Choosing hyperparameters

- What are *hyperparameters*?
- Which hyperparameters have we encountered in the course so far?

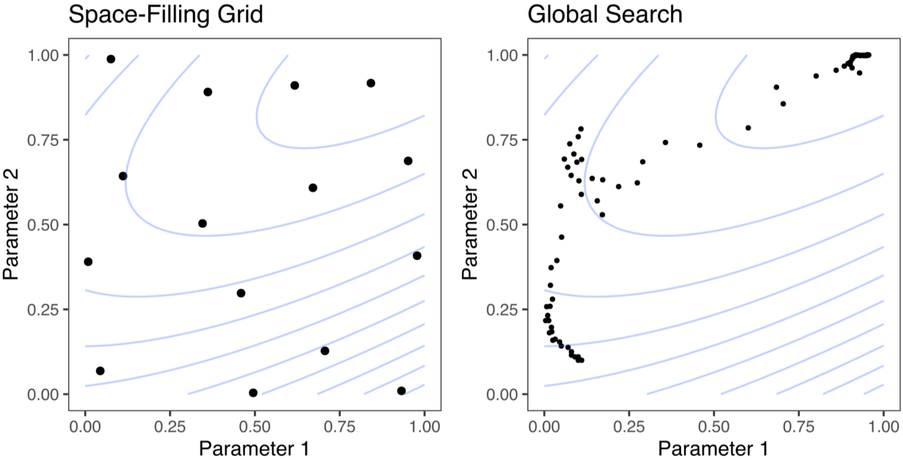
- Hyperparametertuning is performed using a separate validation set or by cross-validation. Different loss functions or selection criteria may be used (MSE, AUC, misclassification rate, ...).
- The hyperparameter tuning is often referred to as a black-box optimization because we (usually) only calculate loss function values (with CV) and do not get to compute gradients.

What may be challenges with hyperparameter optimization?

Some challenges with hyperparameter optimization (Feurer and Hutter, Ch1):

- expensive evaluation of the model under study (large networks, large data sets)
- unclear which of possibly many hyperparameters that need to be selected carefully (refer to the discussion for xgboost)
- gradient of selection criterion with respect to the hyperparameters not (generally) available, and criterion not convex or smooth in the hyperparameters
- and the need for external validation or CV

Grid search vs iterative search



Surrogate methods

We will look at two types of surrogate models: Bayesian regression with Gaussian processes (in Bayesian optimization) and regression-type models in response surface methods (presented by group 2).

Bayesian optimization

Bayesian optimization is an iterative method - where we start with evaluating some loss function at some predefined set of points in the hyperparameter space. New position in the hyperparameter space are chosen iteratively.

Two key ingredients:

- a surrogate model (we will only look at Bayesian regression with Gaussian processes) to fit to the observed values of the loss function in the hyperparameter space
- an *acquisition* function to decide a new point in the hyperparameter space to evaluate next

Underlying idea: given some “observations” in the hyperparameter space, the task is to decide where to place a new point. We should try a point where:

- we expect a good value and/or
- we have little information so far

To do that we need information on both expected value *and* variance - or preferably the distribution of the loss function for your problem.

Multivariate normal distribution

The random vector $\mathbf{Y}_{p\times 1}$ is multivariate normal N_p with mean and (positive definite) covariate matrix Σ. The pdf is:

$$f(\mathbf{Y}) = \frac{1}{(2\pi)^{\frac{p}{2}}|\Sigma|^{\frac{1}{2}}}\exp\{-\frac{1}{2}(\mathbf{Y}-\ )^T\Sigma^{-1}(\mathbf{Y}-\ )\}$$

The conditional distributions of the components are (multivariate) normal.

$$\mathbf{Y}_2 \mid (\mathbf{Y}_1 = \mathbf{Y}_1) \sim N_{p2}(\ {}_2 + \Sigma_{21}\Sigma_{11}^{-1}(\mathbf{Y}_1 - \ {}_1), \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}).$$

Gaussian processes

(Eidsvik 2017, page 6-7, note in TMA4265)

A Gaussian process is defined for

- times or locations x_i, $i = 1, \dots, n$ in $\Re^d$, where
- $Y_i = Y(x_i)$ is a random variable at x_i
- such that $\mathbf{Y} = (Y_1, \dots, Y_n)$ is multivariate Gaussian.

The process is *first order (mean) stationary* if $\text{E}(Y(x)) = \mu$ for all x, and this can be extended to depend on covariates.

The process is *second order stationary* if $\text{Var}(Y(t)) = \sigma^2$ for all x and the correlation $\text{Corr}(Y(x), Y(x'))$ only depends on differences between x and x'.

The multivariate Gaussian distribution is defined by the mean and covariance alone.

Correlation functions

- We assume that points at positions close to each other have a stronger correlation than point far apart.

Power exponential or Gaussian kernel

$$\mathsf{Corr}(Y(x), Y(x')) = \exp(-\phi_G \|x - x'\|^2)$$

where the L2 distance is used and ϕ_G is a parameter that determine the decay in the correlations.

Matern-type kernel

$$\mathsf{Corr}(Y(x), Y(x\acute{})) = (1 + \phi_M \|x - x'\|) \exp(-\phi_M \|x - x'\|)$$

decay-describing parameter ϕ_M.

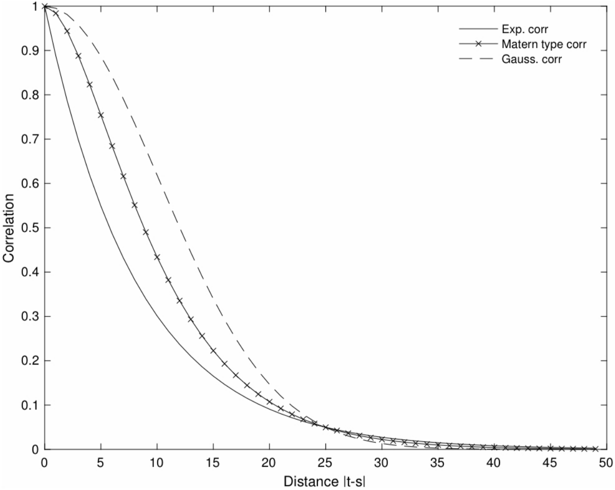


Figure 4: Three different correlation functions.

From correlations into covariance matrix

For simplicty assume that $d = 1$. The number of positions to consider is n.

To get from correlation function to a $n \times n$ covariance matrix first construct a $n \times n$ matrix of distances for each pair of positions, denote this $\mathbf{H}$.

For the Matern-type correlation function the covariance matrix can then be written

$$\Sigma = \sigma^2 (1 + \phi_M \mathbf{H}) \otimes \exp(-\phi_M \mathbf{H}))$$

where $\otimes$ is elementwise multiplication.

Acquisition function: Expected improvement

(Frazier 2018 page 7)
Thought experiment:

1) we have evaluated our function at all possible points x, and must return a solution based on what we already have evaluated. If the evaluation is noise-less we need to return the point with the largest observed value f.
2) Correction: We may perform one more evaluation. If we choose x we observe $f(x)$, and the best point before that was f_n^*. The improvement at the new observation is then

$$\max(f(x) - f_n^*, 0)$$

3) We define the *expected improvement* as

$$\mathsf{EI}_n(x) = \mathsf{E}_n[\max(f(x) - f_n^*, 0)]$$

where the expectation is taken at the posterior distribution given that we have evaluated f at n observations $x_1, \ldots, x_n$, and the posterior distribution is that f conditional on $x_1, \ldots, x_n, y_1, \ldots, y_n$ is normal with mean $\mu_n(x)$ and variance $\sigma_n^2(x)$.

4) How to evaluate the expected improvement? Integration by parts gives

$$\mathsf{EI}_n(x) = \max(\mu_n(x) - f_n^*, 0)]) + \sigma_n(x)\phi(\frac{\max(\mu_n(x) - f_n^*, 0)}{\sigma_n(x)})$$

$$-\mathsf{abs}(\mu_n(x) - f_n^*)\Phi(\frac{\max(\mu_n(x) - f_n^*, 0)}{\sigma_n(x)})$$

$\mu_n(x) - f_n^*$ is expected proposed vs previously best

5) We choose to evaluate the point with the largest expected improvement

$$x_{n+1} = \mathsf{argmaxEI}_n(x)$$

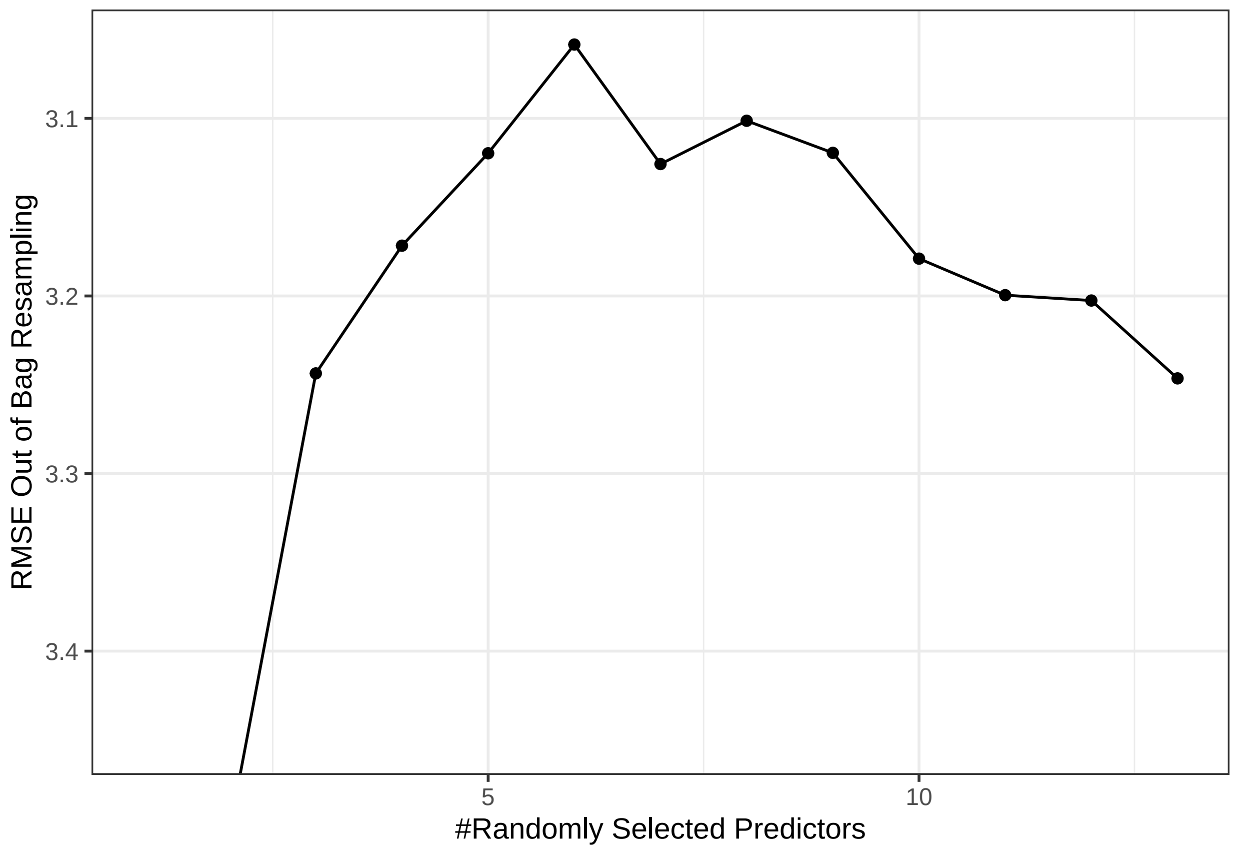
```
Place a Gaussian process prior on $f$.
Observe $f$ at $n_0$ points from some experimental design. Set $n = n_0$.
while $n \leq N$ do
    Update the posterior on f with all available data
    Let $x_n$ be a maximizer of the acquisition function over $x$, computed using the current posterior
    Observe $y_n = f(x_n)$
    Increment $n$
end while
Return a solution: a point with largest $f(x)$ or the point with the largest posterior mean
```

Example

(Kuhn and Silge, Ch 14, the example is for SVM)

First just grid search to test what is best value for `mtry`

```
data(Boston, package = "MASS")
# first using a grid
tune_grid <- expand.grid(
  mtry = (1:13))
#  ntree=seq(100,500,length=10)) # how to also include ntree? primary only mtry, how to define secon
tune_control <- caret::trainControl(
  method = "oob", # cross-validation #eller cv
  #number = 3, # with n folds
  verboseIter = FALSE, # no training log
  allowParallel = FALSE # FALSE for reproducible results
)
rf_tune <- caret::train(
  medv~crim+zn+indus+chas+nox+rm+age+dis+rad+tax+ptratio+black+lstat,
  data=Boston,
  na.action=na.roughfix,
  trControl = tune_control,
  tuneGrid = tune_grid,
  method = "rf", # rf is randomForest, checked at #vhttp://topepo.github.io/caret/train-models-by-ta
  verbose = TRUE
)
tuneplot <- function(x, probs = .90) {
  ggplot(x) +
    coord_cartesian(ylim = c(quantile(x$results$RMSE, probs = probs), min(x$results$RMSE))) +
    theme_bw()
}
tuneplot(rf_tune)
```



```r
rf_tune$bestTune
```

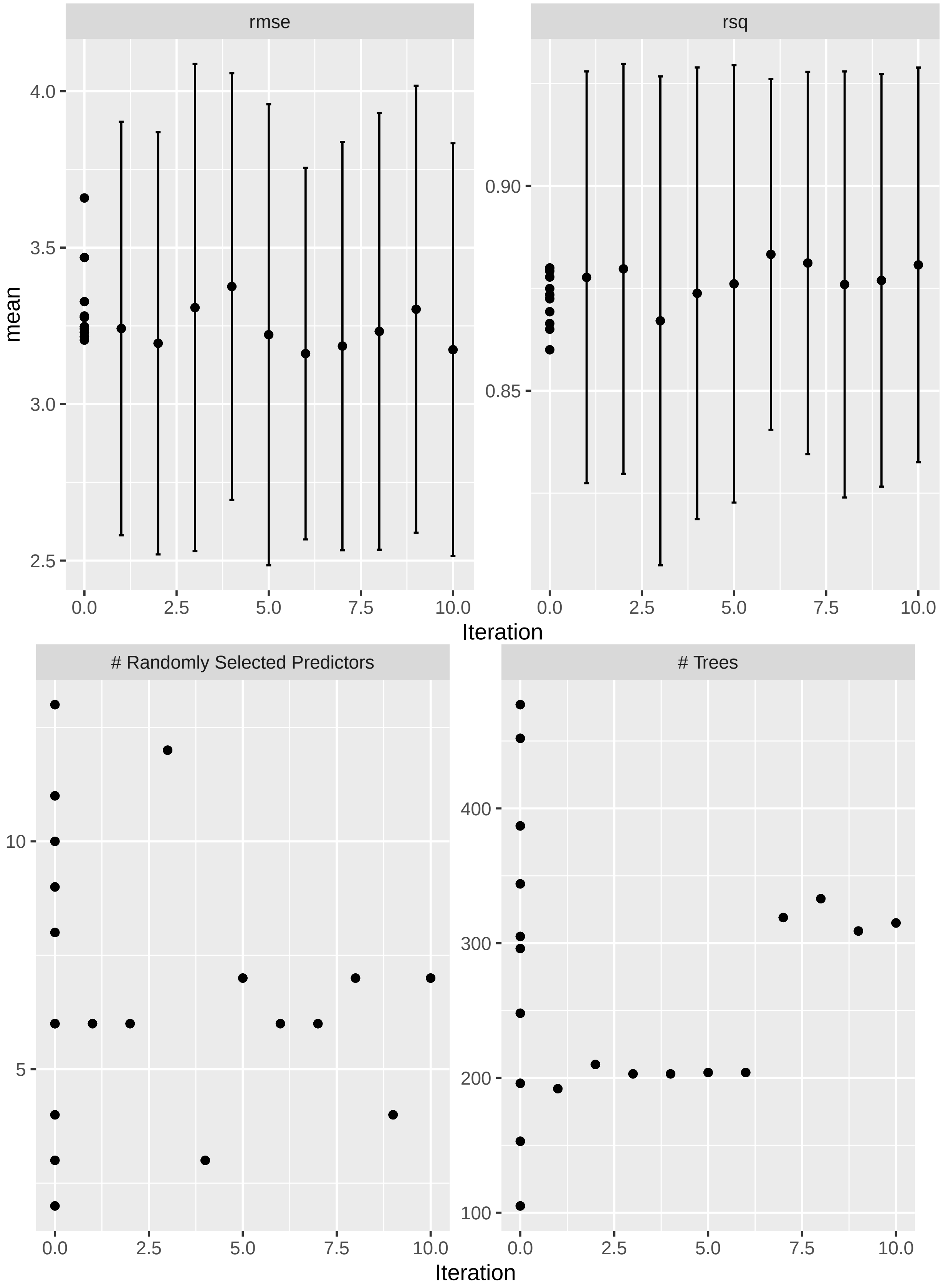
```
##   mtry
## 6    6
```

The R the function `tune_bayes` is available in the package `tune`, and requires that the analyses is done with a workflow. Default in the GP is exponential correlation function, but first we try the Matern.

```r
tree_rec <- recipe(medv~crim+zn+indus+chas+nox+rm+age+dis+rad+tax+ptratio+black+lstat, data = Boston)

tune_spec <- rand_forest( # parsnip interface to random forests models
  mode="regression",
  mtry = tune(),
  trees = tune(),
#  min_n = tune()
) %>%
#  set_mode("regression") %>%
#  set_engine("ranger",objective="reg:rmse") # errors with ranger
  set_engine("randomForest") # randomforest ok

tune_wf <- workflow() %>%
  add_recipe(tree_rec) %>%
  add_model(tune_spec)

tune_param <- tune_spec%>%
  parameters%>%
  update(mtry=mtry(c(1L,13L)),trees=trees(c(100L,500L)))
```

```r
vfold  <- vfold_cv(Boston, v = 5)
# then trying BO
ctrl <- control_bayes(verbose = TRUE)
bayesres<- tune_bayes(tune_wf,
    resamples = vfold,
    #metrics = rmse,
    corr=list(type="matern",nu=5/2),
    #default in corr_mat(GPfit) is "exponential" power 1.95
    initial = 10,
    param_info = tune_param,
    iter = 10,
    objective=exp_improve(),
    control = ctrl
  )
dput(bayesres,"bayesres.dd")
```

```
## # A tibble: 10 x 9
##     mtry trees .metric .estimator  mean     n std_err .config              .iter
##    <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>                <int>
##  1     6   204 rmse    standard    3.16     5   0.231 Iter6                    6
##  2     7   315 rmse    standard    3.17     5   0.257 Iter10                  10
##  3     6   319 rmse    standard    3.19     5   0.254 Iter7                    7
##  4     6   210 rmse    standard    3.19     5   0.262 Iter2                    2
##  5     6   196 rmse    standard    3.20     5   0.252 Preprocessor1_Model~     0
##  6     6   296 rmse    standard    3.22     5   0.256 Preprocessor1_Model~     0
##  7     7   204 rmse    standard    3.22     5   0.287 Iter5                    5
##  8     8   305 rmse    standard    3.23     5   0.280 Preprocessor1_Model~     0
##  9     7   333 rmse    standard    3.23     5   0.271 Iter8                    8
## 10     9   452 rmse    standard    3.24     5   0.283 Preprocessor1_Model~     0
```


rmse
rsq
mean
4.0
3.5
3.0
2.5
0.90
0.85
0.0
2.5
5.0
7.5
10.0
Iteration
# Randomly Selected Predictors
# Trees
10
5
400
300
200
100
Iteration

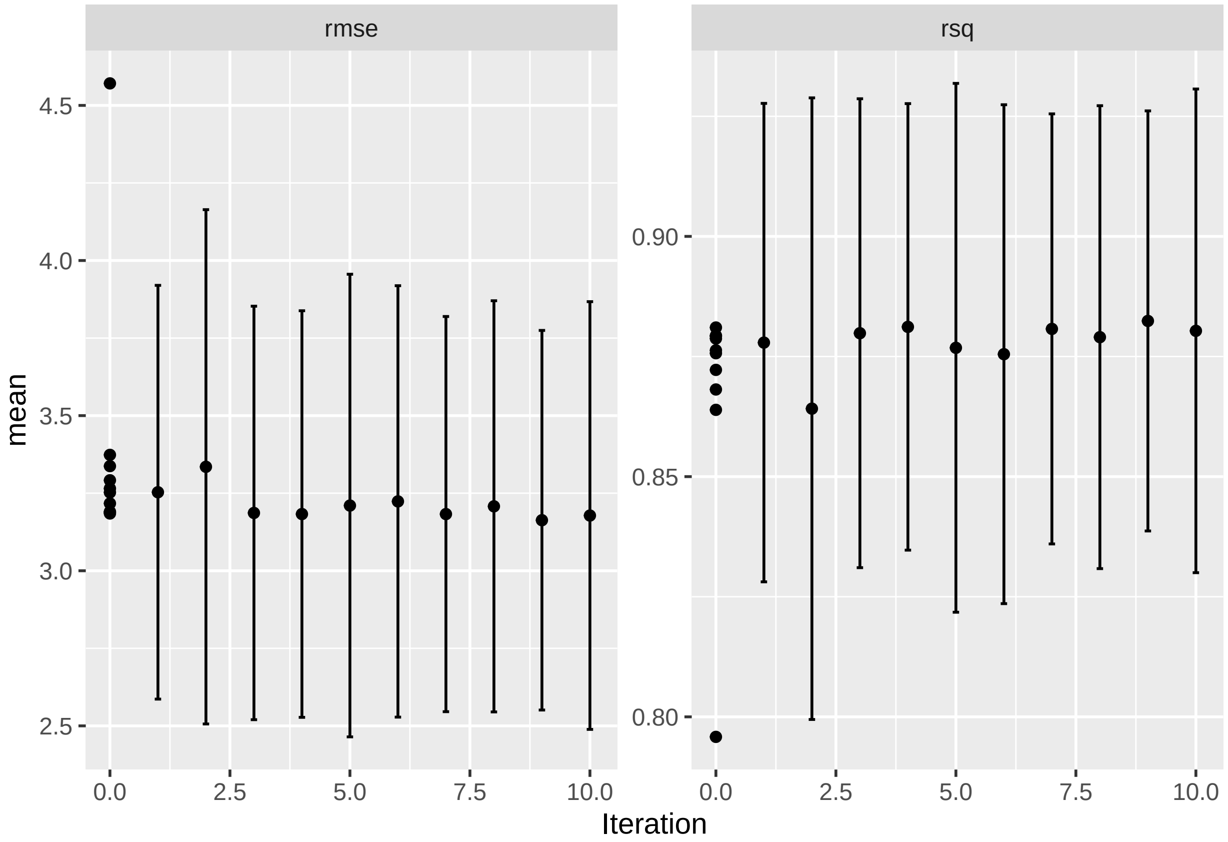
Here we try the default exponential correlation.

```
bayesres2<- tune_bayes(tune_wf,
    resamples = vfold,
    #metrics = rmse,
    #corr=list(type="matern",nu=5/2),
    #default in corr_mat(GPfit) is "exponential" power 1.95
    initial = 10,
    param_info = tune_param,
    iter = 10,
    objective=exp_improve(),
    control = ctrl
  )
dput(bayesres2,"bayesres2.dd")
```

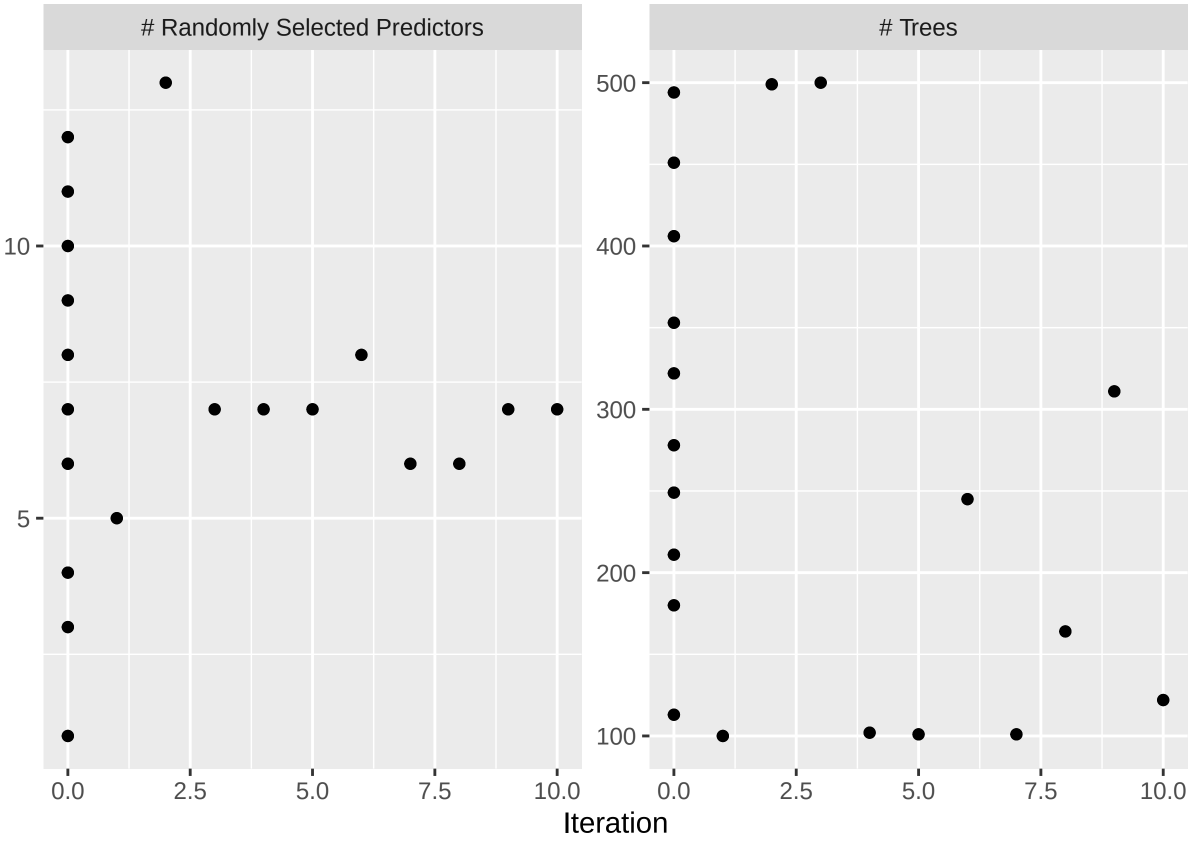
```
bayesres2=dget("bayesres2.dd")
show_best(bayesres2,n=10)
```

```
## # A tibble: 10 x 9
##     mtry trees .metric .estimator  mean     n std_err .config               .iter
##    <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>                 <int>
##  1     7   311 rmse    standard    3.16     5   0.238 Iter9                     9
##  2     7   122 rmse    standard    3.18     5   0.268 Iter10                   10
##  3     6   101 rmse    standard    3.18     5   0.248 Iter7                     7
##  4     7   102 rmse    standard    3.18     5   0.255 Iter4                     4
##  5     7   249 rmse    standard    3.18     5   0.270 Preprocessor1_Model~      0
##  6     7   500 rmse    standard    3.19     5   0.259 Iter3                     3
##  7     6   278 rmse    standard    3.19     5   0.255 Preprocessor1_Model~      0
##  8     8   113 rmse    standard    3.19     5   0.257 Preprocessor1_Model~      0
##  9     6   164 rmse    standard    3.21     5   0.258 Iter8                     8
## 10     7   101 rmse    standard    3.21     5   0.290 Iter5                     5
```

```
autoplot(bayesres2,type="performance")
```



```
autoplot(bayesres2,type="parameters")
```


# Randomly Selected Predictors
# Trees
10
5
500
400
300
200
100
0.0
2.5
5.0
7.5
10.0
Iteration

Suggested software

- R: DiceOptim (on CRAN)
- R: tune_bayes in `tune` (also CRAN)
- Python: Spearmint https://github.com/HIPS/Spearmint
- Python: GPyOpt https://github.com/SheffieldML/GPyOpt
- Python: GPFlow (Tensorflow) https://github.com/GPflow/GPflow and GPYTorch (PyTorch) https://github.com/cornellius-gp/gpytorch

Design of experiments and response surface methodology

Article presentation by group 2.

G. A. Lujan-Moreno, P. R. Howard, O. G. Rojas and D. C. Montgomery (2018): Design of experiments and response surface methodology to tune machine learning hyperparameters, with a random forest case- study. Expert Systems with Applications. 109, 195-205.