07 - Cross-Validation and Model Selection

Data Mining

SYS 6018 | Fall 2019

07-crossval.pdf

Contents

1 Predictive Performance

1.1 Model Complexity

Models can be of varying complexity:

- number of parameters / edof (polynomials, interactions)
- penalty λ or constraint (t) for regularized models
- neighborhood size (knn)
- number of trees, tree depth (classification and regression trees, random forest, boosting)
- etc. (we will cover more models later in course)

Highly adaptable model families can accommodate complex relationships, but easily overemphasize patterns that are not reproducible (e.g. noise or statistical fluctuation)

Goal is to be flexible (complex) enough to find reproducible (true) structure, but not overfit

1.2 Tuning parameters

Tuning parameters are the “control knobs" of a model. It may be better to refer to these as *complexity parameters* because they are usually connected to the flexibility/complexity of a model. E.g., the λ in lasso and ridge are tuning parameters

We will represent the tuning parameters with ω:

- For knn regression, $\omega = k$
- For polynomial regression, $\omega = J$ in the model $\hat{f}(x) = \sum_{j=0}^{J} \beta_j x^j$
- In best subsets regression, $\omega = p$, the total number of features allowed in the model
- In ridge regression, $\omega = \lambda$ in the ridge penalty $\text{Pen}(\beta) = \lambda \sum_{\text{j}=1}^{\text{p}} \beta_{\text{j}}^2$

Given the value of the tuning parameter, it is often straightforward to estimate the *model parameters* from the data (e.g., $\hat{\beta}$)

1.3 Predictive Performance

Because our focus is on predictive performance (not interpretation or inference), the optimal tuning parameter(s) is the value(s) that minimizes the expected prediction error (PE):

$$\text{EPE}(\omega) = \text{E}[\text{PE}(\omega)] = \text{E}_{\tilde{X},\tilde{Y}}[\ell(\tilde{Y}, \hat{f}_\omega(\tilde{X}))]$$

where:
- $\ell()$ is the loss function (e.g., RSS/MSE)
- $\tilde{X}, \tilde{Y}$ are drawn from the same distribution (hopefully) as the sample data
- $\hat{f}_\omega(x) = f_\omega(x; \hat{\theta}_\omega(\mathcal{D}))$ is the prediction function which has been estimated under the tuning parameter ω

Ideally, we want to pick tuning parameter(s) ω to minimize EPE

$$\omega^{opt} = \arg\min_{\omega} \text{EPE}(\omega)$$

1.4 Training Error

There are a few ways to estimate EPE. A *not* very good way is to use the training error (TE):

$$\text{TE}(\omega, \mathcal{D}) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \hat{f}_\omega(x_i))$$

where $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ is the training data.

Using the ω that minimizes the training error:

$$\omega^* = \arg\min_{\omega} \ \text{TE}(\omega, \mathcal{D})$$

Will always overfit (as long as f is flexible enough) Why?

1.5 Training Error vs. Testing Error

Figure Taken from: ESL Fig 7.1

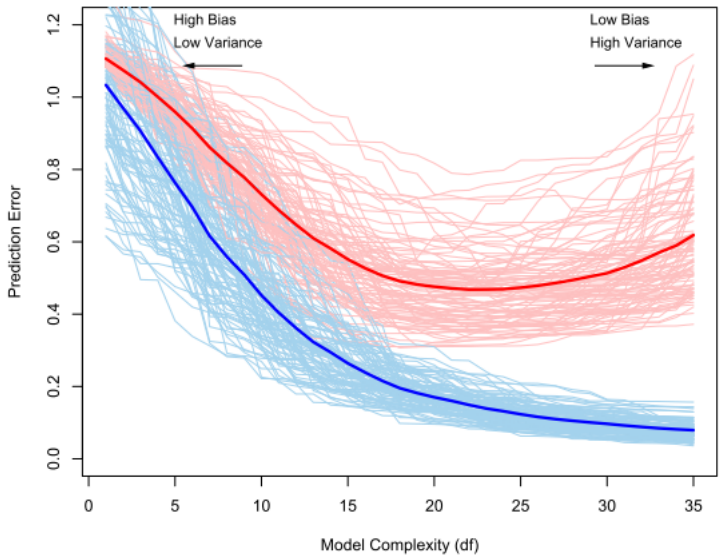


FIGURE 7.1. *Behavior of test sample and training sample error as the model complexity is varied. The light blue curves show the training error* $\overline{\text{err}}$*, while the light red curves show the conditional test error* $\text{Err}_{\mathcal{T}}$ *for* 100 *training sets of size* 50 *each, as the model complexity is increased. The solid curves show the expected test error* Err *and the expected training error* $\text{E}[\overline{\text{err}}]$.

1.6 Model Selection and Assessment

- **Model selection:** estimating the performance of different models in order to choose the best one.

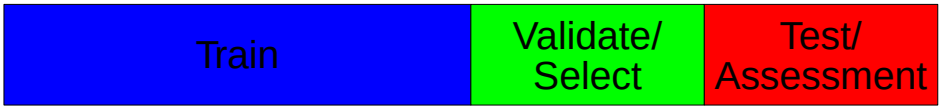
$$\omega^* = \arg\min_{\omega} \text{ EPE}(\omega)$$

- **Model assessment:** having chosen a final model, estimating its prediction error on new data (i.e., estimating EPE for final model).

$$\text{EPE}(\omega^*) = \text{E}_{\tilde{X},\tilde{Y}}[\ell(\tilde{Y}, \hat{f}_{\omega^*}(\tilde{X}))]$$

1.7 Train/Validate/Test

If it were possible to have lots of data (all from the same distribution), then we would split up the data into three pieces:



- Train: Estimate model parameters θ (for given tuning ω) for many models ($\omega_1, \omega_2, \ldots$)
  - Use the training dataset to estimate the *model parameters* (e.g., β) for a set of *tuning parameters* (e.g., ω) (and possibly different model families)
  - The output from this step will be a set of fitted models, $\hat{f}_1, \hat{f}_2, \ldots$
- Validate/Select: Choose optimal tuning parameters ω (i.e., model selection step)
  - Evaluate the performance of each fitted model on the Validate/Select data
  - Choose the best/final model based on the performance
- Test/Assessment: Estimate EPE (i.e., final model assessment)
  - Never use the Test/Assessment data until **all** model fitting, tuning, selection is finished.
  - Then the performance of the best/final model can be assessed (without bias)

Your Turn #1

1. What estimates suffers where there is not enough data in each group?
2. Is the performance of a model on the validation data reflective of its performance on the Test/Assessment data? When/Why do we need Test/Assessment data?

Notation

I usually speak in terms of training and test data only.

- By test data, I am referring to any hold-out data set.
- So most of the time my *test set* is really referring to what is called the Validate/Select data.

2 Model Selection

The goal of model selection is to pick the model that will provide the best *predictive performance* on test data (i.e, model with smallest EPE).

There are three main approaches to estimating the predictive performance (EPE) of a model:

1. Predict on hold-out data
2. Make a mathematical adjustment to the training error that better estimates the test error
3. Resampling methods (cross-validation, bootstrap)
    - i.e., use *multiple* hold-out sets

2.1 Hold out set

An obvious way to assess how well a model will perform on test data is to evaluate it on hold-out data.

Split the data into a training and test set: $\mathcal{D} = (\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}})$.

- Fit models with $\mathcal{D}_{\text{train}}$ to get estimates $\hat{\theta}_\omega = \hat{\theta}_\omega(\mathcal{D}_{\text{train}})$ and $\hat{f}_\omega(\cdot) = f(\cdot; \hat{\theta}_\omega)$.
- Use $\mathcal{D}_{\text{test}}$ to calculate:

$$\text{PE}(\omega, \mathcal{D}_{\text{test}}) = \frac{1}{m} \sum_{i=1}^{m} \ell(\tilde{y}_i, \hat{f}_\omega(\tilde{x}_i))$$

where $(\tilde{x}_i, \tilde{y}_i) \in \mathcal{D}_{\text{test}}$ for $i = 1, 2, \ldots, m$.

2.1.1 Problems with using a single Hold-Out set

1. Need to decide how much data into each set?
    - Too little in training and poor parameter estimates
    - Too little in test and poor performance estimates and model selection
2. We may happen to choose a split that uses a train or test set that poorly represents the data generating process.
    - The chance of *bad* split is reduced when $n - m$ and m are both large.

2.2 Adjustments to Training Error (AIC/BIC)

Another approach is to use all the data for training, but adjust the training error to account for potential over-fitting

- The adjustment is usually a function of model complexity (e.g., edf)

Examples:

- AIC/BIC
- Adjusted R^2

- Mallow's Cp

2.2.1 AIC/BIC

- Let $\mathcal{M}$ be a model (e.g., model family and ω)
- Let $\hat{\mathcal{M}} = \arg\max_\theta \ \mathcal{M}(\theta)$ be the parameters that maximize the liklihood (or penalized log-likelihood).
- $L(M)$ is likelihood of the model
  - Thus, to use AIC/BIC a distributional assumption must be specified
- Let $d(\hat{\mathcal{M}})$ be the *effective degrees of freedom (edf)* (e.g., number of estimated model parameters) of the model

Akaike information criterion (AIC)

$$\text{AIC}(\mathcal{M}) = -2\log L(\hat{\mathcal{M}}) + 2d(\hat{\mathcal{M}})$$

Bayesian information criterion (BIC)

$$\begin{aligned}\text{BIC}(\mathcal{M}) &= -2\log L(\hat{\mathcal{M}}) + d(\hat{\mathcal{M}})\log n \\ &= \text{AIC}(\hat{\mathcal{M}}) + d(\hat{\mathcal{M}})\left(\log(n) - 2\right)\end{aligned}$$

Information Criterion (generic)

$$\begin{aligned}\text{IC}(\mathcal{M}) &= \ell(\hat{\mathcal{M}}) + P(\hat{\mathcal{M}}) \\ &= \text{Loss} + \text{Penalty}\end{aligned}$$

Example

For a linear regression model with $d = p + 1$ estimated coefficients, $\mathcal{M}_d = f(x;\beta) = \sum_{j=0}^{p}\beta_j x_j$,

- d is the tuning parameter, β are the model parameters

$$\log L(\hat{\mathcal{M}}_d) = -\frac{n}{2}\log(\text{MSE}(\hat{\beta})) + C$$

where C is a constant that doesn't depend on any model parameters or tuning parameters

$$\text{AIC}(\hat{\mathcal{M}}_\text{d}) = \text{n}\log(\text{MSE}(\hat{\beta})) + 2\text{d}$$

- Choose the model that *minimizes* the AIC/BIC
- The AIC/BIC can be used for any likelihood (e.g., logistic regression)

2.2.2 Adjusted R^2

$$R^2_{\text{adj}} = 1 - \frac{\text{RSS}(\hat{\beta})}{\text{RSS}(\bar{y})}\frac{n-1}{n-d}$$

where d is the number of estimated parameters in the model.

- R^2_{adj} is only appropriate under a squared error loss function.
- Choose the model with the *largest* R^2_{adj}

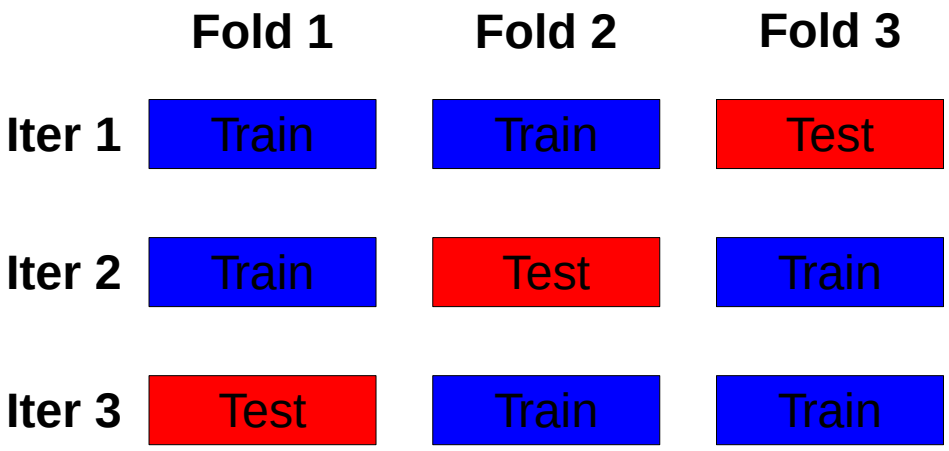
3 Cross-Validation and other Resampling based model selection procedures

3.1 Cross-Validation

Cross-validation is a way to use more of the data for **both** training and testing

- Randomly divide the set of observations into K groups, or folds, of approximately equal size.
- The first fold is treated as a validation set, and the model is fit on the remaining $K-1$ folds. And predictions are made on the hold-out set.
- The performance on each fold is combined to get a more accurate assessment of model performance on future data.

3.1.1 3-fold cross-validation



In practice, the observations should be assigned **randomly** to the folds (not sequentially like in the above figure)

- This will reduce the influence of potential correlation in the data

3.2 Cross-Validation Algorithm

Algorithm: Cross-Validation

1. Split data into K folds (of roughly equal size)
   - $\mathcal{F}_1, \ldots, \mathcal{F}_K$
2. For $k = 1, \ldots, K$ and for all models (e.g., for all ω) :
   a. Use the data in $\mathcal{D} \setminus \mathcal{F}_k$ to estimate the model $\hat{f}_\omega^{-k}$
   b. Predict for data in $\mathcal{F}_k$ and calculate the average loss

$$V_k(\omega) = \frac{1}{n_k} \sum_{i \in \mathcal{F}_k} \ell(y_i, \hat{f}_\omega^{-k}(x_i))$$

   where n_k is the number of observations in fold k
3. Choose tuning parameters (model selection) that minimize cross-validation loss

$$\hat{\omega} = \arg\min_\omega \text{CV}(\omega)$$

   where $\text{CV}(\omega) = \frac{1}{n} \sum_{k=1}^K n_k V_k(\omega)$
4. Refit all data using $\hat{\omega}$ to get the final model ($\hat{\theta}$).
   - An alternative is to use an *ensemble* model by combining the models from all folds with weights $1/K$

3.3 1 SE Rule

The standard error of the average cross-validation error can be estimated by:

$$\text{SE}(\omega) = \frac{\text{standard deviation of } \{V_k\}}{\sqrt{K}}$$

One Standard Error Rule suggests that instead of using $\hat{\omega} = \arg\min_\omega \text{CV}(\omega)$, we should use the least complex model that is within one standard error of $\text{CV}(\hat{\omega})$.

3.4 Choice of K

- $K = 5, 10, n$ are common choices
- $K = n$ is called *leave-one-out (LOOCV)*

3.4.1 Performance Bias/Variance Trade-off

- Note: around $(K-1)/K$ of the data is used for training and $1/K$ for testing
- if K is too small, then not enough training data and poor cv error estimate (**bias** in prediction error)
- if K is too large, then the f_ω^{-k} are correlated and variance is not reduced (**variance** in prediction error)

  - because the training data is similar across folds

- Computational: need to fit each model K times

3.5 Balanced Data Splitting for Cross-Validation

- The most basic approach is to use random sampling to assign observations to folds
- But this can be problematic if there are outliers or classes with small frequency
  - Especially a problem for categorical data. Some levels are not included in training set.
  - So how to predict when they show up in test set?
- Stratified sampling can be used to ensure similar distributions in each fold
  - e.g., equal number of observations in each fold from same quartile of y
- Or based on predictor values: clustering the training data and then assigning into folds such that an equal number of observations from each cluster are in each fold

3.6 Different Model Families

Most of the discussion has been on finding the optimal complexity/tuning parameter for a given *model family*.

But cross-validation can be used to compare across model families.

Consider some options and their corresponding complexity/tuning parameters:

- linear regression with stepwise variable selection
  - number of parameters p, or acceptance/rejection criteria
- elastic net (includes lasso and ridge)
  - α and λ
- k-nearest neighbor
  - k

Use CV to pick best-of-the-best

Be careful to ensure that all aspects of estimation (e.g., variable selection, tuning parameter selection) are **inside** the cross-validation.

3.7 Repeated Cross-Validation

If you have the patience (or computing resources), you can be more certain about the model performance if you repeat cross-validation several times.

- if you repeat K-fold cross-validation 5 times, then you will need to fit each model $5K$ times
- Take the average of the cross-validation score as the performance measure.

3.8 Repeated Train/Test splits

But why even bother about the added code complexity involved in making the folds?

- Just repeatedly split the data into a training and test set
- This will be similar to the repeated cross-validation, but is just simpler to code
  - holding out 10% of the data is equivalent to $K = 10$ fold cross-validation.

3.9 Out-of-Bag

We already saw how the bootstrap can be used for model selection.

- use the out-of-bag observations to estimate predictive performance

Because about 37% of the data will be used for model assessment (testing data), it is similar to $K = 3$ cross-validation.

- But since you still use n observations for model fitting, it will not be equivalent

3.10 Resampling Comparison

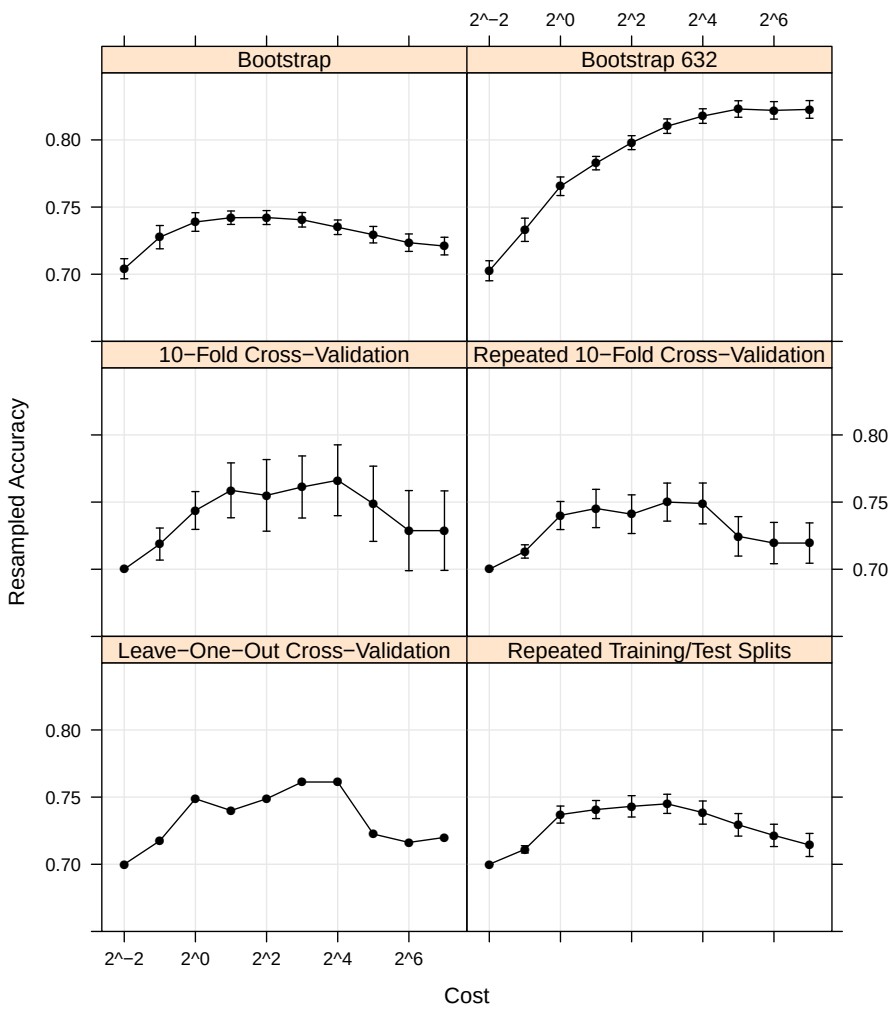


Figure 4.10 from Applied Predictive Modeling, by Kuhn and Johnson. Retreived on Oct 22, 2019 from https://github.com/topepo/APM_Figures

3.11 Linear Smoothers and LOOCV

A *linear smoother* is a model that the fitted training data can be represented by the equation

$$\hat{Y} = HY$$

- For example, in linear regression $\hat{Y} = X\hat{\beta}$ and therefore, $H = X(X^\mathsf{T}X)^{-1}X^\mathsf{T}$.
  - H is called the *hat matrix* or *projection matrix*
  - The diagonal elements of H are called the *leverages* of the observations
  - High leverage points has a large impact on the model fit. Thus, the LOOCV will be especially sensitive to observations with high leverage.

It turns out that for linear smoothers, the LOOCV error (under squared error loss) is

$$\text{LOOCV}(\omega) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{f}_\omega(x_i)}{1-h_{ii}}\right)^2$$

where h_{ii} is the i^{th} diagonal element of H.

- This takes away the computational burden of n model fits! The model is fit once to the full data and *corrected* for in the above equation.
- A similar, but even faster to compute, version is the *Generalized Cross-Validation*

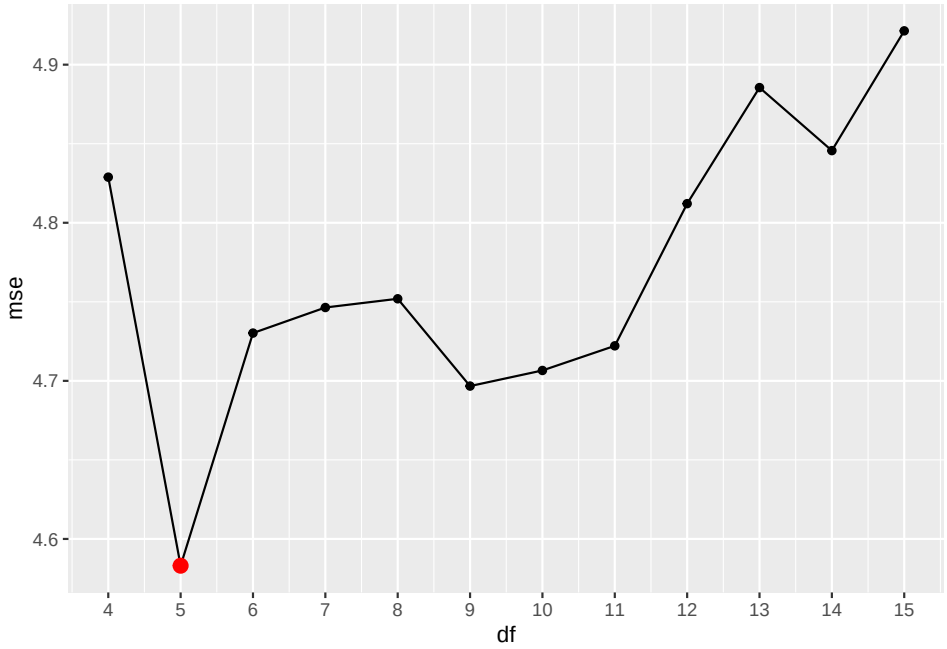
$$\begin{aligned}\text{GCV}(\omega) &= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{f}_\omega(x_i)}{1-tr(H)/n}\right)^2 \\ &= \frac{\text{MSE}_\omega}{1-tr(H)/n}\end{aligned}$$

  where $tr(H) = \sum_{i=1}^{n} h_{ii}$ is the *trace* of H.
  - In unpenalized linear regression $tr(H) = d$, the number of estimated parameters.

4 R Code

```
#- Get K-fold partition
set.seed(2019)                        # set seed for replicability
n.folds = 10                          # number of folds for cross-validation
fold = sample(rep(1:n.folds, length=n))  # vector of fold labels
# notice how this is different than:  sample(1:K,n,replace=TRUE),
#  which won't give almost equal group sizes

#- initialize
DF = seq(4, 15, by=1)      # edfs for spline
kts.bdry = c(-.2, 1.2)
results = tibble()

#- Iterate over folds
for(j in 1:n.folds){

  #-- Set training/val data
  val = which(fold == j)     # indices of validation data
  train = which(fold != j)   # indices of trainin data
  n.val = length(val)        # number of observations in validation

  #-- fit set of spline models
 for(df in DF){
    #- fit with training data
    m_train = lm(y~bs(x, df=df, Boundary.knots=kts.bdry)-1,
                 data=data_train[train,])
    #- predict on val data
    yhat = predict(m_train, newdata=data_train[val, ])
    #- get errors
    sse = sum( (data_train$y[val] - yhat)^2 )
    #- save results
```

```
    results = bind_rows(results,
                        tibble(fold=j, df, sse, n.val))
  }
}

results %>% group_by(df) %>%
  summarize(sse = sum(sse), mse=sse/nrow(data_train)) %>%
  ggplot(aes(df, mse)) + geom_point() + geom_line() +
  geom_point(data=. %>% filter(mse==min(mse)), color="red", size=3) +
  scale_x_continuous(breaks=1:20)
```



- The minimum cross-valiation error occurs at `df=5`. This matches the optimal complexity from the out-of-bag bootstrap analysis and the polynomial fit from the previous lecture notes.