio, quizas basado en la complejidad (clasificaciones con menos o mas certidumbre) para seleccionar el kernel width.

TODO: Leer *[2] In high-dimensional data, data points are sparse. Defining a ''local neighborhood" of the instance of interest may not be straightforward. Importance of the local neighbourhood is presented for example in the article „On the Robustness of Interpretability Methods'' (Alvarez-Melis and Jaakkola 2018). Sometimes even slight changes in the neighbourhood affects strongly obtained explanations.*

Another parameter is the distance function that measures the proximity between the instance of interest and the simulated data points. The default option is Gower's distance but others like Euclidean or Manhattan (see *?dist()* for details) can also be used.

TODO: Elegir la funcion mas adecuada teniendo en cuenta:
- la alta dimensionalidad
- la alta colinearidad
- variables no-normales (pero tampoco exponenciales)
- las variables han sido standardizadas

[3] It is very unclear whether the distance measure should treat all features equally. Is a distance unit for

feature x1 identical to one unit for feature x2? Distance measures are quite arbitrary and distances in different dimensions (aka features) might not be comparable at all.

- The next step is to feed the black box with the simulated data to get the classifications required to fit the local model.

- With the simulated data and the corresponding predictions, a linear model is fit. Several options are available (**feature_select**). We'll choose lasso for two reasons: the high-dimensionality of the data and to get a better comparison with the global surrogate lasso model. Lasso will use the weights from the smoothing kernel to give more influence to the the neighborhood to the original observation to be explained.

- Finally the coefficients with higher absolute values are selected (**n_features**) to explain the output (**n_labels**, *1* if we are just interested in the selected category).

LIME with the original variables

Picking instances to evaluate the interpretation model.

- picking samples that are not close together in the imput space in order to cover different scenarios.
- picking samples with less certainty in the output - samples that lie within the frontier between different categories are tougher to predict and therefore more intersting to understand.
- TODO: mirar el metodo propuesto en "4.SUBMODULAR PICK FOR EXPLAINING MODELS" en el paper [1]

We'll pick up two observations to explain, one from category *PRAD* which data points in the PCA plots look like they don't overlap too much with other categories (easy to predict), and another one from category *LUAD* which data points overlap more with other categories. Each one will be interpreted 4 times to analyse the consistency of the selected features and their weights.

As seen in the plots below, the results are very inconsistent - most of the 20 features are different for the same data point. The problem is probably coming from the high-dimensionality of the data, too many features are "competing" to explain the same variance. Depending on the simulated data created in the interpretation, some predictors will prevail over other predictors, even if the change in the simulated data is subtle.

The low R^2 of the fits ("explanation fit" in the plot) highlights the issue.

To try to work out this problem we could try to increase the number of simulated data points (default is 5000). However even with 10000 permutations (leading to a 10000*20000 matrix for lasso to fit the line, it takes ages) the problem persists. With more than 10000 permutations I get memory allocation errors.

TODO: probar con un kernel width mas grande para reducir la varianza en las interpretaciones?

TODO: idea -> en lugar de depender del smoothing kernel, filtrar del training data que se le pasa al explainer los data points alejados del punto de interes para asi obtener mas densidad alrededor del punto de interes con menos limitaciones en cuanto a CPU/memoria. seria esto distorsionar demasiado la interpretacion?

TODO: sacar las interpretaciones de todos los data points de una misma categoria y comparar con lime?

```
get_instance_explanation <- function(datapoints_index) {

  # the explainer buils the simulated data to fit the local model from the training data and the variab
  explainer_all <- lime(
    x = as.data.frame(x_train),
    model = black_box,
    use_density = TRUE, # marginal kdes
    bin_continuous = FALSE
  )

  lime::explain(
    x = as.data.frame(x_test)[datapoints_index,],
    explainer = explainer_all,
    n_permutations = 10000,
    #kernel_width = 0.75,
    feature_select = "lasso_path",
    n_features = n_features,
    n_labels = 1
  )
}

# PRAD category
```

```
#plot_features(get_instance_explanation(rep(which(cancer[-training_index] == 4)[1],4)))  # training_ind

# LUAD category
#plot_features(get_instance_explanation(rep(which(cancer[-training_index] == 3)[1],4)))
```

TODO: probar con distintos kernel widths (distintas combinaciones: mas o menos categoryoverlap, mas o menos varianza (pcs vs all individual genes)) probar con ruido en el modelo de clusters? buscar literatura como sobre abordar p » n buscar data sets especificos para cada interpretable data representation (time series con segmentos, ...) heatmap de los parametros con distintos data points para ver donde se parece mas lime a global lasso (o algo asin)

LIME with discretized variables

To reduce the variability of the the interpretations, an option is to discretize the data space. This is done in the *lime* package with the parameter **bin_continuous** set to *true* and specifying the number of bins for the variable values (quantiles are computed for each variable to segment the distribution in categories - there is no actual order).

We try again the same samples as before. Some observations:

For the PRAD category (doesn't overlap with other categories)
* Only a few features are significant (between 1 and 4 in my tests).
* The result is more consistent (more common features) than with continuous variables, but still there are some differences.
* R^2 is high.

For the LUAD category (overlaps with other categories)
* Only one feature is significant.
* The dominant feature is the same in all the cases - very consistent.
* R^2 is very low.

TODO: explicar por que ocurre lo anterior descrito.

The interpretation with ranges of values is probably more clear. The discretization of the data in the interpretations will be acceptable depending on the domain, some practitioners using the black box as a tool might not need the level of "granularity" of continuous values and will prefer more "abstract" interpretations.

```
get_discrete_instance_explanation <- function(datapoints_index, explainer) {

  explanation_all_discrete  = lime::explain(
    x = as.data.frame(x_test)[datapoints_index,],
    explainer = explainer,
    n_permutations = 10000,
    #kernel_width = 0.75,
    feature_select = "lasso_path",
    n_features = n_features,
    n_labels = 1
  )
}

explainer_all_discrete = lime(
  x = as.data.frame(x_train),
  model = black_box,
  #use_density = TRUE,
  bin_continuous = TRUE,
  n_bins = 10
)

# PRAD category
#plot_features(get_discrete_instance_explanation(rep(which(cancer[-training_index] == 4)[1],4),explaine

# LUAD category
#plot_features(get_discrete_instance_explanation(rep(which(cancer[-training_index] == 3)[1],4),explaine
```

TODO: relacion con el modelo lasso global (por el momento me esta dando un error de memoria al crear el modelo lasso global con variables categoricas:)

LIME with selected predictors

TODO: Remover predictores que sean colineares. mctest es demasiado lento para tantas dimensiones, incluso con solo 1000 variables le llevaría horas. He probado con forward selection (ver condigo abajo) pero sigue siendo demasiado lento para cubrir todas las variables.
TODO: usar elastic net?
TODO: calcular cuanto se pierde en la prediccion en el modelo lasso

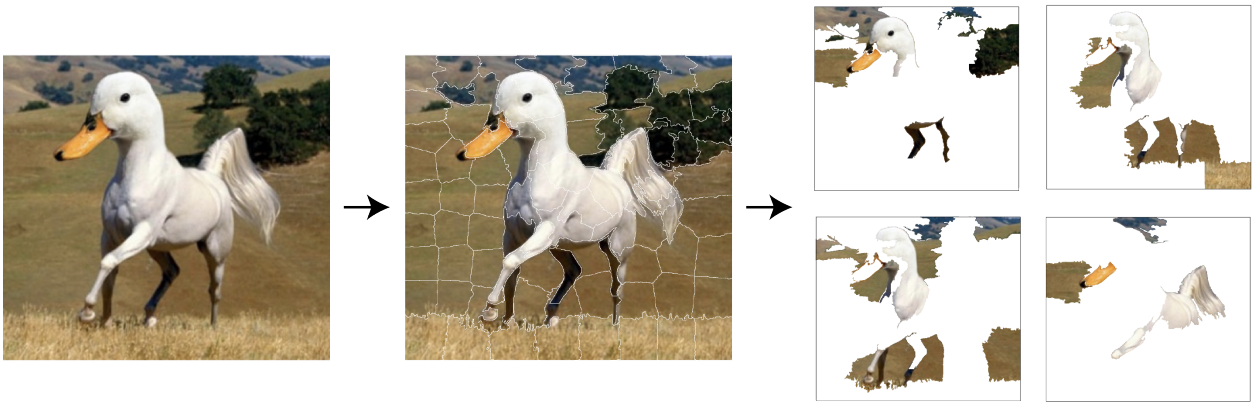
```
library(mctest)

selected = c(colnames(genes)[1])

max = ncol(genes)

if (FALSE)
for (i in 2:max) {
  print(paste(i,"..."))
  selected = c(selected, colnames(genes)[i])
  result = as.data.frame(mctest(genes[,selected], cancer, type="i", method="VIF")[["idiags"]])
  selected = rownames(result[result$detection == 0,])
  print(selected)
}
```

Interpretable data representation (interpretable variable space)

As we have seen with the data space discretization example, the data representation for the interpretations can be different from the data representation used for in the predictions. We could use whatever is more convenient for the interpretation, as long as we keep a mapping between both data representations.

A classic example of this are superpixels from images in image classification. A superpixel represents a segment of an image that group pixeles that are interconnected and share similar colors. As opposed to individual pixels, this representation is natural for humans and simplifies the identification of specific regions that could have high influence in the classification of the image. For instance, if a machine learning model classifies the below picture as a goose, it's very likely that the superpixel representing the beak was selected in the LIME interpretation. The simulated data in this case is represented by copies of the original picture where some superpixels are zeroed (set to white), so the local model is able to distinguish between superpixels that have an impact in the classification and those that are less relevant.

(image taken from https://pbiecek.github.io/ema/LIME.html)



Interpretation with clusters of variables

Taking the superpixel example as inspiration we could look for a more abstract data representation easier to undestand that individual variables in our tabular data. In a picture, pixels are correlated by color similarity and by proximity in the spacial axes. In tabular data, variables could be grouped by linear correlation. The interpretable data representation would be then clusters grouping the original variables. An expert in the domain of our prediction model would be able to advise if the representation is useful.

Clustering could be based on:
- correlation of variables regardless the classification
- multicollinearity using package mctest
- some grouping based on PCA weights
- ...

We'll create 100 clusters based on correlation:

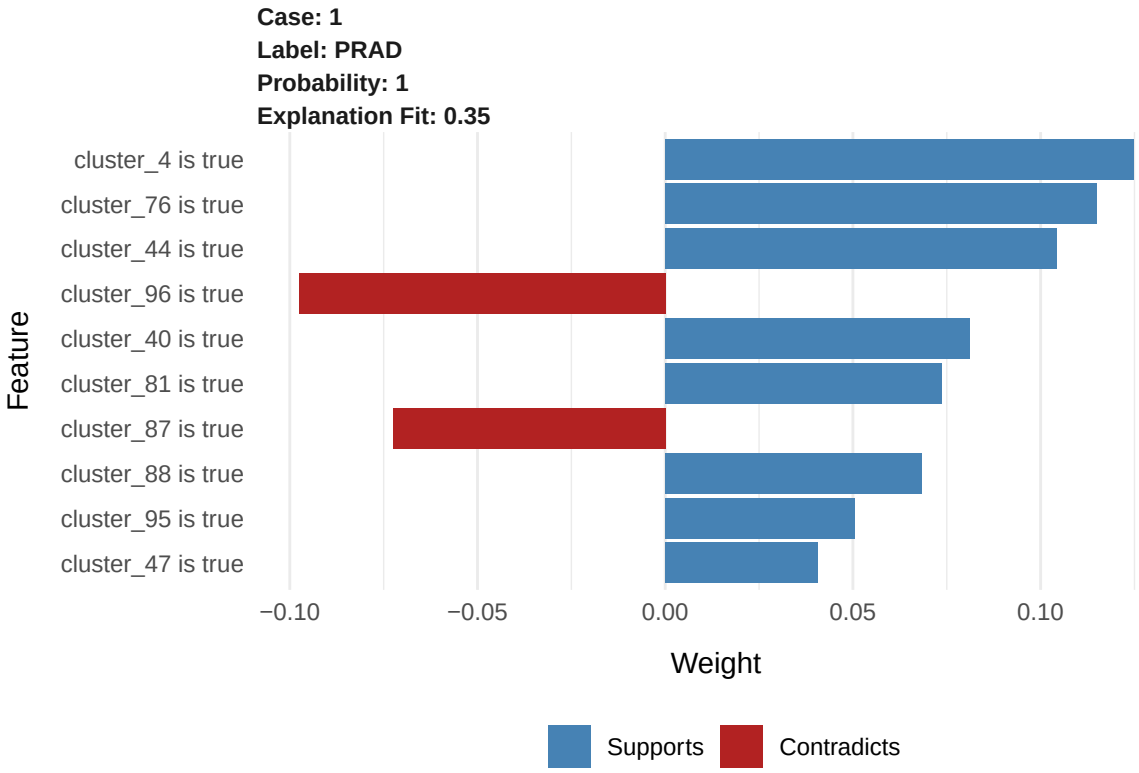
TODO: mirar "grouped lasso" para comparar con las interpretaciones de grupos de genes muy correlacionados.
TODO: hacerlo con con multicolinearidad

```
library(ClustOfVar)
#TODO: dig more into this package (hclustvar, ...)
#TODO: other libraries: corclust

num_clusters = 100
kmeansvar = kmeansvar(X.quanti = as.matrix(genes), X.quali = as.matrix(as.factor(cancer)), init = num_cl

clusters = kmeansvar$cluster
clusters = clusters[1:(length(clusters)-1)]  #X.quali column?
```

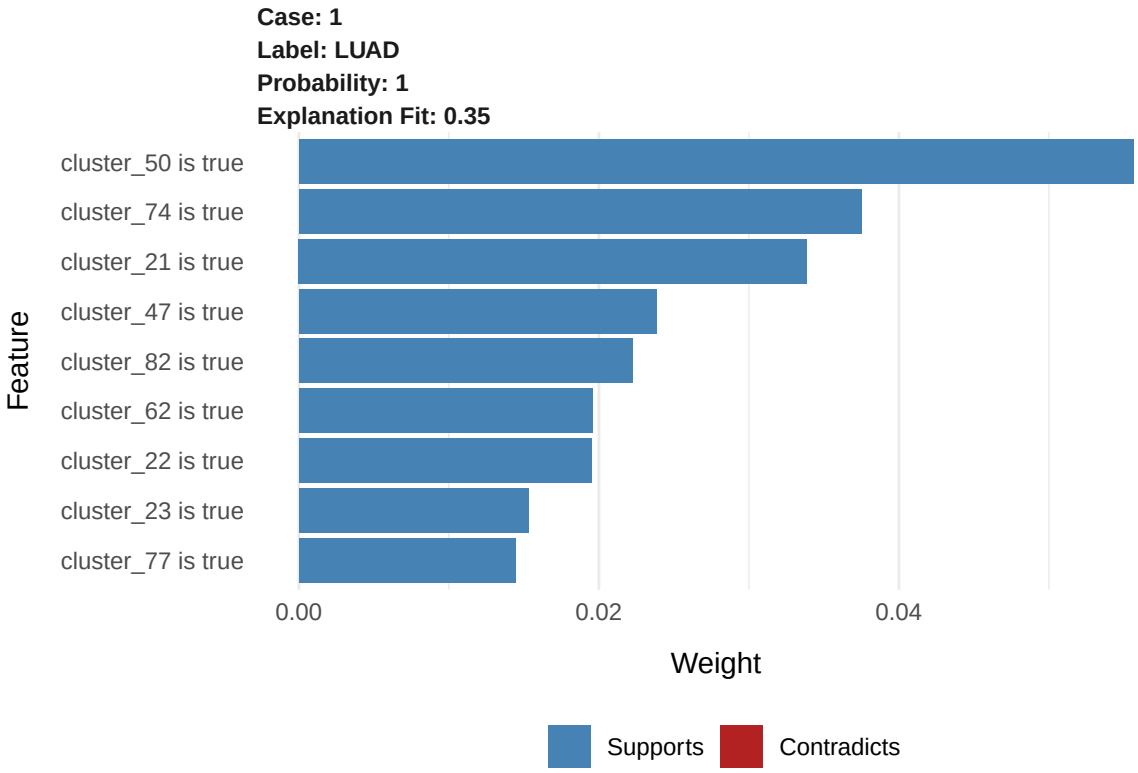
80% of the clusters will be randomly zeroed in each simulated data point in order for the model to find relevant clusters for the classification of the instance of interest.

```
# "p is the percentage of non-zeroed clusters to use in each simulated data point
get_cluster_explanation <- function(datapoint_index, p = 0.2)
{
  # the instance of interest to interpret
  instance = as.data.frame(x_test)[datapoint_index,]

  preprocessing <- function(x) {

    # the instance of interest is replicated n_permutations times
    toblackbox = instance[rep(1, nrow(x)),]

    # then, 0s are set for all the variables contained in an inactive cluster
    # (the permutations will randomly active p% of the clusters of each simulated data point)
    for (i in 1:nrow(x)) {

      for (k in 1:ncol(x))
      {
        if (!x[i,k])
          toblackbox[i,names(clusters[clusters == k])] = 0

        # alternatively, variable means instead of 0s (doesn't change the result too much):
        #vars_in_cluster = names(clusters[clusters == k])
        #
        #if (length(vars_in_cluster) > 1)
```

```
        #  toblackbox[i,names(clusters[clusters == k])] = apply(x_train[,vars_in_cluster], 2, mean)
        #else
        #  toblackbox[i,names(clusters[clusters == k])] = mean(x_train[,vars_in_cluster])
      }
    }

    as.matrix(toblackbox)
  }

  #
  sim_dist = as.data.frame(matrix(FALSE, nrow = 100, ncol = num_clusters))
  sim_dist[1:round(p*100),] = TRUE

  for (i in 1:num_clusters)
    colnames(sim_dist)[i] = as.character(sprintf("cluster_%s",i))

  explainer_cluster <- lime(
    x = as.data.frame(sim_dist),
    model = black_box,
    use_density = TRUE,
    preprocess = preprocessing,
  )

  # vector of active clusters (all 1s for the instance of interest)
  all_ones = as.data.frame(matrix(TRUE, nrow = 1, ncol = num_clusters))
  for (i in 1:num_clusters)
    colnames(all_ones)[i] = as.character(sprintf("cluster_%s",i))

  lime::explain(
    x = all_ones,
    explainer = explainer_cluster,
    n_permutations = 500,
    #kernel_width = 0.75,
    feature_select = "lasso_path",
    n_features = n_features,
    n_labels = 1
  )
}

# PRAD category
plot_features(get_cluster_explanation(which(cancer[-training_index] == 4)[1]))
```



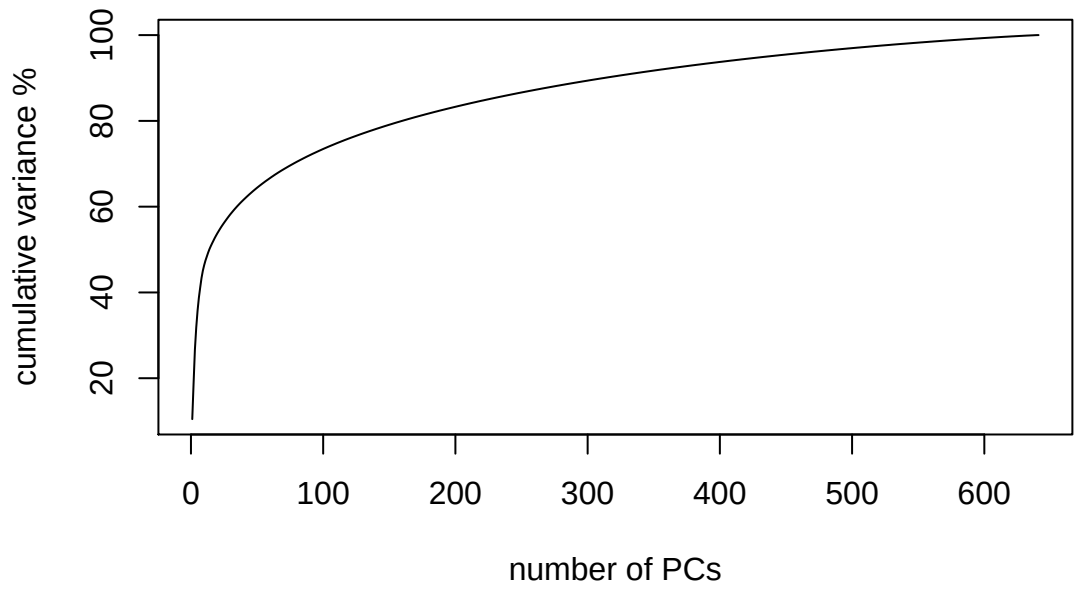

```
# LUAD category
plot_features(get_cluster_explanation(which(cancer[-training_index] == 3)[1]))
```


Case: 1
Label: LUAD
Probability: 1
Explanation Fit: 0.35
cluster_50 is true
cluster_74 is true
cluster_21 is true
cluster_47 is true
cluster_82 is true
cluster_62 is true
cluster_22 is true
cluster_23 is true
cluster_77 is true
Feature
0.00
0.02
0.04
Weight
Supports
Contradicts

Interpretation with PCA

Another way to get a more interpetable data representation would be through dimension reduction like PCA. Again, the expert in the domain has to advise if the representation would be useful for interpretation. The expert might be able to understand the meaning of the main components.

For a fair comparison with the global surrogate lasso model, we'll do interpretations with lime of observations in the training data.
The number of components in the interpretable data representation could be reduced to explain a percentage of the explained variance, since probably only a few dozens of them will be really influencial. We'll start with the first 100 PCs for now.



The black box only understands data as represented originally with individual gene expressions (the optimal representation for prediction rather than PCA which destroys information), therefore the simulated data in PCA format has to be converted back to the original format in order to get predictions (see *preprocessing* function below).

```
# The number of pcs to use in the interpretable data representation:
pcs_n = 100

get_PCA_explanation <- function(datapoints_index, n_permutations = 2000, kernel_width = 0.75)
{
  preprocessing <- function(x){

    # reversing PCA (with the remaining info) to feed the black box which only understands individual g
    a = as.matrix(x) %*% t(x_train_pca$rotation[,1:pcs_n])
    b = t(a) + x_train_pca$center
    t(b)
  }
```

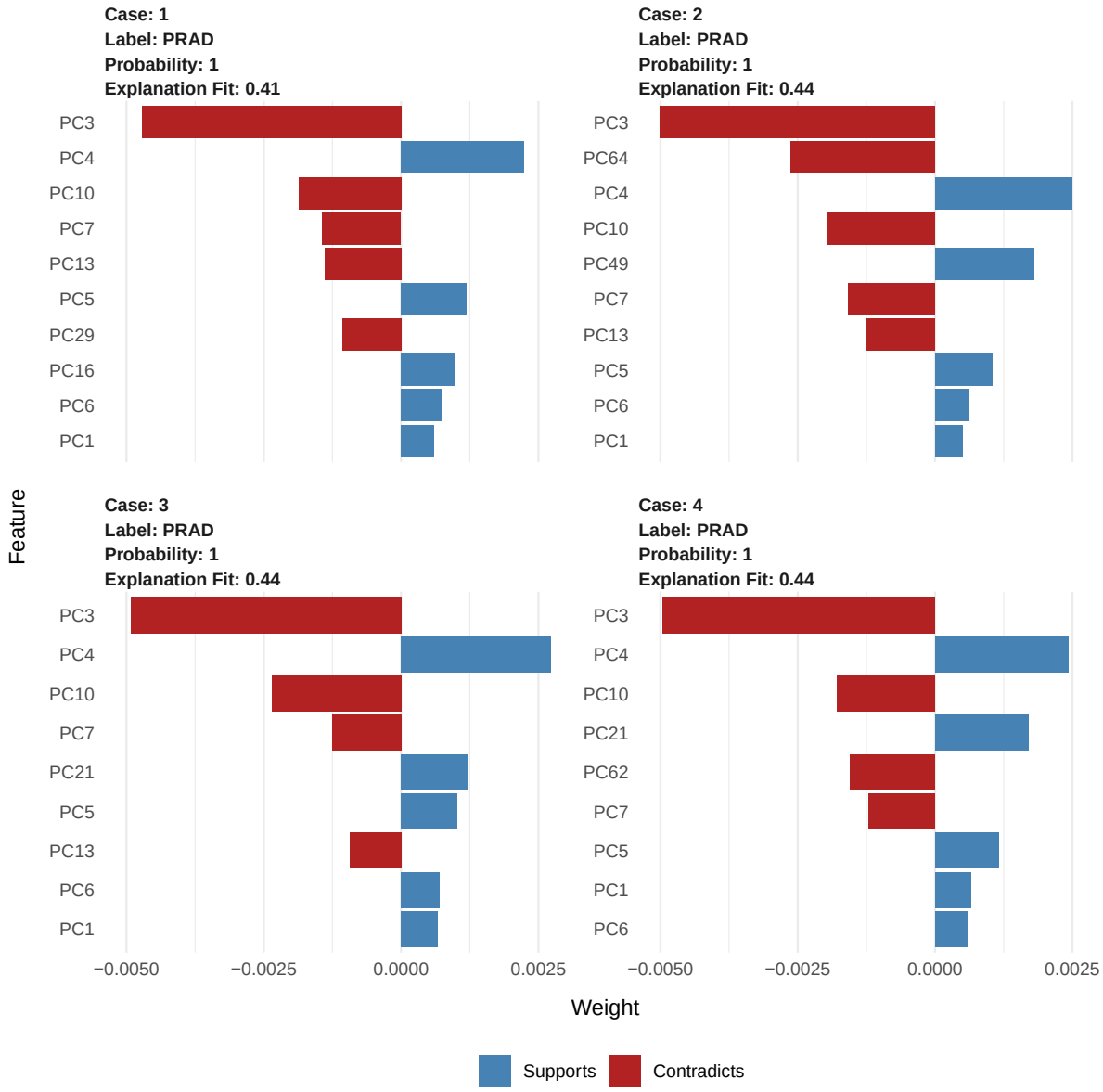
```
  explainer_PCA <- lime(
    x = as.data.frame(x_train_pca$x)[,1:pcs_n],
    model = black_box,
    use_density = TRUE,
    preprocess = preprocessing,
    bin_continuous = FALSE
  )

  lime::explain(
    # converting test data point coordinates to the PCA space:
    x = as.data.frame(x_train[datapoints_index,] %*% x_train_pca$rotation[,1:pcs_n]),
    explainer = explainer_PCA,
    n_permutations = n_permutations,
    kernel_width = kernel_width,
    feature_select = "lasso_path",
    n_features = 10,
    n_labels = 1
  )
}
```

We analyse the interpretation of an observation of category PRAD (this category barely overlaps with other categories), and we try the interpretation of another observation of category LUAD (this category overlaps with other categories as seen in the PCA plots).
For both observations the process is run 4 times to check the consistency of the interpretation. The 10 more influential components are displayed.

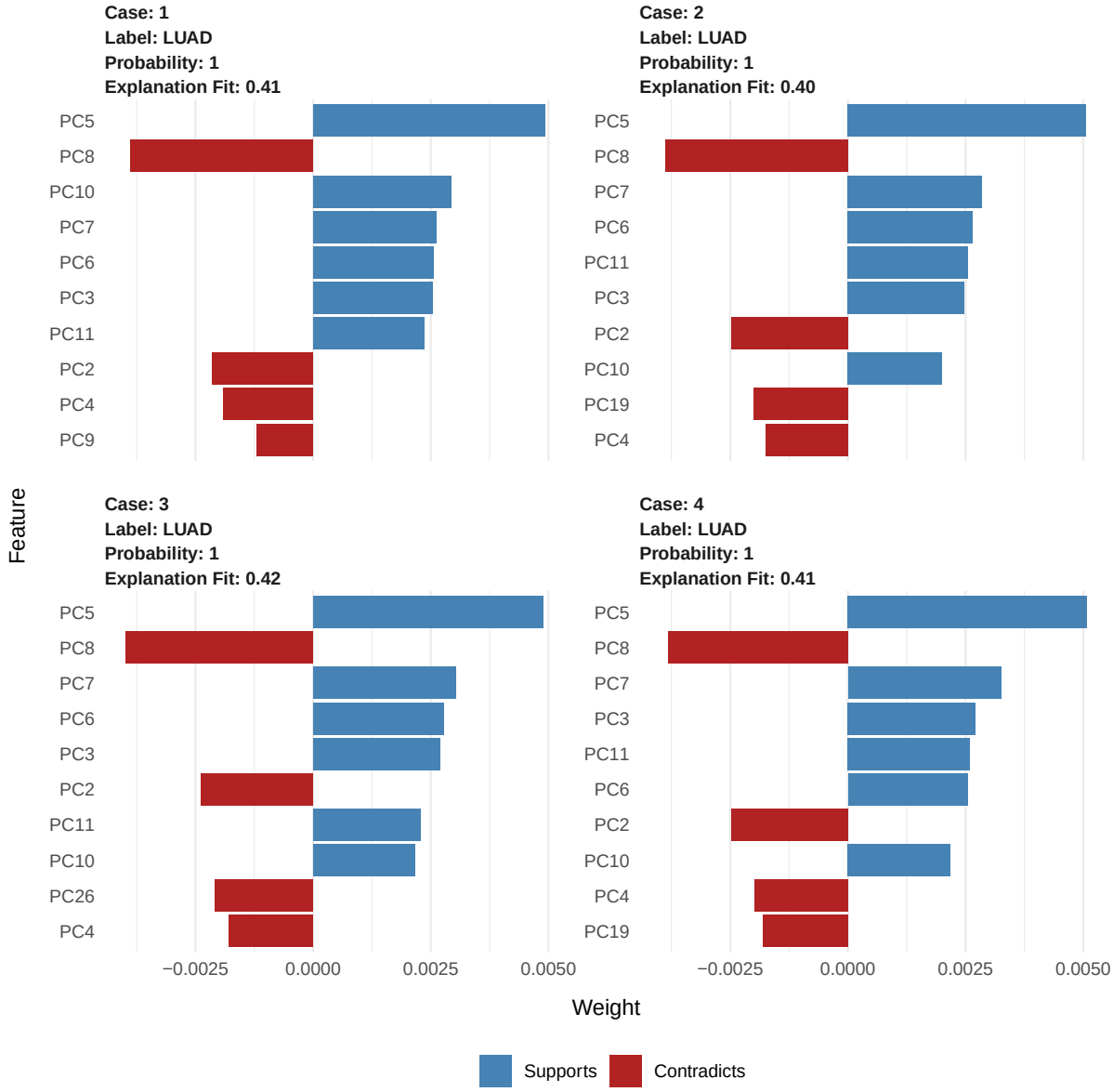
The results are:

- For PRAD (doesn't overlap with other categories) the results are consistent. R^2 is usually high (depends on the randomly selected observation). The interpretation is dominated by 1 or 2 components.
- For LUAD (overlaps with other categories) the selected components are also rather consistent. R^2 is also usually high. The interpretation is also dominated by 1 or 2 components but they are not as dominant as with category PRAD and the non-dominant features are more repeated between interpretations.

```
# PRAD category
plot_features(get_PCA_explanation(rep(which(cancer[training_index] == 4)[1],4),5000))
```




```
# LUAD category
plot_features(get_PCA_explanation(rep(which(cancer[training_index] == 3)[1],4),5000))
```



We now check the correlation between the PCA version of the global surrogate model and the local lime models.

The lasso surrogate model is fitted again with 10 selected components to fairly compare them with the 10 selected features with lime (as close to 10 features as possible, it depends on the lambda grid and the category):

```
##                        PC1           PC2          PC3         PC4         PC5
## BRCA (global)  0.00000000  0.0541381436  0.000000000 -0.06619196 -0.04535951
## COAD (global)  0.00000000  0.0000000000  0.009072814  0.04433518  0.00000000
## KIRC (global) -0.07625818 -0.0003581036  0.000000000  0.00000000 -0.04270580
## LUAD (global)  0.00000000  0.0000000000  0.005459586 -0.01356734  0.04840768
## PRAD (global)  0.00000000  0.0000000000 -0.111228804  0.01432965  0.02443005
##                        PC6         PC7          PC8         PC9         PC10
```

```
## BRCA (global) -0.002134318 0.00000000  0.008411147 0.00000000  0.000000000
## COAD (global) -0.029045550 0.00000000  0.033088989 0.04696238  0.000000000
## KIRC (global)  0.000000000 0.00000000 -0.009069941 0.00000000  0.000000000
## LUAD (global)  0.031320021 0.02925901 -0.037065463 0.00000000  0.004099198
## PRAD (global)  0.017202335 0.00000000  0.000000000 0.00000000 -0.018759535
##                       PC11         PC12         PC13          PC19       PC21
## BRCA (global) -0.012607526  0.000000000  0.009447639  0.0040293573 0.00000000
## COAD (global)  0.000000000  0.000000000  0.000000000  0.0000000000 0.00000000
## KIRC (global)  0.001238345  0.000000000  0.008180170  0.0000000000 0.01184772
## LUAD (global)  0.000000000  0.000000000  0.000000000 -0.0005394707 0.00000000
## PRAD (global)  0.000000000 -0.002659667 -0.001842822  0.0000000000 0.00000000
##                       PC23       PC31         PC33        PC44        PC48
## BRCA (global)  0.00000000 0.00000000 -0.001866413 0.000000000  0.00000000
## COAD (global) -0.00571402 0.00000000  0.000000000 0.000000000 -0.00532328
## KIRC (global)  0.00000000 0.01993513  0.010459089 0.000000000  0.00000000
## LUAD (global)  0.00000000 0.00000000  0.000000000 0.006636278  0.00000000
## PRAD (global)  0.00000000 0.00000000  0.000000000 0.000000000  0.00000000
##                       PC60         PC84       PC87        PC94
## BRCA (global) 0.000000000  0.000000000 0.00000000  0.00000000
## COAD (global) 0.000000000 -0.009969731 0.04150675  0.03610162
## KIRC (global) 0.000000000  0.000000000 0.00000000  0.00000000
## LUAD (global) 0.002374309  0.000000000 0.00000000  0.00000000
## PRAD (global) 0.000000000  0.000000000 0.00000000 -0.05488685
```

The lime local models are fitted for all the observations in categories LUAD and PRAD (more than 100 each) and using 3000 permutations for each interpretation.

```
##                   PC1          PC2          PC3          PC4          PC5
## LUAD 1  0.0000000000 -0.002323763 0.002460679 -0.001851932 0.004821859
## LUAD 2  0.0000000000 -0.002420161 0.002554443 -0.002022234 0.004858894
## LUAD 3  0.0004651804 -0.002246611 0.002557491 -0.001862698 0.004714116
## LUAD 4  0.0000000000 -0.002297402 0.002567524 -0.001813657 0.005033887
## LUAD 5  0.0000000000 -0.002172493 0.002569935 -0.001908158 0.005206666
## LUAD 6  0.0000000000 -0.002250870 0.002735246 -0.001876689 0.004801606
## LUAD 7  0.0000000000 -0.002308387 0.002669860 -0.001852405 0.005105771
## LUAD 8  0.0005418803 -0.002202154 0.002759326 -0.001866455 0.005082815
## LUAD 9  0.0000000000 -0.002278294 0.002700428 -0.002125256 0.004899645
## LUAD 10 0.0000000000 -0.002120400 0.002746543 -0.001676670 0.005109790
##                  PC6          PC7          PC8          PC9         PC10
## LUAD 1  0.002319580 0.002865268 -0.003830479  0.000000000 0.002147113
## LUAD 2  0.002285594 0.002334913 -0.004161241  0.000000000 0.002540231
## LUAD 3  0.002345523 0.002654881 -0.003888931  0.000000000 0.002351031
## LUAD 4  0.002435896 0.003132686 -0.003345379  0.000000000 0.002525356
## LUAD 5  0.002187095 0.003394425 -0.004370338  0.000000000 0.001921459
## LUAD 6  0.002194543 0.003354945 -0.003865404  0.000000000 0.002868216
## LUAD 7  0.002704012 0.003360815 -0.004148382  0.000000000 0.002472940
## LUAD 8  0.002369010 0.003244898 -0.004097025  0.000000000 0.002584850
## LUAD 9  0.002624135 0.002661142 -0.003695984  0.000000000 0.001986101
## LUAD 10 0.002797851 0.003243745 -0.003980478 -0.001140001 0.002282070
##                  PC11 PC12 PC13 PC14 PC15 PC16         PC17         PC19 PC23
## LUAD 1  0.002261385    0    0    0    0    0  0.000000000  0.000000000    0
## LUAD 2  0.003136002    0    0    0    0    0  0.000000000  0.000000000    0
## LUAD 3  0.003148193    0    0    0    0    0  0.000000000  0.000000000    0
```

```
## LUAD 4  0.002469490    0    0    0    0    0  0.000000000  0.000000000    0
## LUAD 5  0.002821844    0    0    0    0    0 -0.002032023  0.000000000    0
## LUAD 6  0.002227375    0    0    0    0    0  0.000000000 -0.002209513    0
## LUAD 7  0.002000432    0    0    0    0    0  0.000000000  0.000000000    0
## LUAD 8  0.002582589    0    0    0    0    0  0.000000000  0.000000000    0
## LUAD 9  0.002846188    0    0    0    0    0  0.000000000  0.000000000    0
## LUAD 10 0.002766673    0    0    0    0    0  0.000000000  0.000000000    0
##                PC26 PC30 PC39 PC43 PC47 PC49 PC60 PC68         PC82 PC93
## LUAD 1   0.000000000    0    0    0    0    0    0    0 -0.002884433    0
## LUAD 2  -0.002293709    0    0    0    0    0    0    0  0.000000000    0
## LUAD 3   0.000000000    0    0    0    0    0    0    0  0.000000000    0
## LUAD 4  -0.002513907    0    0    0    0    0    0    0  0.000000000    0
## LUAD 5   0.000000000    0    0    0    0    0    0    0  0.000000000    0
## LUAD 6   0.000000000    0    0    0    0    0    0    0  0.000000000    0
## LUAD 7   0.000000000    0    0    0    0    0    0    0  0.000000000    0
## LUAD 8   0.000000000    0    0    0    0    0    0    0  0.000000000    0
## LUAD 9  -0.002615767    0    0    0    0    0    0    0  0.000000000    0
## LUAD 10  0.000000000    0    0    0    0    0    0    0  0.000000000    0
```

```
##                 PC1         PC2          PC3         PC4          PC5
## PRAD 1  0.0006046064 0.000000000 -0.004969727 0.002831052 0.0011740098
## PRAD 2  0.0007689730 0.000000000 -0.004639658 0.002535452 0.0009753798
## PRAD 3  0.0006716339 0.000000000 -0.004764693 0.002447236 0.0008181420
## PRAD 4  0.0006258110 0.000350817 -0.005058267 0.002720093 0.0011125384
## PRAD 5  0.0006467580 0.000000000 -0.004604643 0.002479888 0.0012702488
## PRAD 6  0.0004953733 0.000000000 -0.004998899 0.002725104 0.0012063162
## PRAD 7  0.0005337636 0.000000000 -0.004933918 0.002267366 0.0009554057
## PRAD 8  0.0005311648 0.000000000 -0.004922030 0.002663771 0.0010751715
## PRAD 9  0.0006023049 0.000000000 -0.005276429 0.002489265 0.0013102365
## PRAD 10 0.0006219551 0.000000000 -0.004934547 0.002531582 0.0007965765
##                 PC6          PC7           PC9         PC10          PC11
## PRAD 1  0.0009102947 -0.001007275  0.0000000000 -0.001643636  0.0000000000
## PRAD 2  0.0000000000 -0.001303932  0.0000000000 -0.001748691 -0.0010732888
## PRAD 3  0.0009322965 -0.001416357 -0.0007446328 -0.001939112 -0.0008674610
## PRAD 4  0.0006391906 -0.001550046  0.0000000000 -0.001739671  0.0000000000
## PRAD 5  0.0000000000 -0.001258126  0.0000000000 -0.001769528 -0.0010707430
## PRAD 6  0.0000000000 -0.001882749  0.0000000000 -0.001965646 -0.0009204314
## PRAD 7  0.0005299527 -0.001109906  0.0000000000 -0.002020179  0.0000000000
## PRAD 8  0.0006845177 -0.001174426  0.0000000000 -0.002042090  0.0000000000
## PRAD 9  0.0007380430 -0.001666809  0.0000000000 -0.001768815 -0.0012883571
## PRAD 10 0.0008047743 -0.001497558  0.0000000000 -0.002013799  0.0000000000
##                PC12          PC13         PC14 PC15 PC16 PC17         PC18
## PRAD 1  0.000000000  0.0000000000  0.000000000    0    0    0  0.000000000
## PRAD 2  0.000000000 -0.0009562604 -0.001275816    0    0    0  0.000000000
## PRAD 3  0.000000000  0.0000000000  0.000000000    0    0    0  0.000000000
## PRAD 4  0.000000000  0.0000000000  0.000000000    0    0    0  0.000000000
## PRAD 5  0.001003682 -0.0012538969 -0.001074985    0    0    0  0.000000000
## PRAD 6  0.000000000 -0.0010910434  0.000000000    0    0    0  0.000000000
## PRAD 7  0.000000000  0.0000000000  0.000000000    0    0    0  0.000000000
## PRAD 8  0.000000000  0.0000000000  0.000000000    0    0    0  0.000000000
## PRAD 9  0.000000000  0.0000000000  0.000000000    0    0    0  0.000000000
## PRAD 10 0.000000000 -0.0007293540  0.000000000    0    0    0 -0.001518846
##                PC19 PC20         PC21         PC24 PC25 PC26 PC28 PC29 PC31
## PRAD 1   0.00000000    0 0.001173123 0.000000000    0    0    0    0    0
```

```
## PRAD 2  -0.00107045    0 0.000000000 0.000000000    0    0    0    0    0
## PRAD 3   0.00000000    0 0.000000000 0.000000000    0    0    0    0    0
## PRAD 4   0.00000000    0 0.001238474 0.000000000    0    0    0    0    0
## PRAD 5   0.00000000    0 0.000000000 0.000000000    0    0    0    0    0
## PRAD 6   0.00000000    0 0.000000000 0.000000000    0    0    0    0    0
## PRAD 7   0.00000000    0 0.000000000 0.000000000    0    0    0    0    0
## PRAD 8   0.00000000    0 0.001136349 0.000000000    0    0    0    0    0
## PRAD 9   0.00000000    0 0.000000000 0.001591072    0    0    0    0    0
## PRAD 10  0.00000000    0 0.000000000 0.000000000    0    0    0    0    0
##                PC32 PC33 PC34          PC35 PC38 PC39 PC40         PC41 PC42 PC48
## PRAD 1  0.000000000    0    0  0.000000000    0    0    0 0.000000000    0    0
## PRAD 2  0.000000000    0    0  0.000000000    0    0    0 0.000000000    0    0
## PRAD 3  0.000000000    0    0  0.000000000    0    0    0 0.000000000    0    0
## PRAD 4  0.000000000    0    0  0.000000000    0    0    0 0.000000000    0    0
## PRAD 5  0.000000000    0    0  0.000000000    0    0    0 0.000000000    0    0
## PRAD 6  0.000000000    0    0 -0.001519787    0    0    0 0.000000000    0    0
## PRAD 7  0.001825896    0    0  0.000000000    0    0    0 0.001932476    0    0
## PRAD 8  0.001325742    0    0  0.000000000    0    0    0 0.000000000    0    0
## PRAD 9  0.000000000    0    0  0.000000000    0    0    0 0.000000000    0    0
## PRAD 10 0.000000000    0    0  0.000000000    0    0    0 0.000000000    0    0
##                PC49 PC50 PC51 PC52 PC53 PC55 PC56 PC57 PC58 PC59         PC60
## PRAD 1  0.000000000    0    0    0    0    0    0    0    0    0 0.000000000
## PRAD 2  0.000000000    0    0    0    0    0    0    0    0    0 0.000000000
## PRAD 3  0.000000000    0    0    0    0    0    0    0    0    0 0.001871665
## PRAD 4  0.000000000    0    0    0    0    0    0    0    0    0 0.000000000
## PRAD 5  0.000000000    0    0    0    0    0    0    0    0    0 0.000000000
## PRAD 6  0.002029511    0    0    0    0    0    0    0    0    0 0.000000000
## PRAD 7  0.000000000    0    0    0    0    0    0    0    0    0 0.000000000
## PRAD 8  0.000000000    0    0    0    0    0    0    0    0    0 0.001834583
## PRAD 9  0.000000000    0    0    0    0    0    0    0    0    0 0.000000000
## PRAD 10 0.000000000    0    0    0    0    0    0    0    0    0 0.000000000
##         PC62 PC63 PC64 PC65 PC66 PC69 PC70 PC71 PC72 PC74 PC75 PC76 PC77 PC79
## PRAD 1     0    0    0    0    0    0    0    0    0    0    0    0    0    0
## PRAD 2     0    0    0    0    0    0    0    0    0    0    0    0    0    0
## PRAD 3     0    0    0    0    0    0    0    0    0    0    0    0    0    0
## PRAD 4     0    0    0    0    0    0    0    0    0    0    0    0    0    0
## PRAD 5     0    0    0    0    0    0    0    0    0    0    0    0    0    0
## PRAD 6     0    0    0    0    0    0    0    0    0    0    0    0    0    0
## PRAD 7     0    0    0    0    0    0    0    0    0    0    0    0    0    0
## PRAD 8     0    0    0    0    0    0    0    0    0    0    0    0    0    0
## PRAD 9     0    0    0    0    0    0    0    0    0    0    0    0    0    0
## PRAD 10    0    0    0    0    0    0    0    0    0    0    0    0    0    0
##         PC81 PC82        PC83       PC84 PC86 PC87 PC88 PC89         PC91 PC92
## PRAD 1     0    0 0.001738664 0.00000000    0    0    0    0 0.002429354    0
## PRAD 2     0    0 0.000000000 0.00000000    0    0    0    0 0.000000000    0
## PRAD 3     0    0 0.000000000 0.00000000    0    0    0    0 0.000000000    0
## PRAD 4     0    0 0.000000000 0.00240811    0    0    0    0 0.000000000    0
## PRAD 5     0    0 0.000000000 0.00000000    0    0    0    0 0.000000000    0
## PRAD 6     0    0 0.000000000 0.00000000    0    0    0    0 0.000000000    0
## PRAD 7     0    0 0.000000000 0.00000000    0    0    0    0 0.000000000    0
## PRAD 8     0    0 0.000000000 0.00000000    0    0    0    0 0.000000000    0
## PRAD 9     0    0 0.000000000 0.00000000    0    0    0    0 0.000000000    0
## PRAD 10    0    0 0.000000000 0.00000000    0    0    0    0 0.000000000    0
##                PC97 PC98 PC100
```

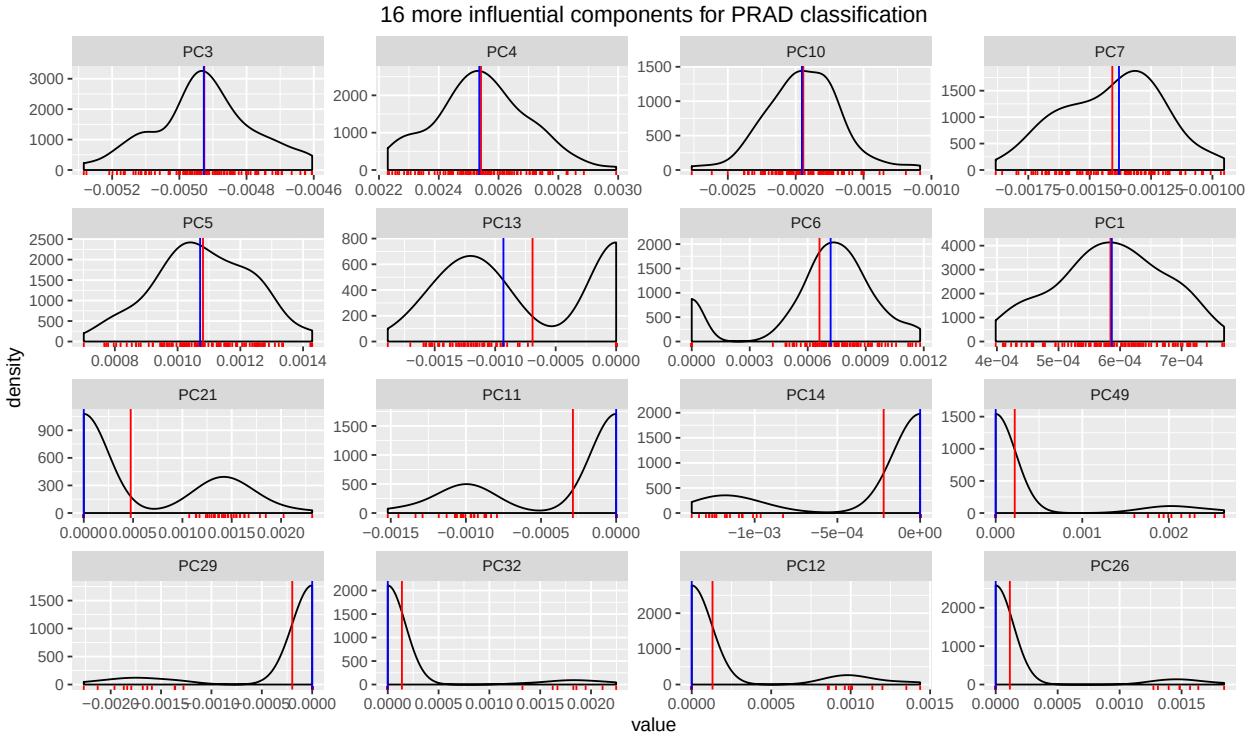
```
## PRAD 1   0.00000000    0     0
## PRAD 2   0.00000000    0     0
## PRAD 3   0.00000000    0     0
## PRAD 4   0.00000000    0     0
## PRAD 5   0.00000000    0     0
## PRAD 6   0.00000000    0     0
## PRAD 7   0.00000000    0     0
## PRAD 8   0.00000000    0     0
## PRAD 9   0.00000000    0     0
## PRAD 10 -0.00294429    0     0
```

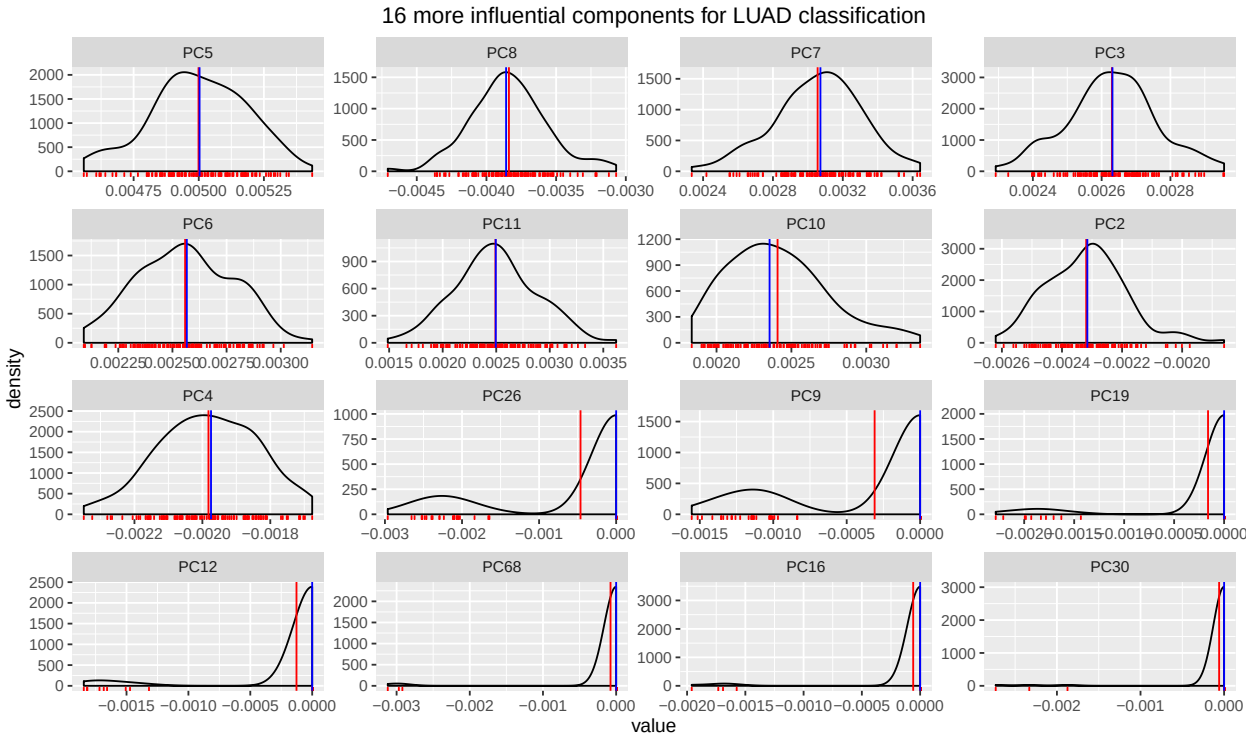
In order to compare the global lasso model with the lime models, the coefficients of the selected features in the interpretations of all the observations for the same category are summarized in a single list of component coefficients. Each summarized component coefficient will be computed with a statistic describing the central tendency of the coefficient across the observations.

To decide which statistic to use we analyse the more influential features for each category. To measure the influence we use the sum of the absolute values of the coefficients of all the observations for each component:

```
##
## sum of the absoulte values of the coefficients for PRAD:

##     PC3     PC4    PC10     PC7     PC5    PC13     PC6     PC1    PC21    PC11
## 0.51721 0.26683 0.20412 0.14782 0.11350 0.07257 0.06938 0.06136 0.04992 0.03022
##    PC14    PC49    PC29    PC32    PC12    PC26
## 0.02314 0.02306 0.02077 0.01461 0.01372 0.01199

##
## sum of the absoulte values of the coefficients for LUAD:

##     PC5     PC8     PC7     PC3     PC6    PC11    PC10     PC2     PC4    PC26
## 0.58489 0.44914 0.35754 0.30769 0.29934 0.29166 0.28170 0.27137 0.23190 0.05410
##     PC9    PC19    PC12    PC68    PC16    PC30
## 0.03625 0.01878 0.01484 0.00901 0.00696 0.00694
```

Below are displayed the distributions of the 16 more influential components for both categories. The vertical red line represents the mean, the blue one represents the median.





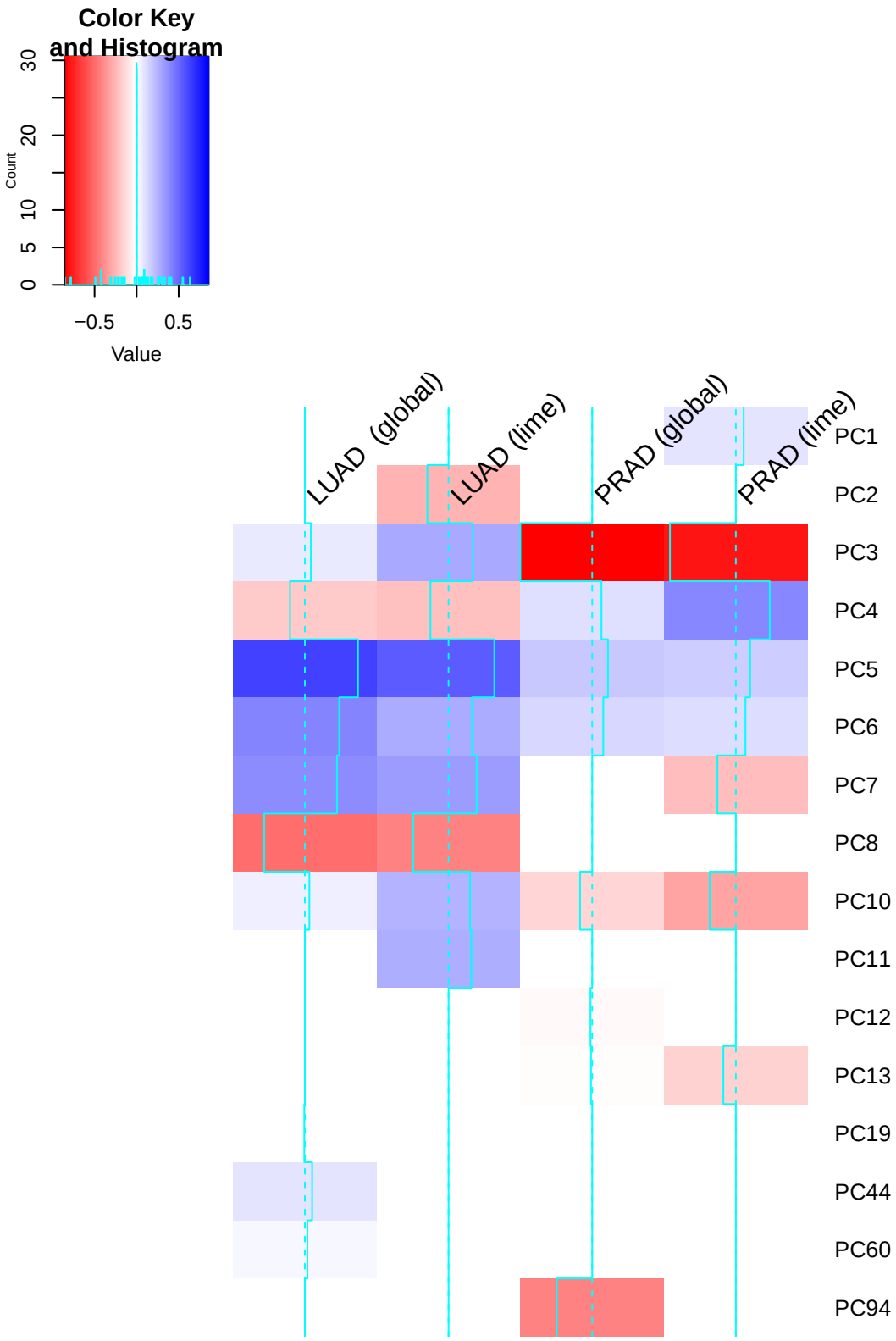
We can see that the more influential components are more or less symmetric, whereas the less influentual ones are bimodal, one of the modes lying on value 0 which represents the absence of influence for a subset of observations.

To avoid the components that only appear as influential for a few observations (and when they are included in the list of 10 more important components their value is very small) we use the median to compute the representing value of the component coefficient for the whole category. The median will ignore components that come up rarely and have small values by setting them to 0 in the summarize component coefficient.

Also note that the variance of the coefficients for the more dominant components is higher in category LUAD (the one overlapping other categories in the data space - harder to predict):

```
##
## standard deviation of first 3 more influential component coefficients for PRAD:

##     PC3     PC4    PC10
## 0.00015 0.00016 0.00028

##
##
## standard deviation of first 3 more influential component coefficients for LUAD:

##     PC5     PC8     PC7
## 0.00018 0.00028 0.00025
```

We compute a single set of coefficients for each category using the median of the components coefficients from all the observations for both categories, and get a histogram to compare them with the surrogate lasso model coefficients:

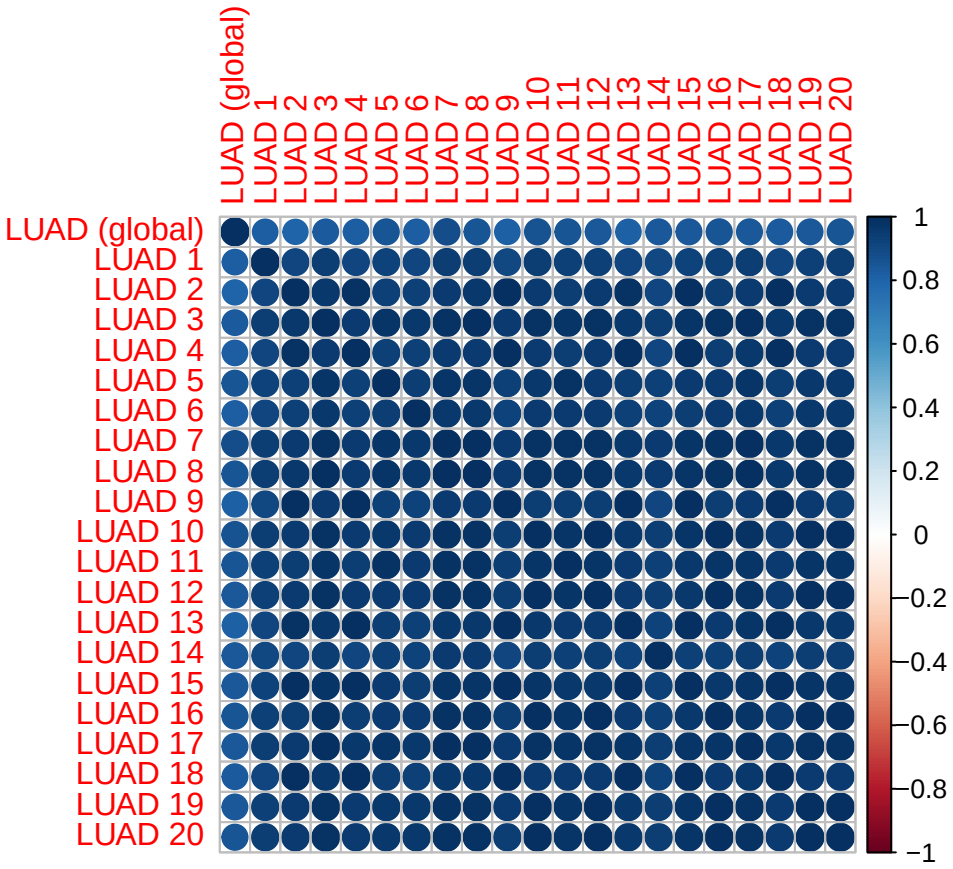

Color Key
and Histogram
Count
0
5
10
20
30
−0.5
0.5
Value
LUAD (global)
LUAD (lime)
PRAD (global)
PRAD (lime)
PC1
PC2
PC3
PC4
PC5
PC6
PC7
PC8
PC10
PC11
PC12
PC13
PC19
PC44
PC60
PC94

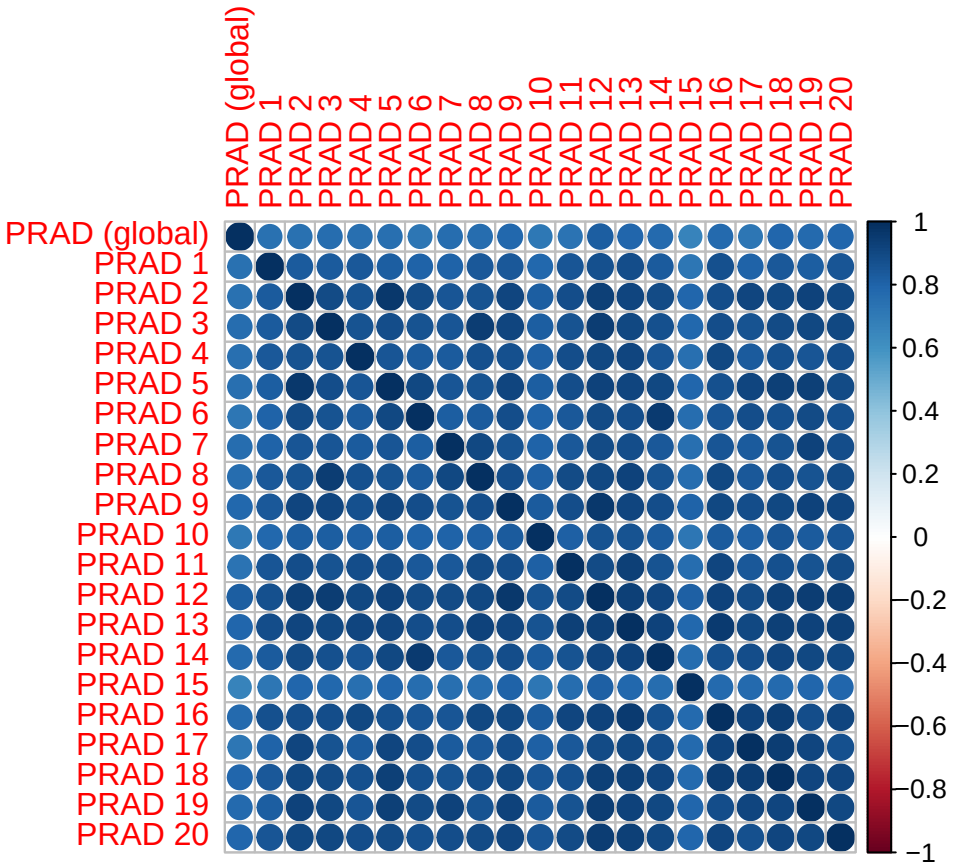
We can see there is high correlation, specially for category LUAD.

TODO: por que LUAD se ajusta mejor?
TODO: por que PC1 y PC2 son tomados en cuenta en lime pero no el global lasso?
TODO: he tenido que normalizar las filas, por que los coeficientes en lime son un orden de magnitud mas pequenos comparados con los coeficientes de global lasso. Por que esa diferencia? Coeficientes originales:

```
##      BRCA (global) COAD (global) KIRC (global) LUAD (global) PRAD (global)
## PC1    0.000000000   0.000000000 -0.0762581832   0.000000000    0.00000000
## PC2    0.054138144   0.000000000 -0.0003581036   0.000000000    0.00000000
## PC3    0.000000000   0.009072814  0.0000000000   0.005459586   -0.11122880
## PC4   -0.066191962   0.044335177  0.0000000000  -0.013567343    0.01432965
## PC5   -0.045359507   0.000000000 -0.0427057954   0.048407684    0.02443005
## PC6   -0.002134318  -0.029045550  0.0000000000   0.031320021    0.01720233
## PC7    0.000000000   0.000000000  0.0000000000   0.029259006    0.00000000
## PC8    0.008411147   0.033088989 -0.0090699412  -0.037065463    0.00000000
## PC9    0.000000000   0.046962381  0.0000000000   0.000000000    0.00000000
## PC10   0.000000000   0.000000000  0.0000000000   0.004099198   -0.01875954
##       LUAD (lime)   PRAD (lime)
## PC1   0.000000000  0.0005865937
## PC2  -0.002315540  0.0000000000
## PC3   0.002631493 -0.0049271707
## PC4  -0.001974565  0.0025354517
## PC5   0.005003060  0.0010717400
## PC6   0.002566378  0.0007183955
## PC7   0.003071912 -0.0013805995
## PC8  -0.003858931  0.0000000000
## PC9   0.000000000  0.0000000000
## PC10  0.002354290 -0.0019541252
```

Correlation plot of coefficients between lime interpretations of 20 observations and the global lasso model for the same category:


LUAD (global)
LUAD 1
LUAD 2
LUAD 3
LUAD 4
LUAD 5
LUAD 6
LUAD 7
LUAD 8
LUAD 9
LUAD 10
LUAD 11
LUAD 12
LUAD 13
LUAD 14
LUAD 15
LUAD 16
LUAD 17
LUAD 18
LUAD 19
LUAD 20
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1



We can see that there is less correlation between coefficients for category PRAD (easier to predict). The reason for this could be that with PRAD we get 1 or 2 dominant components explaining the predictions, while the rest of the components are less relevant and therefore have smaller and more variable values. With LUAD, different observations have more components explaining the prediction in common, even if the variance of the weights of these common components is higher compared to PRAD.