Market Analysis with Econometrics and Machine Learning

2a Prediction Problems and Machine Learning - Lasso Regression and Parameter Tuning

Uni Ulm

Prof. Dr. Sebastian Kranz

SoSe 2020

Predicting y vs. estimating β

- So far we used linear regressions in order to understand how some explanatory variable x is related to or causally affects some dependent variable y.
- This means in a linear regression

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_K x_K + \varepsilon$$

  we studied how we can consistently estimate the coefficients β (or at least some particular coefficient β_k of interest)
- In contrast, in a *pure prediction problem*, we only want to find and estimate a model that allows us to make good predictions $\hat{y}$ of the dependent variable y for new observations (from the same data generating process) where we only know the explanatory variables but don't observe y.
- In a *pure prediction problem*, we don't care about things like the interpretation of β, causal effects, endogeneity problems or whether some explanatory variable is a channel variable or not.

Machine Learning

- Techniques for pure prediction problems 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 for pure prediction problems.
- 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. We will mix both languages in this course. Here is a short overview of expressions that essentially have the same meaning.

- dependent variable = response
- explanatory variable = regressor = predictor = feature

  (Note that if we consider interaction terms like in the regression

  $$y = \beta_0 + \beta_1 x + \beta_2 z + \beta_3 x \cdot z + \varepsilon$$

  often only x and z would be called features, not $x \cdot z$. But $x \cdot z$ would typically be also called an explanatory variable.)
- 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

- Importantly, always remember that a prediction model shall make good predictions $\hat{y}$ for *new* observations, so called *out-of-sample data*.
- Let us first illustrate the trade-offs that affect out of sample prediction accuracy with a simulated data set with one explanatory variable.
- The true data generating process is that y is a polynom of degree 8 of x (an *octic* function) plus some iid error term ε:

  $$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_8 x^8 + \varepsilon$$

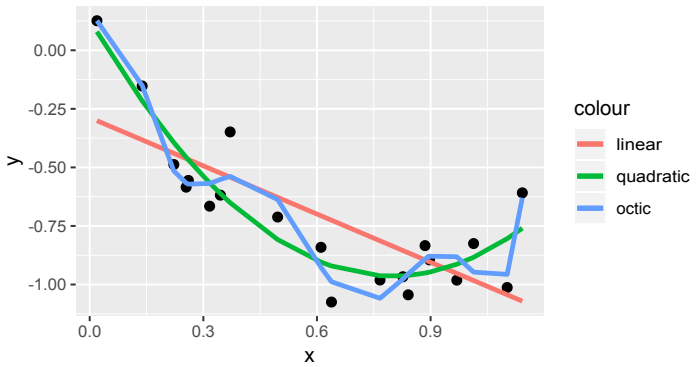
- In our simulation the coefficients β_3 to β_8 have relatively small absolute values.

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: $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \ldots + \hat{\beta}_8 x^8$
- 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 typically sign of "overfitting". That means that the form of the curve is strongly influenced by the realization of the random errors ε in our training data set.

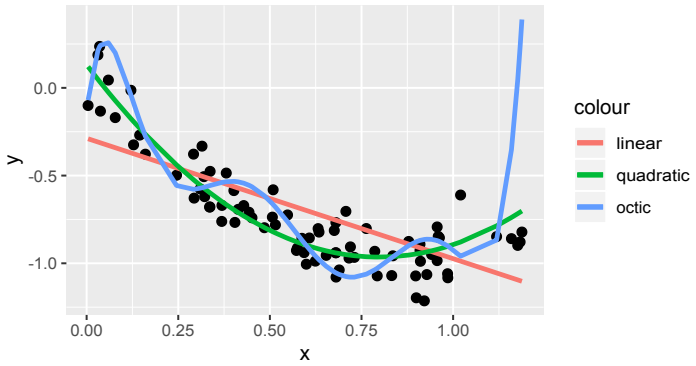


Evaluating out-of-sample prediction accuracy on a test data set

- 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 is captured a lot of random variation in the training data set, which actually creates worse predictions for out-of sample observations.



Mean Squared Prediction Error

- The mean square (out-of-sample) prediction error of a prediction model is given by

$$MSE = E((\hat{y}_i - y_i)^2)$$

  where y_i is a randomly drawn new observation and $\hat{y}_i$ the prediction for it made by the model.
  - This is similar to the MSE of an estimator $\hat{\beta}_k$
  - We call this the population MSE, because it is not computed for a particular sample.
- Note: We will assume throughout our chapter that observations are independently drawn from each other. If new observations are correlated, which is often the case in time-series data, assessing prediction accuracy is more complex.
- The sample equivalent of the MSE for a given data set is

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

  - This is similar to the sum of squared residuals, which an OLS estimator minimizes, except that we divide by the number of observations.

Root Mean Squared Error

- One often reports the square root of the MSE for prediction accuracy, the so called root mean squared error (RMSE). Its population value is:

$$RMSE = \sqrt{E((\hat{y}_i - y_i)^2)}$$

  and the sample equivalent is

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

- Unlike the MSE, RMSE has the same units than y.

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, the **best model** is the one with the **lowest RMSE in the test data**. Here the quadratic model.
- From all three considered models, the quadratric model seems to solve best the trade-off between sufficient flexibility and avoidance of overfitting.
    - Note that even so 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.

Sample RMSE as an estimator of the population RMSE

- In the end we are interested in the population RMSE for a new out-of-sample observation:

$$RMSE = \sqrt{E((\hat{y}_i - y_i)^2)}$$

- The sample MSE from a test data set is typically a consistent estimator of the population MSE. But only if we don't use the test data set for any sort of model selection or parameter tuning.
    - The more observations our test data set has, the more precisely can we estimate the population mean squared prediction error.
    - A very small test data set yields only very noisy estimates of the out-of-sample prediction accuracy.
- The sample RMSE for the training data set is systematically lower (downward bias) than the population out-of-sample prediction error.

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.

Lasso regression

- Lasso regression is a popular machine learning method.
- Consider a multiple linear regression $y = \beta_0 + \beta_1 x_1 + \ldots + \beta_K x_K + \varepsilon$
- The lasso estimator $\hat{\beta}^{lasso}$ solves the following optimization problem:

$$\min_{\hat{\beta}} \sum_{i=1}^{n} \hat{\varepsilon}\,(\hat{\beta})^2 - \lambda \sum_{k=1}^{K} |\hat{\beta}_k|$$

  where $\lambda > 0$ is an exogenously specified *regularization parameter*.
- The first term $\sum_{i=1}^{n} \hat{\varepsilon}\,(\hat{\beta})^2$ is simply the sum of squared residuals, which an OLS estimator minimizes.
- The second term $-\lambda \sum_{k=1}^{K} |\hat{\beta}_k|$ is a so called *regularization term* (or *penalty*) that penalizes too much flexibility.
  - The lower the sum $\sum_{k=1}^{K} |\hat{\beta}_k|$, the less flexibility we have in how the explanatory variables can affect a prediction $\hat{y}_i$.
  - In the extreme case that $\sum_{k=1}^{K} |\hat{\beta}_k| = 0$ predictions $\hat{y}_i$ would not depend on the explanatory variables at all and would be constant.

Standardization of explanatory variables

- In an OLS regression the absolute size of an estimated coefficients $\hat{\beta}_k$ depends on the units of the explanatory variable x_k.
- For example, assume x_k originally measures a distance in meters. If instead we want its unit to be kilometers, we would divide x_k by 1000.
  - The corresponding OLS estimate $\hat{\beta}_k$ for x_k measured in kilometers would then be a 1000 times *larger* than if x_k was measured in meters.
  - But then using the lasso formula above x_k would probably influence y much less if measured in kilometers than in meters.
- Typically one does not want the units of an explanatory variable affect the estmation and resulting prediction quality of a model. Therefore, most lasso implementations automatically *standardize* the explanatory variables before solving the lasso estimation in the following way:
  - Divide each explanatory variable by its standard deviation (and typically also substract the mean).
- Then all explanatory variables have a standard deviation of 1 and are therefore on a similar scale.

More on lasso regression

- One finds that often in a lasso regression many coefficients $\hat{\beta}_k$ are set to exactly 0. The corresponding explanatory variables are then not used at all to predict y.
- In this sense lasso can be seen as a method to *select* explanatory variables (including interaction effects or non-linear terms) used in a regression.
- Also for the selected explanatory variables, the corresponding coefficients $\hat{\beta}_k^{lasso}$ have typically smaller absolute values than if we would just run an OLS regression with the selected variables. One says the coefficients are *shrinked* towards zero.
- Fittingly, lasso is the abbreviation for: *Least absolute shrinkage and selection operator*.
- In the lecture (videos), we will illustrate lasso regression in R using the package `glmnet` for a simulated data set with interaction effects between variables.

Parameter tuning

- The prediction quality of a lasso model depends on the regularization paramater λ.
- Finding a good value of λ is called parameter *tuning*, and we also refer to λ as a tuning parameter.
    - Most machine learning models have one or several tuning parameters.

Parameter tuning with training and test data set?

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

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

- Would the RMSE in our test data set for the best tuning parameter then be an unbiassed estimate of the out-of-sample population RMSE of that lasso model?
    - No. If we use the test data to select a tuning parameter, the corresponding sample RMSE of the lasso model with the best tuning parameter is too optimistic (downward biassed). To get an unbiassed estimate of the out-of-sample RMSE, the test data set must not be used for parameter tuning.

Parameter tuning with a validation data set

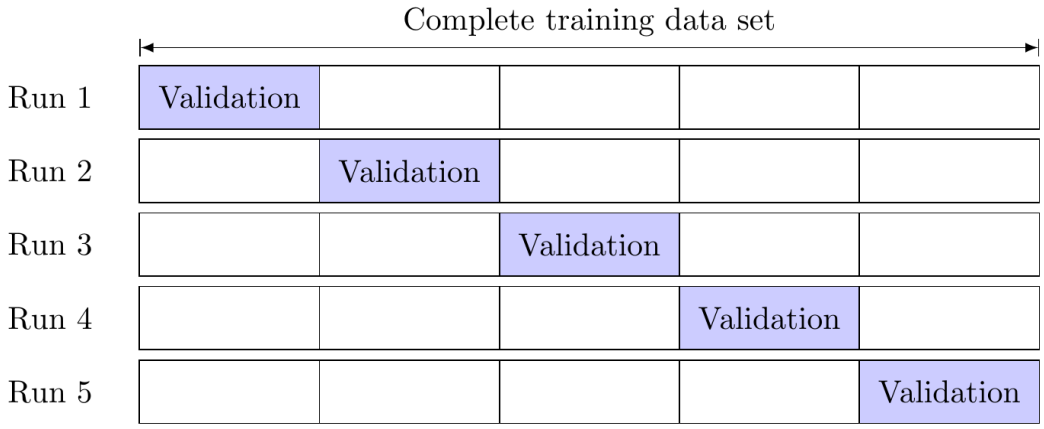
- In practise, a tuning parameter is almost never selected using the test data set.
- Another approach is to split the training data set further into a remaining training data set and a validation data set:

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 tuning parameter(s), on the remaining training data.
3. Compare the prediction accuracy of the models on the *validation data* and select the tuning parameter(s) yielding the best prediction accuracy.
4. (Optional) Train the model again on the complete training data set using the selected tuning parameter(s).
5. Predict y in our *test data* using only the previously selected model to get an unbiassed estimate of the out-of-sample RMSE.

Parameter tuning with k-fold cross validation

- Another very popular approach for parameter 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 parameter tuning in R

- Some packages have a cross-validation function included. E.g. `cv.glmnet` estimates a lasso model by finding the optimal λ via cross-validation.
    - The function `cv.glmnet` uses a special algorithm to quickly solve the lasso model for many values of `lambda` and is therefore likely quicker than if you implement cross-validation yourself or use one of the frameworks below.
- 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.

Ridge Regression

- Ridge regression is a variant of a lasso regression in which the penalty term uses the sum of squared of coefficients (instead of their absolute values).
- The ridge estimator $\hat{\beta}^{ridge}$ solves the following optimization problem:

$$\min_{\hat{\beta}} \sum_{i=1}^{n} \hat{\varepsilon}\,(\hat{\beta})^2 - \lambda \sum_{k=1}^{K} \hat{\beta}_k^2$$

Ridge vs Lasso

- The authors of the `glmnet` package compare lasso and ridge as following (Friedman et. al. 2009):
- **Ridge regression** is known to shrink the coefficients of correlated predictors towards each other, allowing them to borrow strength from each other. In the extreme case of k identical predictors, they each get identical coefficients with 1/kth the size that any single one would get if fit alone.
- **Lasso**, on the other hand, is somewhat indifferent to very correlated predictors, and will tend to pick one and ignore the rest.
- From a Bayesian point of view, the ridge penalty is ideal if one expects there to be many predictors, and all have non-zero coefficients.
- The lasso penalty is better suited if one expects many coefficients to be close to zero, and a small subset to be larger and nonzero.

- Elastic net regression uses a mixture between the ridge and lasso penalty. It solves

$$\min_{\hat{\beta}} \sum_{i=1}^{n} \hat{\varepsilon}\,(\hat{\beta})^2 - \lambda \sum_{k=1}^{K} (1-\alpha)\frac{1}{2}\hat{\beta}_k^2 + (\alpha|\hat{\beta}_k|)$$

  with $0 \leq \alpha \leq 1$.
- For $\alpha = 0$ we have a ridge regression and for $\alpha = 1$ lasso regression.
- Friedman et. al. (2009) state that elastic net regression is particularly useful in a situation in which we have many more possible explanatory variables than observations, or in any situation where there are many correlated explanatory variables.
- One can tune α and λ at the same time via cross validation. This is e.g. implemented by the function cva.glmnet from the package glmnetUtils.