The DALEX package version 2.0 was released on CRAN on September the 4th, followed by the DALEXtra release. At first, I'd like to assure You, we did our best to maintain backward compatibility. I am here to cover major changes and at the same time present workflow using mlr3 and xgboost models. For those who meet DALEX for the first time, I encourage You to visit other posts on this blog. Changes to cover today are as follow:

- Multiclass classification with DALEX
- xgboost model explainer

Let's begin!

Multiclass classification

The first version of the DALEX package was released in 2018. During those two years, it was under constant development bringing new features online. In 2019 DALEXtra—an extension for DALEX was released and in February 2020 DALEX 1.0.0 hit CRAN. Despite that, we still lacked one functionality which is complex support for the multiclass classification model. Of course, thanks to the flexibility of DALEX such models could be explained, but it might have been challenging for some users to specify all custom functions. Now with DALEX 2.0 online, it is no longer necessary.

We still require a model to have probability output, ie. for n-class classification we look for n a column matrix where every column is a probability of the class i. For regression and classification tasks, DALEX requires ay vector to be numeric. On the contrary, when the task is multiclass we require y vector of true labels to be a factor. Residuals of the model are calculated according to the one minus probability of the true class formula.

$$r_i = P(\hat{y}_i = y_i | X)$$

That means if for observation which has the true label ok the model has returned probability 0.6 of belonging to the class ok, calculated residuum is 0.4. Such an assumption allows us to use residual based methods. Obviously, if the package is supported, DALEX will recognize on its own whether the model is a multiclass, binary, or regression problem. Additionally model_performance now calculates measures dedicated to multilabel classification such as macro and micro F1 or macro weighted AUC.

To show examples we will use the HR dataset available in the DALEX package. It is an artificial dataset so results achieved are easy to interpret and we can judge without any doubt if they are true.

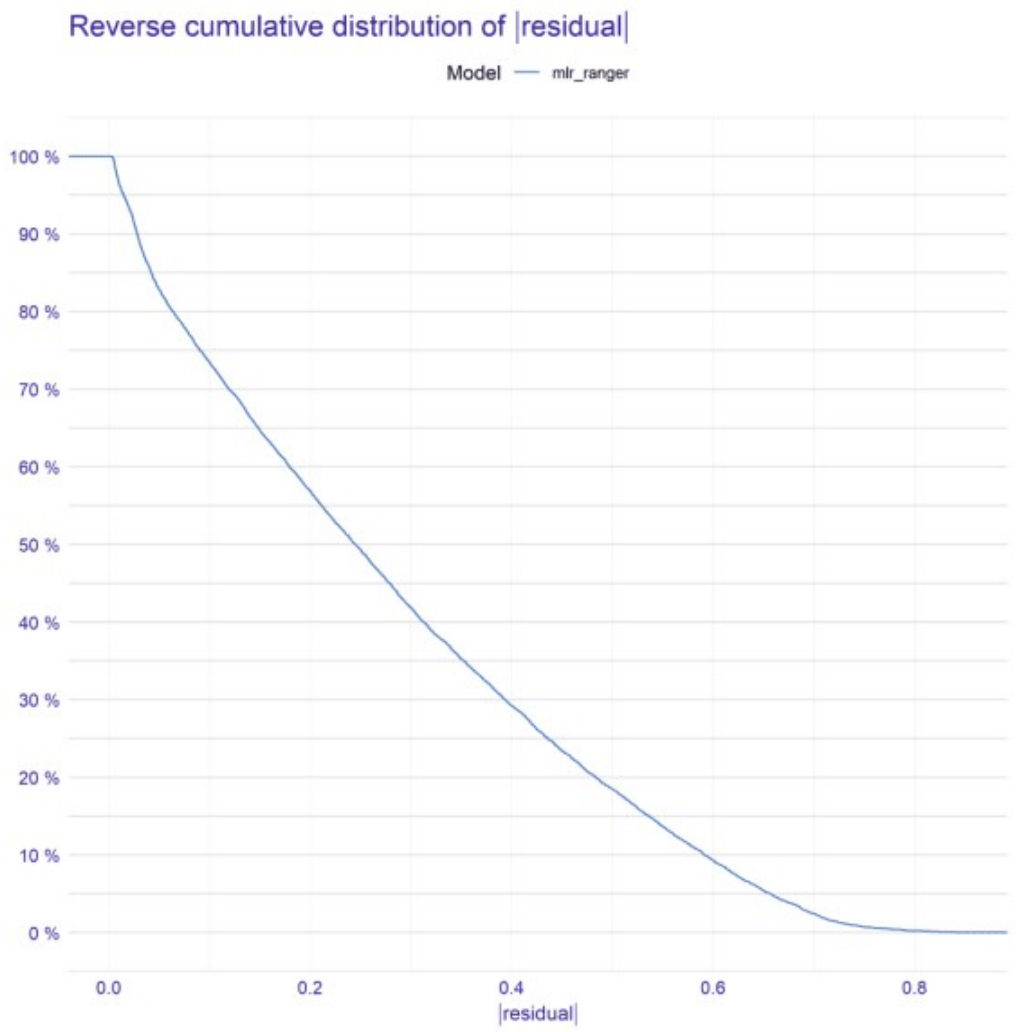
To show examples we will use the HR dataset available in the DALEX package. It is an artificial dataset so results achieved are easy to interpret and we can judge without any doubt if they are true.

```
library(mlr3)
library(mlr3learners)
library(DALEXtra)

HR_task <- TaskClassif$new(id = "HR",
 backend = HR,
 target = "status")
HR_ranger <- lrn("classif.ranger", predict_type = "prob")
HR_ranger$param_set$values <- list(num.trees = 1000)
HR_ranger$train(HR_task)
explainer_ranger <- explain_mlr3(model = HR_ranger,
 data = HR,
 y = HR$status,
 verbose = FALSE,
 label = "mlr_ranger")
```

With the explainer ready, we can use model_performance to inspect the basic parameters of models and calculate measures. We can also analyze residuals distribution.

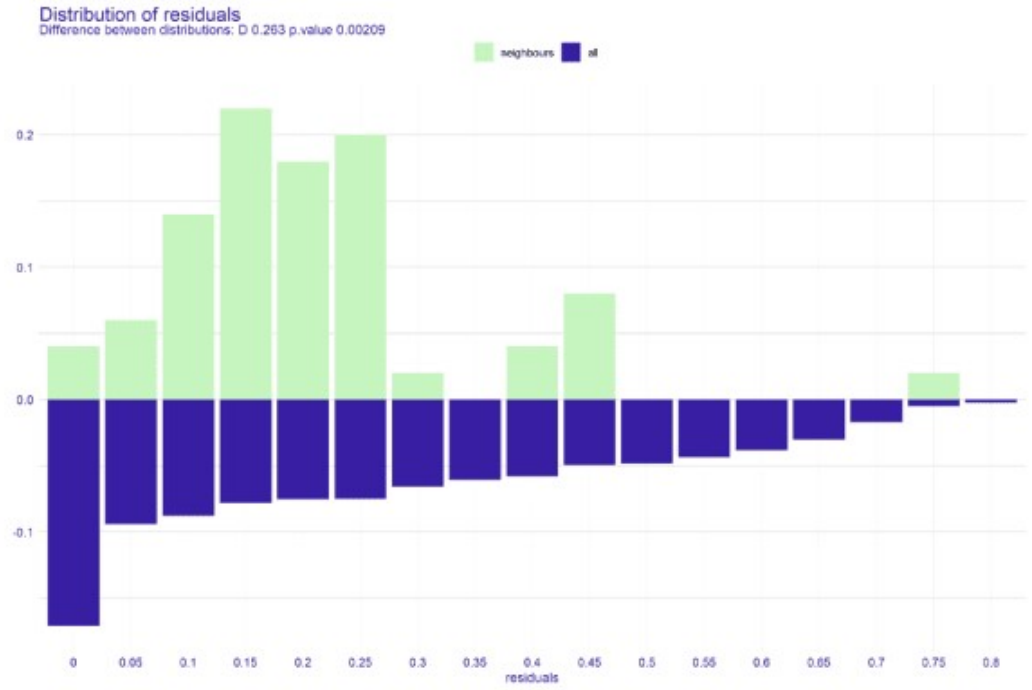
```
mp_ranger <- model_performance(explainer_ranger)
```

```
plot(mp_ranger)
```



Residuals are rather uniformly distributed. The model's behavior is more than acceptable. Obviously accuracy of approximately 87% is not a peak of machine learning possibilities for that task, but also still a very good result. Worth mentioning observation is that micro_F1 score and accuracy are the same. It is not a coincidence, those measures are the same for models that answer exactly one label for one observation. If You want to get to know more about model_performance, visit this book.

```
pd_all_ranger <- predict_diagnostics(explainer_ranger, HR[1,])
plot(pd_all_ranger)
```



The above plot shows a distribution of residuals for all observations versus residuals for 50 neighbors. The difference in distribution brings attention immediately, especially the peak in [0.15, 0.25] compartment and lack of *big* residuals. Moreover, the statistical test proves that those two distributions differ but in favor of the model.

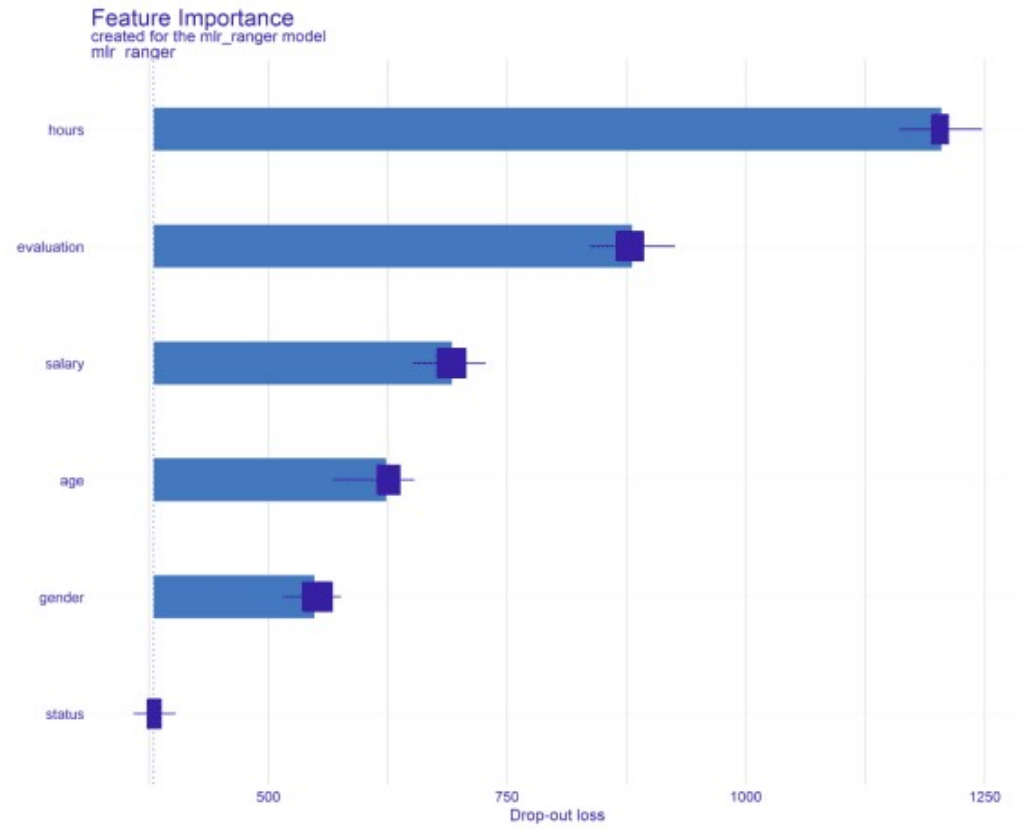
DALEX provides some implemented loss functions, like default one for regression tasks which is the root of mean square error, and for classification tasks—one minus AUC score. Cross entropy is also one of those.

$$H = \sum_{i=1}^{n} -\log(P(\hat{y_i} = y_i|X))$$

cross-entropy formula

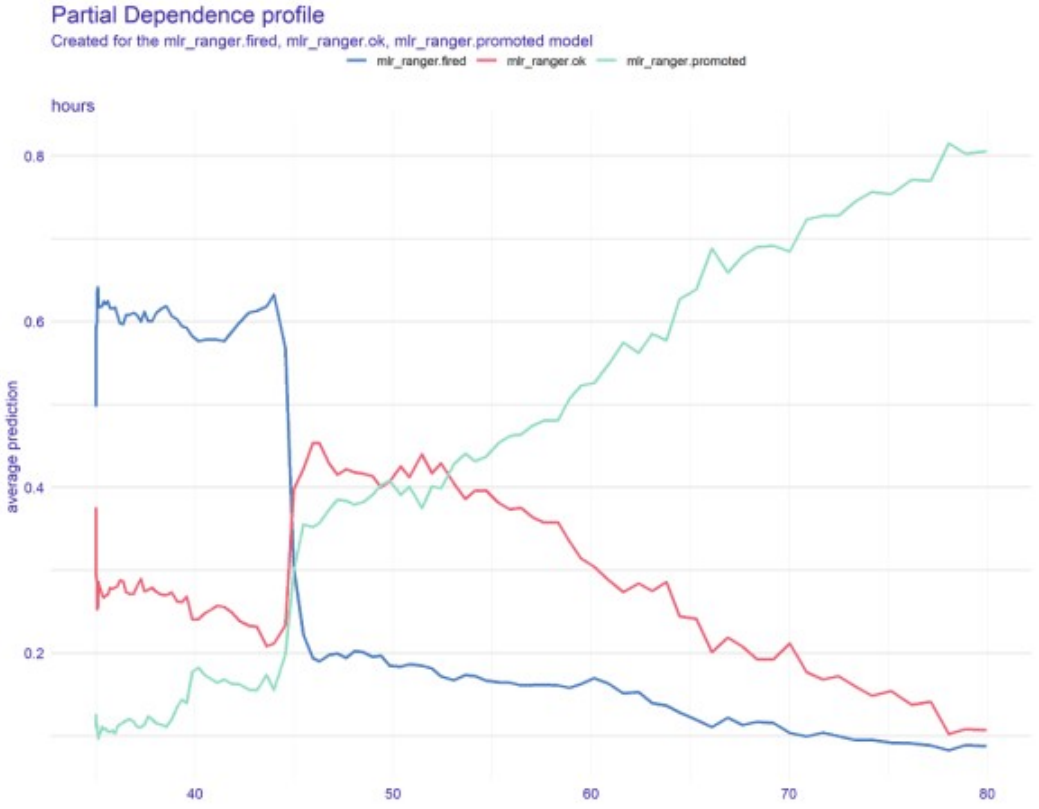
It is dedicated to working with multiclass classification models and will work with any supported models as long as y is provided as a factor vector as was mentioned in the first paragraph. With that knowledge on the board, we can use it to compute the feature importance of our multiclass model.

```
fi_ranger <- model_parts(explainer_ranger,
 loss_function = loss_cross_entropy)
plot(fi_ranger)
```



Surprisingly, the average number of working hours per week is the most important variable, who would've expected that? 😉

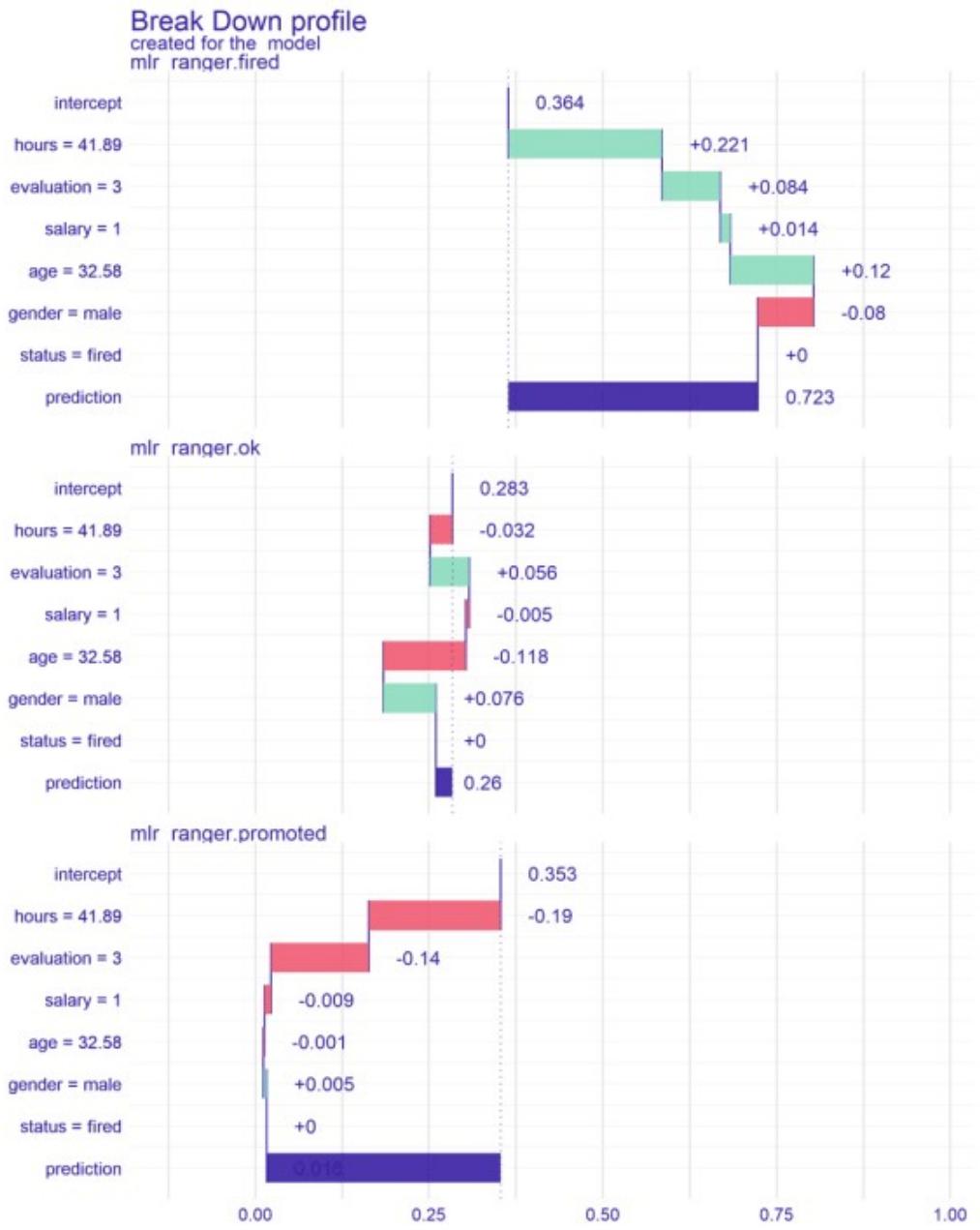
```
adp_ranger <- model_profile(explainer_ranger,
 variables = "hours")
plot(adp_ranger)
```



As we can see, on the average it is more possible to get promoted if we work more and more hours. It is described by the following relation, the more hours we work, the lesser is the probability of belonging to class

fired or *ok*. On average, obviously. It is worth mentioning that DALEX once again beats a multiclass classification problem. You can learn more about accumulated dependency in the ema book.

```
ibd_ranger <- predict_parts(explainer_ranger, HR[1,])
plot(ibd_ranger)
```



iBreakDown plot shows the average contribution specific features had on predicted value for a particular observation. Once again the outcome is split by levels of target variable so every probability output can be properly explained. Thanks to that we know that for example 41.89 hours the employee spent working, made the model improve prediction of being fired by 0.223, lower probability of being ok by 0.034, and of being promoted by 0.189.

xgboost

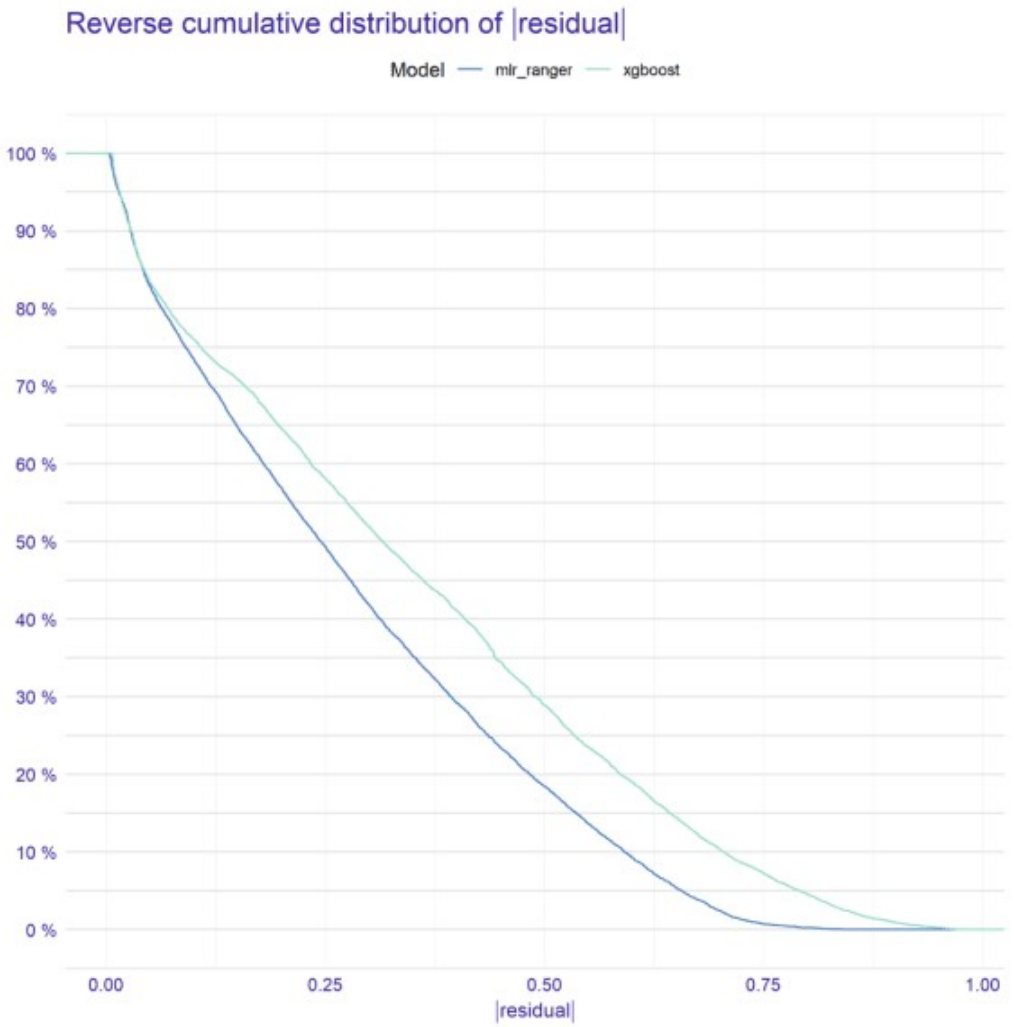
The next feature I'd like to introduce and make it closer to you is related strictly to DALEXtra. I am quite sure You've met with a situation when You wanted to use a specific model straight away, without the involvement of wrappers like *mlr*, but it brought problems. I was in the same spot a few days ago. Had to use xgboost model but the function that creates a model does not handle factors. In fact, it requires a numeric matrix as an X training input. Just create dummy variables—one can say, target encode factor—can say the other. It indeed solves the issue in the matter of creation of a model, but brings a new one, on the explanations field. It is because then some plots may show gender.female, gender.male variables in the case of our dataset or

for instance 100 features named nationality.xyz for different tasks. To solve that issue, explain_xgboost function was made, with an additional parameter – encode_function and true labels. Its purpose is to contain a function that was used to transform the data during the training stage. Thanks to that it can be applied to data before evaluating predictions so the explanations stay consistent. true_labels stores y vector in a form before transformations.

```
library(DALEXtra)
library(xgboost)
library(mlr)
HR_X <- HR[,-6]
HR_y <- HR[,6]

encode_function <- function(X) {
  as.matrix(createDummyFeatures(X))
}

HR_X_enc <- encode_function(HR_X)
HR_y_enc <- as.numeric(HR_y)-1
model <-  xgboost(data = HR_X_enc,
 label = HR_y_enc,
 verbose = FALSE,
 params = list(objective = "multi:softprob",  num_class = 3),
 nrounds = 20)

explainer_xgb <- explain_xgboost(model,
 HR_X,
 as.factor(HR_y),
 encode_function = encode_function,
 true_labels = HR_y,
 verbose = FALSE,
 label = "xgboost")
```

Now with a model ready to work, we cannot omit the opportunity to compare two models, the one we've just created and that from the first paragraph.

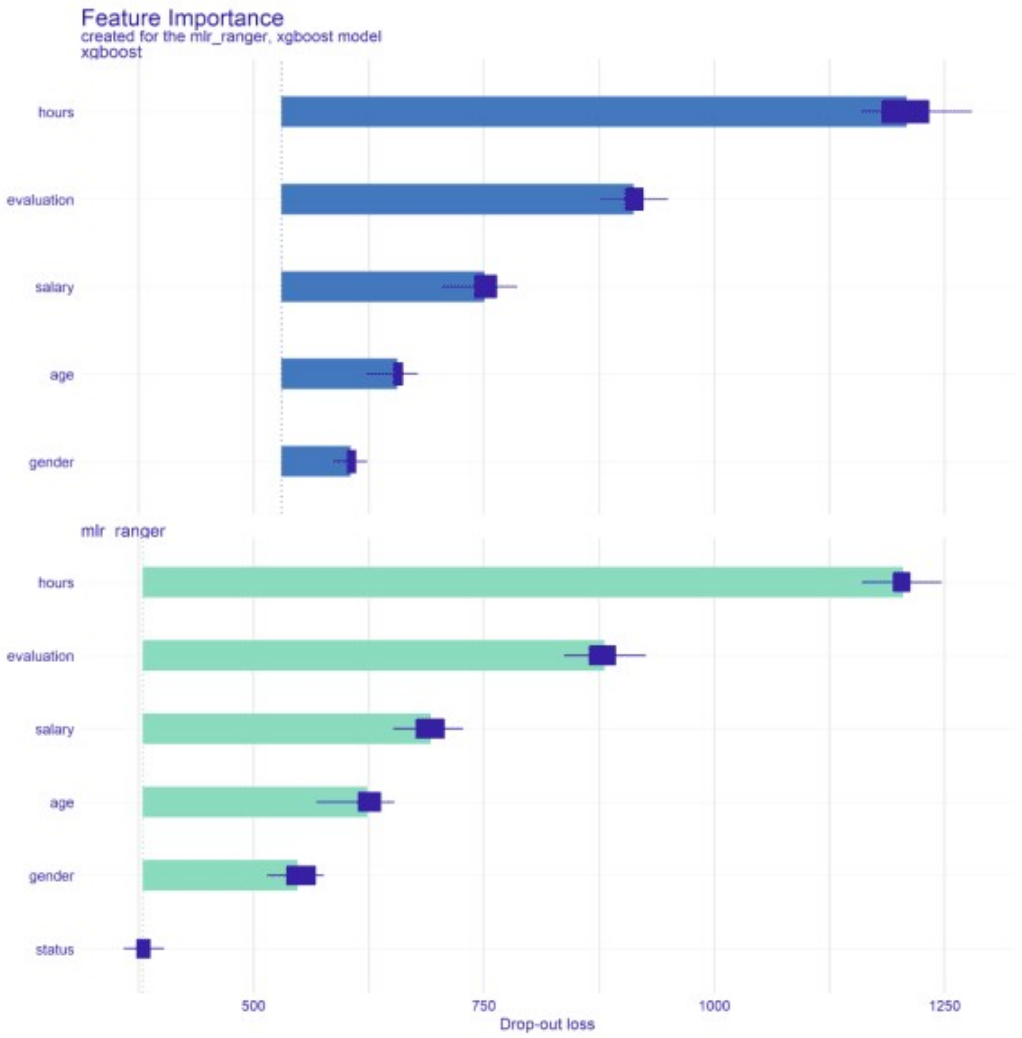
```
mp_xgb <- model_performance(explainer_xgb)
plot(mp_xgb, mp_ranger)
```



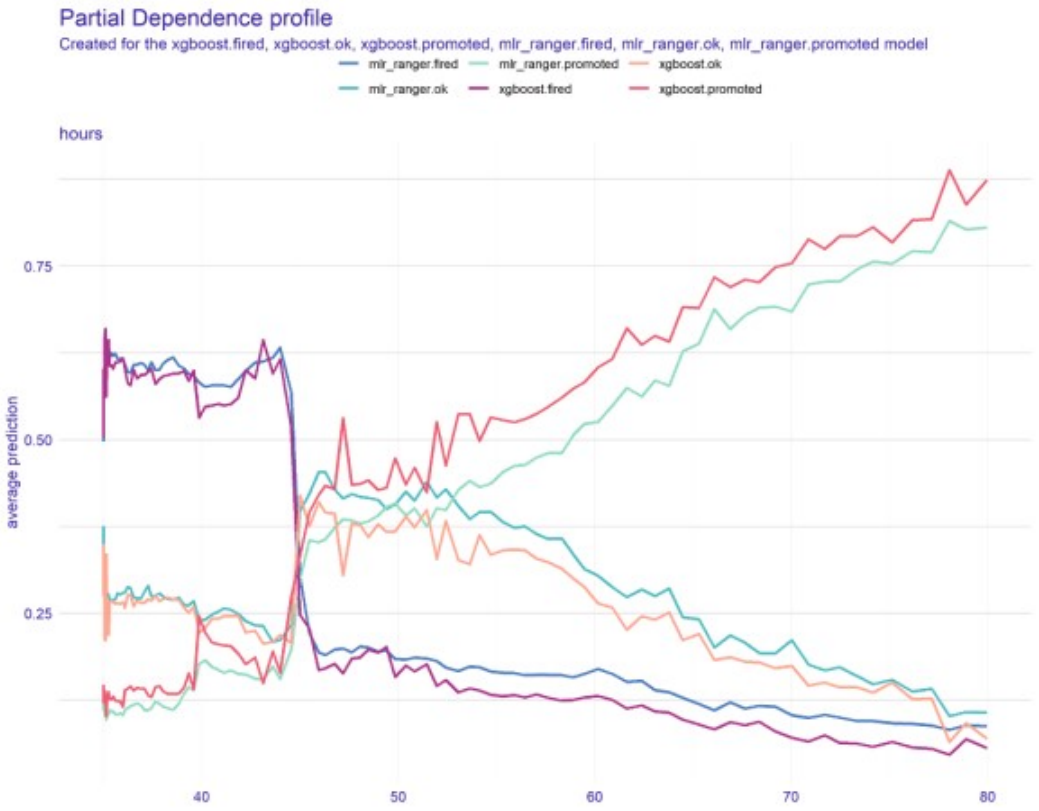
xgboost is slightly worse than a ranger. It has less accuracy by around 10%. Of course, it is related to the usage of default parameters, but overall we are not here for the best scores.

As I mentioned in the first paragraph it is recognized on behalf of DALEX whether the task is regression, binary classification, or multiclass classification. Loss functions are also applied to this logic. You can see below that there is no need to explicitly pass loss_corssentropy function, it will be set as default for multiclass classification models.

```
fi_xgb <- model_parts(explainer_xgb)
plot(fi_xgb, fi_ranger)
```




```
adp_xgb <- model_profile(explainer_xgb,
 variables = "hours")
plot(adp_xgb$agr_profiles,
 adp_ranger$agr_profiles)
```



As You can see the behavior of those models is very similar, therefore we can state that both, xgboost and ranger discovered the same interactions in the data, although ranger did it a little bit better.