Workshop: Data science with R (ZEW)

Session #7: Supervised Learning

Obryan Poyser
2019-04-18

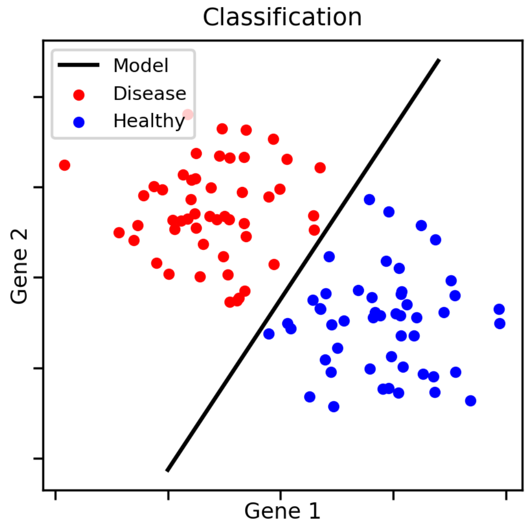
Outline

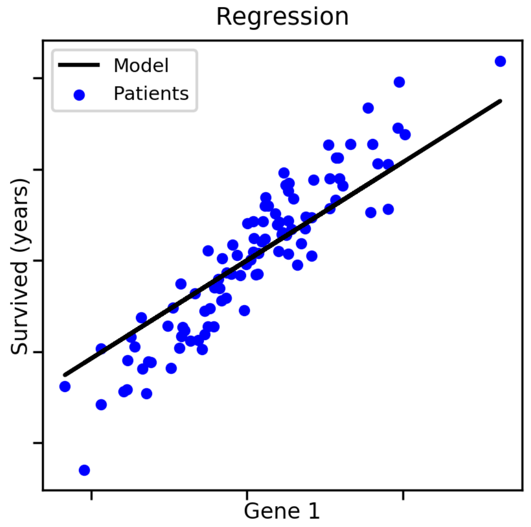
- What is Supervised Learning?
- Machine Learning vs Econometrics
    - Prediction/classification vs inference
- Parametrics vs Non-parametric estimation
- Classification
    - Logistic (softmax)
    - Performance measures
- Prediction
    - Regularization, shrinkage and variable selection
    - Sparsity, Ridge, and LASSO regressions
    - Ensemble learning
    - Performance measures

What is supervised learning?

- Broadly speaking the main two subfields of machine learning are supervised learning and unsupervised learning.
- Supervised statistical involves building statistical models $f()$ for **predicting** or **estimating** and *output* (Y) based on one or more *inputs* given by the design matrix (X).

$$\hat{Y} = \hat{f}(X)$$





Why estimate f?

Prediction & classification

- In classification, the goal is to predict a *class label* within a defined set of elements.
- Prediction is mostly associated with continuous data.
- $\hat{f}$ could be treated as a *"black box"*, that is, we are not concerned on the form of $\hat{f}$, instead, how good this function predicts $\hat{Y}$.
- No matter how accurate $\hat{f}$ is (by choosing a statistical learning technique), there always be non-reducible error term (irreducible error).
- We can improve $\hat{f}$ by choosing a more flexible form. But it has a cost!

Inference

- We want to understand the effect of $X_1, \ldots, X_p$ on Y.
- By definition, it can not be a *"black box"* function, because we need to know the exact form.
- ML statistics differentiate from Econometrics in this element.
    - Econometrics: *"the quantitative analysis of actual economic phenomena based on the concurrent development of theory and observation, related by appropriate methods of inference."* source
    - Machine learning: *"Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed"* Arthur Samuel, 1959.
- Causal inference is getting attention from ML supporters, but there is much work to do.

The use of $\hat{}$ denote estimates.

Econometrics vs Machine Learning: the $\hat{\beta}$ vs $\hat{y}$ dilemma

- Many economic applications revolve around *parameter estimation*
  - Produce good estimates that unveil the true relationship between y and X
- Machine learning algorithms are not designed for inference purposes
  - One has to be aware of the properties and goals of the estimators, the typical parameters' interpretation (i.e. asymptotic theory least square) is no necessary longer valid.
- Applications
  - New data for traditional questions: for instance: measuring the level of economic activity from satellite maps using light-intensity measures.
  - Pre-processing
    - Propensity Score Matching, Linear Instrumental Variables Regression, Heterogeneous treatment effects.
- ML algorithms are technically easy to use in Python or R
  - Threats: *naive interpretations!*

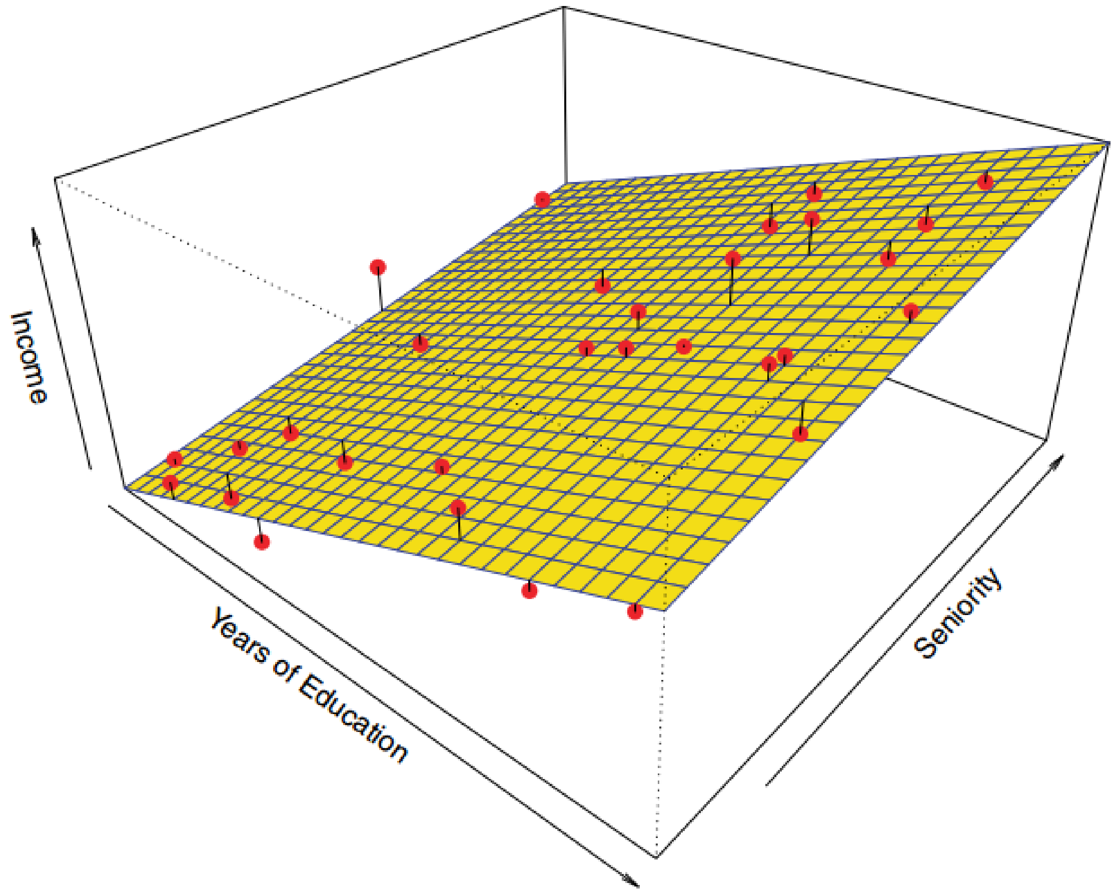
Predicting poverty from satellite maps

Sustainability and artificial intelligence lab (Stanford University)

How to estimate $\hat{f}$?

Parametric methods

- There is a reason for imposing a function form of f: interpretability. In econometrics the conditional expectation function $E(Y|X = x) = f(x)$, we restrict f from a theoretical grounds. Nonetheless, the CEF is by default nonparametric.
- Generalized Linear Model (GLMs) assume the design matrix can be expressed as a linear combination of a set of parameters (typically denoted as β) obtained by OLS. The estimated parameters are linear by construction, while the $X = x_i$ can be of any form or distribution, however, some transformation may be captured easier. Example: $income = \beta_0 + \beta_1 education + \beta_2 seniority$

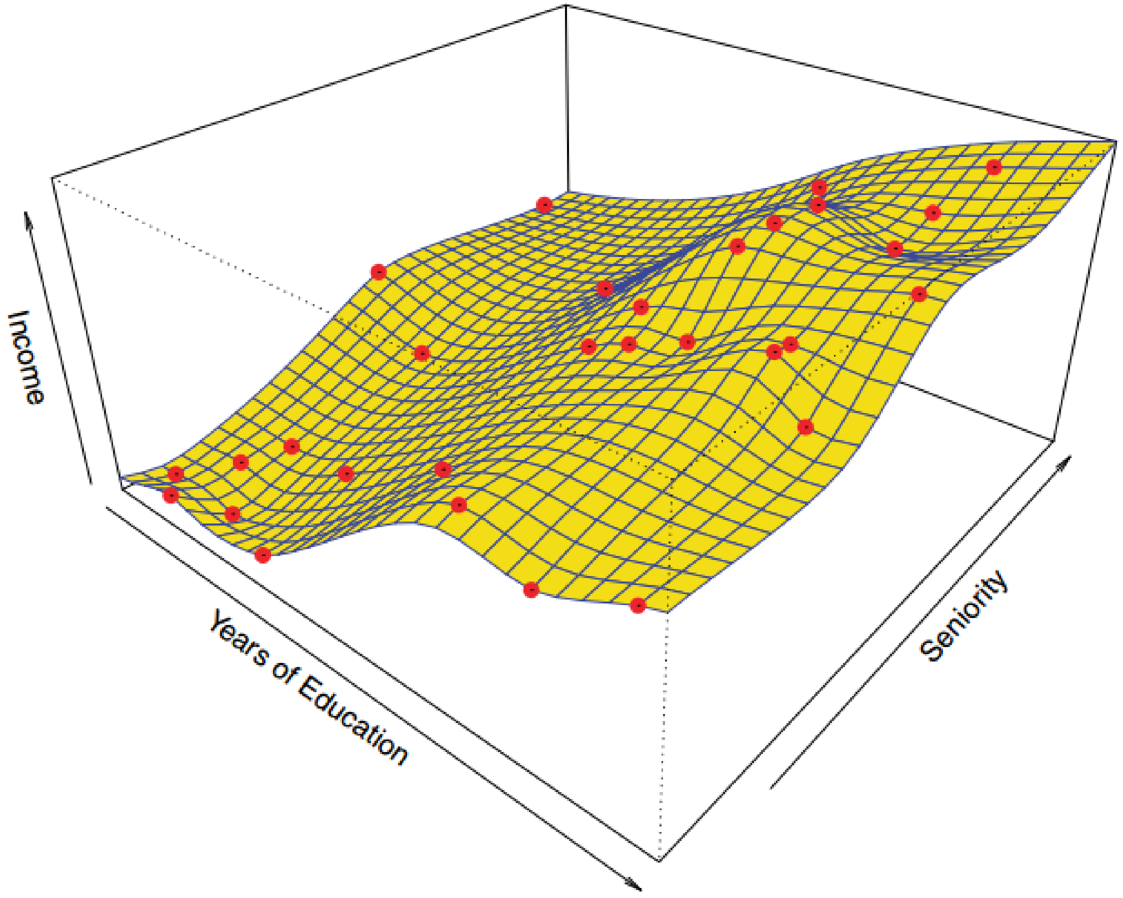


Parametric Linear Model. Hastie et al. 2013.

How to estimate $\hat{f}$?

Non-parametric methods

- Non-parametric methods do not make explicit assumptions about the functional form of f.
- Avoiding the assumption of a particular functional form for f, they have the potential to accurately fit a wider range of possible shapes for f.
- Disadvantage: the amount of parameters increases drastically!
- Examples:
  - Multiple Adaptive Regression Splines: $\hat{f}(X) = \sum_{j=1}^{k} c_i B_i(X_j)$
  - Generalized Additive Models: $\hat{f}(X) = s_0 + \sum_{j=1}^{p} s_j(X_j)$
  - Random Forest
  - Neural Network
  - ...



Non-parametric model. Hastie et al. 2013.

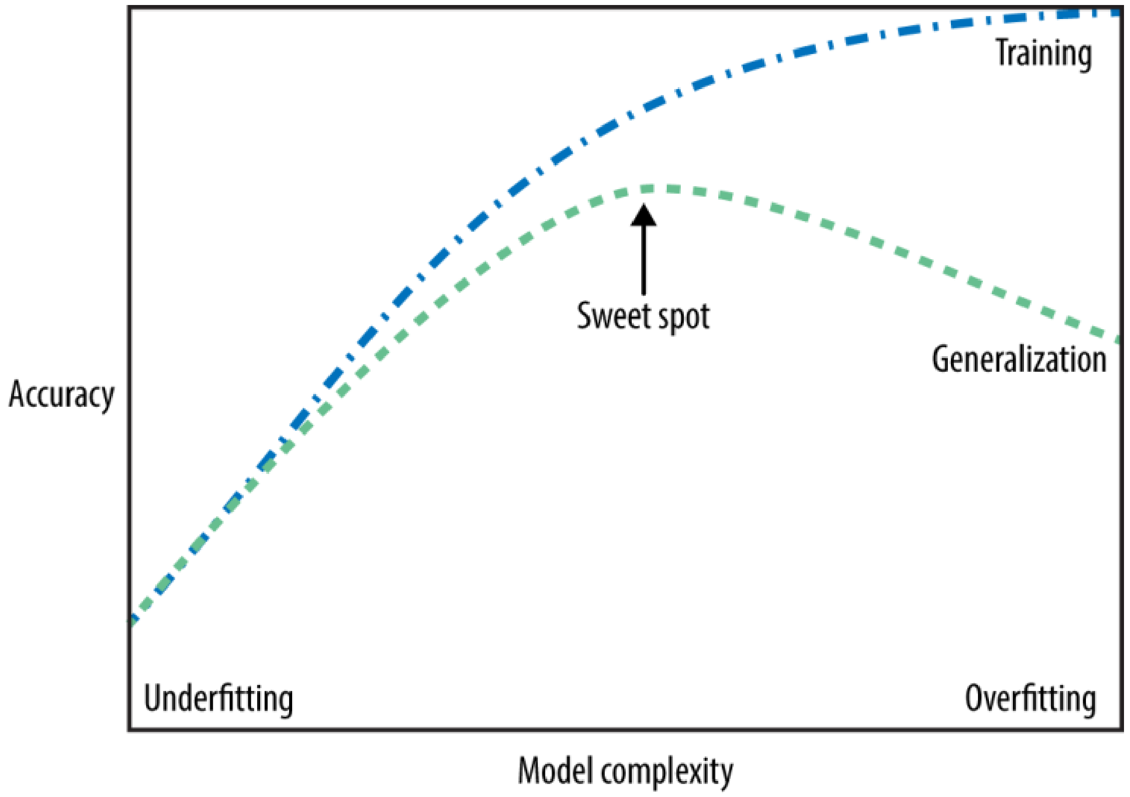
Trade-offs of selecting f

Interpretation wise

- The more complex (flexible) the function form, the best it will fit the design matrix data generating process.
- The simplest a functional form is, the more interpretable are their parameters (GLM for instance), but, its fit is generally worse.

Generalization wise

- Complex models tend to do a good job explaining the data used to estimate f, but it does a very bad job explaining data not used in the estimation process, this is called **overfitting**.
- Complex models often have **hyperparameters**, also name smoothing or complexity parameters, that cannot be estimated from the data. Examples: penalty term, the width of the kernel.
- **Underfitting** happens when the model is too simple,



Müller, Guido (2017)

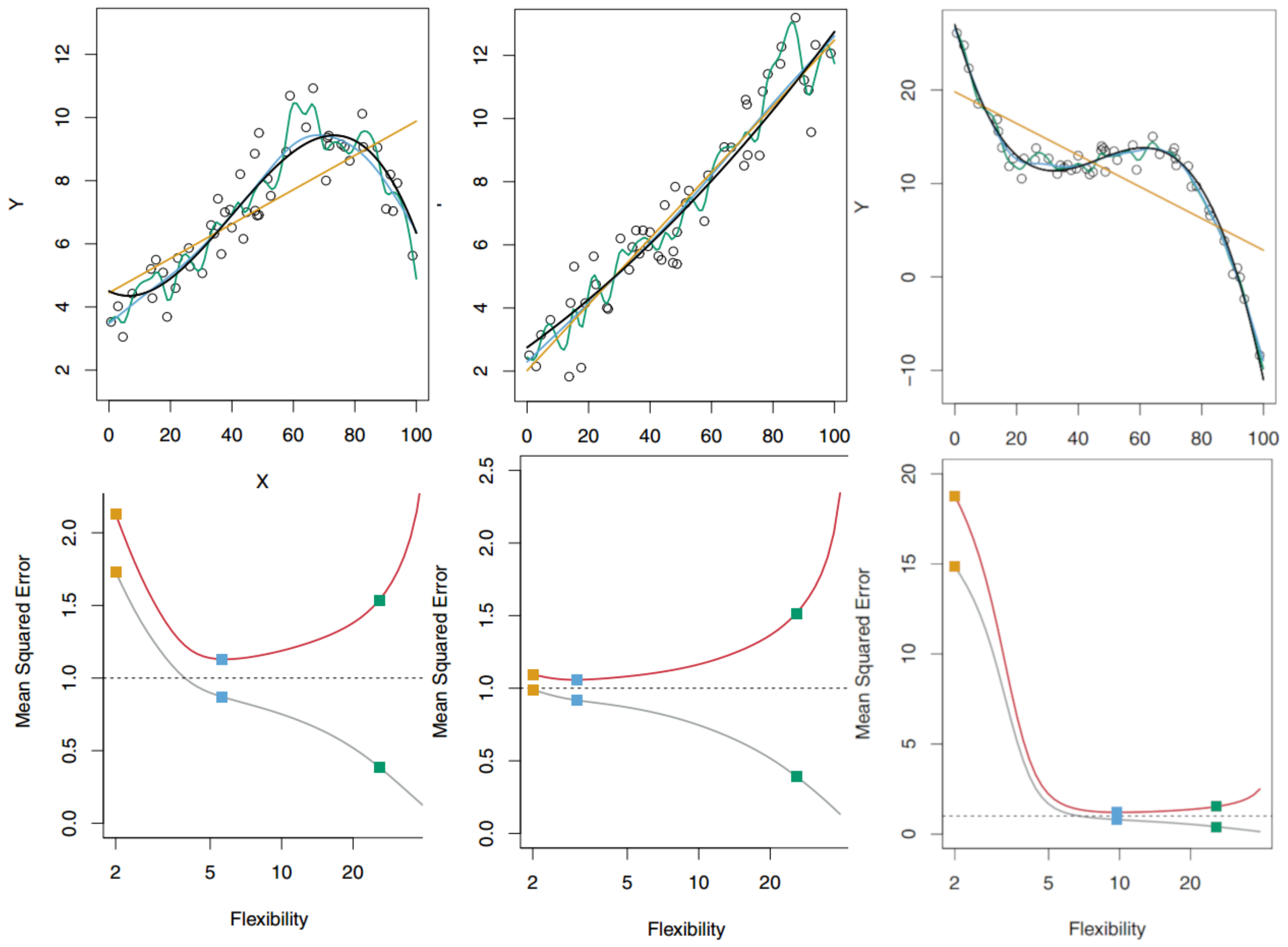
Accuracy

- In order to evaluate the performance of a statistical learning method on a given data set, we need some way to measure how well its predictions actually match the observed data.
- We are interested in the accuracy of the *unseen* data, that is, a subsample of information which has not been used to estimate $\hat{f}$.
    - In practice, one has to divide the total number of observation into **training** and **testing** subsets. The first step is to use the **train** data and observe how good it predicts the **test** data.
    - As mentioned, complex models do a good job finding a function that captures as much variance for training design matrix, but they tend to extrapolate badly on **new information**, which make them useless.
- The fundamental problem of selecting f is the trade-off between **bias** and **variance**.

$$E(Y - \hat{f}(X))^2 = Var(\hat{f}(X)) + [Bias(\hat{f}(X))]^2 + Var(\epsilon)$$

- Variance refers to the amount by which f ˆ would change if we estimated it using a different training data set.
- Bias refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model.

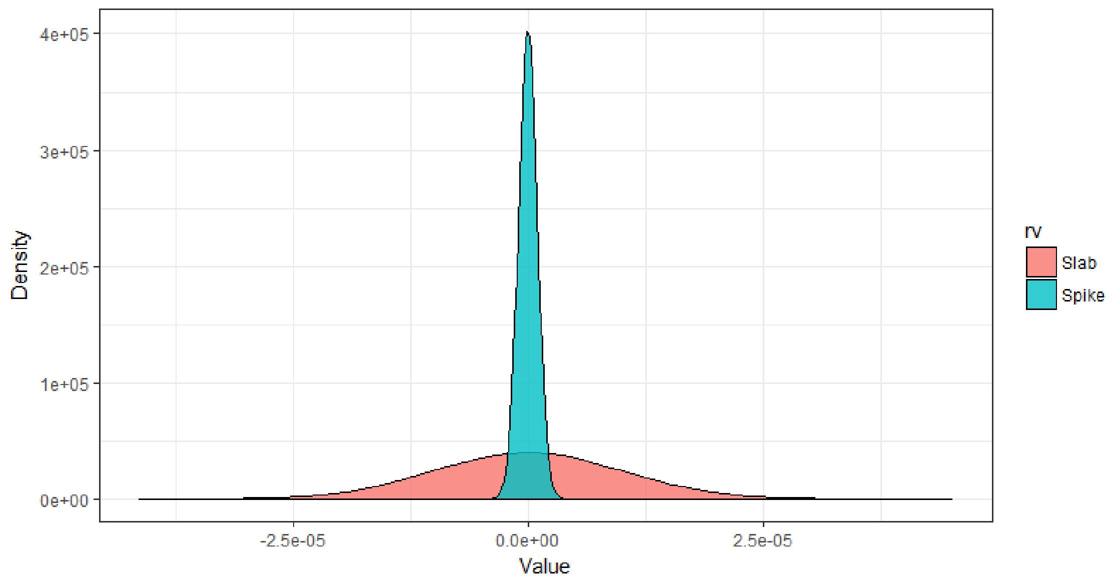
Bias variance trade-off



James et al. 2013

Lasso and Ridge regression

- Least Square estimates present commonly two inconvenient
  - Low prediction accuracy: low bias but large variance.
  - Interpretation: With a lower set of predictors is easier to visualize the *"big picture"*.
  - Subset selection can be applied following *(for)backward-stepwise selection ad-hoc methods*.
- Another alternative is to use shrinkage (regularization) techniques that penalized parameter space, and reduce the sparsity of the design matrix, that is, when $N < X$.
- The goal is to reduce the space of regressors by imposing restrictions. Examples:
  - Spike and slab: In Bayesian models, it is possible to impose a strong prior to shrinkage the parameter space.
  - Principal component regression
  - **Ridge and Lasso**



Spike and Slab priors. (Poyser, 2018)

Ridge regression

- Shrinks the regression coefficients by imposing a penalty on their size
- The ridge coefficients minimize a penalized residual sum of squares

$$\begin{aligned}\hat{\beta}^{ridge} =& \arg\max_{\beta}\left\{\sum_{i=1}^{N}\left(y_i-\beta_0-\sum_{j=1}^{p}x_{ij}\beta_j\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2\right\}\\ =&(X^TX+\lambda I)^{-1}X^Ty\end{aligned}$$

Here $\lambda \geq 0$ is the tuning parameter and controls the complexity of the estimation. The greater λ the greater the shrinkage.

- As expected, scaling affects the shrinkage component, therefore is necessary to standardize the inputs.
  - **Coefficients are no longer scale invariant**
- Ridge uses the so-called L_2 norm

Lasso regression

- Similar to Ridge, but instead of shrinking the coefficients with a squared rule of the parameters, it imposes an absolute value
- The penalty is a L_1 norm $||$
- Solution in y_i is non-linear
- Lasso does a continuous subset selection

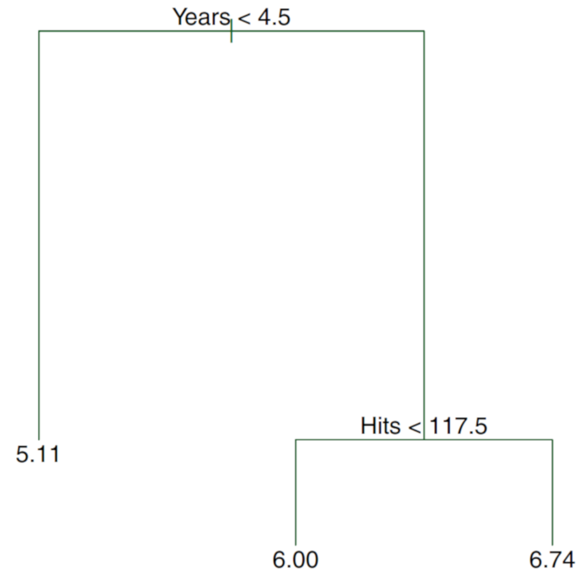
$$\hat{\beta}^{lasso} = \arg\max_{\beta}\left\{\sum_{i=1}^{N}\left(y_i-\beta_0-\sum_{j=1}^{p}x_{ij}\beta_j\right)^2+\lambda\sum_{j=1}^{p}|\beta_j|\right\}$$

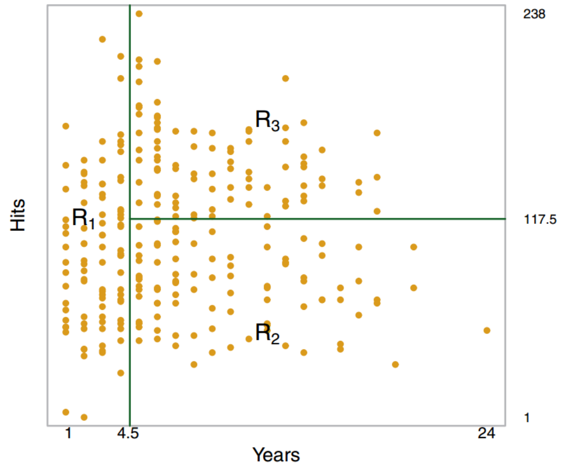
- Lasso λ shrinkage paramets is chosen with the goal of minimizen the expected prediction error.

Decision-tree based models

- Decision trees are widely used models for classification and regression tasks.
- Essentially, they learn a hierarchy of if/else questions, leading to a decision.
- Tree-based methods are simple and useful for interpretation. However, they typically are not competitive with the best-supervised learning approaches
- R_1, R_2, R_3 are known as *leaves* or *terminal nodes* of the tree
- Steps:
    - Dividing the predictor space X_p into $R_1, \ldots, R_J$ non-overlapping regions
    - For every observation that falls into R_J, take the mean of the dependent variable
- The objective is to minimize the Residual Sums of Squares of:

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$





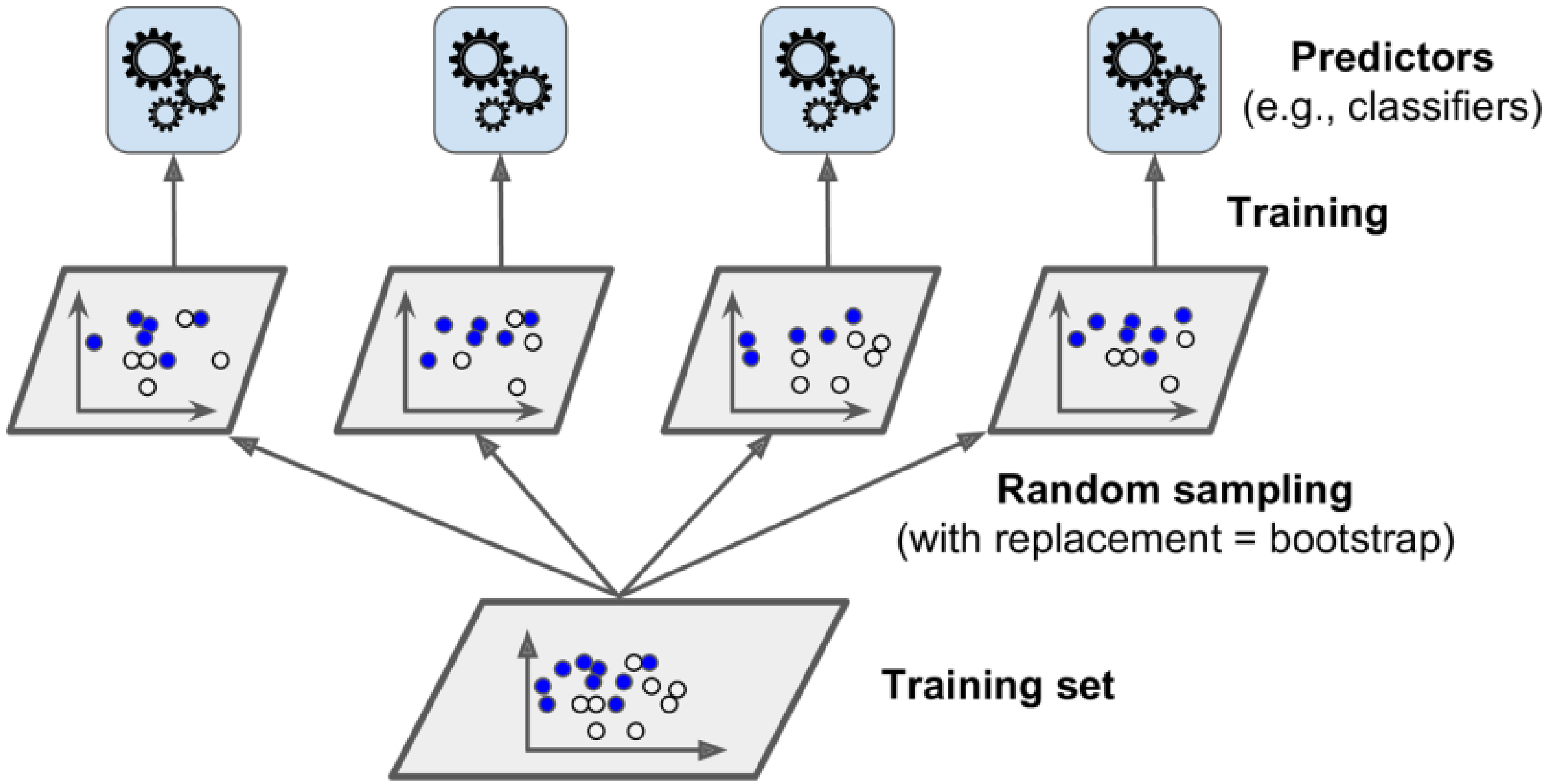
Bagging

- Decision trees have the advantage of interpretability, however, they tend to show high variance
- One way to reduce the variance is to take the average of all the predictions
- Since the actual design matrix does not change, it is not possible to take the average of inexistent samples. To solve this problem we can use **bootstrapping**
    - What bootstrap does is taking B random samples with replacement from a single sample, creating several artificial samples?
- Bagging uses a bootstrap rule to create pseudo-training sets, then average the prediction in order to reduce the variance.

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x)$$

- Even though bagging improves accuracy, it reduces interpretability. Nevertheless, there is one way to assess the importance of one variable: variable importance
- The process works as follows:
    - Take one variable at a time, then calculate how much the RSS decreases in each split

Bagging



Bagging process. Géron (2017)

Random-forest

- Works similar to bagging, but with the difference of taking a random sample of predictors over the design matrix X_p
- Process:
  - Draw a bootstrap sample Z^* of size N from the training data.
  - Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_min is reached.
  - Select m variables at random from the p variables.
  - Pick the best variable/split-point among the m.
  - Split the node into two daughter nodes.

Considerations

- There are several algorithms and tweaks to improve prediction accuracy, however, each of them deserves a whole session to be studied.
  - Boosting
  - Ensemble
  - Boosting
  - SVM
  - Neural Networks
  - …
- For a more detailed explanation of such techniques see the references section.

Packages

- `broom`: is an attempt to bridge the gap from untidy outputs of predictions and estimations to the tidy data we want to work with.

    - `tidy`: constructs a data frame that summarizes the model's statistical findings. This includes coefficients and p-values for each term in a regression, per-cluster information in clustering applications, or pre-test information for multtest functions.
    - `augment`: add columns to the original data that was modeled. This includes predictions, residuals, and cluster assignments.
    - `glance`: construct a concise one-row summary of the model. This typically contains values such as R^2, adjusted R^2, and residual standard error that are computed once for the entire model.

- `parsnip`

    - It is designed to solve a specific problem related to model fitting in R, **the interface**.
    - Many functions have different interfaces and arguments names and parsnip standardizes the interface for fitting models as well as the return values

- `tidymodels`: Collection of modeling packages, that aim to create a common structure (similar to sklearn in Python)

- `rsample`: Classes and functions to create and summarize different types of resampling objects (e.g. bootstrap, cross-validation).

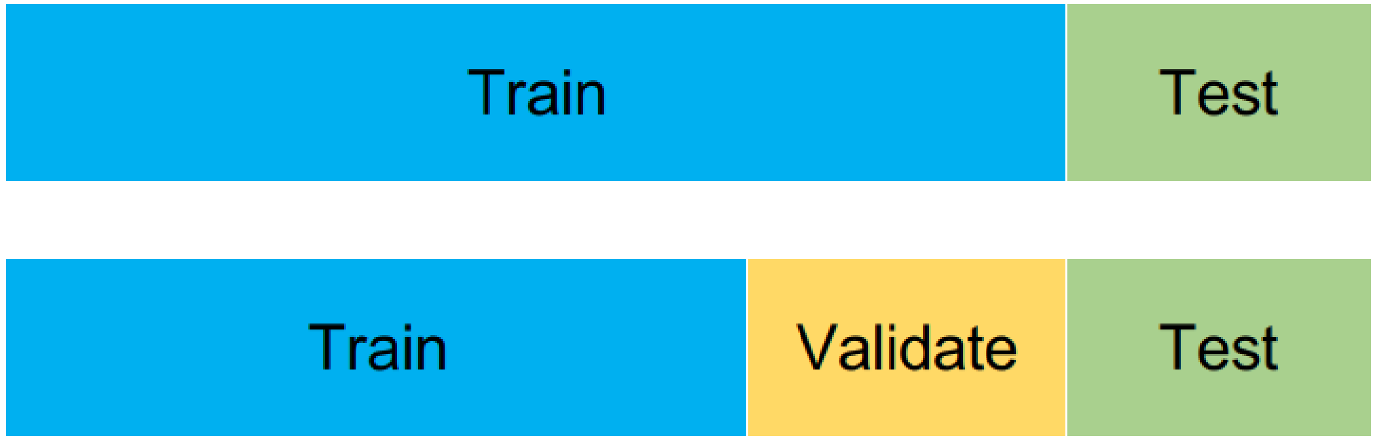
Dataset

Wine quality

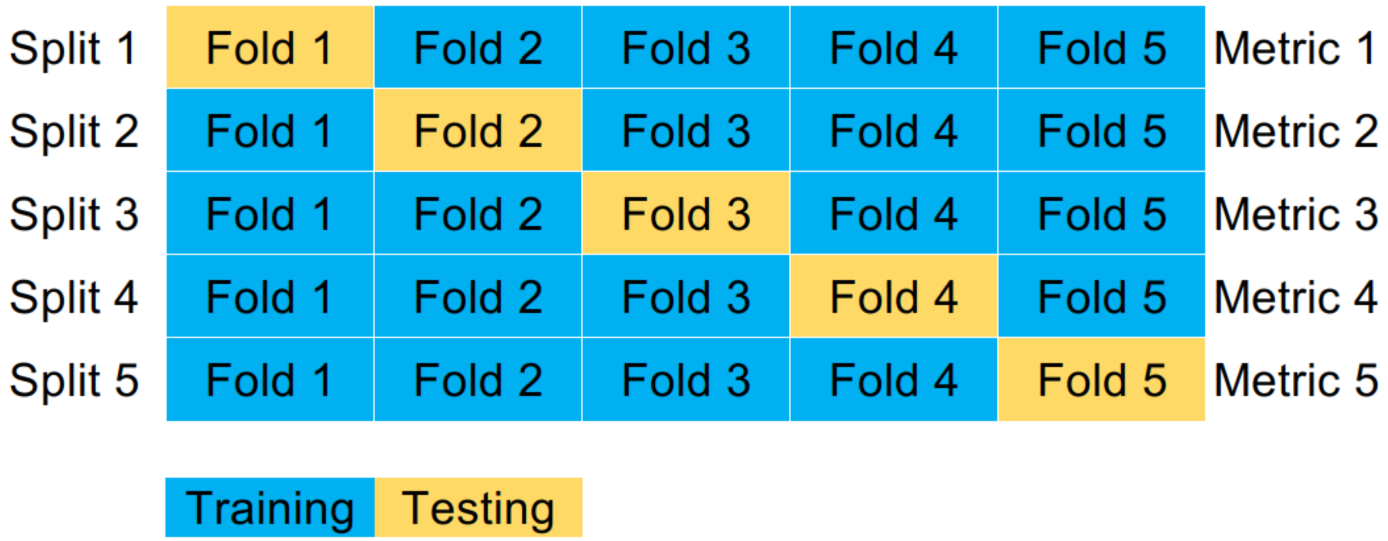
```r
wine_w <- readr::read_rds("datasets/session_7/wine_white.rds")
skimr::skim(wine_w) %>% skimr::kable()
```

```
## Skim summary statistics
##  n obs: 4898
##  n variables: 13
##
## Variable type: numeric
##
##        variable         missing   complete    n      mean      sd      p0      p25      p50     p75    p100      hist
## ---------------------- --------- ---------- ------ -------- ------- ------- ------- ------- ------ ------ ----------
##        alcohol             0        4898     4898   10.51    1.23      8      9.5     10.4    11.4   14.2   ▁▇▆▆▃▃▂
##       chlorides            0        4898     4898   0.046   0.022   0.009   0.036   0.043   0.05   0.35   ▇▂▁▁▁▁▁
##      citric_acid           0        4898     4898    0.33    0.12      0     0.27    0.32    0.39   1.66   ▁▇▂▁▁▁▁
##        density             0        4898     4898    0.99   0.003   0.99    0.99    0.99     1     1.04   ▇▇▁▁▁▁▁
##     fixed_acidity          0        4898     4898    6.85    0.84    3.8     6.3     6.8    7.3    14.2   ▁▃▇▂▁▁▁
##   free_sulfur_dioxide      0        4898     4898   35.31   17.01     2      23      34      46     289    ▇▅▁▁▁▁▁
##          p_h               0        4898     4898    3.19    0.15   2.72    3.09    3.18    3.28   3.82   ▁▂▆▇▃▂▁
##        quality             0        4898     4898    5.88    0.89     3       5       6      6      9     ▁▁▅▇▁▃▁
##       quality_b            0        4898     4898    0.67    0.47     0       0       1      1      1     ▅▁▁▁▁▁▁
##     residual_sugar         0        4898     4898    6.39    5.07    0.6     1.7     5.2    9.9    65.8   ▇▃▁▁▁▁▁
##       sulphates            0        4898     4898    0.49    0.11   0.22    0.41    0.47    0.55   1.08   ▁▆▇▃▁▁▁
##  total_sulfur_dioxide      0        4898     4898   138.36   42.5     9      108     134     167    440    ▁▆▇▃▁▁▁
##    volatile_acidity        0        4898     4898    0.28     0.1   0.08    0.21    0.26    0.32   1.1    ▃▇▂▁▁▁▁
```

Training, testing



Cross-validation



Split 1	🟨 Fold 1	🟦 Fold 2	🟦 Fold 3	🟦 Fold 4	🟦 Fold 5	Metric 1
Split 2	🟦 Fold 1	🟨 Fold 2	🟦 Fold 3	🟦 Fold 4	🟦 Fold 5	Metric 2
Split 3	🟦 Fold 1	🟦 Fold 2	🟨 Fold 3	🟦 Fold 4	🟦 Fold 5	Metric 3
Split 4	🟦 Fold 1	🟦 Fold 2	🟦 Fold 3	🟨 Fold 4	🟦 Fold 5	Metric 4
Split 5	🟦 Fold 1	🟦 Fold 2	🟦 Fold 3	🟦 Fold 4	🟨 Fold 5	Metric 5

🟦 Training 🟨 Testing

Training, testing, and cross-validation

```r
data_split <- initial_split(wine_w, prop = .7)
(train_data <- training(data_split)) %>% dim()
```

```
## [1] 3429   13
```

```r
(test_data <- testing(data_split)) %>% dim()
```

```
## [1] 1469   13
```

Cross-validation

```
(cv_data <- vfold_cv(train_data, v = 10) %>%
    mutate(train=map(splits, ~training(.x))
         , validate=map(splits, ~testing(.x))
         , lm=map(train, ~lm(quality~.-quality_b, data = .x))
         , rf=map(train, ~rand_forest(mtry = 5, trees = 200) %>%
                        set_engine("ranger", importance="impurity") %>%
                        fit(quality~.-quality_b, data=.x)
                    )
    )
)
```

```
## #  10-fold cross-validation
## # A tibble: 10 x 6
##    splits           id     train                 validate         lm       rf
##  * <list>           <chr>  <list>                <list>           <list>   <list>
##  1 <split [3.1K/3… Fold01 <tibble [3,086 ×… <tibble [343 ×… <S3: l… <fit[+…
##  2 <split [3.1K/3… Fold02 <tibble [3,086 ×… <tibble [343 ×… <S3: l… <fit[+…
##  3 <split [3.1K/3… Fold03 <tibble [3,086 ×… <tibble [343 ×… <S3: l… <fit[+…
##  4 <split [3.1K/3… Fold04 <tibble [3,086 ×… <tibble [343 ×… <S3: l… <fit[+…
##  5 <split [3.1K/3… Fold05 <tibble [3,086 ×… <tibble [343 ×… <S3: l… <fit[+…
##  6 <split [3.1K/3… Fold06 <tibble [3,086 ×… <tibble [343 ×… <S3: l… <fit[+…
##  7 <split [3.1K/3… Fold07 <tibble [3,086 ×… <tibble [343 ×… <S3: l… <fit[+…
##  8 <split [3.1K/3… Fold08 <tibble [3,086 ×… <tibble [343 ×… <S3: l… <fit[+…
##  9 <split [3.1K/3… Fold09 <tibble [3,086 ×… <tibble [343 ×… <S3: l… <fit[+…
## 10 <split [3.1K/3… Fold10 <tibble [3,087 ×… <tibble [342 ×… <S3: l… <fit[+…
```

Performance

Linear model

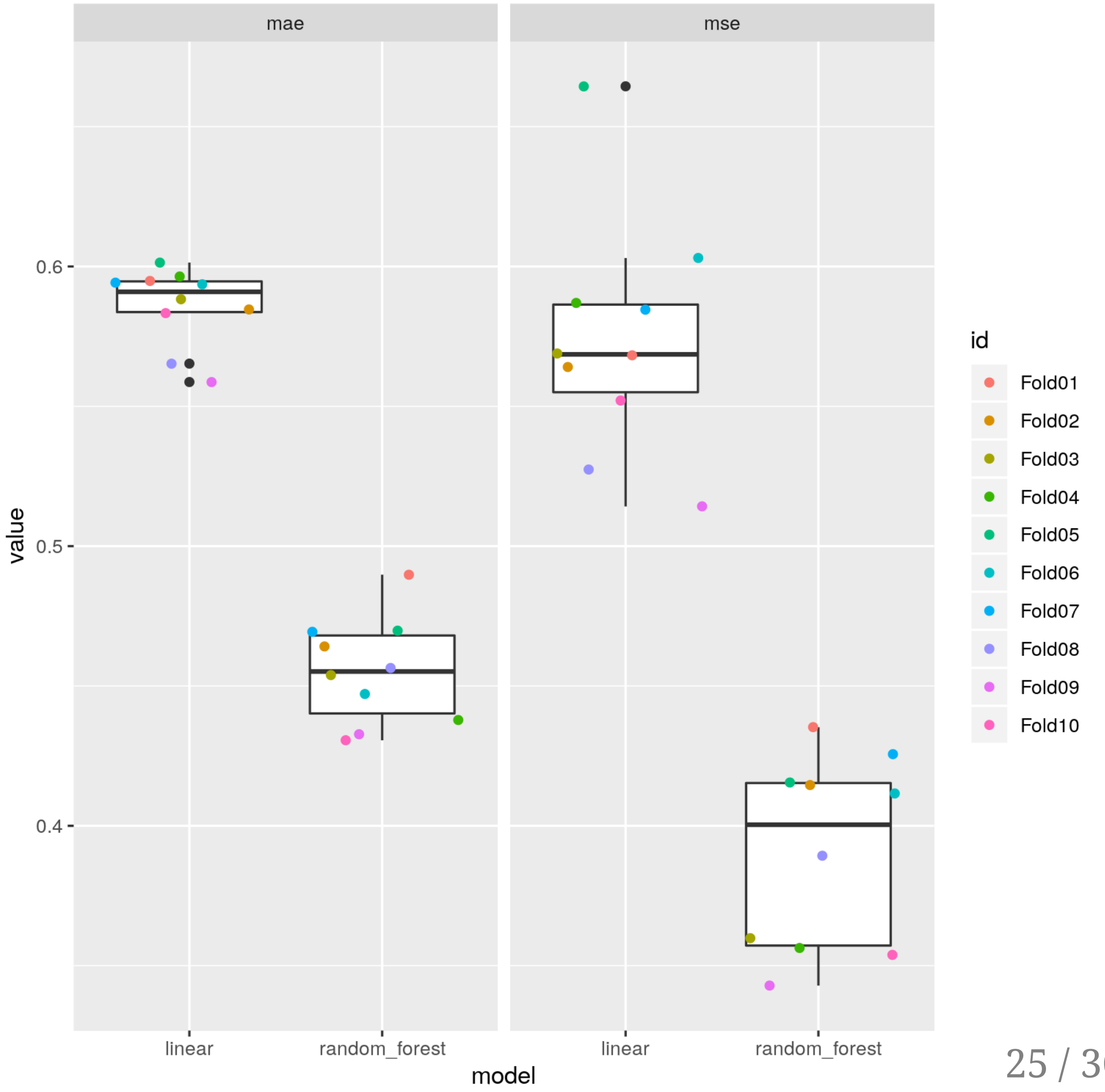
```r
lm_model <- cv_data %>%
    mutate(pred=map2(.x=lm, .y=validate, ~predict(.x, .y) %>%
                         as_tibble %>%
                         set_names("pred"))
           , mae=map2_dbl(.x=pred, .y=validate
                          , ~measures::MAE(truth = .y$quality
                                           , response = .x$pre
                          )
           , mse=map2_dbl(.x=pred, .y=validate
                          , ~measures::MSE(truth = .y$quality
                                           , response = .x$pre
           ))
```

Random forest

```r
rf_model <- cv_data %>%
    mutate(pred=map2(.x=rf, .y=validate, ~predict(.x, .y) %>%
                         as_tibble %>%
                         set_names("pred"))
           , mae=map2_dbl(.x=pred, .y=validate
                          , ~measures::MAE(truth = .y$quality
                                           , response = .x$pre
                          )
           , mse=map2_dbl(.x=pred, .y=validate
                          , ~measures::MSE(truth = .y$quality
                                           , response = .x$pre
           ))
```

Model performance

```r
lm_model %>%
    dplyr::select(id, mae, mse) %>%
    bind_rows(rf_model %>%
                dplyr::select(id, mae, mse)
              , .id = "model") %>%
    mutate(model=case_when(
        model==1~"linear"
      , T~"random_forest"
    )) %>%
    pivot_longer(cols = c(mae, mse)
                 , names_to = "measure") %>%
    ggplot(aes(model, value))+
    geom_boxplot() +
    geom_jitter(aes(col=id))+
    facet_grid(~measure, scales = "free")
```



Hyperparameter tuning: grid search

```r
(rf_tun <- vfold_cv(train_data, v = 5) %>%
    crossing(mtry=1:5) %>%
    mutate(train=map(splits, ~training(.x))
           , validate=map(splits, ~testing(.x))
           , rf=map2(.x=train, .y=mtry, ~rand_forest(mtry = .y, trees = 200) %>%
                        set_engine("ranger", importance="impurity") %>%
                        fit(quality~.-quality_b, data=.x)
           ))
)
```

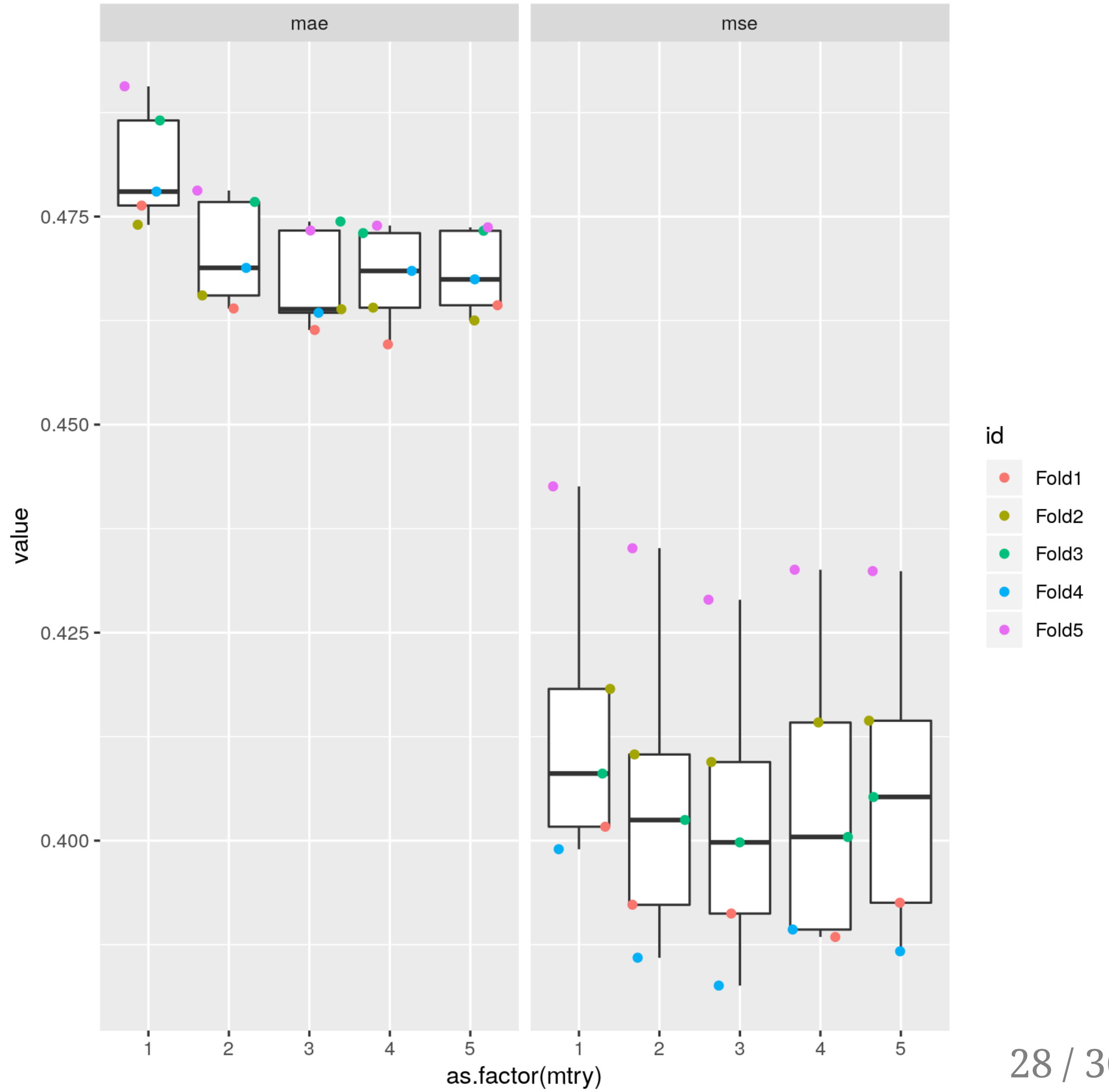
```
## # A tibble: 25 x 6
##    splits             id     mtry train                validate           rf
##    <list>             <chr> <int> <list>               <list>             <list>
##  1 <split [2.7K/68… Fold1     1 <tibble [2,743 × … <tibble [686 × … <fit[+…
##  2 <split [2.7K/68… Fold1     2 <tibble [2,743 × … <tibble [686 × … <fit[+…
##  3 <split [2.7K/68… Fold1     3 <tibble [2,743 × … <tibble [686 × … <fit[+…
##  4 <split [2.7K/68… Fold1     4 <tibble [2,743 × … <tibble [686 × … <fit[+…
##  5 <split [2.7K/68… Fold1     5 <tibble [2,743 × … <tibble [686 × … <fit[+…
##  6 <split [2.7K/68… Fold2     1 <tibble [2,743 × … <tibble [686 × … <fit[+…
##  7 <split [2.7K/68… Fold2     2 <tibble [2,743 × … <tibble [686 × … <fit[+…
##  8 <split [2.7K/68… Fold2     3 <tibble [2,743 × … <tibble [686 × … <fit[+…
##  9 <split [2.7K/68… Fold2     4 <tibble [2,743 × … <tibble [686 × … <fit[+…
## 10 <split [2.7K/68… Fold2     5 <tibble [2,743 × … <tibble [686 × … <fit[+…
## # … with 15 more rows
```

Model performance: post-grid search

```r
rf_model_tun <- rf_tun %>%
    mutate(pred=map2(.x=rf, .y=validate, ~predict(.x, .y) %>%
                         as_tibble %>%
                         set_names("pred"))
           , mae=map2_dbl(.x=pred, .y=validate
                          , ~measures::MAE(truth = .y$quality
                                           , response = .x$pred)
                          )
           , mse=map2_dbl(.x=pred, .y=validate
                          , ~measures::MSE(truth = .y$quality
                                           , response = .x$pred)
           ))
```

Model performance: post-grid search

```
rf_model_tun %>%
    dplyr::select(mtry, id, mae, mse) %>%
    pivot_longer(cols = c(mae, mse)
                 , names_to = "measure") %>%
    ggplot(aes(as.factor(mtry), value))+
    geom_boxplot() +
    geom_jitter(aes(col=id))+
    facet_grid(~measure, scales = "free")
```



Regression: final model

```r
best_model <- rand_forest(mtry = 5, trees = 200) %>%
                        set_engine("ranger", importance="impurity") %>%
                        fit(quality~.-quality_b, data=train_data)
test_pred <- predict(best_model, test_data) %>%
  as_tibble %>%
  set_names("pred")

mae_rf <- measures::MAE(truth = test_data$quality, response = test_pred$pred)
mse_rf <- measures::MSE(truth = test_data$quality, response = test_pred$pred)
```

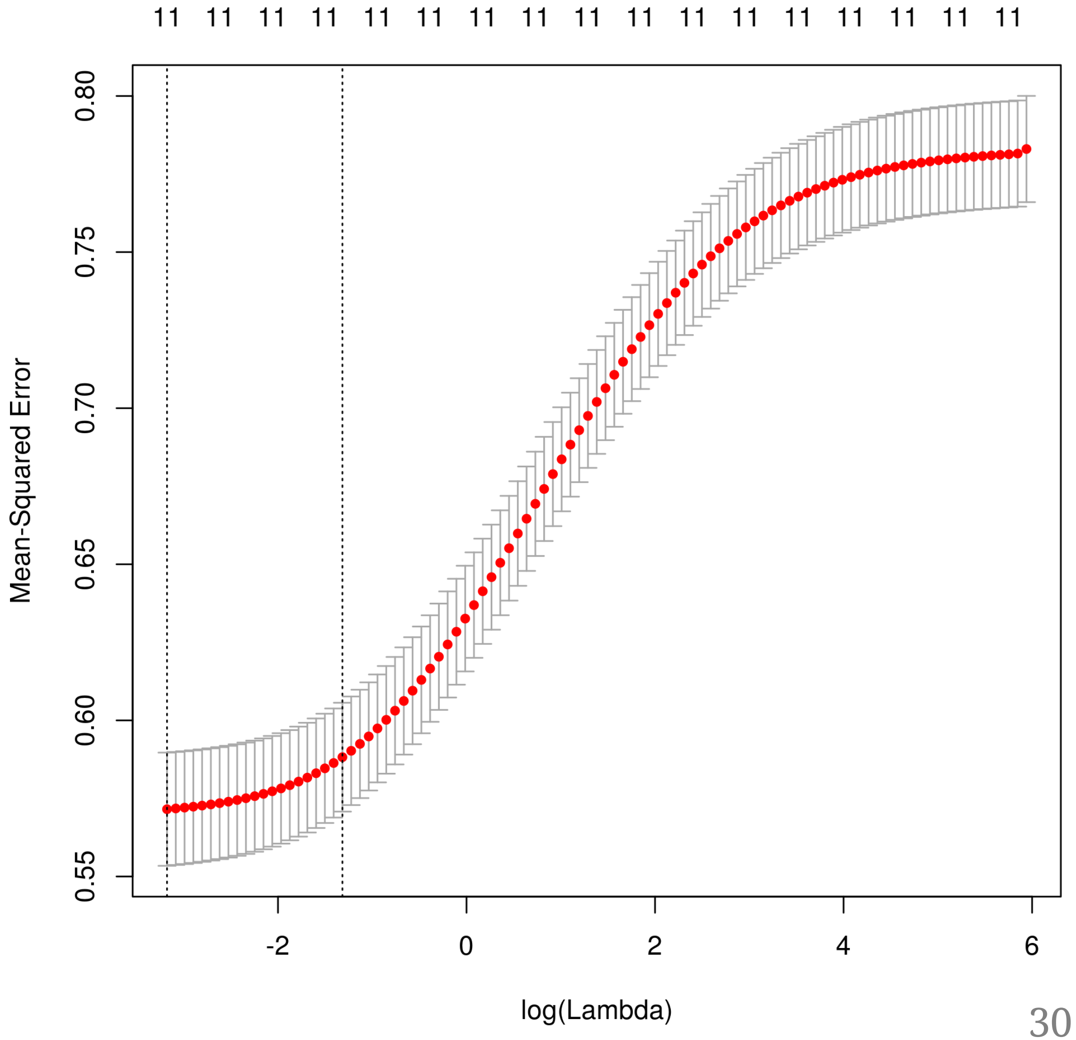
Ridge

Tweak data

```
X <- train_data %>%
  dplyr::select(-quality, -quality_b) %>% as.matrix()
y <- train_data %>%
  dplyr::select(quality) %>% as.matrix()
X_test <- test_data %>%
  dplyr::select(-quality, -quality_b) %>% as.matrix()
y_test <- test_data %>%
  dplyr::select(quality) %>% as.matrix()
```

Model

```
ridge_cv <- cv.glmnet(X, y, alpha = 0, standardize = T, nfolds
```

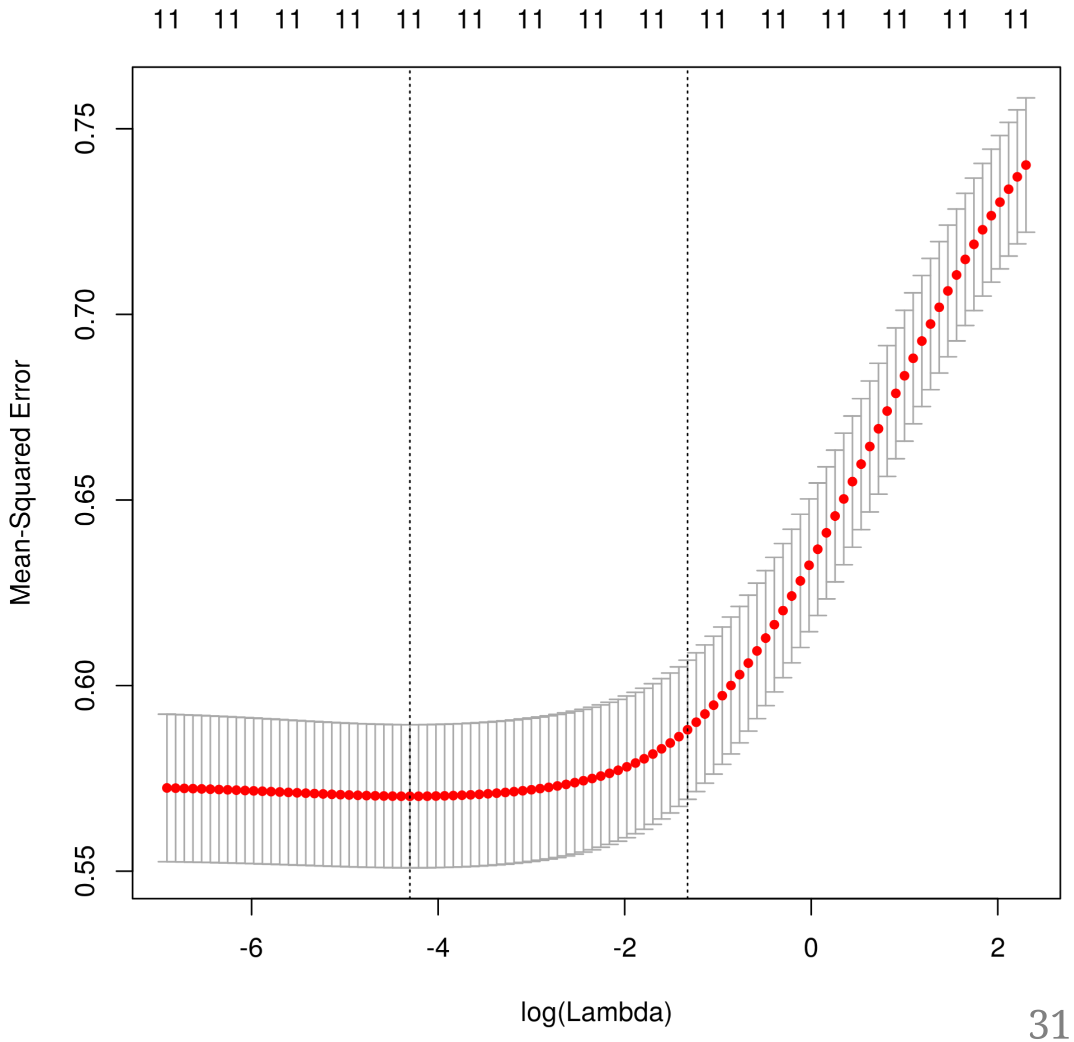


Ridge

Tweak model

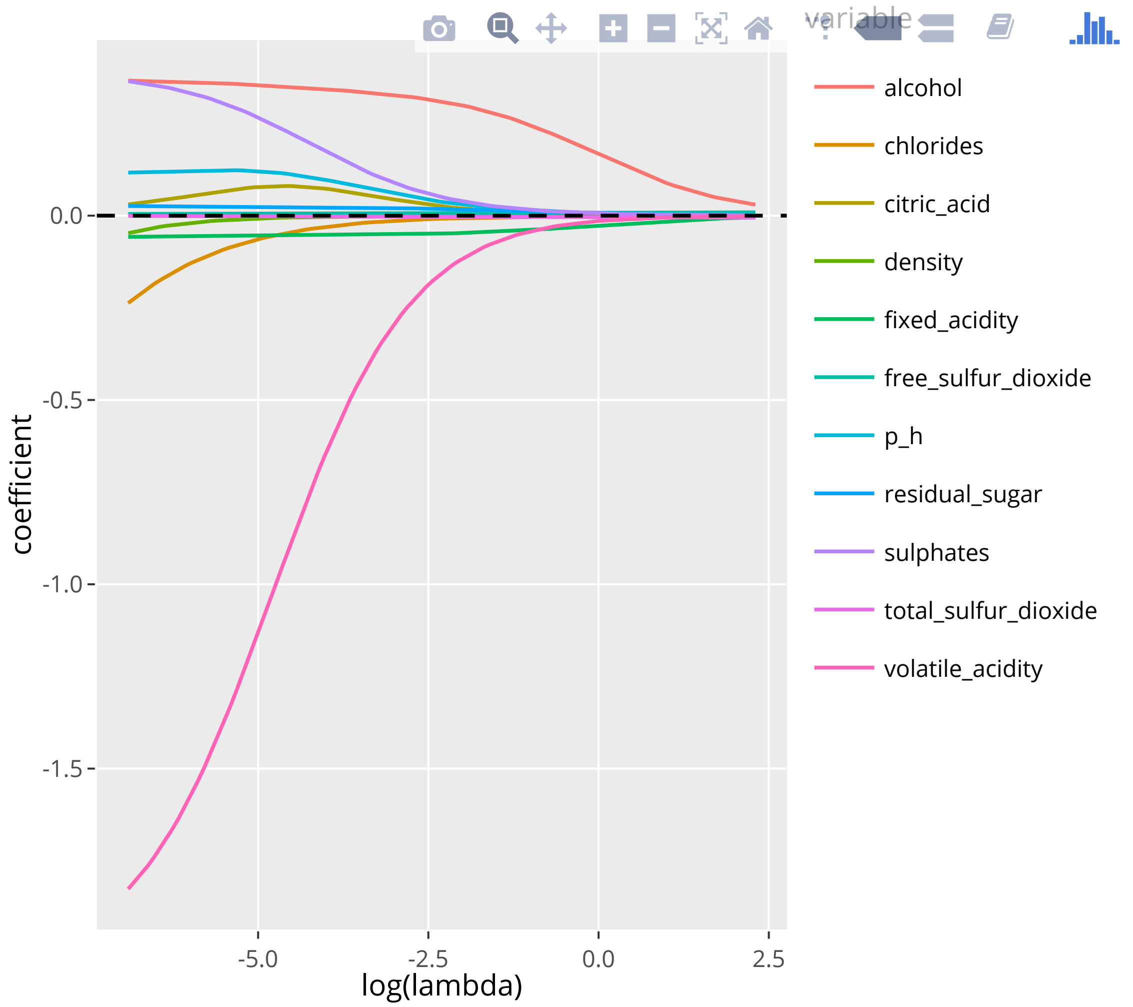
```
ridge_lambda <- 10^seq(-3, 1, length.out = 100)
ridge_cv_a1 <- cv.glmnet(X, y, alpha = 0, lambda = ridge_lambd
                       , standardize = T, nfolds = 10)
```

```
ridge_best <- glmnet(X, y, alpha = 0, lambda = ridge_cv_a1$lam
                       , standardize = T)
ridge_pred <- predict(ridge_best, X_test)
mae_ridge <- measures::MAE(truth = test_data$quality, response
```



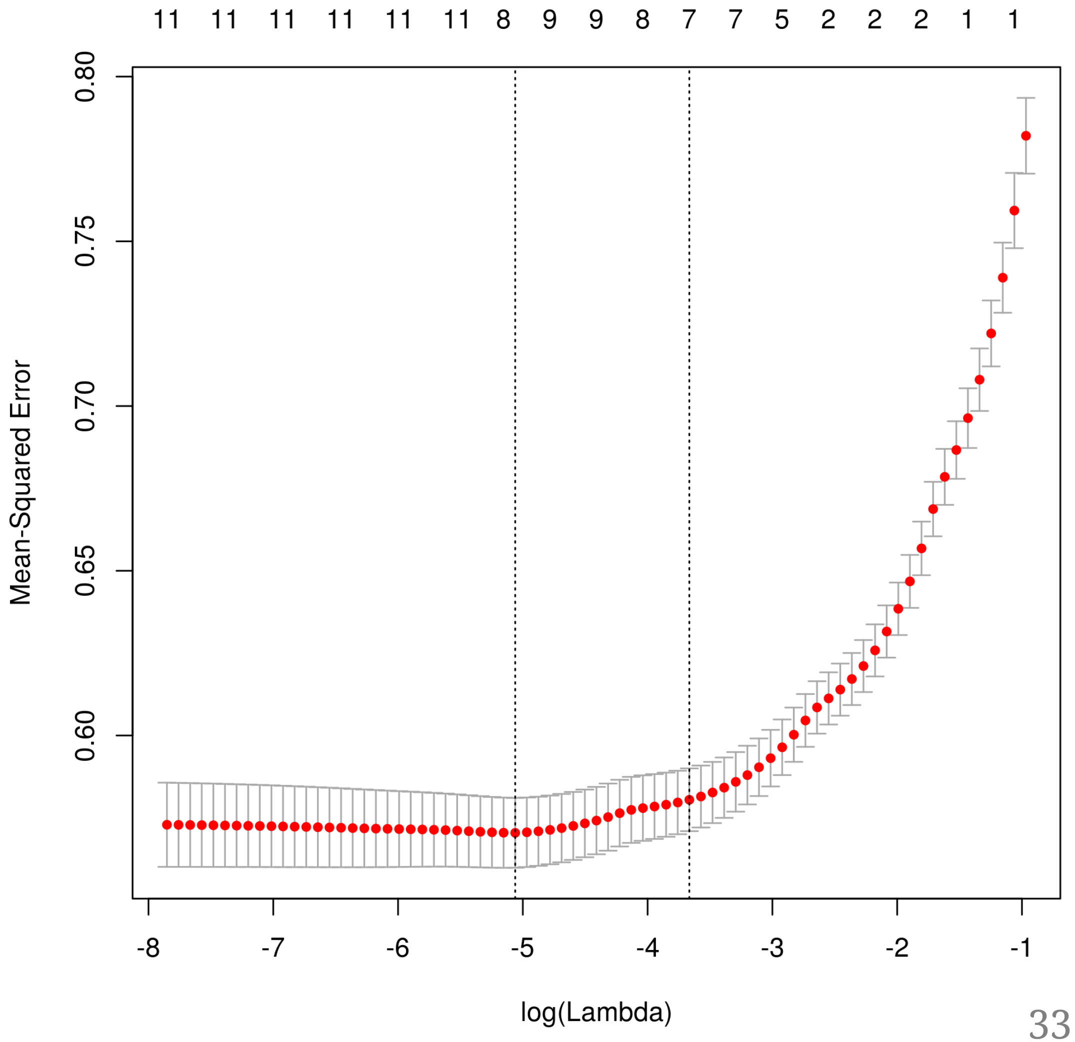
Ridge

```
ridge_shrink <- glmnet(X, y, alpha = 0
                       , lambda = ridge_lambda
                       , standardize = F)
```



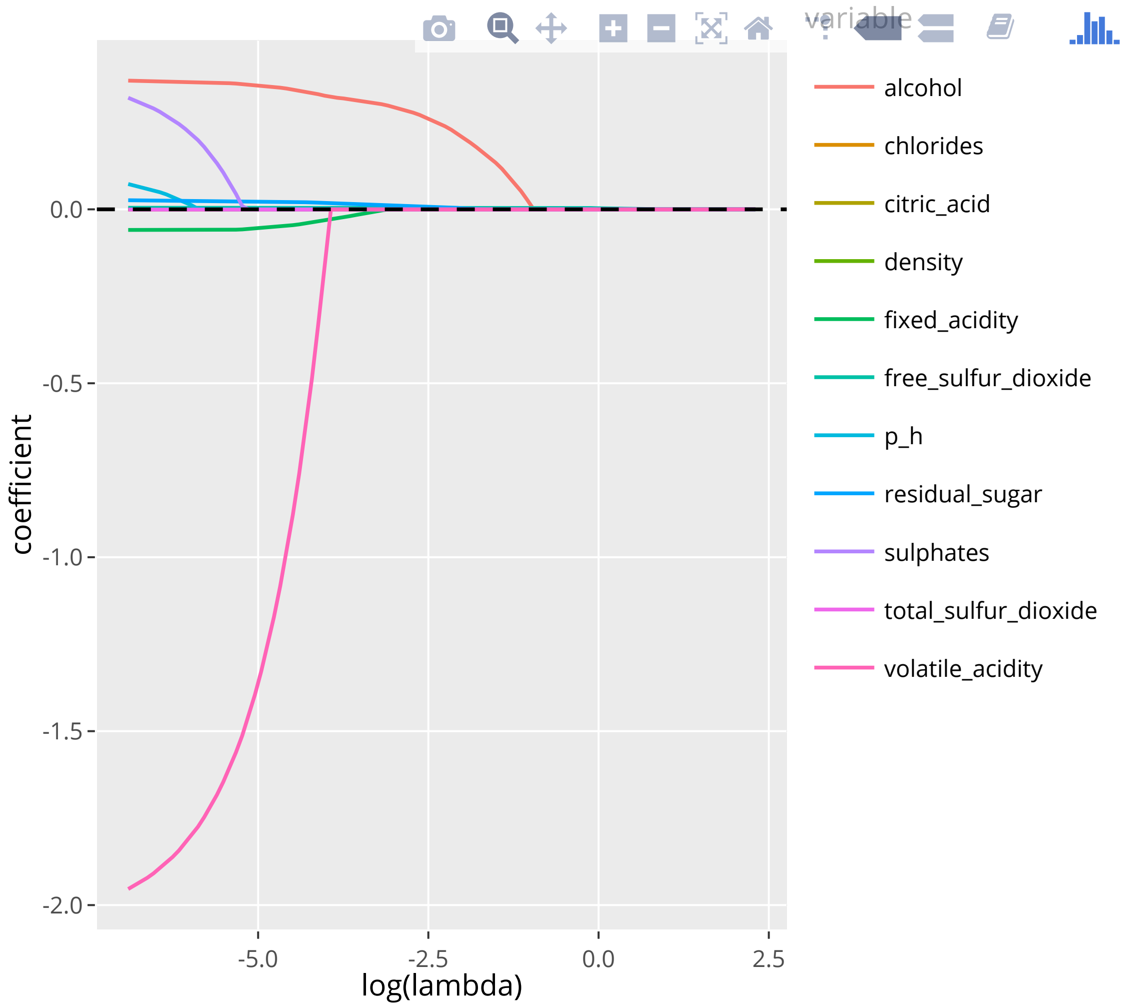
Lasso

```
lasso_cv <- cv.glmnet(X, y, alpha = 1
                      , standardize = TRUE, nfolds = 10)
lasso_best <- glmnet(X, y, alpha = 1
                     , lambda = lasso_cv$lambda.min
                      , standardize = TRUE)
lasso_pred <- predict(lasso_best, X_test)
mae_lasso <- measures::MAE(truth = test_data$quality
                           , response = lasso_pred)
```



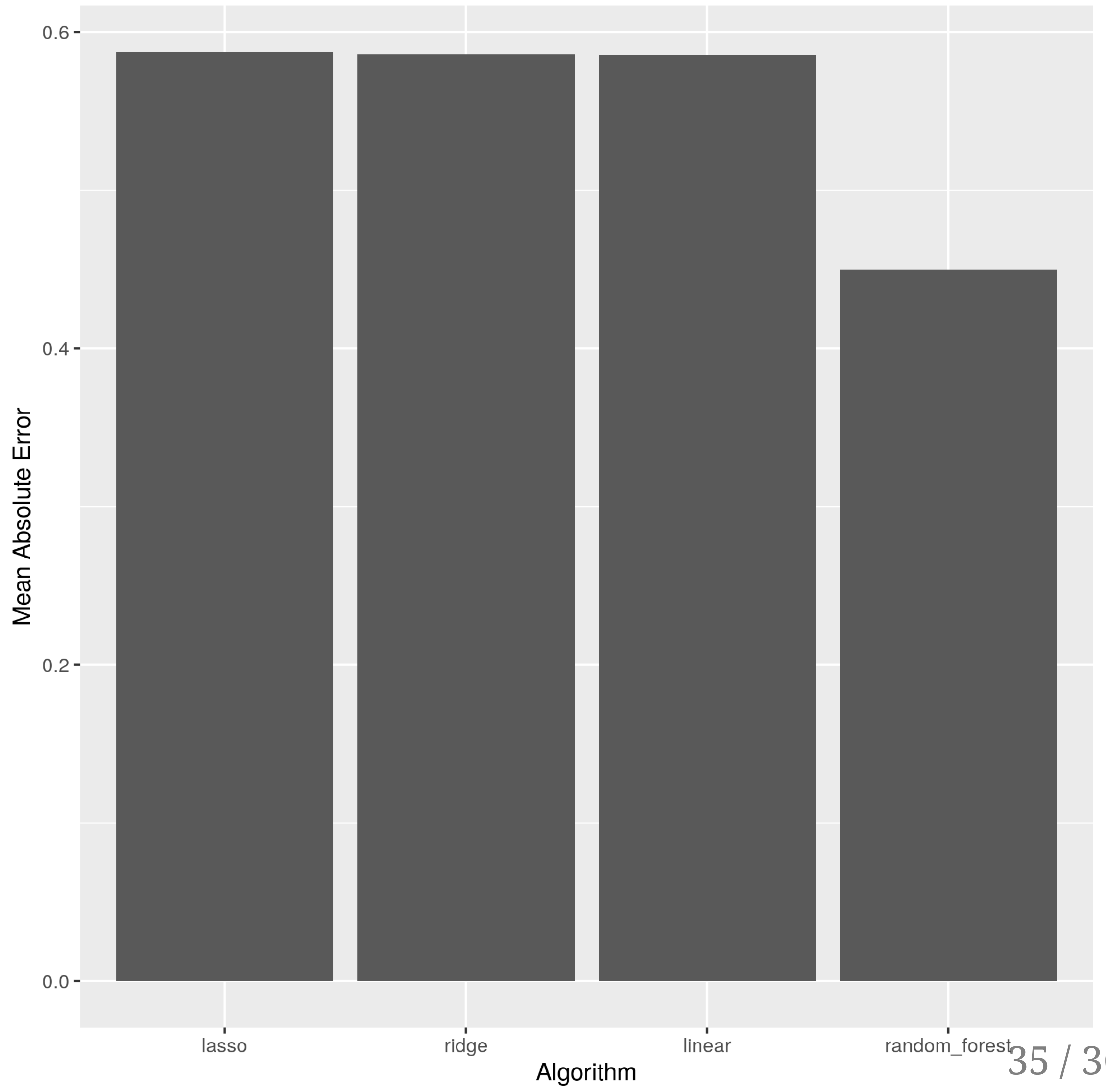
Lasso shrinkage

```r
lasso_lambda <- 10^seq(-3, 1, length.out = 100)
lasso_shrink <- glmnet(X, y, alpha = 1
                       , lambda = lasso_lambda
                       , standardize = F)
```



Comparison

```r
data.frame(linear=mae_lm, random_forest=mae_rf
           , ridge=mae_ridge, lasso=mae_lasso) %>%
  t() %>%
  as.data.frame() %>%
  tibble::rownames_to_column("MAE") %>%
  as_tibble() %>%
  ggplot(aes(reorder(MAE, desc(V1)), V1))+
  geom_col()+
  labs(x="Algorithm", y="Mean Absolute Error")
```



References

- Géron, A. (2017). Hands-On Machine Learning with Scikit-Learn & TensorFlow.
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning. Springer, 27(2), 83–85.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning. Springer. New York.
- Müller, A. C., & Guido, S. (2017). Introduction to machine learning with Python. O'Reilly.