Model Transportability and Privacy Protection

31st International Biometric Conference

Daniel Schalk[a,b], Begum Irmak On[b,c], Alexander Hapfelmeier[b,d,e] Ulrich Mansmann[b,c], Verena S. Hoffmann[b,c]

July 12, 2022

[a] Department of Statistics, LMU Munich, Munich, Germany
[b] DIFUTURE (Data Integration for Future Medicine, www.difuture.de), LMU Munich, Munich, Germany
[c] Institute for Medical Information Processing, Biometry and Epidemiology, LMU Munich, Munich, Germany
[d] Institute for AI and Informatics in Medicine, School of Medicine, Technical University of Munich, Munich, Germany
[e] Institute of General Practice and Health Services Research, School of Medicine, Technical University of Munich

Aim of this talk

The goal of this talk is to give you insights about our findings when using predictive/statistical/prognostic models in private settings:

1. Definition of the problem.
2. Transportable models and the practical situation.
3. Guideline to build transportable models.

Practical problems of model building

- Starting point: Development of a treatment decision score for newly diagnosed patients using a clinical data set (ProVal-MS study in DIFUTURE, https://difuture.de/, DRKS: 00014034)
- The model: We choosed a random forest [1] variant called transformation forest [3] implemented in the R [4] function `traforest` of the package `trtf` [2].
- The problem: After developing a potential model we recognized that we cannot publish the model or share it with other researchers because the data set was attached to the R object at multiple locations.

⇒ How to deal with these situations and ensure models to not publish privacy protected information?

Model transportability - theoretically

- A model $\hat{f}$ is the result of a learning algorithm a applied to training data $\mathcal{D}$ to estimate an underlying functional dependency f between a response variable y and input features x:

$$\hat{f} = a(\mathcal{D}), \quad \hat{y} = \hat{f}(x)$$

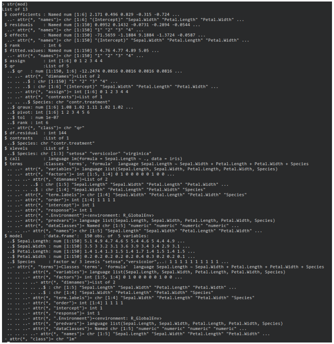
- After the training step, the model is defined by its estimated model parameters $\hat{\theta}$. E.g. the linear model simply equates a linear combination:

$$\hat{f}(x) = x^T\hat{\theta}$$

$\Rightarrow$ After training a model, the training data $\mathcal{D}$ is (in most cases) no longer needed. An exception, for example, is k-NN.

Model transportability - in practise i

- Most R models are storing the data at some place, e.g. creating a linear model with the iris data and looking at the structure (with `str`) reveals much more then just the model parameters:



```
> str(mod)
List of 13
 $ coefficients : Named num [1:6] 2.171 0.496 0.829 -0.315 -0.724 ...
  ..- attr(*, "names")= chr [1:6] "(Intercept)" "Sepal.Width" "Petal.Length" "Petal.Width" ...
 $ residuals    : Named num [1:150] 0.0952 0.1432 -0.0731 -0.2894 -0.0544 ...
  ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
 $ effects      : Named num [1:150] -71.5659 -1.1884 9.1884 -1.3724 -0.0587 ...
  ..- attr(*, "names")= chr [1:150] "(Intercept)" "Sepal.Width" "Petal.Length" "Petal.Width" ...
 $ rank         : int 6
 $ fitted.values: Named num [1:150] 5 4.76 4.77 4.89 5.05 ...
  ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
 $ assign       : int [1:6] 0 1 2 3 4 4
 $ qr           :List of 5
  ..$ qr   : num [1:150, 1:6] -12.2474 0.0816 0.0816 0.0816 0.0816 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:150] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:6] "(Intercept)" "Sepal.Width" "Petal.Length" "Petal.Width" ...
  .. ..- attr(*, "assign")= int [1:6] 0 1 2 3 4 4
  .. ..- attr(*, "contrasts")=List of 1
  .. .. ..$ Species: chr "contr.treatment"
  ..$ qraux: num [1:6] 1.08 1.02 1.11 1.02 1.02 ...
  ..$ pivot: int [1:6] 1 2 3 4 5 6
  ..$ tol  : num 1e-07
  ..$ rank : int 6
  ..- attr(*, "class")= chr "qr"
 $ df.residual  : int 144
 $ contrasts    :List of 1
  ..$ Species: chr "contr.treatment"
 $ xlevels      :List of 1
  ..$ Species: chr [1:3] "setosa" "versicolor" "virginica"
 $ call         : language lm(formula = Sepal.Length ~ ., data = iris)
 $ terms        :Classes 'terms', 'formula'  language Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width + Species
  .. ..- attr(*, "variables")= language list(Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, Species)
  .. ..- attr(*, "factors")= int [1:5, 1:4] 0 1 0 0 0 0 0 1 0 0 ...
  .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. ..$ : chr [1:5] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" ...
  .. .. .. ..$ : chr [1:4] "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
  .. ..- attr(*, "term.labels")= chr [1:4] "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
  .. ..- attr(*, "order")= int [1:4] 1 1 1 1
  .. ..- attr(*, "intercept")= int 1
  .. ..- attr(*, "response")= int 1
  .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. ..- attr(*, "predvars")= language list(Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, Species)
  .. ..- attr(*, "dataClasses")= Named chr [1:5] "numeric" "numeric" "numeric" "numeric" ...
  .. .. ..- attr(*, "names")= chr [1:5] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" ...
 $ model        :'data.frame':  150 obs. of  5 variables:
  ..$ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
  ..$ Sepal.Width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
  ..$ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
  ..$ Petal.Width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
  ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
  ..- attr(*, "terms")=Classes 'terms', 'formula'  language Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width + Species
  .. .. ..- attr(*, "variables")= language list(Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, Species)
  .. .. ..- attr(*, "factors")= int [1:5, 1:4] 0 1 0 0 0 0 0 1 0 0 ...
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:5] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" ...
  .. .. .. .. ..$ : chr [1:4] "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
  .. .. ..- attr(*, "term.labels")= chr [1:4] "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
  .. .. ..- attr(*, "order")= int [1:4] 1 1 1 1
  .. .. ..- attr(*, "intercept")= int 1
  .. .. ..- attr(*, "response")= int 1
  .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. .. ..- attr(*, "predvars")= language list(Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, Species)
  .. .. ..- attr(*, "dataClasses")= Named chr [1:5] "numeric" "numeric" "numeric" "numeric" ...
  .. .. .. ..- attr(*, "names")= chr [1:5] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" ...
 - attr(*, "class")= chr "lm"
```

Figure 1: Structure of an R `lm` object.

Model transportability - in practise ii

- Here, the data is directly stored in the `$model` slot:

```
$ model        :'data.frame':  150 obs. of  5 variables:
 ..$ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 ..$ Sepal.Width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 ..$ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 ..$ Petal.Width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
 ..- attr(*, "terms")=Classes 'terms', 'formula'  language Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width + Species
 .. .. ..- attr(*, "variables")= language list(Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, Species)
 .. .. ..- attr(*, "factors")= int [1:5, 1:4] 0 1 0 0 0 0 0 1 0 0 ...
 .. .. .. ..- attr(*, "dimnames")=List of 2
 .. .. .. .. ..$ : chr [1:5] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" ...
 .. .. .. .. ..$ : chr [1:4] "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
 .. .. ..- attr(*, "term.labels")= chr [1:4] "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
 .. .. ..- attr(*, "order")= int [1:4] 1 1 1 1
 .. .. ..- attr(*, "intercept")= int 1
 .. .. ..- attr(*, "response")= int 1
 .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
 .. .. ..- attr(*, "predvars")= language list(Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, Species)
 .. .. ..- attr(*, "dataClasses")= Named chr [1:5] "numeric" "numeric" "numeric" "numeric" ...
 .. .. .. ..- attr(*, "names")= chr [1:5] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" ...
- attr(*, "class")= chr "lm"
```

Figure 2: Structure of an R `lm` object.

- **But**, parts of the data, the response variable $y = \hat{y} + \hat{\varepsilon}$, can be recreated by accessing the fitted values $\hat{y}$ (`$fitted.values`) and the residuals $\hat{\varepsilon}$ (`$residuals`):

Model transportability - in practise iii

```
> str(mod)
List of 13
 $ coefficients : Named num [1:6] 2.171 0.496 0.829 -0.315 -0.724 ...
  ..- attr(*, "names")= chr [1:6] "(Intercept)" "Sepal.Width" "Petal.Length" "Petal.Width" ...
 $ residuals    : Named num [1:150] 0.0952 0.1432 -0.0731 -0.2894 -0.0544 ...
  ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
 $ effects      : Named num [1:150] -71.5659 -1.1884 9.1884 -1.3724 -0.0587 ...
  ..- attr(*, "names")= chr [1:150] "(Intercept)" "Sepal.Width" "Petal.Length" "Petal.Width" ...
 $ rank         : int 6
 $ fitted.values: Named num [1:150] 5 4.76 4.77 4.89 5.05 ...
  ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
 $ assign       : int [1:6] 0 1 2 3 4 4
 $ qr           :List of 5
```

Figure 3: Structure of an R `lm` object.

⇒ Not just the data, but also everything that enables a recalculation of the data set or parts of it leads to privacy breaches.

How to fix this problem?

Solution 1: Remove crucial parts from the object.

$\Rightarrow$ Sometimes data is hidden at various locations.

Solution 2: Re-implement the algorithm.

$\Rightarrow$ Takes too much time for complex algorithms.

How to fix this problem?

Solution 1: Remove crucial parts from the object.

$\Rightarrow$ Sometimes data is hidden at various locations.

Solution 2: Re-implement the algorithm.

$\Rightarrow$ Takes too much time for complex algorithms.

We will focus on solution 1.

Challenges when removing data

- Nested data structures. Sometimes, a method depends on another method and hence stores data in each of these sub-objects. E.g. the `traforest` implementation uses other `R` objects such as `lm` objects.
- Method to calculate predictions sometimes depends on the data.
- Sometimes transformations of the data are stored. Cholesky decomposition, SVD, or QU decomposition from which original data can be recreated.

$\Rightarrow$ Each method and the resulting object requires an extensive investigation of its structure.

Model transportability - example `traforest`

Making a `traforest` object transportible requires to remove all data:

```
> tf_cmod = trtf::traforest(m, formula = y ~ horTh | age, control = ctrl,
+   ntree = 50, mtry = 1, trace = TRUE, data = GBSG2)
  |======================================================================| 100%
>
> tf_cmod$data = NULL
> tf_cmod$info$model$model = NULL
> tf_cmod$info$call$data = NULL
> tf_cmod$fitted = NULL
> tf_cmod$mltobj$object$data = NULL
```

Figure 4: Training of a `traforest` model and removing all the data from the object.

Model transportability - example `traforest`

Trying to predict with this data-free object throws an error:

```
> tf_cmod = trtf::traforest(m, formula = y ~ horTh | age, control = ctrl,
+   ntree = 50, mtry = 1, trace = TRUE, data = GBSG2)
  |======================================================================| 100%
>
> tf_cmod$data = NULL
> tf_cmod$info$model$model = NULL
> tf_cmod$info$call$data = NULL
> tf_cmod$fitted = NULL
> tf_cmod$mltobj$object$data = NULL
> predict(tf_cmod, newdata = GBSG2)
Error in 1:ncol(nd) : argument of length 0
```

Figure 5: Predicting a `traforest` object after removing all data.

- We have to adjust the predict function to get predictions with a transportable `traforest` object.
- Even worse, for the `traforest` we also have to adjust the train method.

How did we fix that issue?

- Carefully searching for data in an object is mandatory:
    - First overview over the structure of an object using the `str` command.
    - We designed an R package called `rmmodeldata` (`github.com/difuture-lmu/rmmodeldata`) to search through an object to detect numbers. Use this method to search for individual numbers of the original data.
    - Always ask yourself if there are transformations of the original data and try to get detailed information about suspicious objects.
- Remove these parts and, if necessary, re-write parts of the original code to ensure proper functionality.
- Our package `rmmodeldata` contains wrapper around selected methods (`traforest`, `lm`, `ctree`) to get an object without data and still ensure working functionality.

Model transportability - example `traforest`

Using `rmmodeldata` returns a transportable `traforest` object which can be used to calculate predictions:

```
> tf_cmod_wd = rmmodeldata::traforest(m, formula = y ~ horTh | age, control = ctrl,
+   ntree = 50, mtry = 1, trace = TRUE, data = GBSG2)
[2022-06-27 16:30:27] Using `rmmodeldata::traforest`
[2022-06-27 16:30:27] Using `rmmodeldata::cforest`
  |======================================================================| 100%
[2022-06-27 16:30:51] Remove data `$data`
[2022-06-27 16:30:51] Remove data `$info$model$model`
[2022-06-27 16:30:51] Remove data `$info$call$data`
[2022-06-27 16:30:51] Remove data `$fitted`
[2022-06-27 16:30:51] Remove data `$mltobj$object$data`
> tf_cmod$data
NULL
> tf_cmod$info$model$model
NULL
> tf_cmod$info$call$data
NULL
> tf_cmod$fitted
NULL
> tf_cmod$mltobj$object$data
NULL
> pred = predict(tf_cmod_wd, newdata = GBSG2)
[2022-06-27 16:31:01] Using `rmmodeldata::predict.traforest.nodat`
[2022-06-27 16:31:01] Using `rmmodeldata::predict.cforest.nodat`
> str(pred)
 num [1:686, 1:686] 1.61 0 0 0 1.67 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:686] "1" "2" "3" "4" ...
```

Figure 6: Training of a `traforest` model and predicting using `rmmodeldata`.

Advantages and disadvantages of transportability

Advantages

- The model object does not contain confidential data.
- The object size becomes smaller. Especially striking for big data situations.
- We have to deal with the object and are getting a better understanding about the functionality of the method.

Disadvantages

- The data is gone, obviously.
- Reproducibility is more elaborate without data.
- Read and understand other code can quickly become exhausting.

General considerations

- When developing a package/method, think about how to make the results independent of the data. E.g. include an option `store_data = FALSE` to return models without data.
- If possible, and not too time consuming, implement the method by yourself. Not feasible for most complex algorithms such as Boosting, Random Forests, etc.
- Making a model object transportable requires a lot of post processing of the object and advanced knowledge about the programming language and methods.

Thanks for your attention!
Any Questions?

References

[1] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[2] T. Hothorn. *trtf: Transformation Trees and Forests*, 2020. R package version 0.3-7.

[3] T. Hothorn and A. Zeileis. Transformation forests. *arXiv preprint arXiv:1701.02110*, 2017.

[4] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2022.