Empirical Economics with R

2 Introduction to Machine Learning

Uni Ulm

Prof. Dr. Sebastian Kranz

WiSe 20/21

Machine Learning

- Prediction methods have been substantially advanced by interdisciplinary field of *machine learning* (also called *statistical learning*).
- There are many powerful prediction methods (e.g. random forests, gradient boosted trees, lasso and ridge regression, deep neural networks,...) that can often, but not always, substantially outperform linear regressions.
- There are established procedures for prediction problems, like splitting your sample into a *training* and *test* data set and to use *cross validation* for *parameter tuning*.
- A good textbook for starters is "Introduction to Statistical Learning", which you can download for free here: http://faculty.marshall.usc.edu/gareth-james/ISL/

Thesaurus

The machine learning literature sometimes uses different expressions than econometrics. Here is a short overview of expressions that essentially have the same meaning.

- dependent variable = response
- explanatory variable = regressor = predictor = feature
- estimate a model = train a model
- nominal variable = categorical variable = factor variable = non-numeric variable
- dummy variable = one-hot encoded variable

Example: A polynomial regression for prediction

- The main criterion to evaluate a machine learning prediction model is how well the dependent variable y can be predicted for *new* observations, the so called *out-of-sample prediction accuracy*.
- We first illustrate the trade-offs that affect out of sample prediction accuracy with a simulated data set.
- The data is simulated from the following R function:

```
sim.data = function(n) {
  u = rnorm(n, 0, 0.12)
  x = runif(n, 0, 1.2)
  y = -2*x + x^2 + 0.01*x^3 -0.01*x^4 +
      0.1*x^5 -0.01*x^6 + 0.01*x^7 - 0.01*x^8+u
}
```

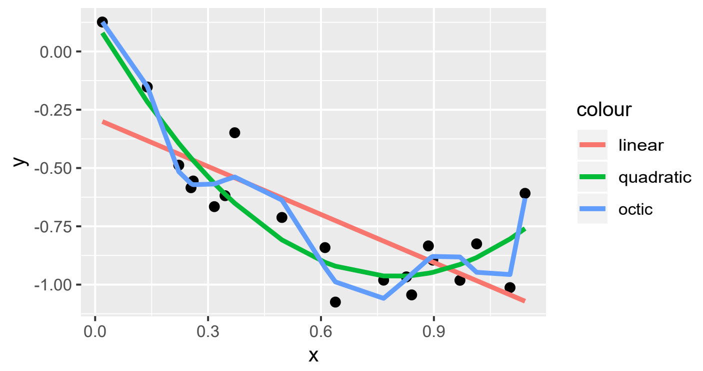
- This means y is a polynom of degree 8 of x (an *octic* function) plus some iid normally distributed error term u.

Example

- We have only a small sample of $n = 20$ observations, that we call *training* data set. We use it to estimate (in machine learning one also says "train") 3 different linear regression models to predict y:
  - linear: $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$
  - quadratic: $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2$
  - octic: $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + ... + \hat{\beta}_8 x^8$
- The following plot shows for the training data set the observations and predicted values $\hat{y}$ for all three estimated models.

Predictions for training data set

The octic curve can predict the training data points most closely. But it looks quite wiggely, which is a typical sign of "overfitting". That means that the form of the curve is strongly influenced by the realization of the random errors u in our training data set.

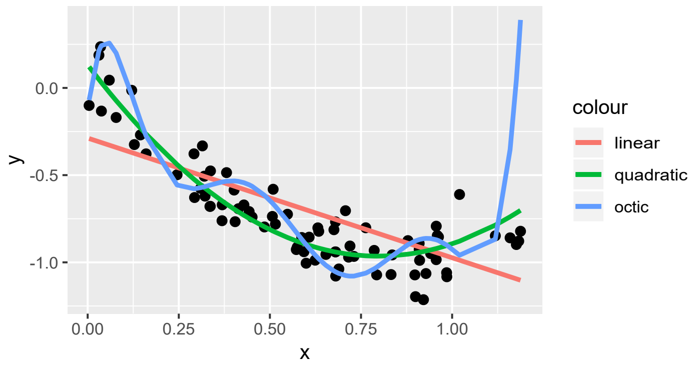


Evaluating out-of-sample prediction accuracy on a test data set

- In machine learning one cares nothing (or only very little) about how well a model can predict y on the training data set that is used to estimate the model.
- To evaluate the out-of-sample prediction accuracy, one checks how well models predict y on a *test* data set that has not been used to estimate the model.
- The 3 curves in the following plot show the predicted values $\hat{y}$ for our 3 models on a test data set with 100 new observations that have not been used to estimate a model:

Predictions for test data set

The most flexible regression (with an octic polynomial) seems to fit the test data in parts very poorly. This is a typical sign *overfitting*. This means the model was so flexible estimated that it captured a lot of random variation in the training data set. This causes worse predictions for out-of sample observations.



Mean Absolute Error and Root Mean Squared Error

- There are different measures to summarize the prediction inaccuracy of a machine learning model.
- The (sample) mean absolute error (MAE) is given by

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$$

- The (sample) root mean squared error (RMSE) is given by

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)^2}$$

- While the MAE weights all prediction errors proportional to their size, the RMSE penalizes large devations more strongly.
- While the MAE is easier to interpret, the RMSE tends to be a more popular criterion and is closer connected to the least squares idea.

RMSE in our example

- In our example of fitting a polynomial function, we find for the three estimated models the following RMSE:

RMSE	linear	quadratic	octic
training data	0.2	0.117	0.077
test data	0.19	0.12	0.268

- If we are only interested in pure prediction quaility, the **best model** is the one with the **lowest RMSE in the test data** (or lowest MAE if you prefer MAE as criterion). Here it is the quadratic model.
- From all three considered models, the quadratic model seems to solve best the trade-off between sufficient flexibility and avoidance of overfitting.
    - Note that even though the true data generating process was actually a polynomial of degree 8, the octic model performs worst. So simpler functional forms than the true one can yield better out-of-sample prediction accuracy.
    - If we would have substantially more observations in our training data, more complex models like the octic model could be estimated more precisely and may then also outperform the quadratic model with respect to out-of-sample prediction accuracy.

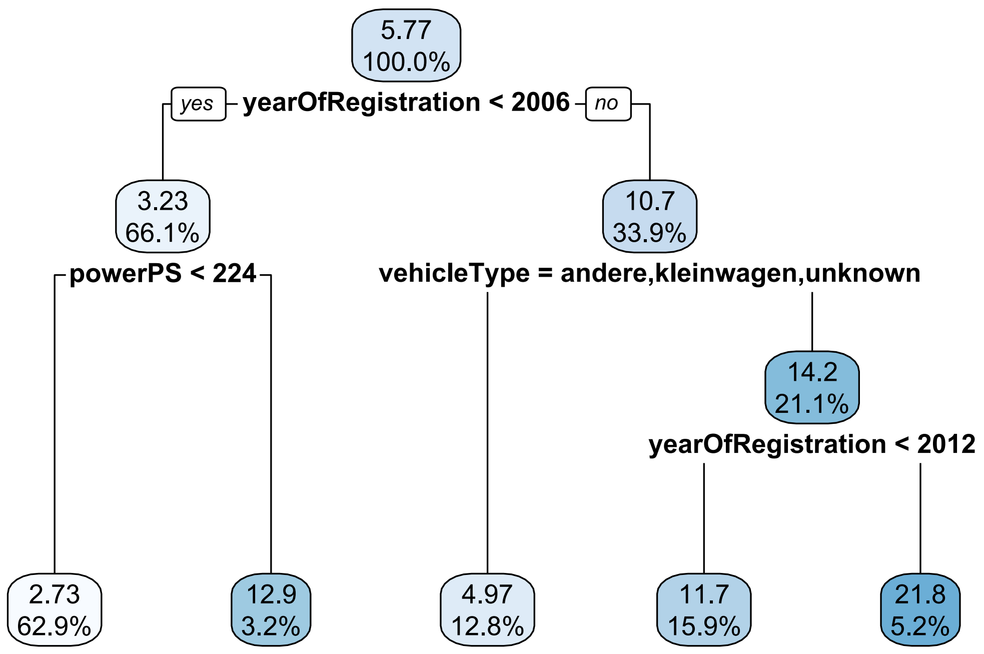
The trade-off between enough flexibility and the risk of overfitting

- Good prediction models balance a trade-off between being flexible enough to capture the relevant systematic relationship while at the same time avoiding overfitting.
- Sometimes this trade-off is called the "Bias-Variance Tradeoff". If you want to know why look at Section 2.2 in the "Introduction to Statistical Learning".
- Many machine learning methods have been constructed with this trade-off in mind.

Tree based methods

- In this section we will discuss regression *trees* and their very successful extension: *random forests*
    - Random forests typically have good prediction accuracy. They are flexible and can well find interaction effects between explanatory variables while beeing constructed in a way that reduces overfitting.
- First look on the graphical representation of a small estimated regression tree on the next slide. It is based on a used cars data set. The model predicts the posted `price` (in 1000 Euro) of a used car.

Regression tree to predict used car price (in 1000 Euro)



Quiz: What is the predicted price of a `Limusine` , registered in `2009` with `150 PS` ?

Using the regression tree for predictions

- The resulting estimated tree above uses for the price prediction 3 variables (even though more variables were given) `yearOfRegistration` , `vehicleType` and `powerPS` .
- Let's predict the price for a car with `yearOfRegistration=2009` , `vehicleType="Limusine"` and `powerPS=150` .
    - We start with the first decision node on the top with the condition `yearOfRegistration < 2006` . If the condition is true for our considered car, we continue on the left, otherwise on the right. Since we consider a car that is registered in 2009, we continue on the right.
    - The next node asks whether the `vehicleType` is one of the values `andere` , `Kleinwagen` , or `unknown` (`unknown` is how I have encoded missing values). Since we have a `Limusine` , we continue again on the right.
    - The final node asks whether `yearOfRegistration < 2012` . This is true for our car. So we continue on the left and end up in a the 4th node from the left in the bottom row. Such a terminal node that is not split anymore is called a *leaf*.
    - The `11.7` (thousand Euro) shown in the leaf is the predicted price for our car. It is simply the average price of all used cars from our training data set which have ended up into this node when following the tree as described above.
    - The `15.9%` means that 15.9% of the cars in the training data set have ended up in this node.
- We also can get the average price and shares of observation for nodes higher up in the tree. E.g. from the left node in the second line we find that 66.1% of cars have been registered before 2006 and those cars have an average price of 3.23 thousand Euros.
- Note that this tree is of course too simplistic to make good predictions, usually one estimates larger trees.

How is a regression tree estimated?

The most common algorithm to estimate a regression tree is called *recursive binary splitting*.

Let us first explain how we make a single binary split:

- Consider a set of n observations that we split into two subsets S_1 and S_2.
- We denote by $\bar{y}_{S_1}$ and $\bar{y}_{S_2}$ the average values of the dependent variable y in each subset.
- The predicted value $\hat{y}_i$ of an observation i shall be either $\bar{y}_{S_1}$ or $\bar{y}_{S_2}$, depending to which subset observation i is assigned to.
- The residual sum of squares of the split observations therefore is

$$RSS(S_1, S_2) = \sum_{i \in S_1} (y_i - \bar{y}_{S_1})^2 + \sum_{i \in S_2} (y_i - \bar{y}_{S_2})^2$$

Computing a single split (continued)

- A split into S_1 and S_2 will be based on an explanatory variable, call it x:
  - If x is numeric (like `yearOfRegistration`) we split by choosing a threshold x^* and putting all observations i with $x_i < x^*$ into S_1 and all other observations into S_2. We will choose that threshold x^* that minimizes the resulting residual sum of squares $RSS(S_1, S_2)$.
  - The case that x is a nominal variable (like `vehicleType`) will be explained on a later slide.
- We go through every explanatory variable and compute the resulting RSS if we would base the split on it. We then base the split on the explanatory variable that yields the lowest RSS.

Computing the whole tree

- To estimate a whole tree we first compute the initial split for all observations in the training data set and thereby get two new child nodes.
- Then in every round, we look at all nodes who have not yet been split and pick the node whose subset has the highest RSS. We split this node again into two child nodes using the procedure above.
- The step above is repeated until some stopping criterion is reached (e.g. as long as the number of nodes is below some specified maximum number of nodes). Stopping criteria involve parameters, which can be determined by usual parameter tuning methods (like cross-validation).

Optional: Optimal splitting based on a nominal variable

- A split based on a factor variable x with M different categories is achieved by finding an optimal subset $\mathcal{M}_1$ of categories such that all observations i with $x_i \in \mathcal{M}_1$ belong to subset S_1 and all others to S_2. The optimal subset of categories $\mathcal{M}_1$ is the one that minimizes the residual sum of squares $RSS(S_1, S_2)$.
- There are $2^{(M-1)}$ ways to split M categories into two non-empty subsets. For example if we have 40 categories there are more than 549 billion ways to split them.
- Luckily, there is a quick way to compute the split that minimizes the RSS. One first computes for each category m the mean of the dependent variable $\bar{y}_m$ of all observations with $x_i = m$. Then one splits by finding a threshold y^* such that all observations where x_i is a category m with $\bar{y}_m < y^*$ belong to S_1 and all others to S_2. (For more details see Section 5 here)

Optional: Replicating a regression tree with a linear regression

- We can in principle replicate the predictions of a regression tree by estimating an equivalent linear regression.
- Consider a regression tree that has K different leaves (terminal nodes) and the corresponding linear regression:

$$y = \beta_1 \delta_1 + \ldots + \beta_K \delta_K + \varepsilon$$

  where δ_k is a dummy variable that corresponds to the `k'th` leaf in our regression tree.
- More concretely $\delta_{k,i}$ is 1 if observation i satisfies all conditions to end up in the k'th leaf of our regression tree. Otherwise $\delta_{k,i}$ is zero.
  - For an example, look again at the previously estimated regression tree of our car data set.
  - The first dummy variable $\delta_{1,i}$ is 1 if and only if `yearOfRegistration_i< 2006` and `powerPS_i < 224`.
  - The second dummy variable $\delta_{2,i}$ is 1 if and only if `yearOfRegistration_i< 2006` and `powerPS_i >= 224`,
  - ...

Optional: Replicating a regression tree with a linear regression

- If we estimate that linear regression

$$y = \beta_1 \delta_1 + \ldots + \beta_K \delta_K + \varepsilon,$$

  the OLS estimate $\hat{\beta}_k$ will be equal to the average value $\bar{y}_k$ of the dependent variable for all observations of the training data that fall into leaf k.
- In practice there is no need to estimate such a regression, after we have estimated the corresponding regression tree.
- Conceptually, however, we see that a regression tree can be interpreted as a method to pick explanatory variables with good predictive power by discretizing the original variables and by finding important interaction effects between those discretized variables.

Discussion and Extensions

- Regression trees are very intuitive and can be nicely illustrated.
- However, a single tree is prone to over-fitting and tends not to make very good predictions.
- Yet, two extensions: *random forests* and *gradient boosted trees* are very successful machine learning methods. Both estimate a large number of trees (called an ensemble) and combine their predictions.
- We will only cover random forests in this course, but there are many good tutorials for *gradient boosting* in the internet. The most popular R package to estimate gradient boosted trees probably is xgboost.

Random Forests

22 / 32

- A (regression) random forest estimates a large number of trees (e.g. 500) and takes the mean of the predicted values of every tree as total prediction.
- Every tree is estimated a bit differently be introducing two random factors (explained further below):

1. Bagging
2. Select a random subset of considered features for each node split.

- The goal of the random factors is to make the overall prediction robust against over-fitting.

23 / 32

Bagging in Random Forests

- Bagging is a technique inspired by bootstrap re-sampling.
- Instead of using the complete training data set to estimate a tree, a random sample with replacement of the same size than the training data set will be drawn to train the tree.
- This means some rows of the training data will not be used to train a particular tree, others occur twice or even more often.
- The non-used-observations are called *out-of-bag* sample, and can be used as a sort of validation data set to assess the prediction accuracy of a single tree.

24 / 32

Random feature subset for splitting nodes

- When splitting a specific node in a tree of a random forest not all explanatory variables (aka features) are considered.
- Instead, a random subset of explanatory variables is drawn (by default one third of all features in a regression tree) and one chooses the explanatory from that subset which creates the lowest residual sum of squares to be the splitting variable.

25 / 32

Discussion of random forests

- Random forests typically yield good out-of-the-box prediction quality even without parameter tuning.
- A drawback compared to linear regression, lasso and trees is that random forests are more like a black-box. It is hard to intuitively reconstruct how 500 estimated trees come up with particular predictions.
    - There are several approaches to make black box models easier interpretable, see e.g. here for an overview, or the R packages modelStudio, Dalex or iml.
- A good implementation of random forests in R is the package ranger. It can be called as comfortably as `lm` via a formula interface. You see an example in the exercise sheet.

Hyperparameter tuning

- The prediction quality of a tree depends on a complexity parameter `cp` that depends on how large the resulting tree shall be.
  - This is a so called *hyperparameter* or tuning parameter.
  - A regular parameter would be a parameter that is estimated, like coefficient estimates $\hat{\beta}$ of a linear regression.
- Random forests have several hyperparameters, like tree complexity, or the share of explanatory variables that is considered for each split.
- Most packages set sensible default values for the hyperparameters.
  - Sometimes prediction accuracy can be substantially improved by hyperparameter tuning, i.e. systematically finding good values for the hyperparameters.
- For some methods hyperparameter tuning is very important (e.g. single trees) for others there often only are modest improvements over the default values (e.g. random forests).

Hyperparameter tuning with training and test data set?

- Theoretically, we could tune our hyperparameters in the following way:

1. Train several models, which differ by their hyperparameter(s), on the *training data*.
2. Compare the prediction accuracy of the models on our *test data* and select the hyperparameter(s) yielding the best prediction accuracy.

- Problem: If we use the test data to select a hyperparameter, the corresponding sample RMSE of the selected model with the best tuning parameter is too optimistic. To get an unbiased estimate of the true out-of-sample RMSE, the test data set must not be used for hyperparameter tuning.

Hyperparameter tuning with a validation data set

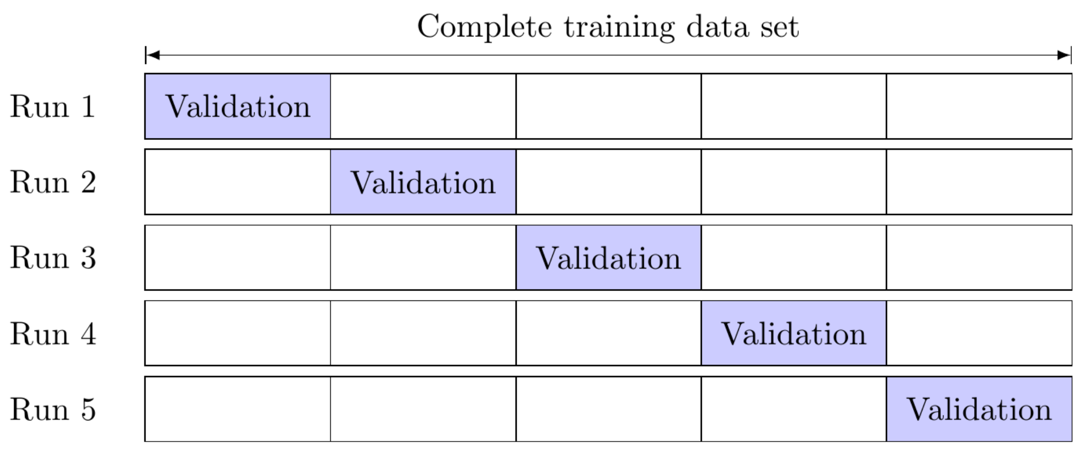
- In practice, a hyperparameter is almost never selected using the test data set.
- Another approach is to split the training data set further into a remaining training data set and a validation data set:

1. Split the training data set further into a *validation data set* and a remaining training data set.
2. Train several models, which differ by their hyperparameter(s), on the remaining training data.
3. Compare the prediction accuracy of the models on the *validation data* and select the hyperparameter(s) yielding the best prediction accuracy.
4. (Optional) Train the model again on the complete training data set using the selected hyperparameter(s).
5. Predict y in our *test data* using only the previously selected model to get an unbiased estimate of the out-of-sample RMSE.

Hyperparameter tuning with k-fold cross validation

- Another very popular approach for hyperparameter tuning is *k-fold cross validation*.
    - Split the complete training data set randomly into k equal sized subsets, which are called *folds*. Common values for k are 5 or 10.
    - Now repeat the steps 1 and 2 from the algorithm from the previous slide k times in the following way:
        - On the 1st run the validation data set is the data of the 1st fold and the observations from all remaining folds are the remaining training data set used to estimate the models for all considered tuning parameter values.
        - On the 2nd run use the 2nd fold as validation data set and so on...
    - We can now aggregate for each model the prediction error for each of the k runs to compute an RMSE and select the model with the lowest RMSE. Afterwards we perform step 4 and 5 as in the algorithm with a single validation set.

Illustration of 5-fold cross validation



Discussion k-fold cross validation

- k-fold cross validation has better performance than using a single validation set, since we use in total more data for validation and training.
- However, parameter tuning also takes k times as long as if we would use a single validation data set.
- In particular if your data set is not too big and speed is no big concern, you should use cross validation instead of a single validation set for parameter tuning.
- There are also other forms of cross-validation like "leave-one-out cross validation" which use a similar idea that k-fold cross validation.

Performing hyperparameter tuning in R

- Some packages have a cross-validation function included.
- There are frameworks in R like caret, tidymodels or mlr3 that have general convenience functions for parameter tuning and many other common machine learning tasks that can be used with a lot of different estimation methods. Getting used to a framework takes some time, but it may pay off if you perform a lot of machine learning with different estimation methods.
- Alternatively, you can always write the cross validation code yourself. It is not too complicated.