STA314, Lecture 2

Sept 17, 2018

Stanislav Volgushev

Recap from previous lecture

K-nn estimator

$$\widehat{f}(x_0) = \frac{\sum_{i=1}^{n} Y_i I\{x_i \text{ among closest K to } x_0\}}{K}$$

Bias-Variance decomposition

$$\mathbb{E}[(\hat{Y}_0 - f(x_0))^2] = \mathrm{Var}\,(\hat{Y}_0) + \{\mathbb{E}[\hat{Y}_0] - f(x_0)\}^2.$$

Bias and variance of K-nn (derived in lectures for specific setting)

$$\mathrm{Var}\,[\hat{f}(x_0)] = \sigma^2/K,$$

$$\mathbb{E}[\hat{f}(x_0)] - f(x_0) \approx \frac{1}{3} f''(x_0)(K/n)^2.$$

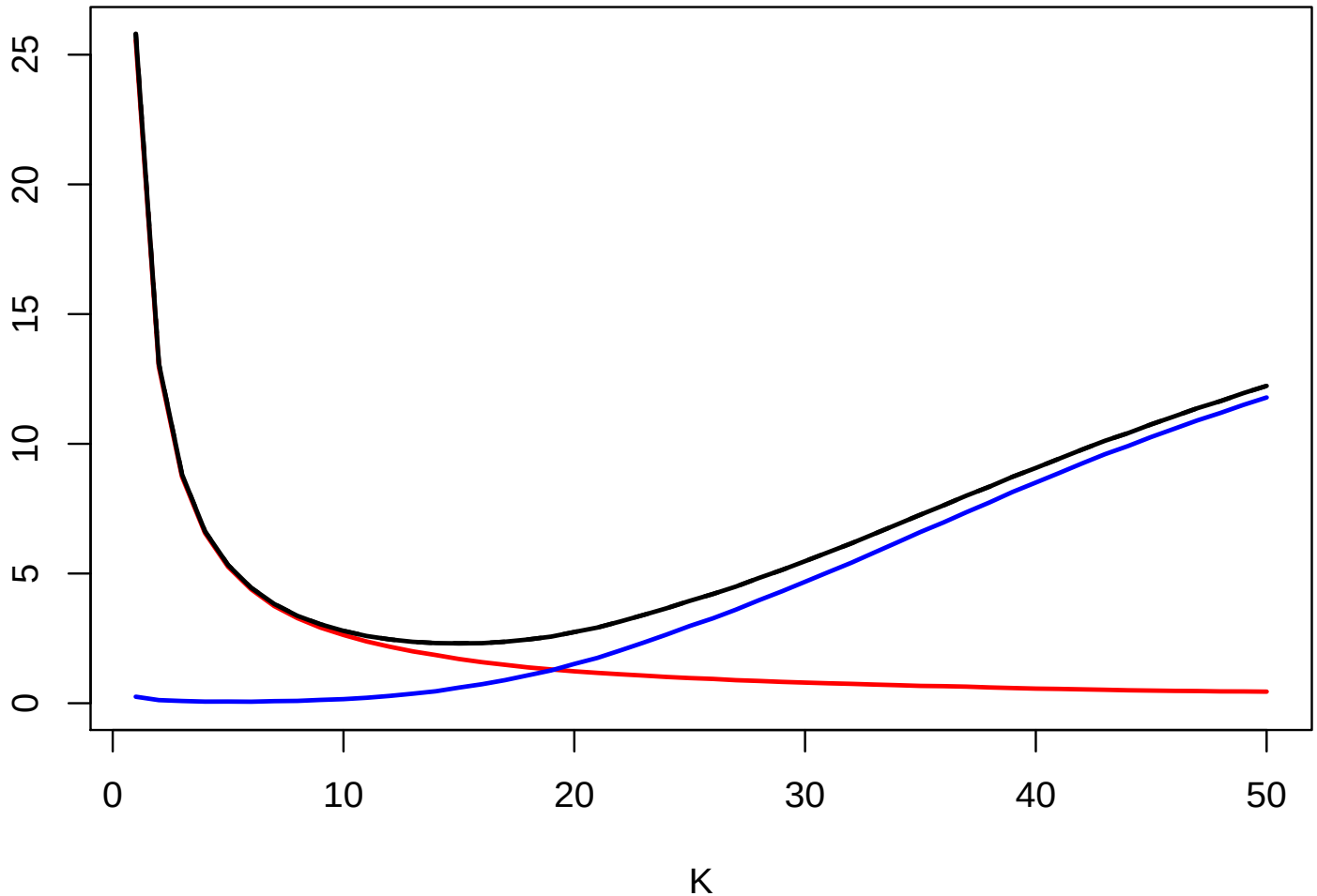
Could minimize over K to obtain value which gives smallest MSE (exercise), would obtain $K = cn^{4/5}$ for some constant c. Note: optimal K depends on n but also on unknown quantities such as f''.

Remark: often the average MSE is more interest. More precisely, given new observations $(X_i^{new}, y_i^{new})_{i=1,\ldots,M}$ we are interested in

$$\sum_{j=1}^{M} (\hat{f}(X_i^{new}) - y_i^{new})^2.$$

Let's look at some R code (simulated bias, variance and MSE of K-nn).

Bias and variance of K-nn



Bias, Variance and MSE (which curve is which?) of K-nn for $f(x) = 0.5 \sin(-2 + 12x)$ as a function of K.

K-nn: choosing K

Main message from previous slides: need to select a 'good' value of K to obtain good predictions. How do we go about that?

More general question: given any method for prediction, how do we evaluate the performance of our method?

First try: utilize available data set $(x_1, Y_1), ..., (x_n, Y_n)$ to compute *training error (in-sample prediction error)* :

$$E_{tr} = \frac{1}{n} \sum_{i=1}^{n} (\hat{f}(x_i) - Y_i)^2$$

and use this to measure quality of our prediction. Let's see how it works with K-nn.

The training error of K-nn regression...

Test error

- what we are really interested in: error when predicting outcomes for *new observations* that were not used to compute $\hat{f}$.
- this is called *test error (out-of sample prediction error, generalization error)*.
- if we had an additional sample, called *test sample*, $(x_1^{te}, y_1^{te}), ..., (x_N^{te}, y_N^{te})$ which was not used to build $\widehat{f}$, we could try to approximate this error by

$$E_{te} = \frac{1}{N} \sum_{i=1}^{N} (\widehat{f}(x_i^{te}) - y_i^{te})^2.$$

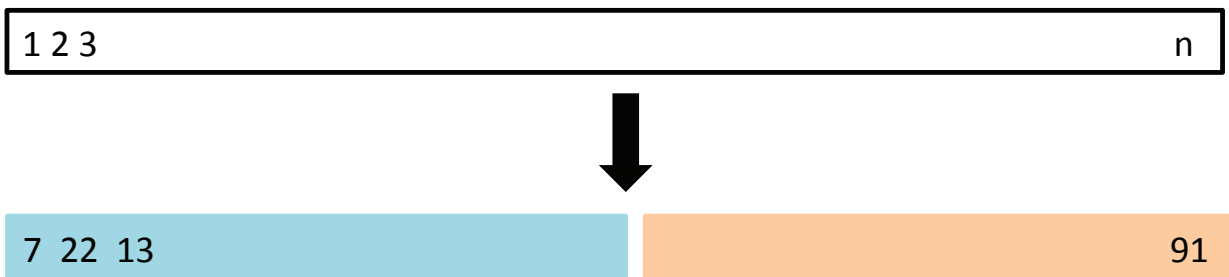
- if we knew the data-generating process, we could simulate additional data to obtain a test sample.
- in real-world scenarios we can not obtain more data, or only do so at additional cost. We have to work with the available sample.

The validation set approach

In real-world scenarios we can not obtain more data, or only do so at additional cost. So what do we do about it?

One possible idea: split the available data set into two parts, estimate $\widehat{f}$ from one part (*the training set*) and use the other part (*validation set*) to select tuning parameters.

The simplest version of this is to split into two (approximately) equal parts. This is sometimes called *validation set approach*.



Potential problems:

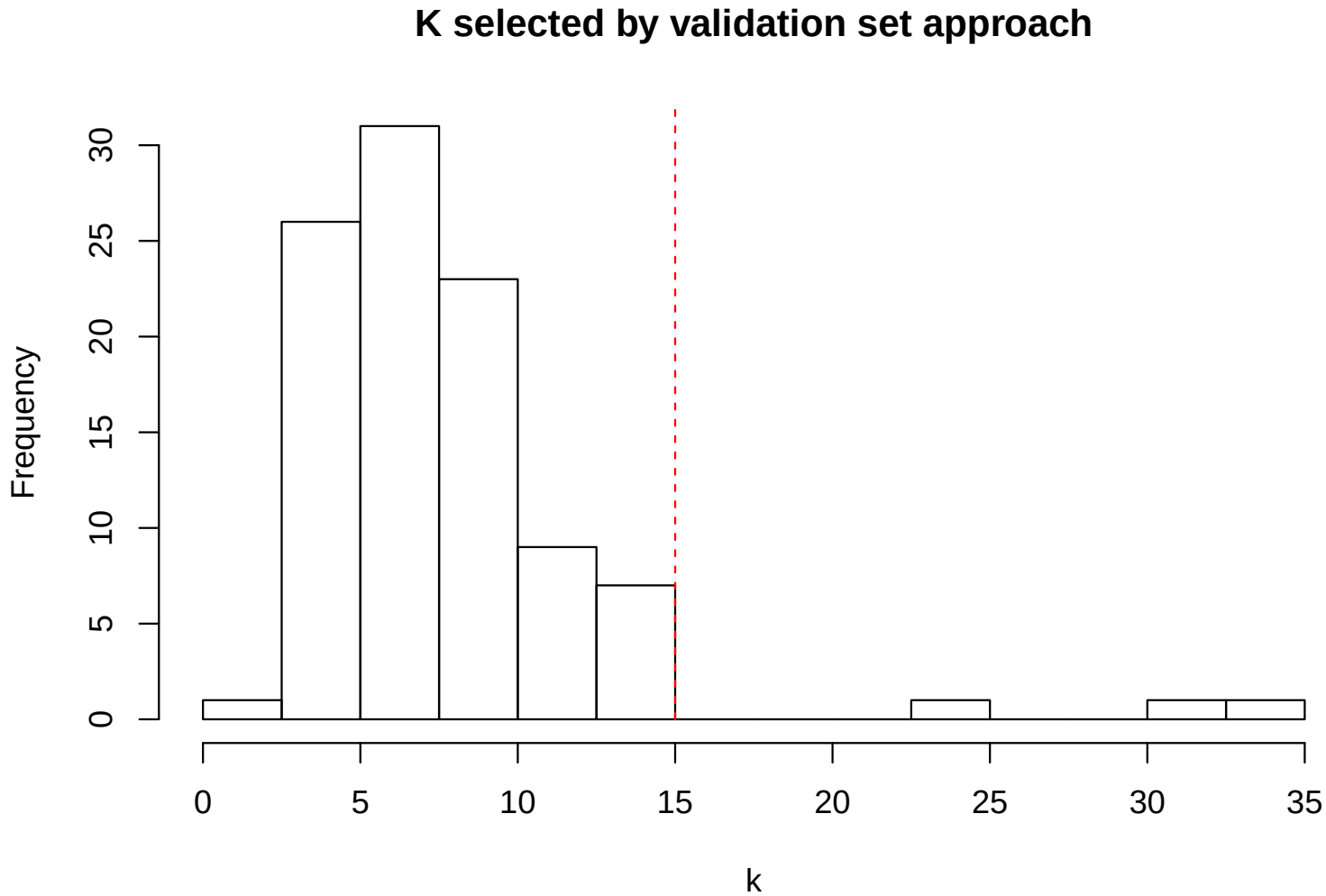
1. Since the split is random the answer we get might strongly depend on the split.
2. Only half of the data are used for learning the model. Usually more data for learning means better performance – splitting half-half might give biased answers.

Example of a function in R

```
valset_knn = function(X,Y,krange){
  n = length(Y)
  l.tr = floor(n/2)
  l.val = n - l.tr
  train_index = sample(1:n,l.tr) # randomly sample l.tr points
  # this will correspond to the training set

  # create array where results of validation error will be stored
  valset_error = array(0,length(krange))
  # loop over values of K, foer ach store error on validation set
  for(k in 1:length(krange)){
     K = krange[k]
     # only use data with index in train_index to fit the model
     fit = knnreg(as.matrix(x[train_index]),y[train_index],k=K)
     # now use fitted model to predict data which are not in train_index
     pr = predict(fit, newdata = data.frame(x = x[-train_index]))
     # compute and store the validation error
     valset_error[k] = mean((y[-train_index] - pr)^2)
  } # end loop over k
  return(valset_error) # this will be function output
} # end of function
```

The validation set approach: example



Histogram after 100 runs. Red line: k that gives optimal MSE (from previous simulation). Observation: tends to select too small k.

Improving the validation set approach

Problem: the split is random so the answer we get might strongly depend on the split.

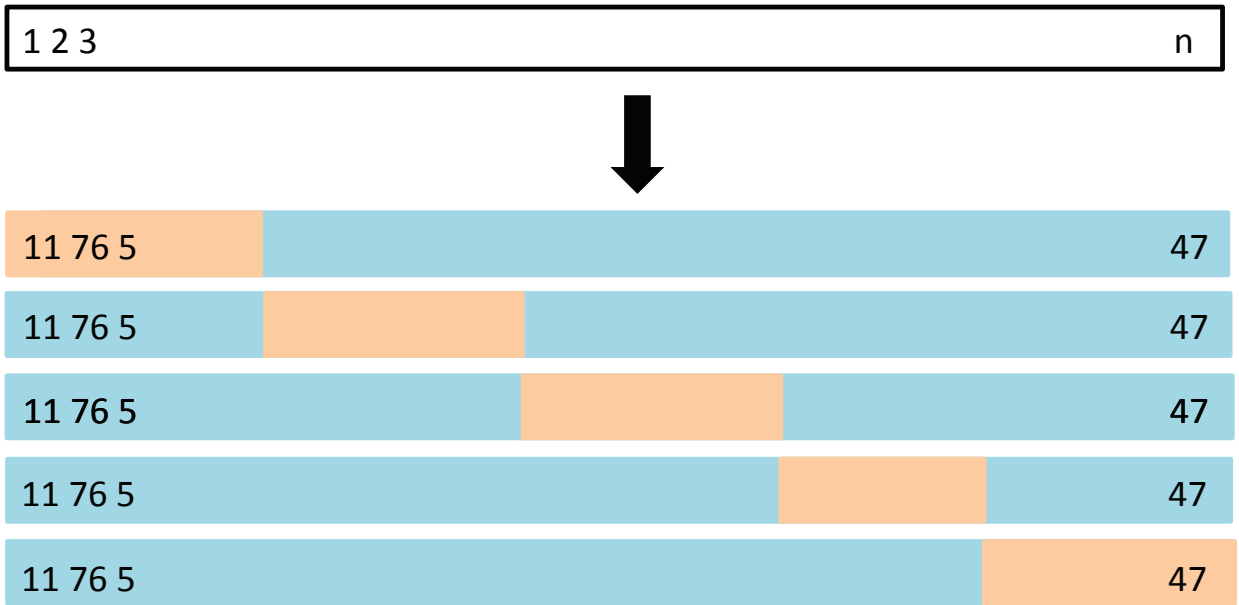
- Perform averaging over several random splits.

Problem: Only half of the data are used for learning the model. Usually more data for learning means better performance, so splitting half-half might give biased answers. Specifically for K-nn regression: K will usually be too small (see computations on bb).

- Use a larger portion of data for learning and a smaller portion for validation.

The above ideas are combined in **K-fold cross-validation**.

K-fold cross-validation



1. Randomly split $\{1, ..., n\}$ into non-overlapping subsets (also called *folds*) $S_1, ..., S_K$ of roughly equal size.
2. For k taking values $1, ..., K$: use data from all subsets except S_k to estimate f, call estimators $\widehat{f}^{(k)}$.
3. The cross-validated training error is

$$\frac{1}{n} \sum_{k=1}^{K} \sum_{j \in S_k} (\widehat{f}^{(k)}(x_j) - y_j)^2.$$

Comments on cross-validation

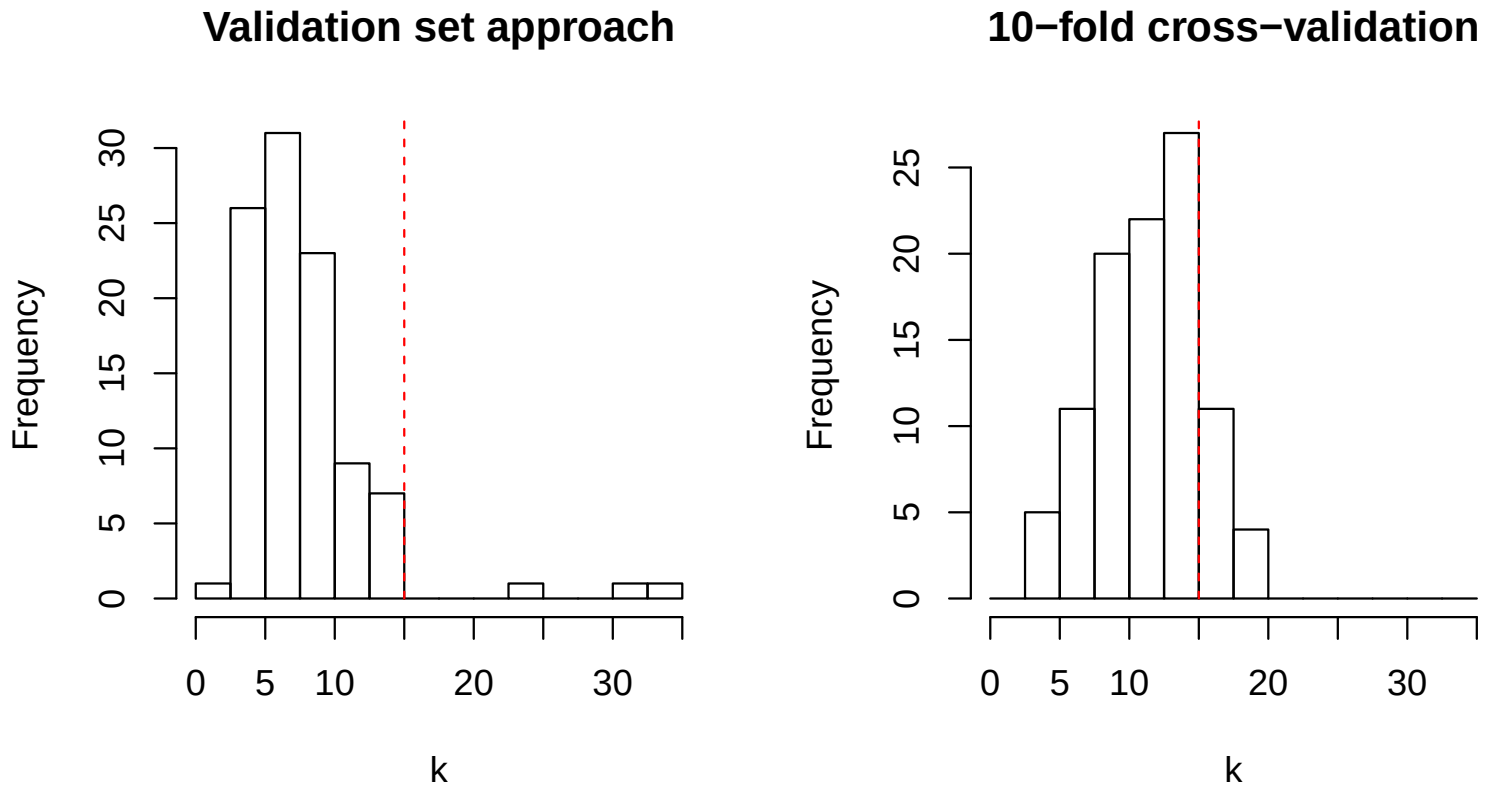
So how do we choose the number of folds? There are several aspects to consider...

- A small number of folds corresponds to more variation and smaller training sample sizes. Might result in highly variable and distorted results.
- Choosing more folds means more computation. Most extreme case: leave out one observation at a time, also known as *leave one out cross validation, LOOCV*
- LOOCV does not involve randomness, so for a single data set we get a single answer regarding the best model.
- There are reasons that go beyond the scope of this lecture for why LOOCV is not always the best choice.
- Theoretical comparisons show that LOOCV is not the best approach. We will not do theory, but we will run some computer experiments to compare the performance of different methods.

The 'rule of thumb' recommended in textbook and elsewhere: usually 5 or 10-fold cross-validation gives a good compromise.

Let's try some R code!

Comparison of validation set approach and 10-fold CV



Comparison based on same randomness. 10-fold cross-validation does better.

- MSE comparison: Validation set 4.41, 10 CV: 3.27, optimal k: 2.42.
- Advantage in selecting k carries over to MSE, but not as good as optimal k.

The one standard error rule I

- Recall: for each value of k, K-fold cross-validation returns estimated errors on each of the K folds. Call those errors $e_1(k), ..., e_K(k)$.
- Those errors $e_1(k), ..., e_K(k)$ are random variables (random splits, randomness in data...).
- To reduce noise in model selection, and to select simpler models, one can use the 'one standard error rule'.

---

1. Find k_0 which minimizes $K^{-1} \sum_{j=1}^{K} e_j(k)$ (classical cross validation would output this as solution).
2. Compute $\widehat{sd}(k_0)$, the sample standard deviation of $e_1(k_0), ..., e_K(k_0)$.
3. Select the smallest k^* among those k which satisfy

$$K^{-1} \sum_{j=1}^{K} e_j(k) \leq K^{-1} \sum_{j=1}^{K} e_j(k_0) + K^{-1/2} \widehat{sd}(k_0)$$

See picture on blackboard for more explanations.

The one standard error rule II

1. Find k_0 which minimizes $K^{-1}\sum_{j=1}^{K} e_j(k)$ (classical cross validation would output this as solution).
2. Compute $\widehat{sd}(k_0)$, the sample standard deviation of $e_1(k_0), ..., e_K(k_0)$.
3. Select the smallest k^* among those k which satisfy

$$K^{-1}\sum_{j=1}^{K} e_j(k) \leq K^{-1}\sum_{j=1}^{K} e_j(k_0) + K^{-1/2}\widehat{sd}(k_0)$$

- This idea is applicable to any problem when selecting tuning and will appear again.
- Will tend to select 'simpler' models than classical cross-validation (never selects more complicated models), i.e. larger k for k-nn. Good for building models that are more interpretable.
- Recommended in textbook and other places, but bot universally accepted. No clear formal justification.

Some of the figures in this presentation are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.