University of BRISTOL

Selecting hyper-parameters with cross validation

Henry W J Reeve

henry.reeve@bristol.ac.uk

Statistical Computing & Empirical Methods (EMATM0061)

MSc in Data Science, Teaching block 1, 2021.

What will we cover today?

- We will return to the fundamental topic of hyper-parameters and regularization.
- We will consider the concepts of overfitting, underfitting and the gap between train and test error.
- We will also think about some limitations of having a single train, validation, test split.
- Will discuss the method of k-fold cross validation to improve hyper-parameter selection.

The challenge of regression

There is an unknown distribution P over pairs (X, Y) taking values in $\mathcal{X} \times \mathbb{R}$.

We want to learn a regression model $\phi : \mathcal{X} \rightarrow \mathbb{R}$ to minimize the mean squared error,

$$\mathcal{R}_{\mathrm{MSE}}(\phi) := \mathbb{E}[(\phi(X) - Y)^2]\,.$$

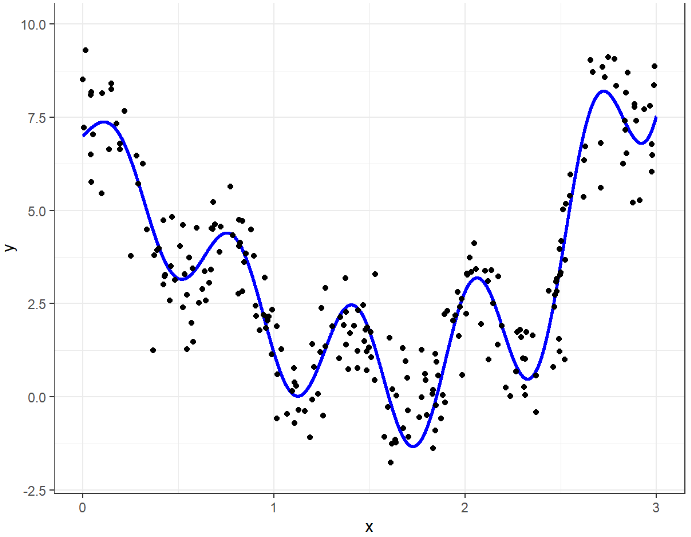
A low value of $\mathcal{R}_{\mathrm{MSE}}(\phi)$ corresponds to a good performance on unseen data.

The optimal function is $\eta(x) = \mathbb{E}(Y|X = x)$.

A low $\mathcal{R}_{\mathrm{MSE}}(\phi)$ requires $\hat{\phi}(X) \approx \eta(X)$ for typical $(X, Y) \sim \mathrm{P}$

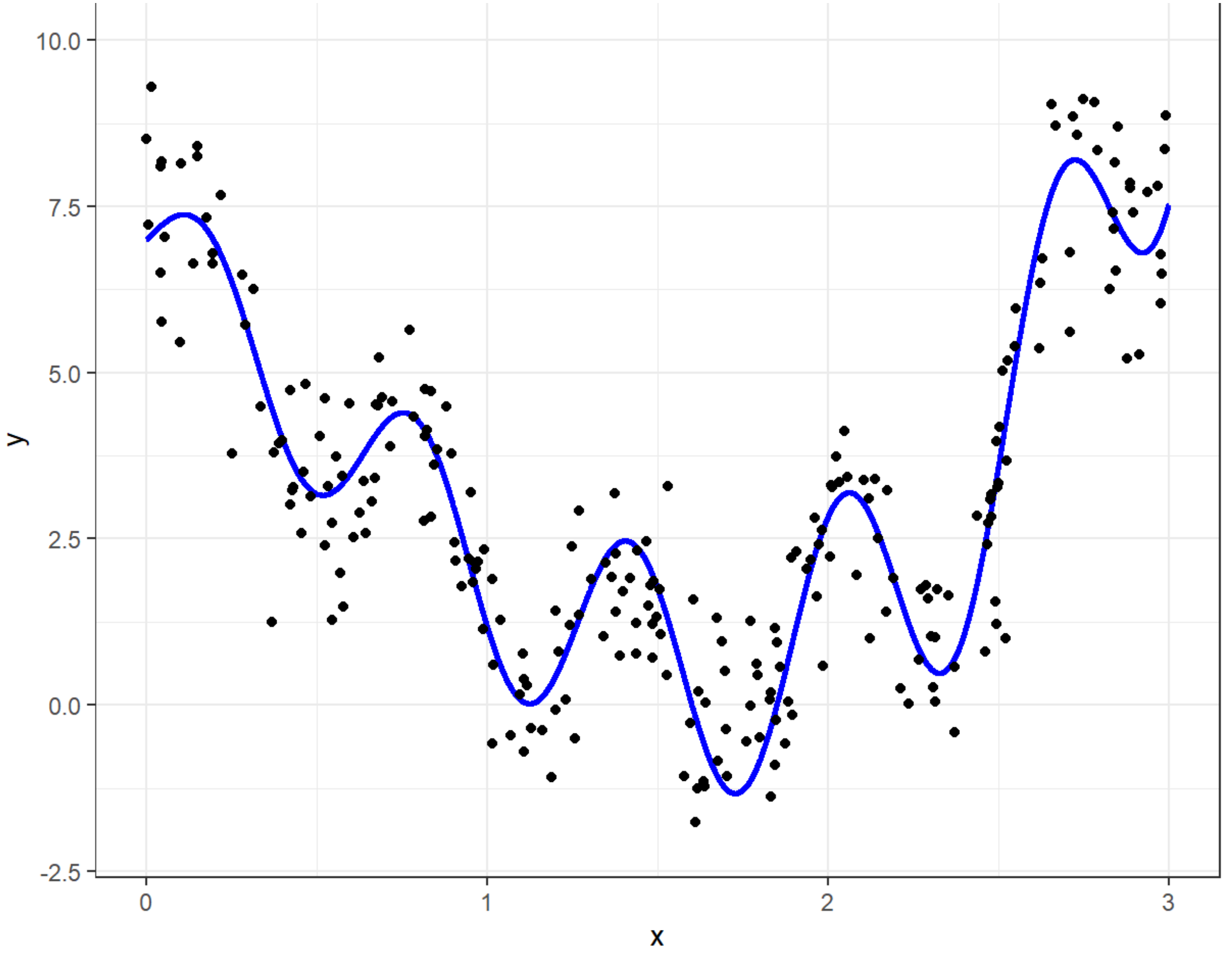
We learn a regression model $\hat{\phi} : \mathcal{X} \rightarrow \mathbb{R}$ based on data

$\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ with $(X_i, Y_i) \sim \mathrm{P}$ i.i.d.



The challenge of regression

Let’s consider the following regression problem.



● (X_i, Y_i)

● $(x, \eta(x))$

Here $\eta(x) = \mathbb{E}(Y|X = x)$ is the optimal predictive function.

The challenge of regression

There is an unknown distribution P over pairs (X, Y) taking values in $\mathcal{X} \times \mathbb{R}$.

We want to learn a regression model $\phi : \mathcal{X} \to \mathbb{R}$ to minimize the mean squared error,

$$\mathcal{R}_{\mathrm{MSE}}(\phi) := \mathbb{E}[(\phi(X) - Y)^2] \, .$$

A low value of $\mathcal{R}_{\mathrm{MSE}}(\phi)$ corresponds to a good performance on unseen data.

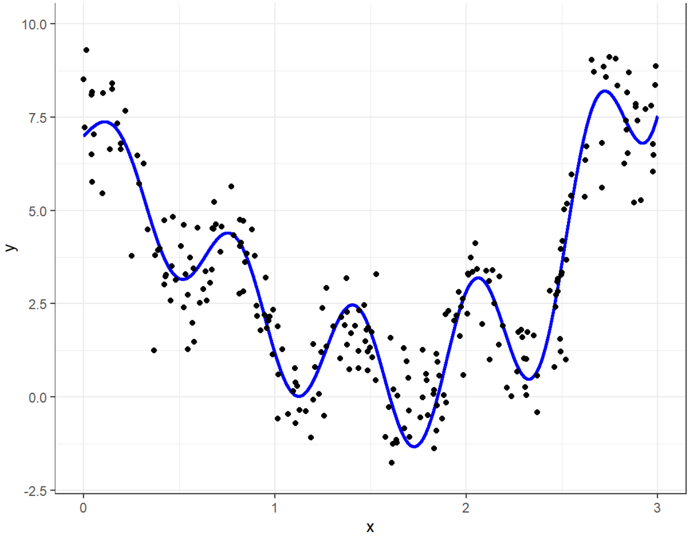
We learn a regression model $\hat{\phi} : \mathcal{X} \to \mathbb{R}$ based on data

$$\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n)) \quad \text{with} \quad (X_i, Y_i) \sim \mathrm{P} \quad \text{i.i.d.}$$

We can compute

$$\hat{\mathcal{R}}_{\mathrm{MSE}}(\phi) := \frac{1}{n} \sum_{i=1}^{n} (\phi(X_i) - Y_i)^2$$

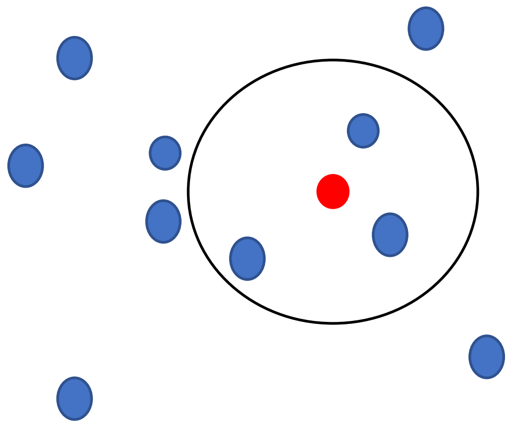
This is the training error, which is not the same as the test error.



The k-nearest neighbour regression method

The optimal solution is $\eta : \mathcal{X} \to \mathbb{R}$ defined by $\eta(x) = \mathbb{E}(Y|X = x)$.

We attempt to approximate η with the k-nearest neighbour method:



$$\hat{\phi}_k(x) = \frac{1}{k} \sum_{j=1}^{k} Y_{\tau_j(x)}$$

where $X_{\tau_1(x)}, \cdots, X_{\tau_k(x)}$

are the k –nearest neighbours of x .

The k-nearest neighbour regression method

We attempt to approximate η with the k-nearest neighbour method:

Suppose we have a data set $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ with (X_i, Y_i) in $\mathbb{R}^d \times \mathbb{R}$.

Fix $k \in \mathbb{N}$. Given a test point $x \in \mathbb{R}^d$,

We take a permutation (a reordering) $\tau_1(x), \cdots, \tau_n(x)$ of the set $\{1, \cdots, n\}$ with

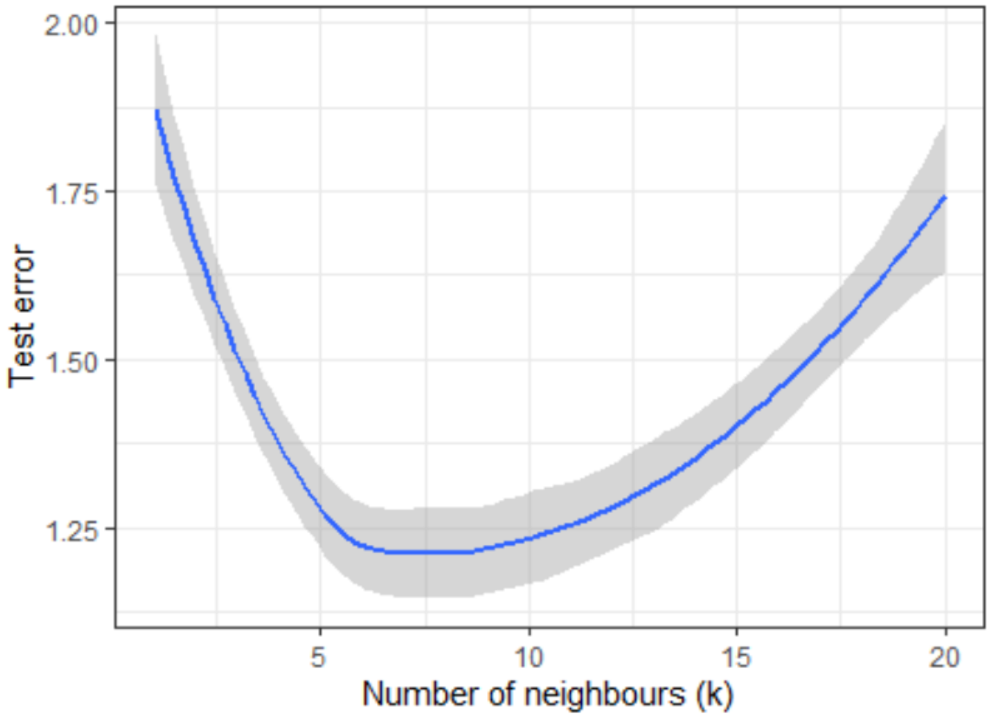
$$\|x - X_{\tau_1(x)}\|_2 \leq \|x - X_{\tau_1(x)}\|_2 \leq \cdots \leq \|x - X_{\tau_n(x)}\|_2.$$

Set $\hat{\phi}_k(x) = \frac{1}{k}\sum_{j=1}^{k} Y_{\tau_j(x)}$ where $X_{\tau_1(x)}, \cdots, X_{\tau_k(x)}$ are the k-nearest neighbours.

The output $\hat{\phi}_k(x)$ is the average over the k-nearest neighbours.

The k nearest neighbor method

Let’s look at the test error as a function of the number of neighbours k for a simulated example.



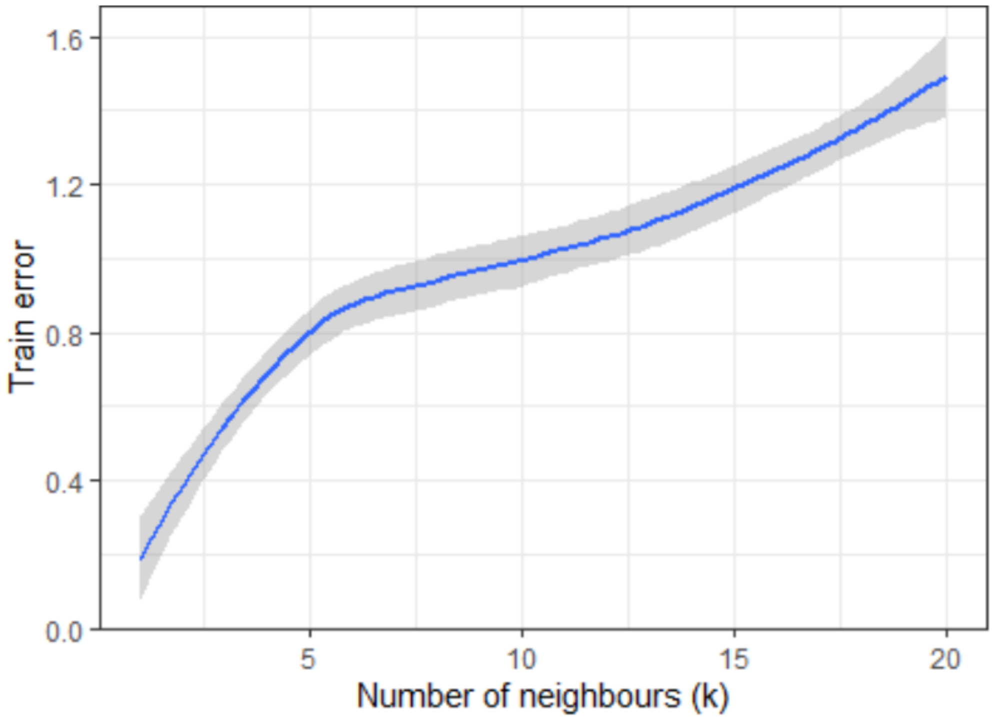
The test error is given by

$$\mathcal{R}_{\text{MSE}}(\phi) := \mathbb{E}[(\phi(X) - Y)^2]$$

Why do we observe this pattern?

The k nearest neighbor method

Let’s look at the train error as a function of the number of neighbours k for a simulated example.



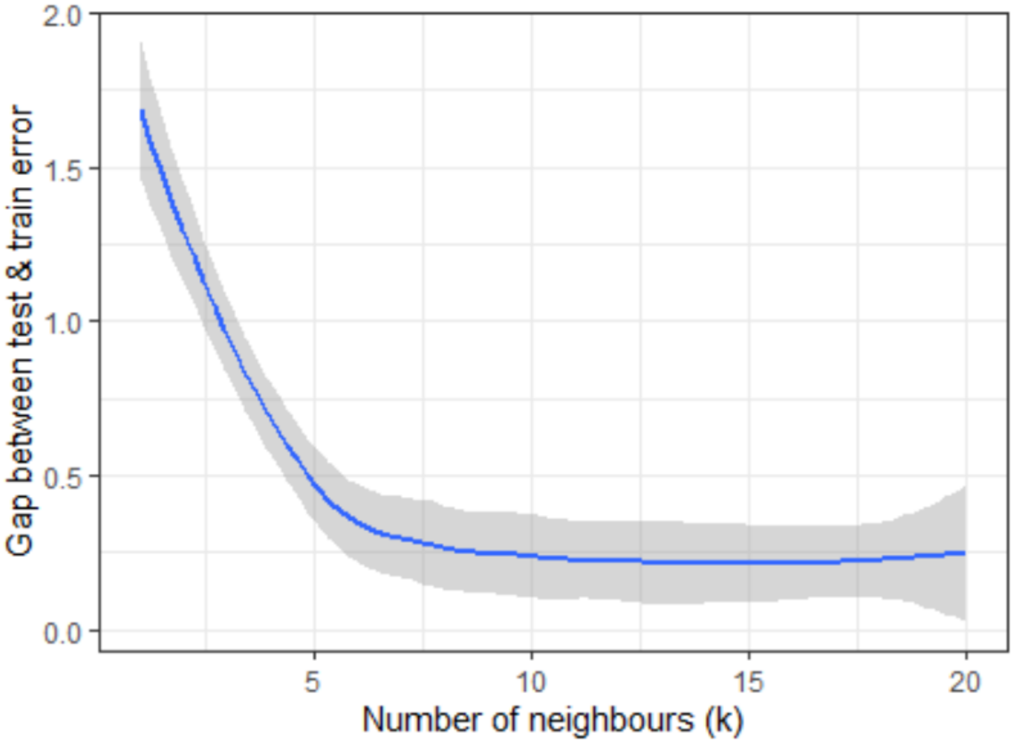
The train error is given by

$$\hat{\mathcal{R}}_{\mathrm{MSE}}(\phi) := \frac{1}{n}\sum_{i=1}^{n}\left(\phi(X_i) - Y_i\right)^2$$

Training error increases monotonically as we increase the number of neighbours k.

The k nearest neighbor method

Let’s look at the gap between test and train error as a function of the number of neighbours k



The gap between test and train error is

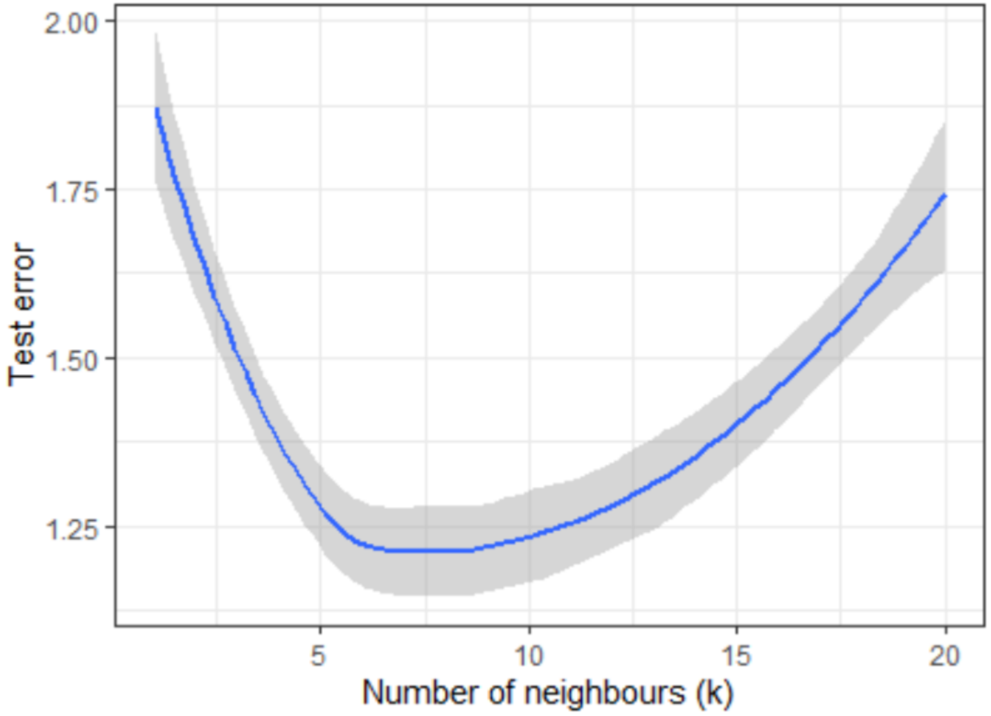
$$\Delta_{\mathrm{MSE}}(\phi) = \mathcal{R}_{\mathrm{MSE}}(\phi) - \hat{\mathcal{R}}_{\mathrm{MSE}}(\phi)$$

$$= \mathbb{E}\left[(\phi(X) - Y)^2\right] - \frac{1}{n}\sum_{i=1}^{n}(\phi(X_i) - Y_i)^2 .$$

The gap between test and train error falls monotonically as we increase k before flattening out.

The k nearest neighbor method

The test error may be viewed as a sum of training error plus the gap between test and train error.



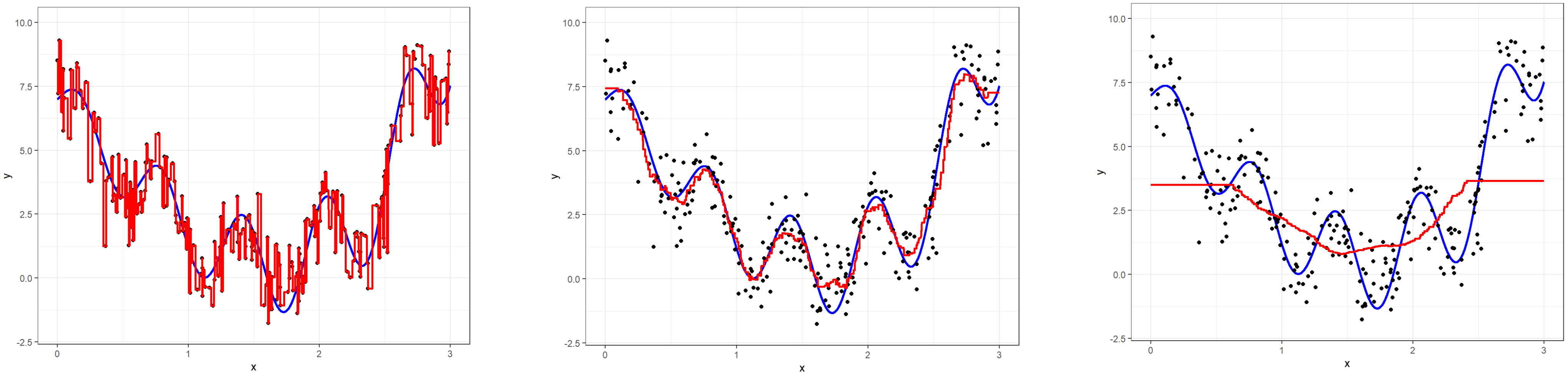
The test error is

$$\underset{\text{Test error}}{\underset{\uparrow}{\mathcal{R}_{\mathrm{MSE}}(\phi)}} = \underset{\text{Train error}}{\underset{\uparrow}{\hat{\mathcal{R}}_{\mathrm{MSE}}(\phi)}} + \underset{\text{Error gap}}{\underset{\uparrow}{\Delta_{\mathrm{MSE}}(\phi)}}$$

Combining train error and the test train gap leads to a convex shape for the test error curve.

The k nearest neighbor method

The performance of the k -nearest neighbour method depends critically upon k .



When k is very small the gap between test and train is very large – **Overfitting**.

When k is too large the train and test errors are both very large – **Underfitting**.

How should we select our hyper-parameters?

Hyper-parameters

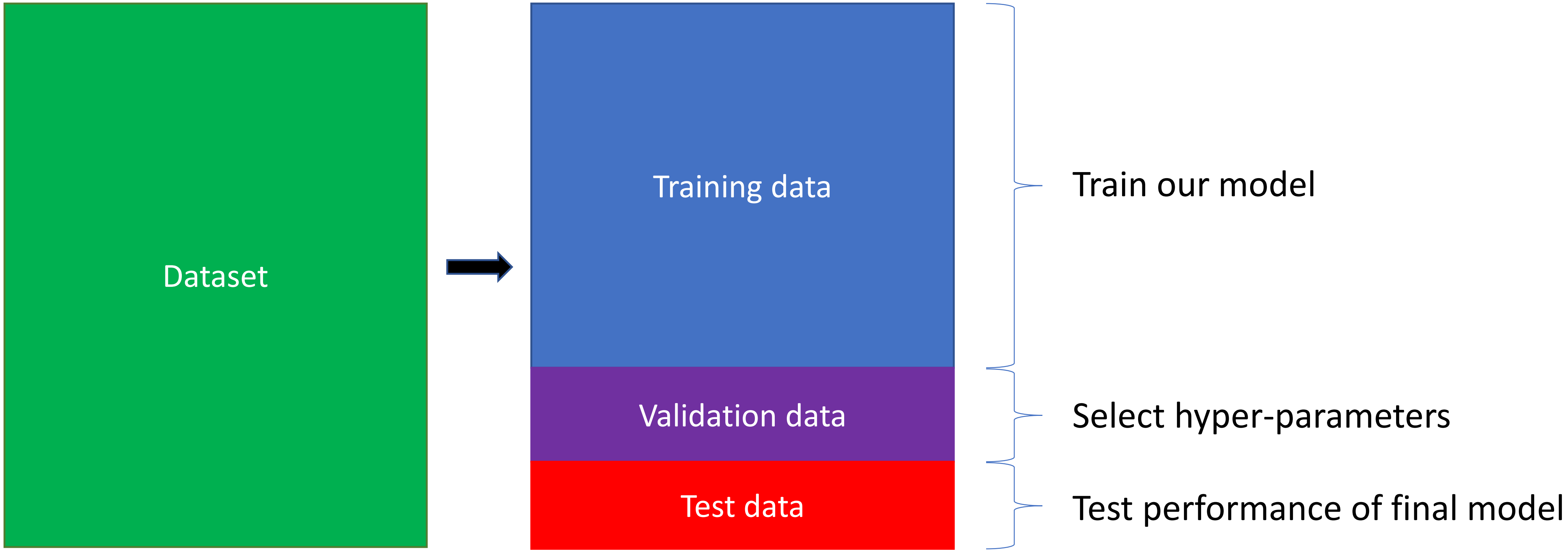
Almost every widely used Machine Learning method has one or more hyper-parameters:

- Linear regression models: ℓ_1 or ℓ_2 norm penalization.
- Logistic regression models: ℓ_1 or ℓ_2 norm penalization.
- Neural networks: ℓ_1 or ℓ_2 norm penalization, learning rate, dropout, …..
- Random forests: Number of leaf nodes, maximum depth of trees, features per split.
- Gradient boosting: Number of trees, maximum depth of trees, examples per round…..
- Support vector machines: Choice of kernel, hyper-parameters of kernel, ℓ_2 regularization.

How should we select our hyper-parameters?

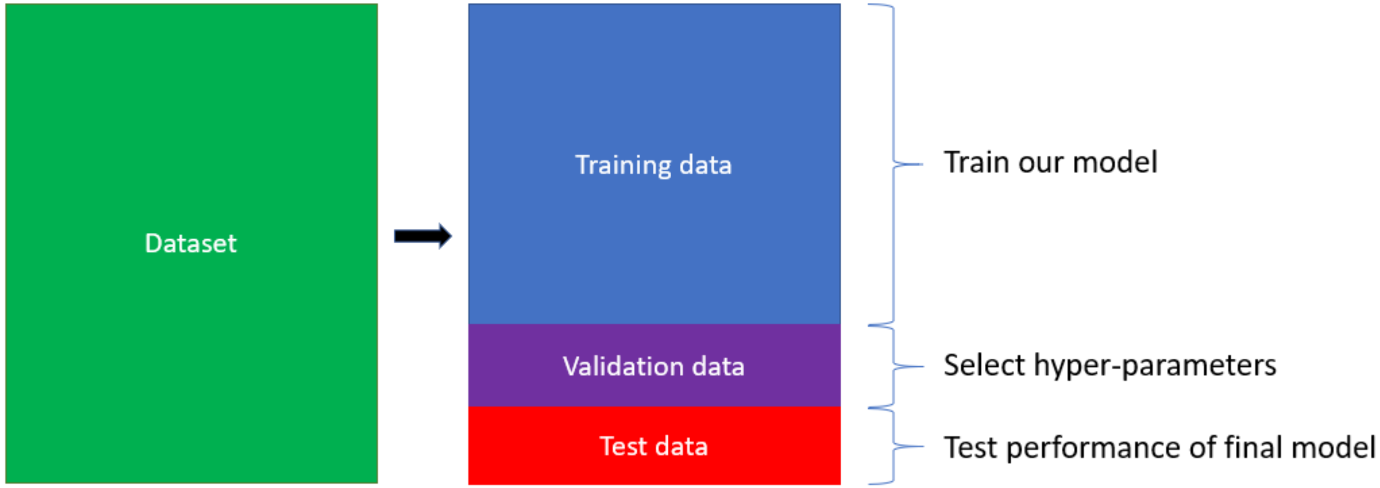
The train-validation-test split

One approach to selecting hyper-parameters is based upon a single train-validation split.



The train-validation-test split

One approach to selecting hyper-parameters is based upon a single validation split.



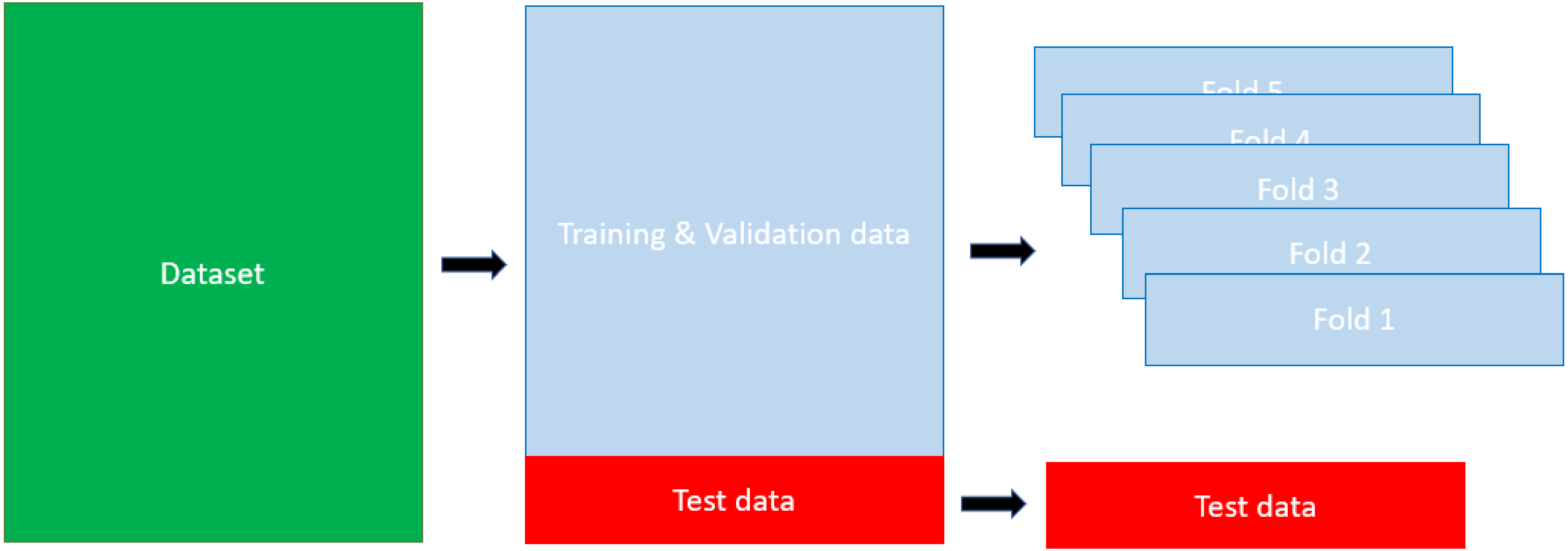
A single train-validation-test split works well for relatively large data sets.

For smaller data sets we have a small validation set, and performance depends crucially upon the split.

This can lead to instabilities and a relatively poor selection of hyper-parameters.

The cross-validation approach

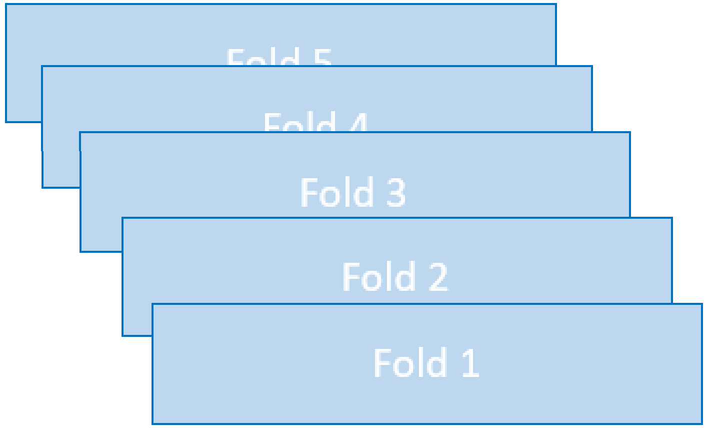
An alternative approach to selecting hyper-parameters is cross-validation.



The cross-validation approach

Suppose we have a data split into several folds and potential hyper-parameters $\{\lambda_1, \cdots, \lambda_Q\}$.

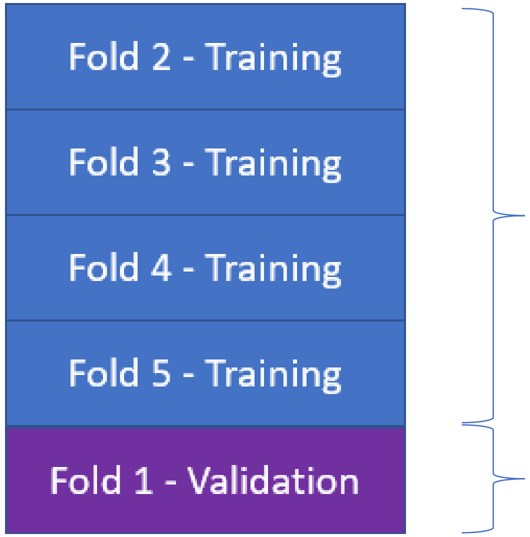
For $q = 1, \cdots, Q$ we use all of the folds to estimate the performance of the model with λ_q.



The cross-validation approach

Suppose we have a data split into several folds and potential hyper-parameters $\{\lambda_1, \cdots, \lambda_Q\}$.

For $q = 1, \cdots, Q$ we use all of the folds to estimate the performance of the model with λ_q .



Fold 2 - Training

Fold 3 - Training

Fold 4 - Training

Fold 5 - Training

Fold 1 - Validation

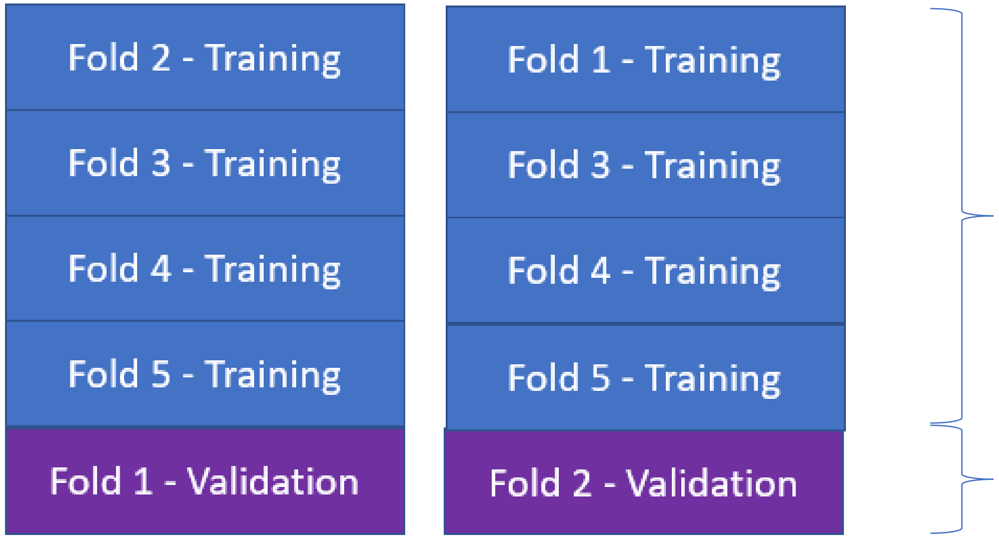
Train the model $\hat{\phi}^{(1)}_{\lambda_q}$ with hyper-parameter λ_q .

Compute the validation error $V_1(\lambda_q)$ of $\hat{\phi}^{(1)}_{\lambda_q}$ on the first fold.

The cross-validation approach

Suppose we have a data split into several folds and potential hyper-parameters $\{\lambda_1, \cdots, \lambda_Q\}$.

For $q = 1, \cdots, Q$ we use all of the folds to estimate the performance of the model with λ_q.



Fold 2 - Training
Fold 3 - Training
Fold 4 - Training
Fold 5 - Training
Fold 1 - Validation

Fold 1 - Training
Fold 3 - Training
Fold 4 - Training
Fold 5 - Training
Fold 2 - Validation

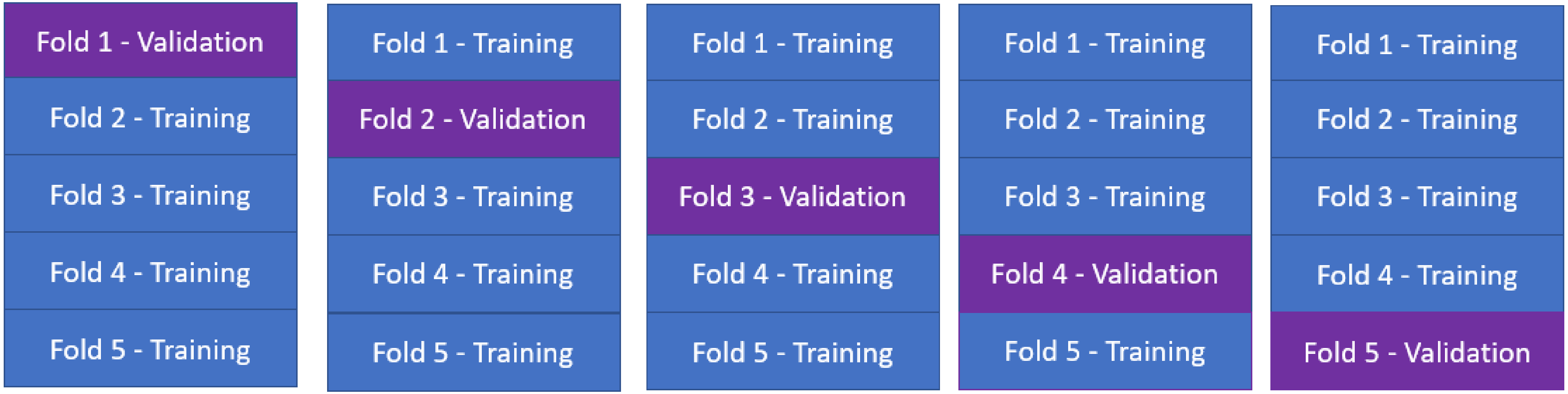
Train the model $\hat{\phi}^{(2)}_{\lambda_q}$ with hyper-parameter λ_q.

Compute the error $V_2(\lambda_q)$ of $\hat{\phi}^{(2)}_{\lambda_q}$ on the second fold.

The cross-validation approach

Suppose we have a data split into several folds and potential hyper-parameters $\{\lambda_1, \cdots, \lambda_Q\}$.

For $q = 1, \cdots, Q$ we use all of the folds to estimate the performance of the model with λ_q .



Fold 1 - Validation	Fold 1 - Training	Fold 1 - Training	Fold 1 - Training	Fold 1 - Training
Fold 2 - Training	Fold 2 - Validation	Fold 2 - Training	Fold 2 - Training	Fold 2 - Training
Fold 3 - Training	Fold 3 - Training	Fold 3 - Validation	Fold 3 - Training	Fold 3 - Training
Fold 4 - Training	Fold 4 - Training	Fold 4 - Training	Fold 4 - Validation	Fold 4 - Training
Fold 5 - Training	Fold 5 - Training	Fold 5 - Training	Fold 5 - Training	Fold 5 - Validation

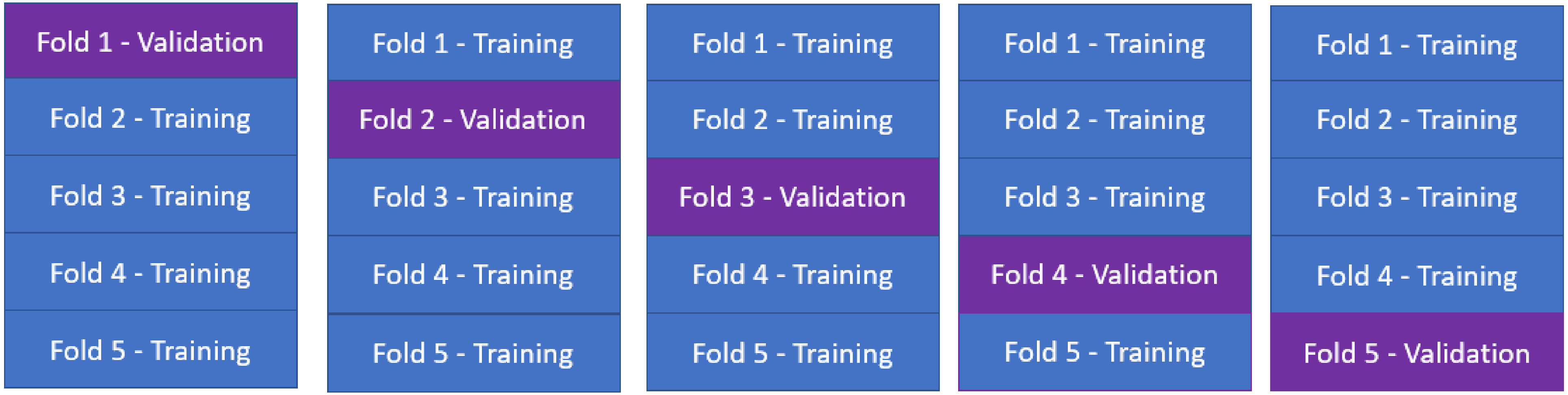
We compute the average of these validation errors over the different folds.

We select the hyper-parameter $\hat{\lambda} \in \{\lambda_1, \cdots, \lambda_Q\}$ with the lowest average validation error.

The cross-validation approach

Suppose we have a data split into several folds and potential hyper-parameters $\{\lambda_1, \cdots, \lambda_Q\}$.

For $q = 1, \cdots, Q$ we use all of the folds to estimate the performance of the model with λ_q.



Fold 1 - Validation	Fold 1 - Training	Fold 1 - Training	Fold 1 - Training	Fold 1 - Training
Fold 2 - Training	Fold 2 - Validation	Fold 2 - Training	Fold 2 - Training	Fold 2 - Training
Fold 3 - Training	Fold 3 - Training	Fold 3 - Validation	Fold 3 - Training	Fold 3 - Training
Fold 4 - Training	Fold 4 - Training	Fold 4 - Training	Fold 4 - Validation	Fold 4 - Training
Fold 5 - Training	Fold 5 - Training	Fold 5 - Training	Fold 5 - Training	Fold 5 - Validation

Note: This procedure is often referred to as “k-fold cross validation”.

Here “k” refers to the number of folds (in this case 5) – not the number of neighbours!

I will use J here for the number of folds to avoid confusion.

The cross-validation approach

Suppose we have a data split into J folds and potential hyper-parameters $\{\lambda_1, \cdots, \lambda_Q\}$.

```
For $q = 1, \cdots, Q$,
    For $j = 1, \cdots, J$ ,
        Train the model $\hat{\phi}^{(j)}_{\lambda_q}$ with hyper-parameter $\lambda_q$ using all folds except for $j$ .
        Compute the validation error $V_j(\lambda_q)$ of the model $\hat{\phi}^{(j)}_{\lambda_q}$ on the $j$ -th fold.
    Compute the average validation error $\overline{V}(\lambda_q) = \frac{1}{J}\left(V_1(\lambda_q) + \cdots + V_J(\lambda_q)\right)$ .
Select the hyper-parameter $\hat{\lambda} \in \{\lambda_1, \cdots, \lambda_Q\}$ to minimize $\overline{V}(\hat{\lambda})$ .
```

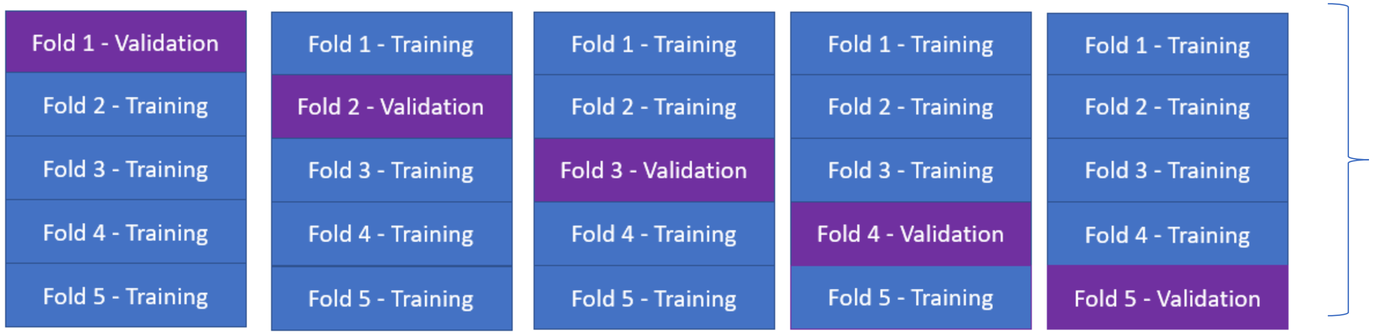
The cross-validation approach

Suppose we have a data split into J folds and potential hyper-parameters $\{\lambda_1, \cdots, \lambda_Q\}$.

Here there are J folds

```
For $q = 1, \cdots, Q$,
    For $j = 1, \cdots, J$,
        Train the model $\hat{\phi}^{(j)}_{\lambda_q}$ with hyper-parameter $\lambda_q$ using all folds except for $j$ .
        Compute the validation error $V_j(\lambda_q)$ of the model $\hat{\phi}^{(j)}_{\lambda_q}$ on the $j$ -th fold.
    Compute the average validation error $\overline{V}(\lambda_q) = \frac{1}{J}\left(V_1(\lambda_q) + \cdots + V_J(\lambda_q)\right)$.
Select the hyper-parameter $\hat{\lambda} \in \{\lambda_1, \cdots, \lambda_Q\}$ to minimize $\overline{V}(\hat{\lambda})$ .
```

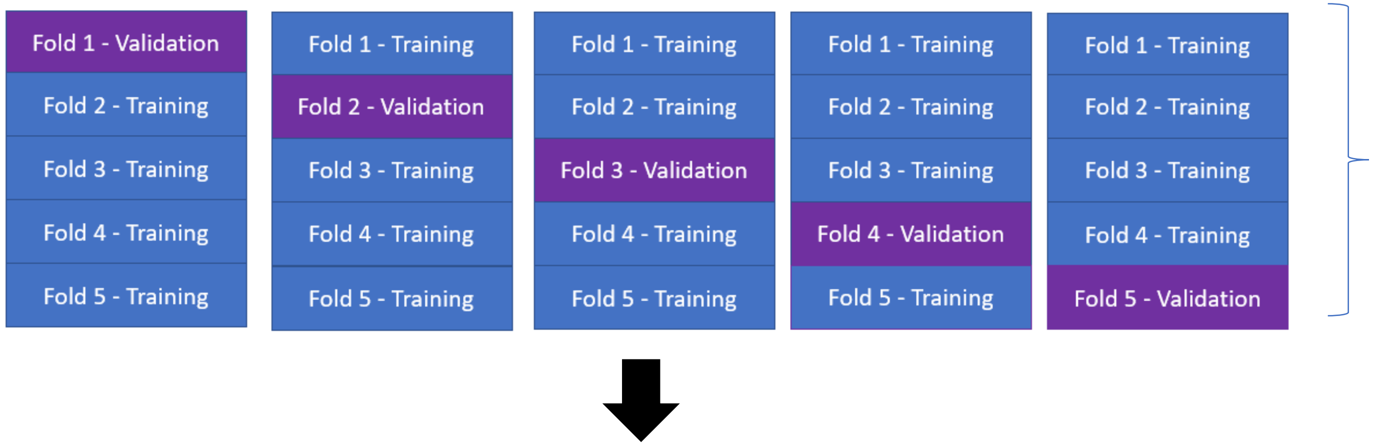
The cross-validation approach



Fold 1 - Validation	Fold 1 - Training	Fold 1 - Training	Fold 1 - Training	Fold 1 - Training
Fold 2 - Training	Fold 2 - Validation	Fold 2 - Training	Fold 2 - Training	Fold 2 - Training
Fold 3 - Training	Fold 3 - Training	Fold 3 - Validation	Fold 3 - Training	Fold 3 - Training
Fold 4 - Training	Fold 4 - Training	Fold 4 - Training	Fold 4 - Validation	Fold 4 - Training
Fold 5 - Training	Fold 5 - Training	Fold 5 - Training	Fold 5 - Training	Fold 5 - Validation

Select our hyper-parameters to minimize the average validation error.

The cross-validation approach



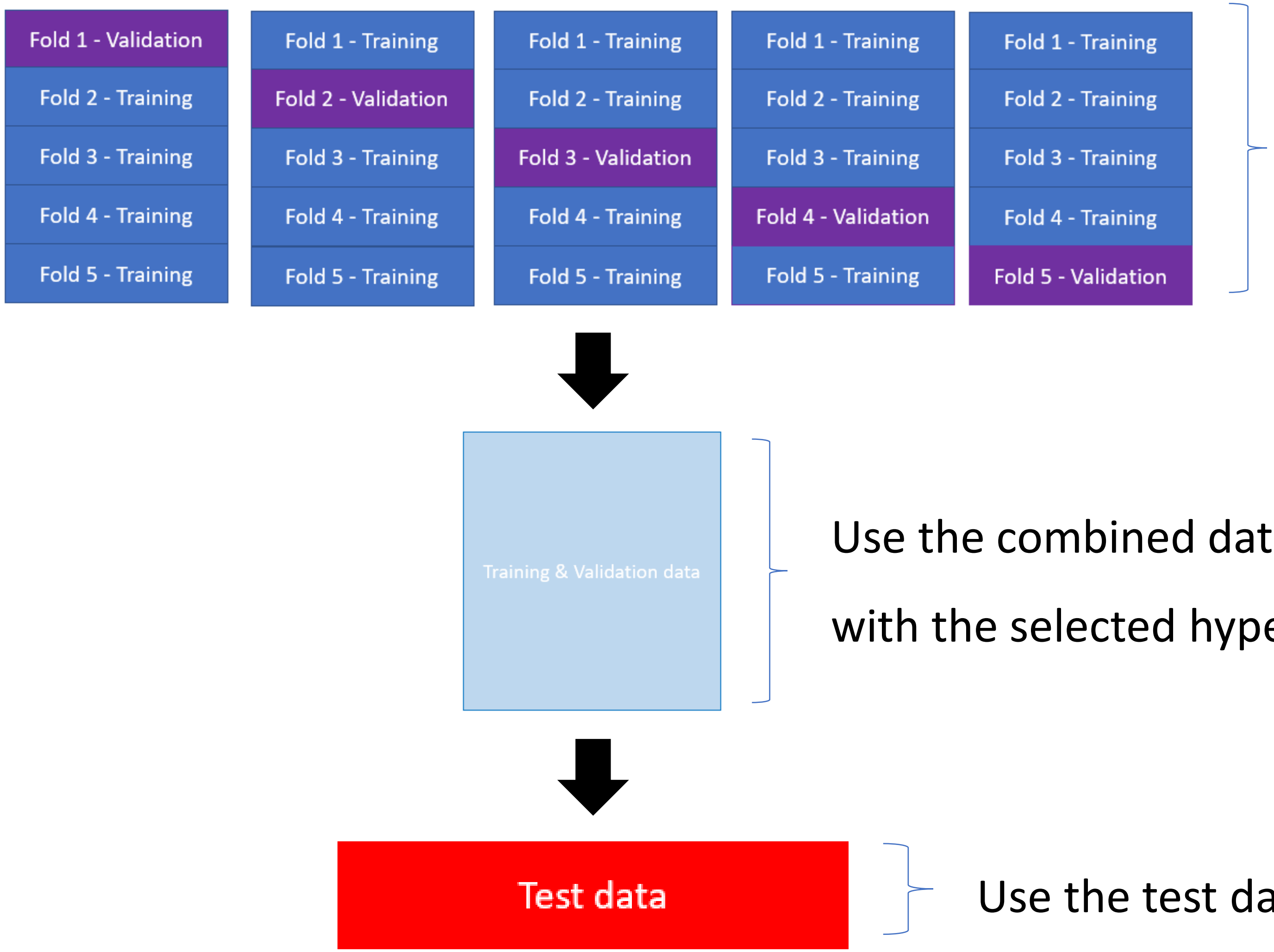
Fold 1 - Validation	Fold 1 - Training	Fold 1 - Training	Fold 1 - Training	Fold 1 - Training
Fold 2 - Training	Fold 2 - Validation	Fold 2 - Training	Fold 2 - Training	Fold 2 - Training
Fold 3 - Training	Fold 3 - Training	Fold 3 - Validation	Fold 3 - Training	Fold 3 - Training
Fold 4 - Training	Fold 4 - Training	Fold 4 - Training	Fold 4 - Validation	Fold 4 - Training
Fold 5 - Training	Fold 5 - Training	Fold 5 - Training	Fold 5 - Training	Fold 5 - Validation

Select our hyper-parameters to minimize the average validation error.

Training & Validation data

Use the combined data to retrain the model with the selected hyper-parameter.

The cross-validation approach



Fold 1 - Validation	Fold 1 - Training	Fold 1 - Training	Fold 1 - Training	Fold 1 - Training
Fold 2 - Training	Fold 2 - Validation	Fold 2 - Training	Fold 2 - Training	Fold 2 - Training
Fold 3 - Training	Fold 3 - Training	Fold 3 - Validation	Fold 3 - Training	Fold 3 - Training
Fold 4 - Training	Fold 4 - Training	Fold 4 - Training	Fold 4 - Validation	Fold 4 - Training
Fold 5 - Training	Fold 5 - Training	Fold 5 - Training	Fold 5 - Training	Fold 5 - Validation

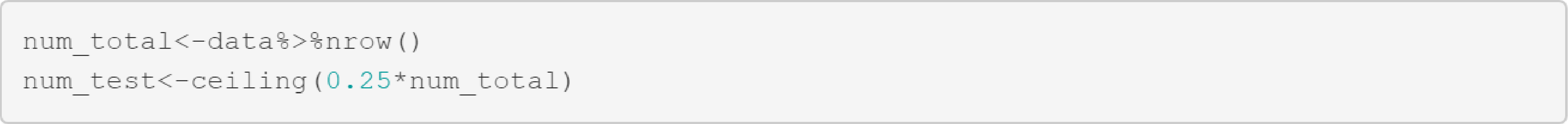
Select our hyper-parameters to minimize the average validation error.

Use the combined data to retrain the model with the selected hyper-parameter.

Use the test data to test the final model.

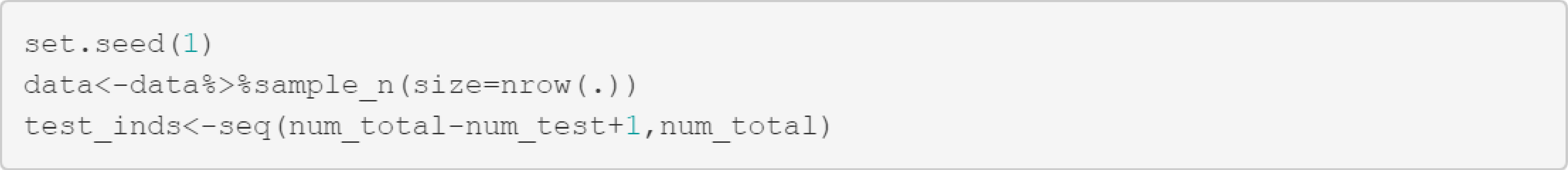
The cross-validation approach in R

Suppose we have a data frame “data”. We take 25% of our data as test data.



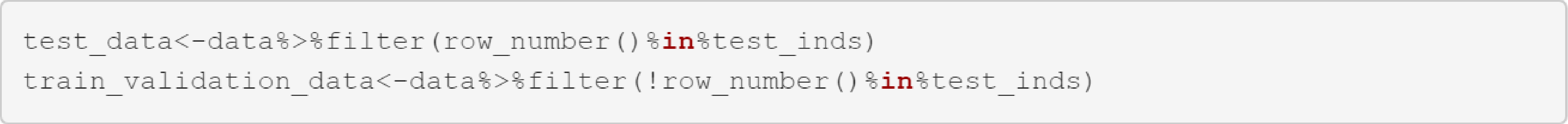
```
num_total<-data%>%nrow()
num_test<-ceiling(0.25*num_total)
```

Randomly shuffle and select a subset of the data for testing.



```
set.seed(1)
data<-data%>%sample_n(size=nrow(.))
test_inds<-seq(num_total-num_test+1,num_total)
```

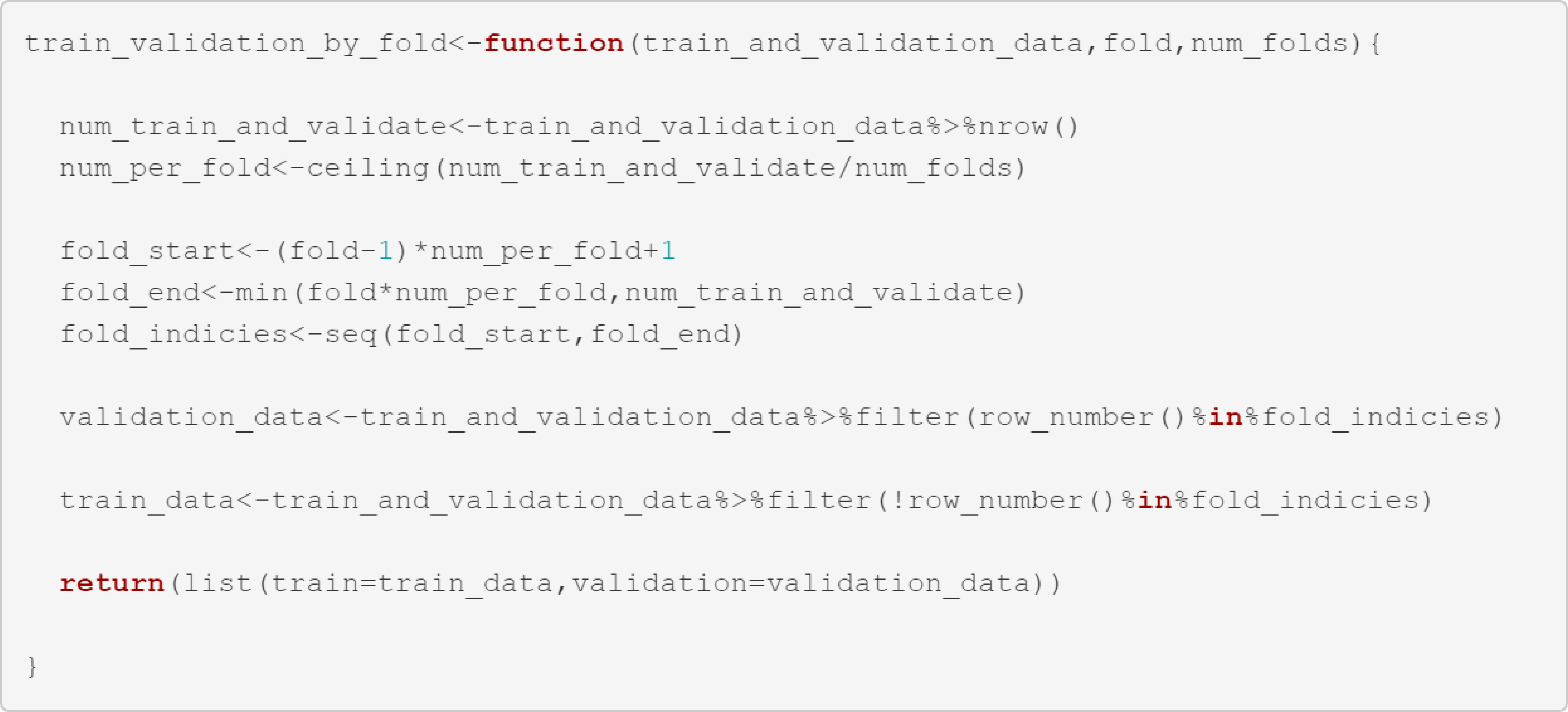
Split the data into a test sample and a train/validation sample.



```
test_data<-data%>%filter(row_number()%in%test_inds)
train_validation_data<-data%>%filter(!row_number()%in%test_inds)
```

The cross-validation approach in R

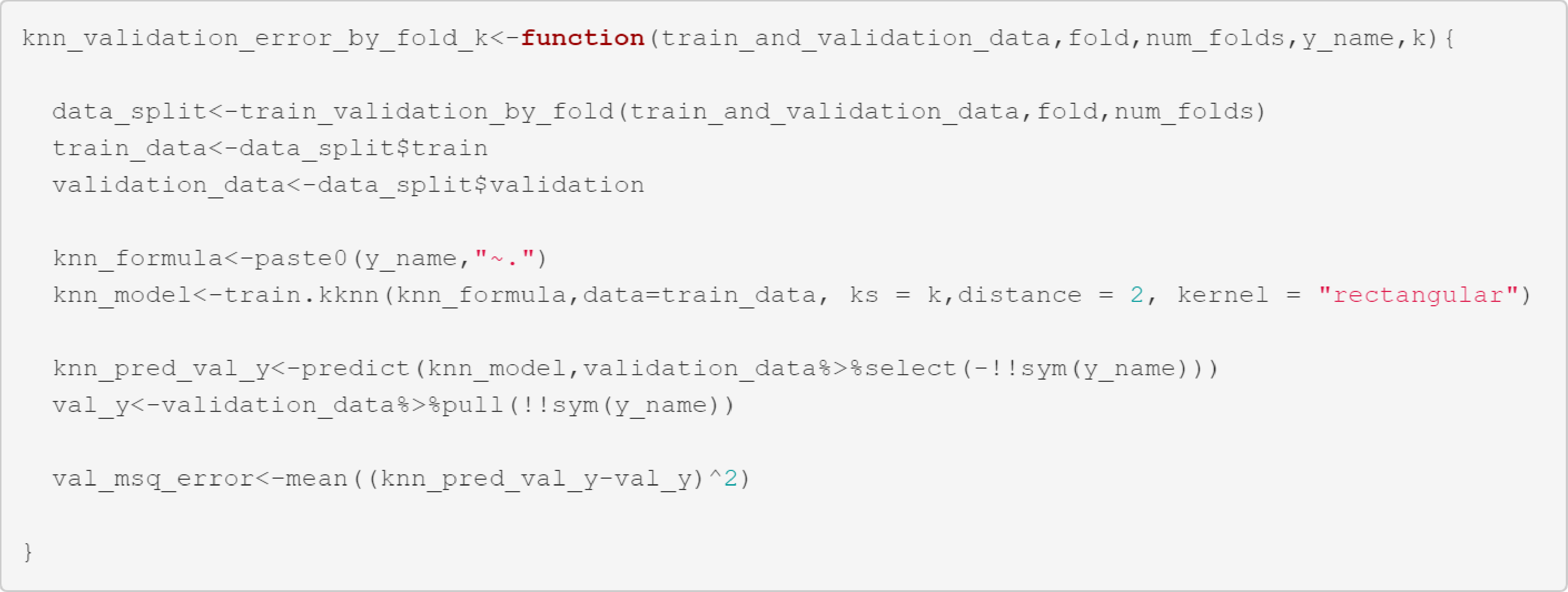
Let’s generate a function extract split the train/validation data into train and validation by fold.



```r
train_validation_by_fold<-function(train_and_validation_data,fold,num_folds){

  num_train_and_validate<-train_and_validation_data%>%nrow()
  num_per_fold<-ceiling(num_train_and_validate/num_folds)

  fold_start<-(fold-1)*num_per_fold+1
  fold_end<-min(fold*num_per_fold,num_train_and_validate)
  fold_indicies<-seq(fold_start,fold_end)

  validation_data<-train_and_validation_data%>%filter(row_number()%in%fold_indicies)

  train_data<-train_and_validation_data%>%filter(!row_number()%in%fold_indicies)

  return(list(train=train_data,validation=validation_data))

}
```

The cross-validation approach in R

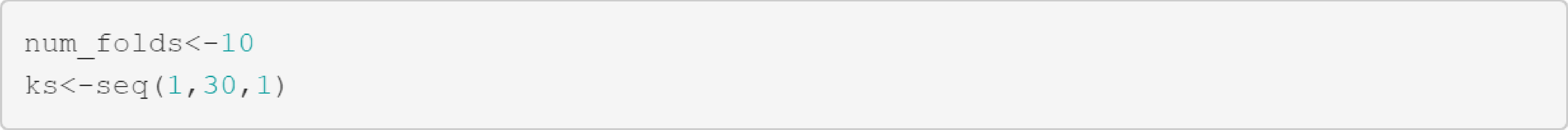
Next a function to estimate validation error by fold and number of neighbours k.



```r
knn_validation_error_by_fold_k<-function(train_and_validation_data,fold,num_folds,y_name,k){

  data_split<-train_validation_by_fold(train_and_validation_data,fold,num_folds)
  train_data<-data_split$train
  validation_data<-data_split$validation

  knn_formula<-paste0(y_name,"~.")
  knn_model<-train.kknn(knn_formula,data=train_data, ks = k,distance = 2, kernel = "rectangular")

  knn_pred_val_y<-predict(knn_model,validation_data%>%select(-!!sym(y_name)))
  val_y<-validation_data%>%pull(!!sym(y_name))

  val_msq_error<-mean((knn_pred_val_y-val_y)^2)

}
```

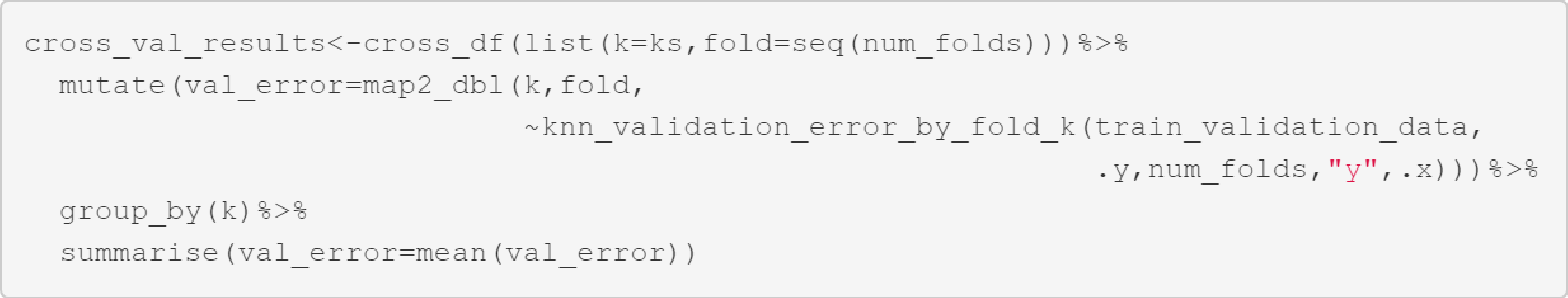
The cross-validation approach in R

We specify a number of folds and a selection of possible numbers of neighbours.



```r
num_folds<-10
ks<-seq(1,30,1)
```

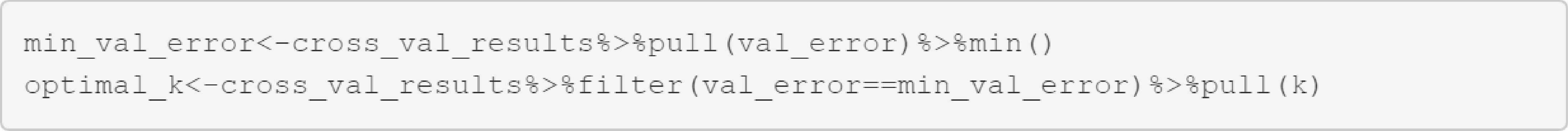
Compute the validation error for each possible choice of hyper-parameter and each fold.



```r
cross_val_results<-cross_df(list(k=ks,fold=seq(num_folds)))%>%
  mutate(val_error=map2_dbl(k,fold,
                            ~knn_validation_error_by_fold_k(train_validation_data,
                                                            .y,num_folds,"y",.x)))%>%
  group_by(k)%>%
  summarise(val_error=mean(val_error))
```

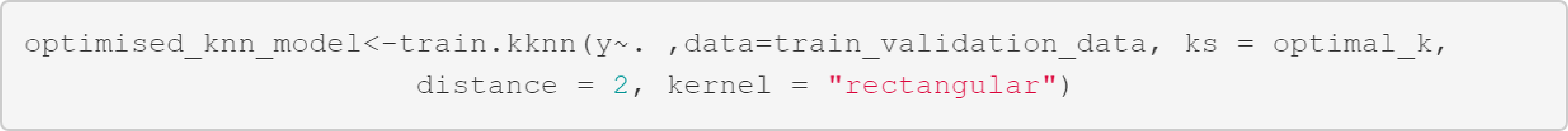
The cross-validation approach in R

Find the hyper-parameter which minimizes the validation error.



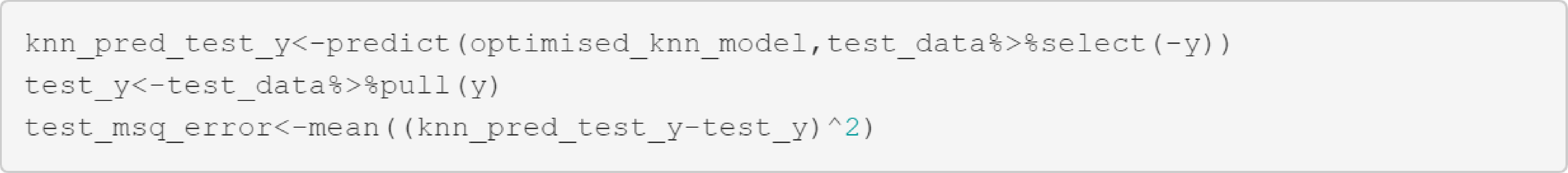
```
min_val_error<-cross_val_results%>%pull(val_error)%>%min()
optimal_k<-cross_val_results%>%filter(val_error==min_val_error)%>%pull(k)
```

Retrain the model with the optimal hyper-parameter using the combined train & validation data.



```
optimised_knn_model<-train.kknn(y~. ,data=train_validation_data, ks = optimal_k,
                      distance = 2, kernel = "rectangular")
```

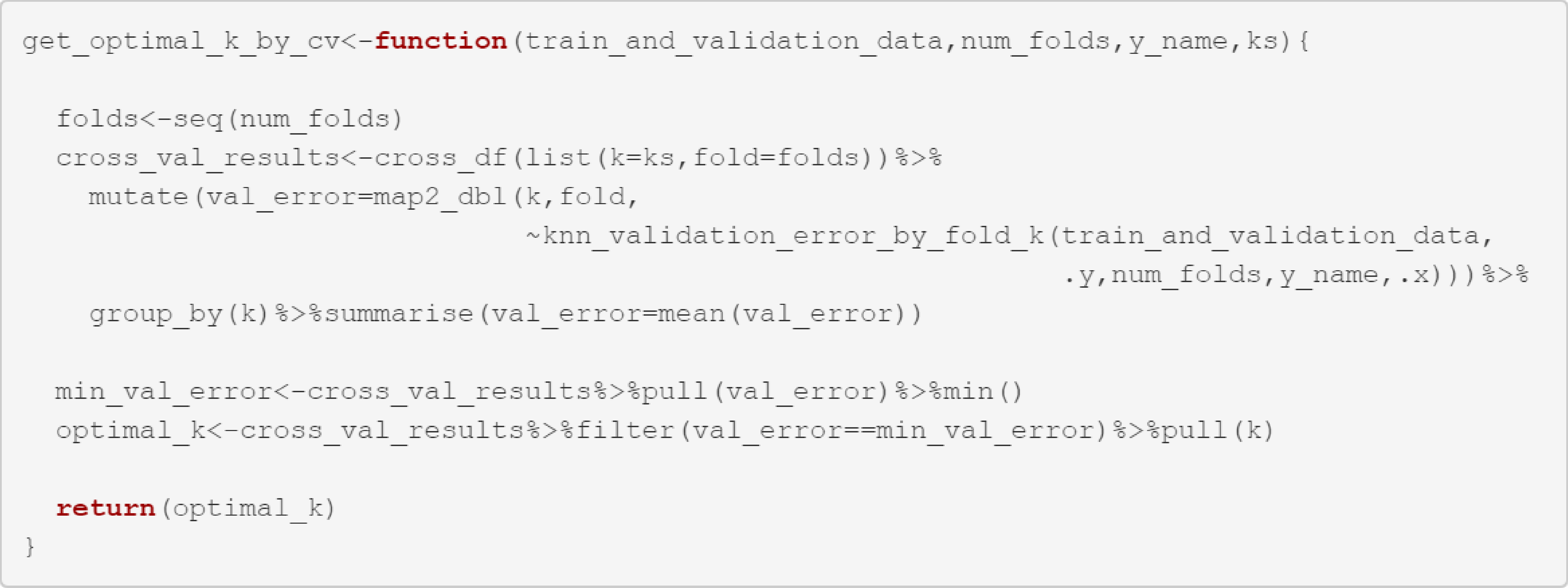
We can now make predictions and compute the test error.



```
knn_pred_test_y<-predict(optimised_knn_model,test_data%>%select(-y))
test_y<-test_data%>%pull(y)
test_msq_error<-mean((knn_pred_test_y-test_y)^2)
```

The cross-validation approach in R

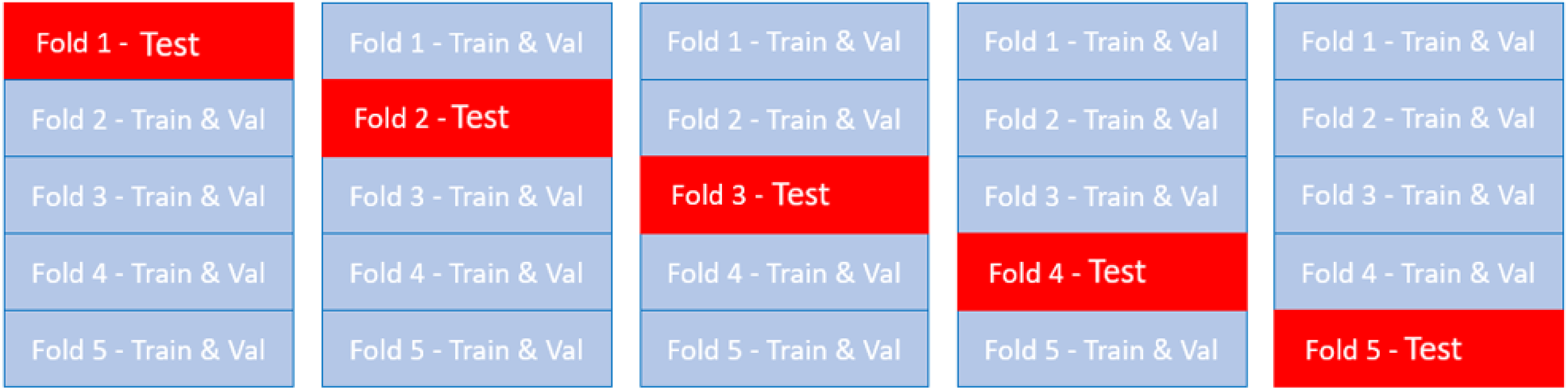
We can combine the process of selecting the optimal hyper-parameter by CV within a single function.



```r
get_optimal_k_by_cv<-function(train_and_validation_data,num_folds,y_name,ks){

  folds<-seq(num_folds)
  cross_val_results<-cross_df(list(k=ks,fold=folds))%>%
    mutate(val_error=map2_dbl(k,fold,
                              ~knn_validation_error_by_fold_k(train_and_validation_data,
                                                              .y,num_folds,y_name,.x)))%>%
    group_by(k)%>%summarise(val_error=mean(val_error))

  min_val_error<-cross_val_results%>%pull(val_error)%>%min()
  optimal_k<-cross_val_results%>%filter(val_error==min_val_error)%>%pull(k)

  return(optimal_k)
}
```

The cross-validation approach in R

We can also use cross-validation to get a better understanding of performance on unseen data.



Fold 1 - Test	Fold 1 - Train & Val	Fold 1 - Train & Val	Fold 1 - Train & Val	Fold 1 - Train & Val
Fold 2 - Train & Val	Fold 2 - Test	Fold 2 - Train & Val	Fold 2 - Train & Val	Fold 2 - Train & Val
Fold 3 - Train & Val	Fold 3 - Train & Val	Fold 3 - Test	Fold 3 - Train & Val	Fold 3 - Train & Val
Fold 4 - Train & Val	Fold 4 - Train & Val	Fold 4 - Train & Val	Fold 4 - Test	Fold 4 - Train & Val
Fold 5 - Train & Val	Fold 5 - Train & Val	Fold 5 - Train & Val	Fold 5 - Train & Val	Fold 5 - Test

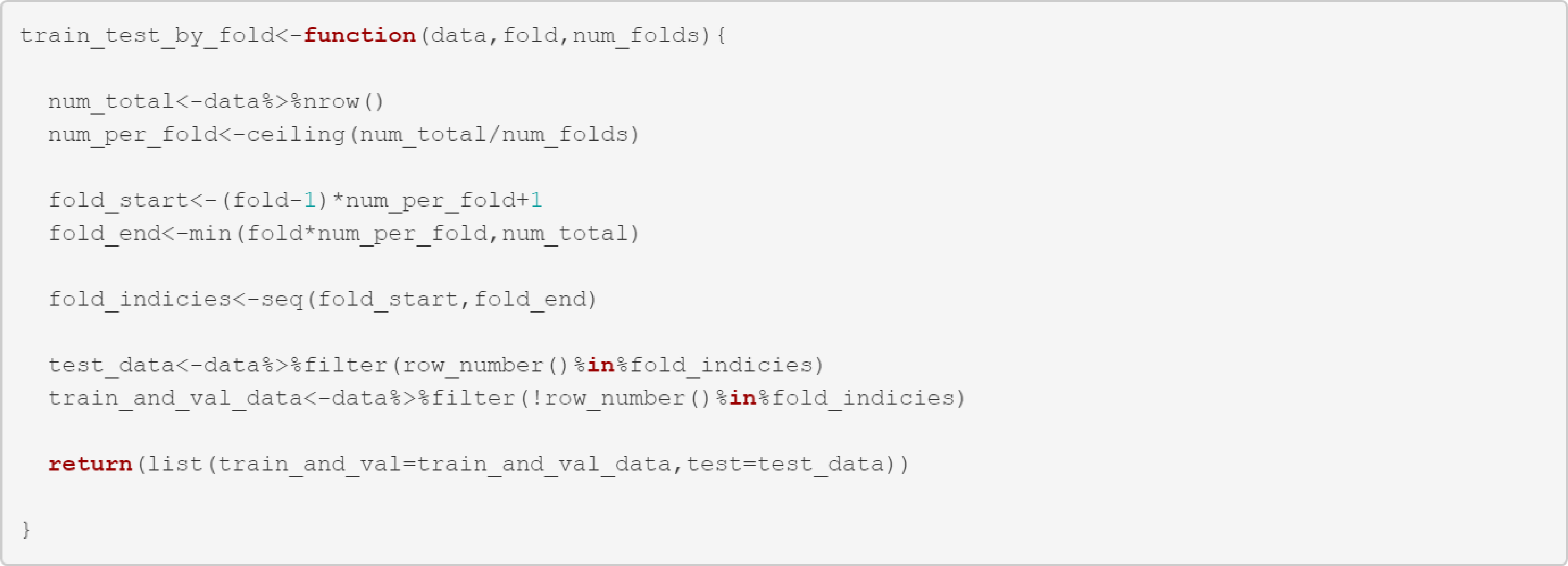
Using cross-validation to estimate test error reduces the dependency on a single piece of test data.

This can be computationally expensive since it requires another outer-loop through the data.

This nested procedure is sometimes referred to as “k*l-fold cross validation”.

The cross-validation approach in R

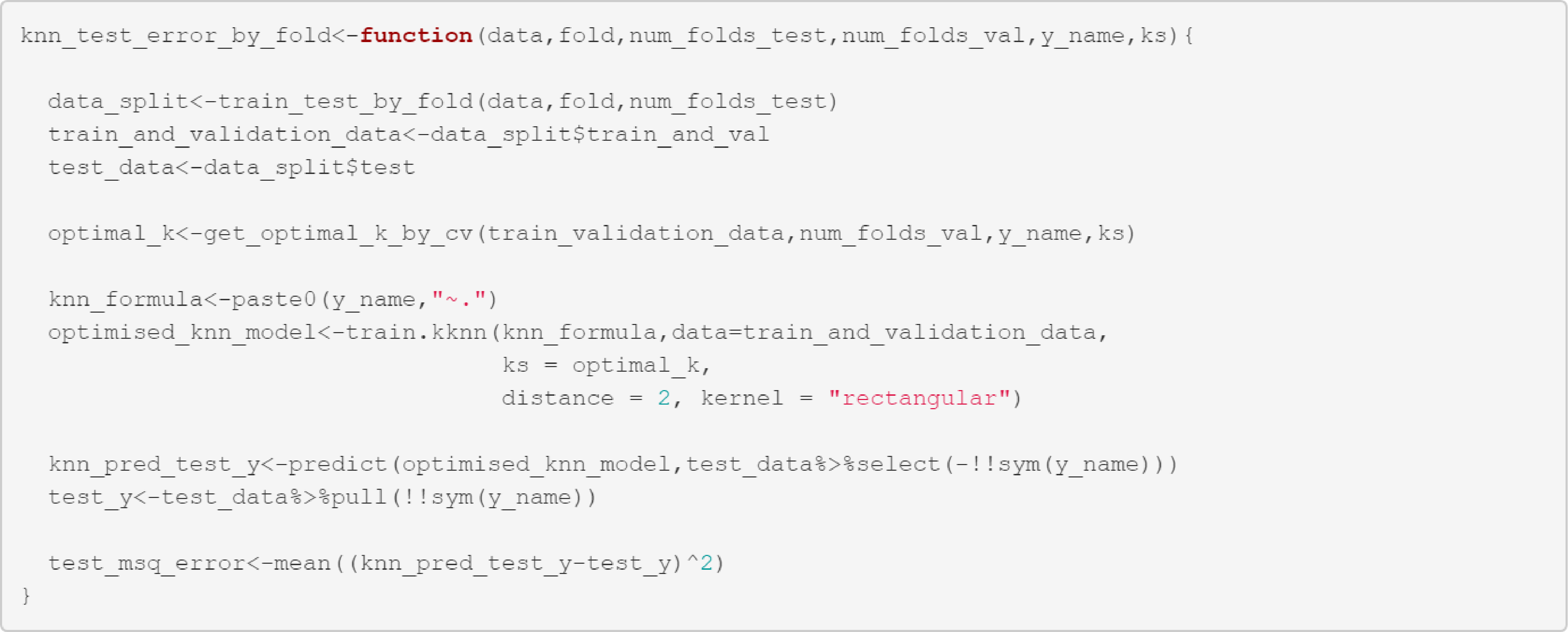
We create a function for extracting a split into test and train+validation by fold.



```
train_test_by_fold<-function(data,fold,num_folds){

  num_total<-data%>%nrow()
  num_per_fold<-ceiling(num_total/num_folds)

  fold_start<-(fold-1)*num_per_fold+1
  fold_end<-min(fold*num_per_fold,num_total)

  fold_indicies<-seq(fold_start,fold_end)

  test_data<-data%>%filter(row_number()%in%fold_indicies)
  train_and_val_data<-data%>%filter(!row_number()%in%fold_indicies)

  return(list(train_and_val=train_and_val_data,test=test_data))

}
```

The cross-validation approach in R

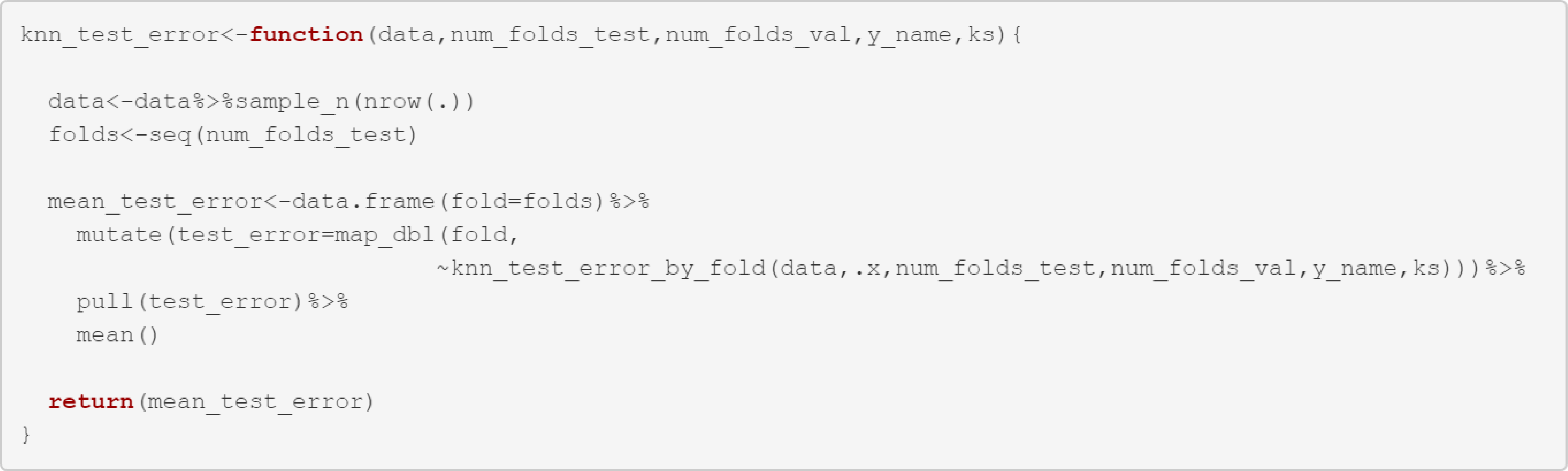
Next a function for estimating the test error of the knn with a validation optimised choice of k.



```r
knn_test_error_by_fold<-function(data,fold,num_folds_test,num_folds_val,y_name,ks){

  data_split<-train_test_by_fold(data,fold,num_folds_test)
  train_and_validation_data<-data_split$train_and_val
  test_data<-data_split$test

  optimal_k<-get_optimal_k_by_cv(train_validation_data,num_folds_val,y_name,ks)

  knn_formula<-paste0(y_name,"~.")
  optimised_knn_model<-train.kknn(knn_formula,data=train_and_validation_data,
                                ks = optimal_k,
                                distance = 2, kernel = "rectangular")

  knn_pred_test_y<-predict(optimised_knn_model,test_data%>%select(-!!sym(y_name)))
  test_y<-test_data%>%pull(!!sym(y_name))

  test_msq_error<-mean((knn_pred_test_y-test_y)^2)
}
```

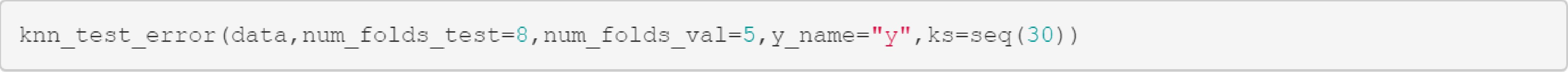
The cross-validation approach in R

Finally a procedure for estimating the out-of-sample error based on validation optimised knn.



```
knn_test_error<-function(data,num_folds_test,num_folds_val,y_name,ks){

  data<-data%>%sample_n(nrow(.))
  folds<-seq(num_folds_test)

  mean_test_error<-data.frame(fold=folds)%>%
    mutate(test_error=map_dbl(fold,
                             ~knn_test_error_by_fold(data,.x,num_folds_test,num_folds_val,y_name,ks)))%>%
    pull(test_error)%>%
    mean()

  return(mean_test_error)
}
```

We can apply this to estimating out of sample optimised knn performance as follows.



```
knn_test_error(data,num_folds_test=8,num_folds_val=5,y_name="y",ks=seq(30))
```

What have we covered today?

- If our hyper-parameter places to much emphasis on regularization under-fitting will occur:
  - Both the train and the test error will be large.

- If our hyper-parameter places too little emphasis on regularization over-fitting will occur:
  - The train error will be small but the gap between test and train error will be large.

- Good performance requires careful parameter tuning – We can do this based on validation data.

- A single train-validation-test split can lead to unstable hyper-parameter selection.

- We can improve our hyper-parameter selections via the k-fold cross-validation method.

University of BRISTOL

Thanks for listening!

henry.reeve@bristol.ac.uk

Include EMATM0061 in the subject of your email.