In this tutorial I want to show how you can use alluvial plots to visualise model response in up to 4 dimensions. easyalluvial generates an artificial data space using fixed values for unplotted variables or uses the partial dependence plotting method. It is model agnostic but offers some convenient wrappers for caret models.

**Introduction**

**Taking a peek**

When building machine learning models we are usually faced with a trade-off between accurracy and interpretability. However even if we tend to lean towards accuracy and pick a modelling method that results in nearly uninterpretable models we can still make use of model agnostic interporetation techniques that have been summarized.

Whithout getting to theoretical I personally always feel the urge to simply take a peek simulate some data and see how the model reacts a method. In order to simulate data we can generate a a vector with a sequence of values over the entire range of a predictor variable of interest while setting all the others to their median or mode and use this artificial data space to obtain model predictions which we can plot against our variable of interest. An R package that will do this for you is (plotmo. Instead of ranging over 1 predictor variable we can create a data grid using 2 predictor variables and plot the response as a third dimension. However this is as far as you can go in a conventional plot. Alluvial plots can line up much more than 3 dimensions on a plane next to each other only limited by the number of flows as it will get too cluttered when there are too many of them.

**Which variables to plot**

When using conventional model response plotting beeing limited two 2 variables we can simply resolve this by generating many plots and look at them one by one. Alluvial plots require a bit more attention and cannot easily be screened and compared since visually there is so much going on. Therefore I do not recommend to brute force it by simply creating a lot of random combinations of predictor variables and multiple alluvial plots but instead to focus on those that have the highest calculated feature importance. Feature importance values are natively provided by most modelling packages. So the question is how many can we plot and it turns out 4 features will usually result in well-interpretable plot.

**Generating the data space**

suppressPackageStartupMessages( require(tidyverse) )

suppressPackageStartupMessages( require(easyalluvial) )

suppressPackageStartupMessages( require(mlbench) )

suppressPackageStartupMessages( require(randomForest) )

We start by creating a model

data('BostonHousing')

df = as\_tibble( BostonHousing )

m = randomForest( lstat ~ ., df )

and looking at the importance features

imp = m$importance %>%

tidy\_imp(df) #

knitr::kable(imp)

| **vars** | **imp** |
| --- | --- |
| medv | 8766.4579 |
| rm | 3492.9529 |
| age | 2628.2033 |
| indus | 2308.9869 |
| crim | 2244.6537 |
| dis | 1767.7827 |
| nox | 1591.9735 |
| b | 696.5375 |
| tax | 662.6383 |
| ptratio | 420.4378 |
| rad | 196.8711 |
| zn | 137.9414 |
| chas | 119.5868 |

When generating the data space we cannot screen an infinite amount of values per variable but will have to limit ourselves to a fixed number. Then we want to create all possible combinations between these values which determines the number of flows. The visual limit for flows on an alluvial plot is somewhere around 1000 flows. Thus I recommend to go with 5 values which will result in 5 x 5 X 5 X 5 –> 625 combinations and the same number of flows. That also leaves some wiggeling room if one of the top 4 variables is a factor with more than 5 levels. get\_data\_space() will split the range of a variable into 3 and pick the median of each split and add the variable minimum and the maximum to the set.

dspace = get\_data\_space(df, imp

, degree = 4 # specifies the number of variables

, bins = 5 # the number of values per variable

)

knitr::kable( head(dspace, 10) )

| **medv** | **rm** | **age** | **indus** | **crim** | **dis** | **nox** | **b** | **tax** | **ptratio** | **rad** | **zn** | **chas** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 3.561 | 2.9 | 0.46 | 0.25651 | 3.20745 | 0.538 | 391.44 | 330 | 19.05 | 5 | 0 | 0 |
| 5 | 3.561 | 2.9 | 4.27 | 0.25651 | 3.20745 | 0.538 | 391.44 | 330 | 19.05 | 5 | 0 | 0 |
| 5 | 3.561 | 2.9 | 9.90 | 0.25651 | 3.20745 | 0.538 | 391.44 | 330 | 19.05 | 5 | 0 | 0 |
| 5 | 3.561 | 2.9 | 18.10 | 0.25651 | 3.20745 | 0.538 | 391.44 | 330 | 19.05 | 5 | 0 | 0 |
| 5 | 3.561 | 2.9 | 27.74 | 0.25651 | 3.20745 | 0.538 | 391.44 | 330 | 19.05 | 5 | 0 | 0 |
| 5 | 3.561 | 32.2 | 0.46 | 0.25651 | 3.20745 | 0.538 | 391.44 | 330 | 19.05 | 5 | 0 | 0 |
| 5 | 3.561 | 32.2 | 4.27 | 0.25651 | 3.20745 | 0.538 | 391.44 | 330 | 19.05 | 5 | 0 | 0 |
| 5 | 3.561 | 32.2 | 9.90 | 0.25651 | 3.20745 | 0.538 | 391.44 | 330 | 19.05 | 5 | 0 | 0 |
| 5 | 3.561 | 32.2 | 18.10 | 0.25651 | 3.20745 | 0.538 | 391.44 | 330 | 19.05 | 5 | 0 | 0 |
| 5 | 3.561 | 32.2 | 27.74 | 0.25651 | 3.20745 | 0.538 | 391.44 | 330 | 19.05 | 5 | 0 | 0 |

Total rows in dspace: 625

dspace %>%

summarise\_all( ~ length( unique(.) ) ) %>%

knitr::kable()

| **medv** | **rm** | **age** | **indus** | **crim** | **dis** | **nox** | **b** | **tax** | **ptratio** | **rad** | **zn** | **chas** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 5 | 5 | 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Generating model response**

pred = predict(m, newdata = dspace)

**Plotting**

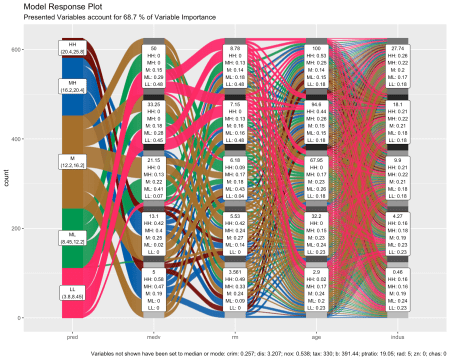
The predictions will be binned as well into 5 bins. Binning options can be passed as a list via the params\_bin\_numeric\_pred parameter.

p = alluvial\_model\_response(pred, dspace, imp

, degree = 4, bins = 5

, stratum\_label\_size = 2.8 )

p



We can see the binned range of predictions and explore by which variable combination they have been created by tracing the coloured flows. The stratum labels of the prediction variables indicate the value of the variable and which fraction of the flows of that colour (prediction variable bin range) pass through that stratum.

**Marginal histograms**

As well as for other easyalluvial plots we can add marginal histograms and as a bonus also the feature importance.

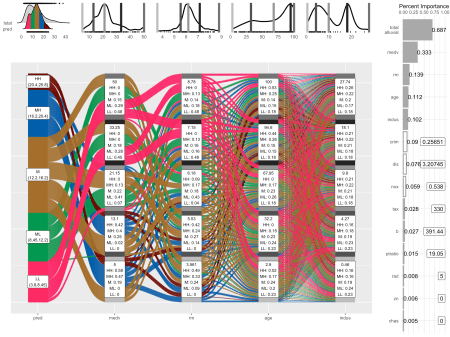
p\_grid = add\_marginal\_histograms(p, data\_input = df

, plot = F # plot only after adding feature importance

, scale = 50 # to increase distance between ridge plots, Default: 400

) %>%

add\_imp\_plot( p = p, data\_input = df)



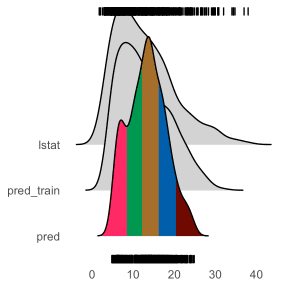
We can see the original distribution of the variables, the lines indicate the position of the values as picked by get\_data\_space(). When comparing the distribution of the predictions against the original distribution of lstat we see that the range of the predictions in response to the artificial dataspace do not cover all of the range of lstat. Which most likely means that all possible combinations of the 4 plotted variables in combination with moderate values for all other predictors will not give any extreme values. Which we can further explore, first we will need to check whether the model is capable of making predictions in the lower and upper ranges of lsat. We can use plot\_hist() to only plot the distributions and add the prediction for the training data set using the pred\_train parameter.

pred\_train = predict(m)

plot\_hist('pred', p, df

, pred\_train = pred\_train # pred\_train can also be passed to add\_marginal\_histograms()

, scale = 50)



We see that the training prediction also do not completely cover all of the lstat range but more of it than the predictions from the aritficial data space.

If we wanted to emphasize this we can bin the data space predictions on the basis of the training predictions. In this case it makes sense to increase the number of bins for the predictions in order not to loose resolution on the plot.

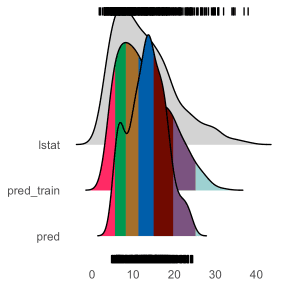
p = alluvial\_model\_response(pred, dspace, imp, degree = 4, bins = 7

, bin\_labels = c("LLL","LL", "ML", "M", "MH", "HH", "HHH")

, pred\_train = pred\_train

, stratum\_label\_size = 2.8 )

plot\_hist('pred', p, df, scale = 50)



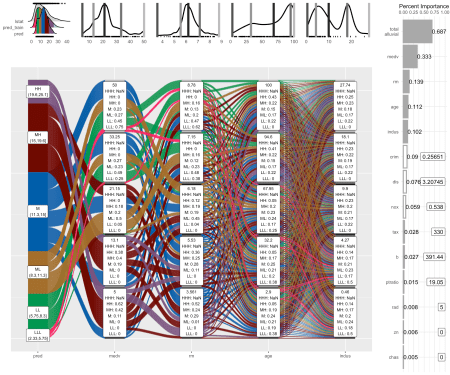
p\_grid = add\_marginal\_histograms(p, data\_input = df

, plot = F # plot only after adding feature importance

, scale = 50 # to increase distance between ridge plots, Default: 400

) %>%

add\_imp\_plot( p = p, data\_input = df)



**What feature combinations are needed to obtain predictions in the lower and higher ranges of lstat?**

We can add the training predictions to the training data and assign the variables not covered by the model response plot to bins. We can then create an alluvial plot over the entire training dataframe in order to trace the lstat ranges not covered by the model response alluvial plot above. Since we are only interested in those observations we remove all other flows from the plot by setting their color to NA.

breaks = c( min(pred\_train) - 1,min(pred),max(pred),max(pred\_train) + 1 )

df\_inv = df %>%

select(-lstat) %>%

mutate( pred\_train = pred\_train

, pred\_train = cut(pred\_train, breaks) ) %>%

select( pred\_train, one\_of(imp$vars) ) # order by feature importance

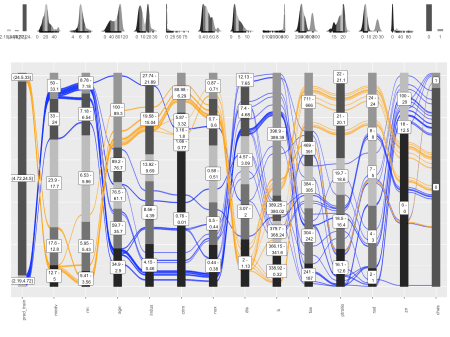
p = alluvial\_wide(df\_inv, bin\_labels = 'min\_max'

, stratum\_label\_size = 3

, col\_vector\_flow = c('blue', NA, 'orange')

, colorful\_fill\_variable\_stratum = F)

p\_grid = add\_marginal\_histograms(p, df\_inv)



We can see that for the upper ranges of lstat a pretty specific combination values is required from 8 feature variables from medv until tax only then begin the flows to scatter over the eintire range of the following variables. The lower range values of lstat start to scatter at crim or dis. Interestingly there seems to be a set of houses that branches off from the other houses in the lower range of lstat predictions already at the third feature, age. We could potentially explore this further, as a reminder you can get the binned input data for each alluvial plot by calling p$data\_key and trace the variables of interest.

**caret wrapper**

caret provides a uniformous interface for training and calling a lot of machine learning models as well as for calculating feature importance. easyalluvial provides a function that wraps the above described workflow into one single call. Note that feature importance values are slightly different. randomForest returns a combined importance for all levels of a factor variable while caret splits them up. tidy\_imp() aggregates them and uses the maximum value of all values. This behaviour can be adjusted by passing another aggregating function to the .f parameter.

train = caret::train( lstat ~ .

, df

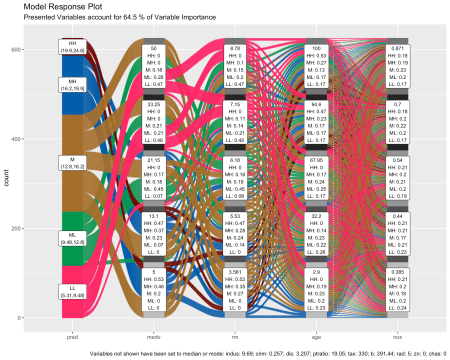
, method = 'rf'

, trControl = caret::trainControl( method = 'none' )

, importance = TRUE )

alluvial\_model\_response\_caret(train, degree = 4, bins = 5

, stratum\_label\_size = 2.8)



**Advantages**

Model response alluvial plots can help us to get an immediate intuitive understanding how predictions of a certain range can be generated by the model. They can be understood by none-statistical stakeholders and invite the viewer to start exploring and question the decision making process of the model while also conveying an appreciation for the model complexity as flows branch out to the variables of lower feature importance.

**Partial Dependence Plotting Method**

When simulating artificial data space we need to be aware that we will create data space that is outside the feature distribution space of our original training data. In this example we could be creating houses that cannot exist as such maybe a 5 square meter appartment with 10 bedrooms. We have to keep this in mind when interpretating the plot. Moreover by setting all the none-plotted values to median/mode we might also be creating observations outside the possible feature distribution space but this will be hidden and not foreseable even by somebody who has some subject matter expertise. Simply imagine how likely it would be if I meassurre a hundred different feautures of the inhabitants of a certain country how likely will it be that I find one person that is exactly the average for all of those measurements? So to alleviate this we can use the partial dependency plotting method:

1. construct the data space as described above, but instead of setting all none-plotted variables to median/mode we set them to the values found in the first row of the training data.
2. Repeat 1. for each row
3. Use all data spaces to get model predictions
4. Average model predictions

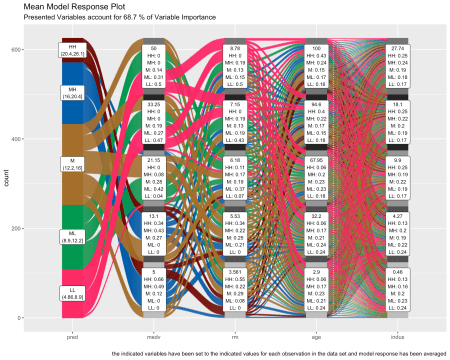
Using this method we reduce the risk of simulating data outside the feature distribution space concerning the variables that are not plotted.

pred = get\_pdp\_predictions(df, imp, m, degree = 4, bins = 5)

alluvial\_model\_response(pred, dspace, imp, degree = 4

, method = 'pdp' # changes title and caption

, stratum\_label\_size = 2.8 )

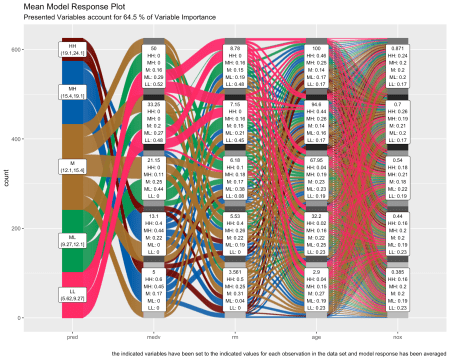


thanks to the uniformous modelling interface of caret alluvial\_model\_response\_caret() can call get\_pdp\_predictions() directly.

alluvial\_model\_response\_caret(train, degree = 4, bins = 5

, method = 'pdp'

, stratum\_label\_size = 2.8 )



**A note on feature importance**

Calculating feature importance in a dataset with strongly correlating variables will lead to inacurrate results. It usually means that two strongly correlating variables share the importance that would be accredited to them if only one of them was present in the data set. Further some feature importance calculation methods require the model to be trained with scaled, centered and potentially normalized (transformed to be more normally distributed) numerical variables in order to deliver meaningful results. We can use the recipes package to perform these tasks. Using the below described workflow we can create the artifical dataspace with untransformed numerical values which will give us more meaningful output.

suppressPackageStartupMessages( require(recipes) )

# filter correlating variables recipe- ------------------------------

# caret tends to complain if recipe filters any variables, therefore

# we use two recipes

rec\_filt\_corrs = recipe(df, lstat ~ . ) %>%

step\_corr( all\_numeric()

, - all\_outcomes()

, threshold = 0.7 ) %>%

prep()

df\_filt\_corrs = bake(rec\_filt\_corrs, df)

# transformation recipe --------------------------------------------

rec\_trans = recipe( df\_filt\_corrs, lstat ~ .) %>%

step\_center( all\_numeric(), - all\_outcomes() ) %>%

step\_scale( all\_numeric(), - all\_outcomes() ) %>%

step\_YeoJohnson( all\_numeric(), - all\_outcomes() ) %>%

prep()

# train and plot ---------------------------------------------------

set.seed(1)

train = caret::train( rec\_trans

, df\_filt\_corrs

, method = 'earth'

, trControl = caret::trainControl( method = 'cv'

, search = 'random')

, tuneLength = 100)

pred\_train = predict(train, df\_filt\_corrs)

p = alluvial\_model\_response\_caret(train, degree = 4, bins = 5

, pred\_train = pred\_train

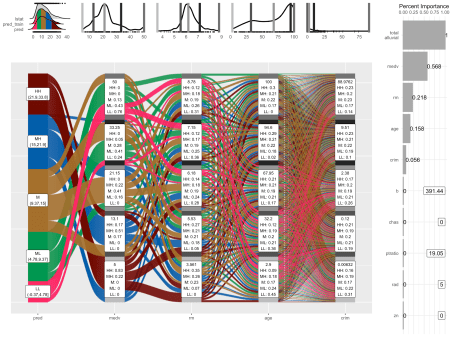
, stratum\_label\_size = 2.8 )

p\_grid = add\_marginal\_histograms(p, df, plot = F

, pred\_var = 'lstat'

, scale = 50) %>%

add\_imp\_plot(p, df)



**None-regression models**

Works the same way.

df = select(mtcars2, -ids)

m = randomForest::randomForest( am ~ ., df)

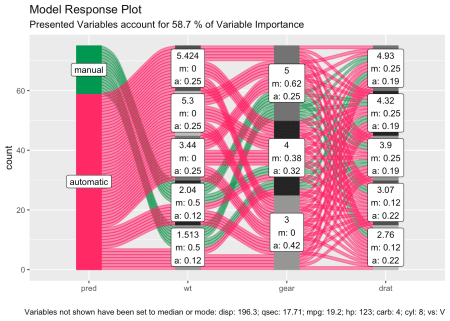
imp = m$importance

dspace = get\_data\_space(df, imp, degree = 3)

pred = predict(m, newdata = dspace,type = 'response')

p = alluvial\_model\_response(pred, dspace, imp, degree = 3)

p



**Limitations**

* There is a loss of information when binning the numerical variables
* The combinations generated when making the grid might be outside the feature distribution space (generate combinations that are impossible)
* We only look at the combination of 4 features and disregard the others

To alleviate this you can reduce the complexity of the model by reducing features (take out correlating variables) or use additional model exploration methods such as classical PDPs, ALE plots, Shapely values, etc, …