Selecting hyper-parameters with cross validation

Statistical Computing and Empirical Methods
Unit EMATM0061, Data Science MSc

Rihuan Ke
rihuan.ke@bristol.ac.uk

Teaching Block 1, 2024

University of BRISTOL

What we will cover today

We will return to the fundamental topic of hyperparameters and regularisation.

We will consider the concepts of overfitting, under-fitting and the gap between train and test error.

We will also think about some limitations of having a single train-validation-test split.

We will discuss the method of k-fold cross validation to improve hyperparameter selection.

Regularisation and hyper-parameters selection

The challenge of regression

Assume that we have a pair of random variables (X, Y) with unknown distribution $\mathbf{P}$.

- $X \in \mathcal{X}$ denotes a feature vector and $Y \in \mathbb{R}$ denotes a continuous variable.

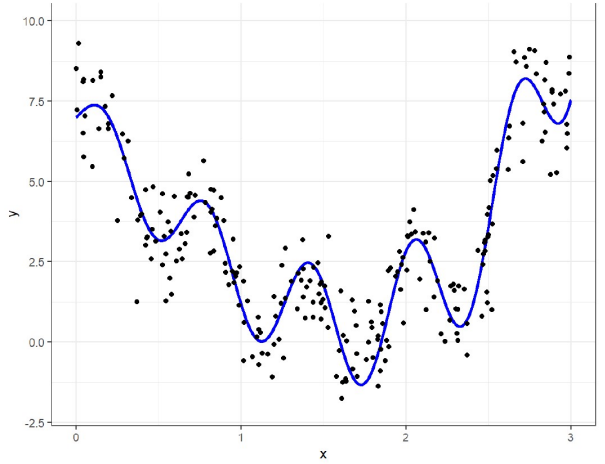
We want to learn a regression model $\phi : \mathcal{X} \to \mathbb{R}$ that minimises the mean squared error

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

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

The optimal function (minimising $\mathcal{R}_{\text{MSE}}$) is $\eta(x) = \mathbb{E}(Y \mid X = x)$.

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



(X_i, Y_i)

$(x, \eta(x))$

Learning a regression model

We don't know $\mathbf{P}$, so we can not compute directly the optimal function η.

If we have data $\mathcal{D} := ((X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n))$ where $(X_i, Y_i) \sim \mathbf{P}$ i.i.d.

We can learn from the data a regression model $\hat{\phi}$ that minimises the training error

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

The problem here is that minimising $\hat{\mathcal{R}}_{\text{MSE}}$ is not equavilent to minimising $\mathcal{R}_{\text{MSE}}$.

There are models ϕ with very low value of $\hat{\mathcal{R}}_{\text{MSE}}(\phi)$ and large value of $\mathcal{R}_{\text{MSE}}(\phi)$

These models perform very well on training data but poorly on unseen data

However, we are interested in models with good performance on unseen data.

How can we reduce the performance gap on training data and test data?

Regularisation

The k-nearest neighbour method

The optimal function (minimising $\mathcal{R}$) is $\eta(x) = \mathbb{E}(Y \mid X = x)$.

Idea: If $x_0 \in \mathcal{X}$ is close to $x_1 \in \mathcal{X}$, then we expect $\eta(x_0)$ to be close to $\eta(x_1)$.

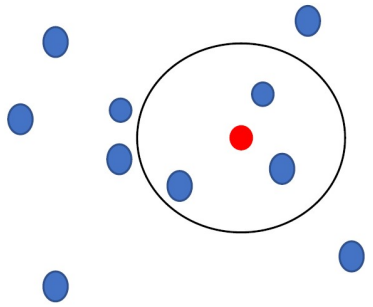
Suppose we have data $\mathcal{D} := ((X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n))$ where $(X_i, Y_i) \sim \mathbf{P}$

To estimate the conditional mean $\eta(x)$, we take the average of all Y_i satisfying that the associated X_i are close to x.

Step 1. Find k-nearest neighbours

Given a point $x \in \mathcal{X}$, we want to find k points from $(X_1, \cdots, X_n)$ with smallest distance to x.

$X_{\tau_1(x)}, X_{\tau_2(x)}, \cdots, X_{\tau_k(x)}$ are the k-nearest neighbours of x.



Step 2. Averaging

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

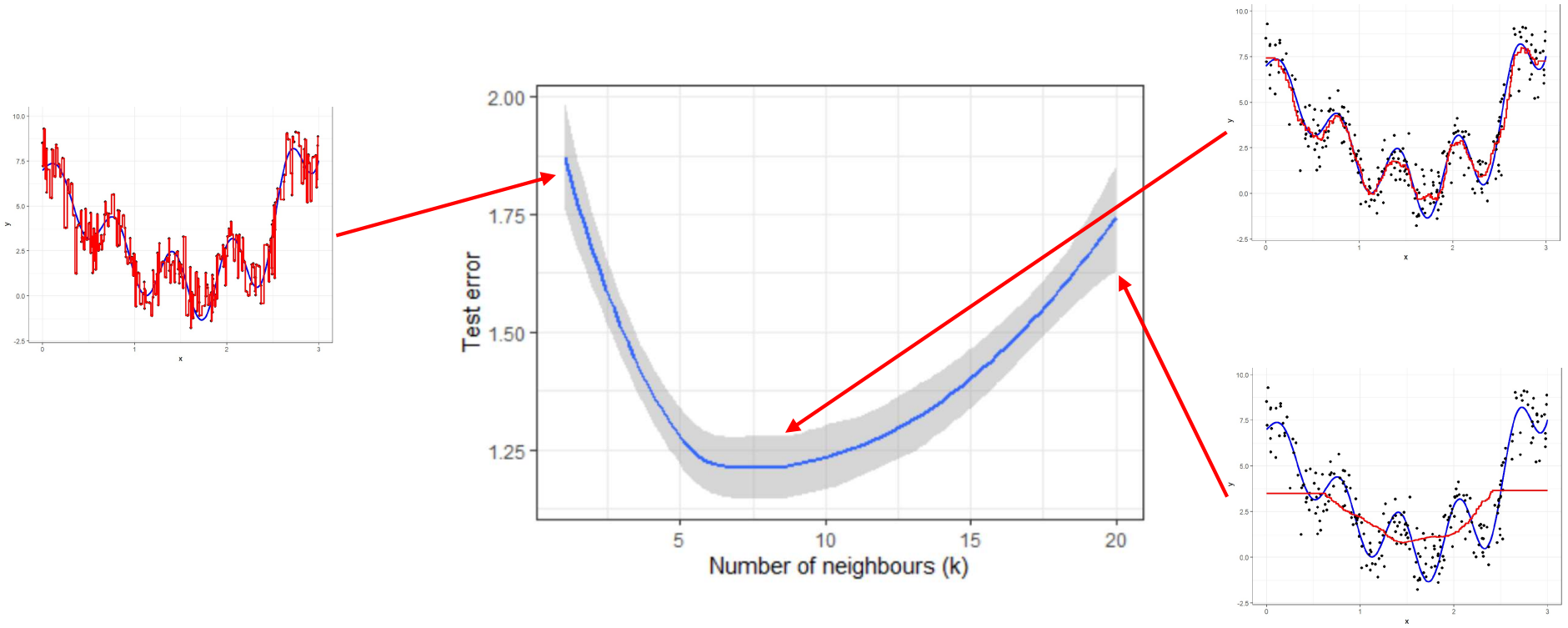
$\hat{\phi}(x)$ is the output of the regression model (can be viewed as an estimate of $\eta(x)$).

The k nearest neighbour method

Hyper-parameter k: the number of neighbour

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

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



Why do we observe this pattern?

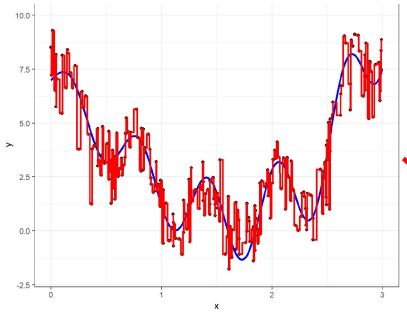
We can understand this by factoring the test error into two components (see next slides).

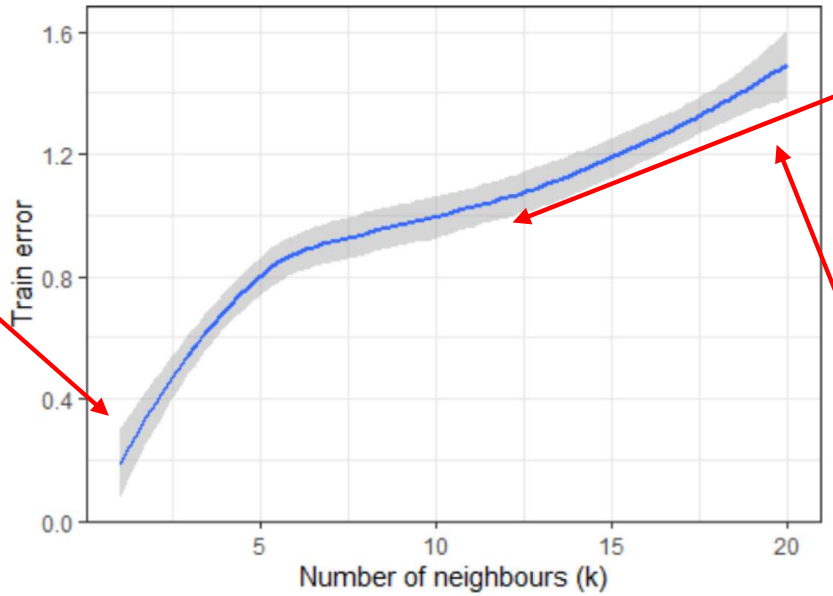
The k nearest neighbour method

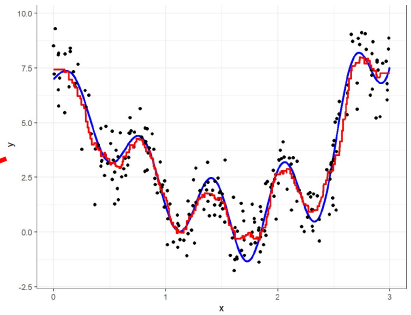
Hyper-parameter k: the number of neighbour

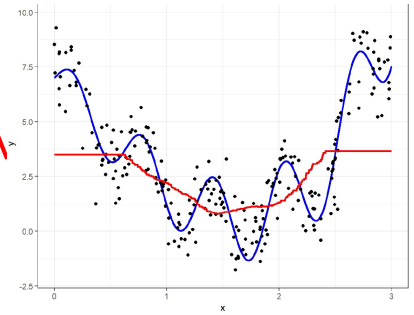
Let's look at the train error as a function of the number of neighbours k for a simulated example.

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









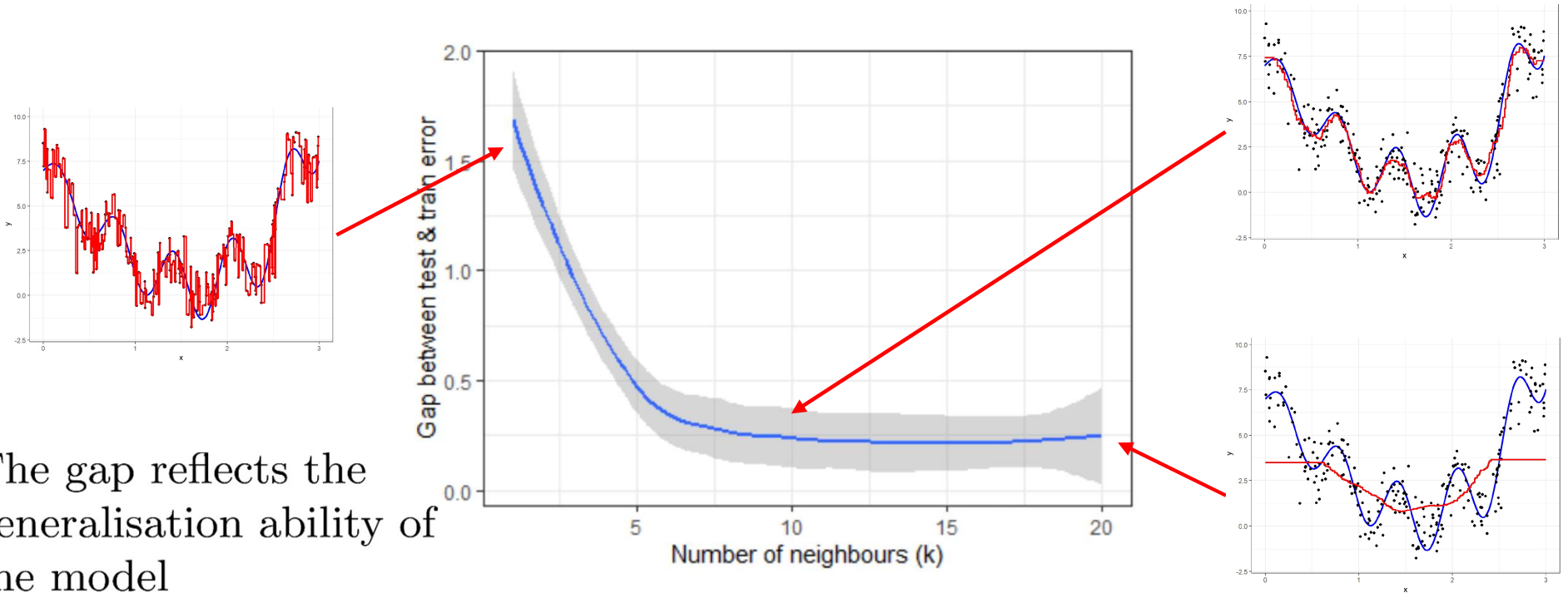
Note: training error increases monotonically as we increase the number of neighbours k.

The k nearest neighbour method

Hyper-parameter k: the number of neighbour

Let's look at the gap between test and train error as a function of the number of neighbours k. The gap is given by:

$$\Delta_{\mathrm{MSE}}(\phi) = \mathcal{R}_{\mathrm{MSE}}(\phi) - \hat{\mathcal{R}}_{\mathrm{MSE}}(\phi) = \mathbb{E}\left[(\phi(X) - Y)^2\right] - \frac{1}{n}\sum_{i=1}^{n}(\phi(X_i) - Y_i)^2.$$

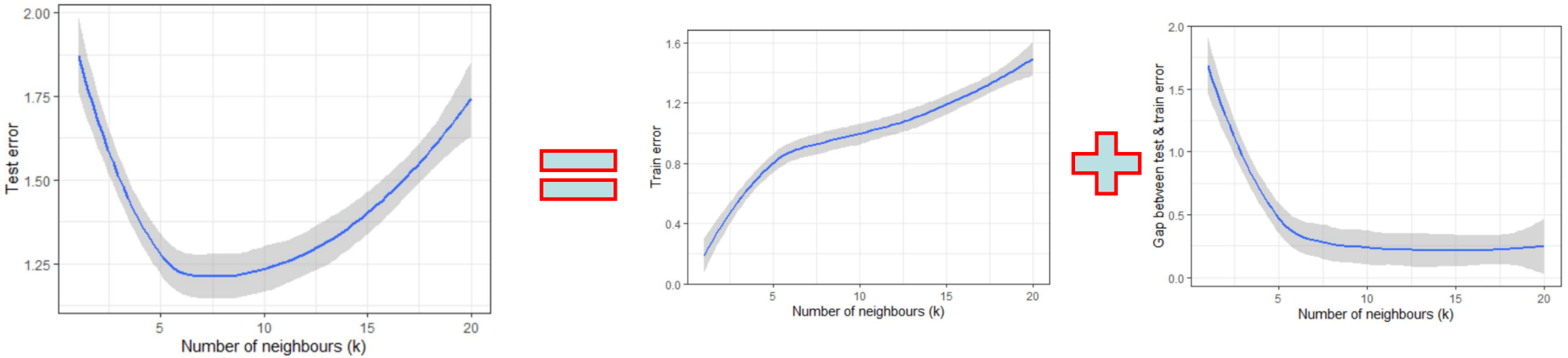


The gap reflects the generalisation ability of the model

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

Understand the test error

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



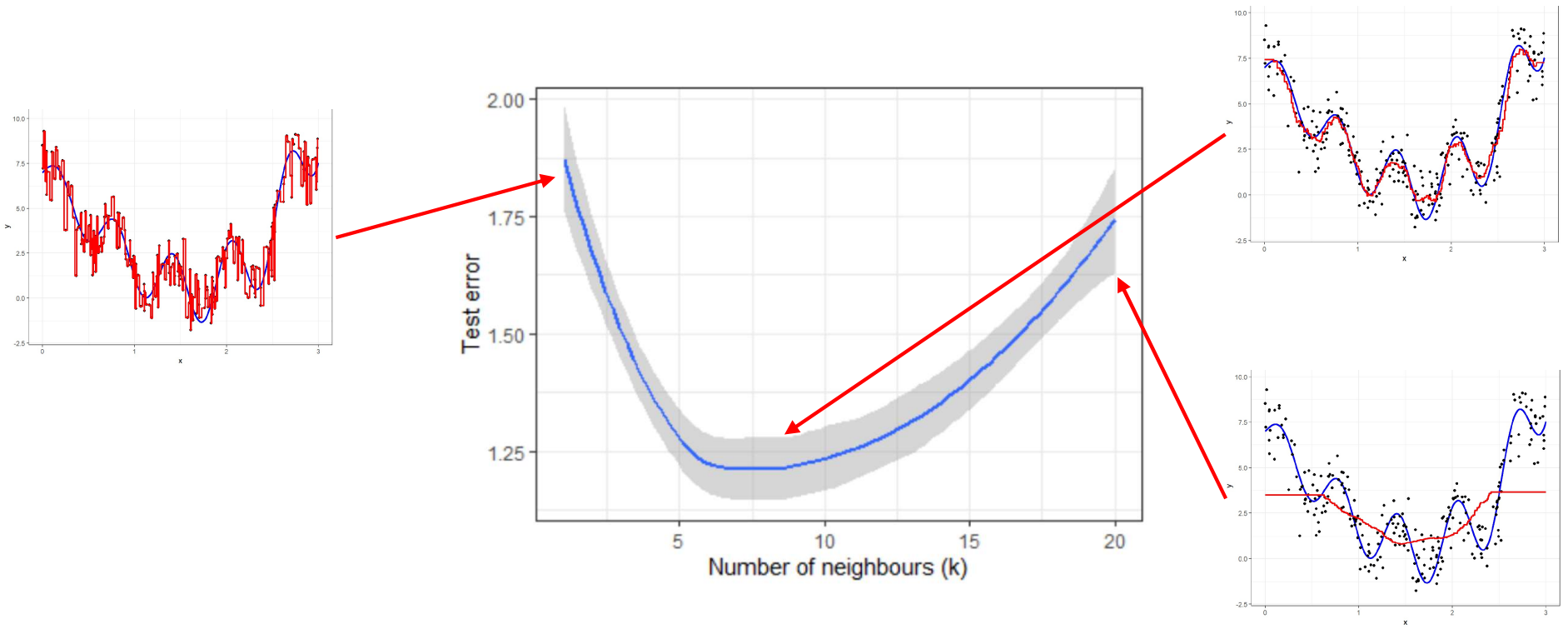
$$\mathcal{R}_{\text{MSE}}(\phi) = \hat{\mathcal{R}}_{\text{MSE}}(\phi) + \Delta_{\text{MSE}}(\phi) \ .$$

Test error — training error — error gap (generalisation ability)

Therefore, to have a small test error, we want to have a small training error as well as a small error gap.

The k nearest neighbour method

Hyper-parameter k: the number of neighbour



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 penalisation.

- Logistic regression models: l_2 or l_2 norm penalisation.

- Neural networks: l_1 or l_2 norm penalisation, learning rate, dropout, $\cdots$

- Random forests: Number of leaf nodes, maximum depth of trees, features per split.

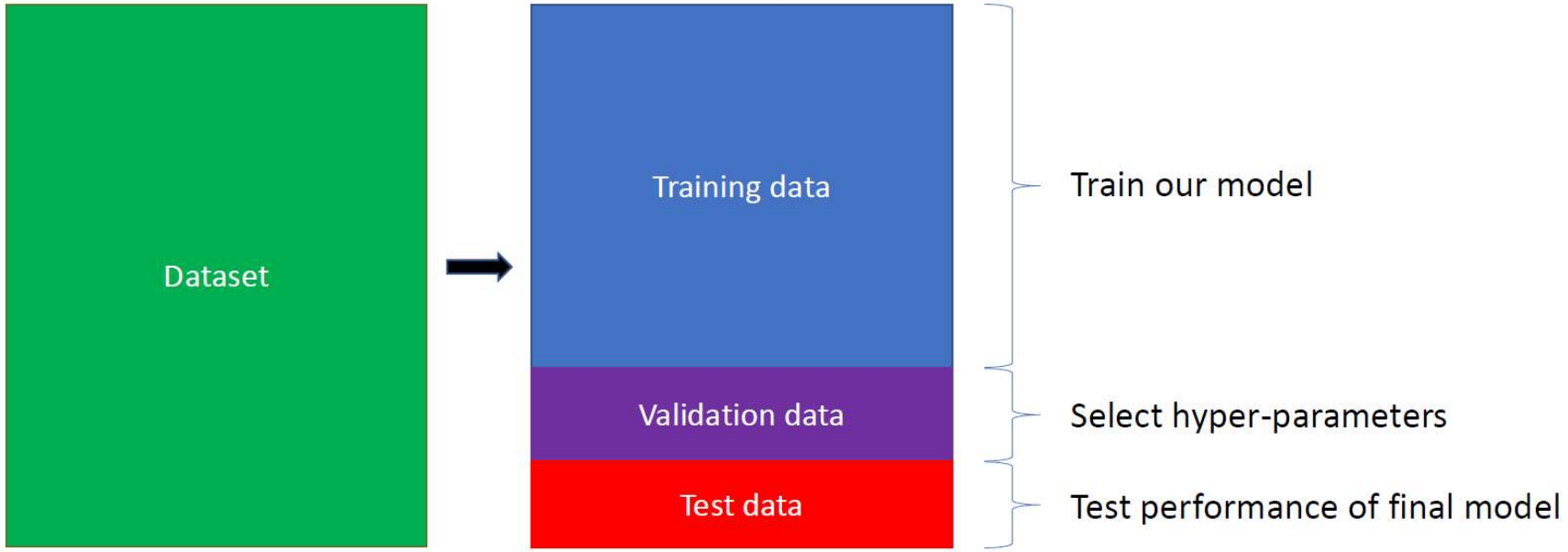
- Gradient boosting: Number of trees, maximum depth of trees, examples per round $\cdots$

- Support vector machines: Choice of kernel, hyper parameters of kernel, l_2 regularisation.

How should we select our hyper-parameters? We can select them by using a validation dataset.

The train-validation-test split

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

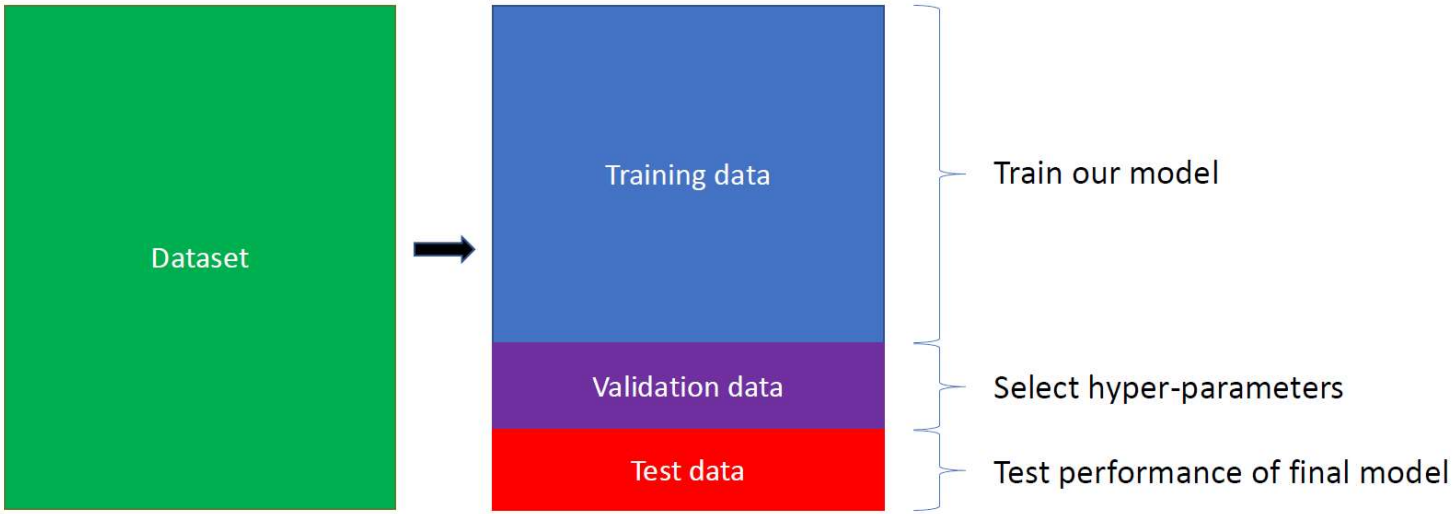


Limitation of a single train-validation-test split

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.



How can we mitigate this problem?

The cross-validation approach

Cross-validation

A single train-validation-test split: 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.

One option is to have a larger validation set, but this may not always be feasible.

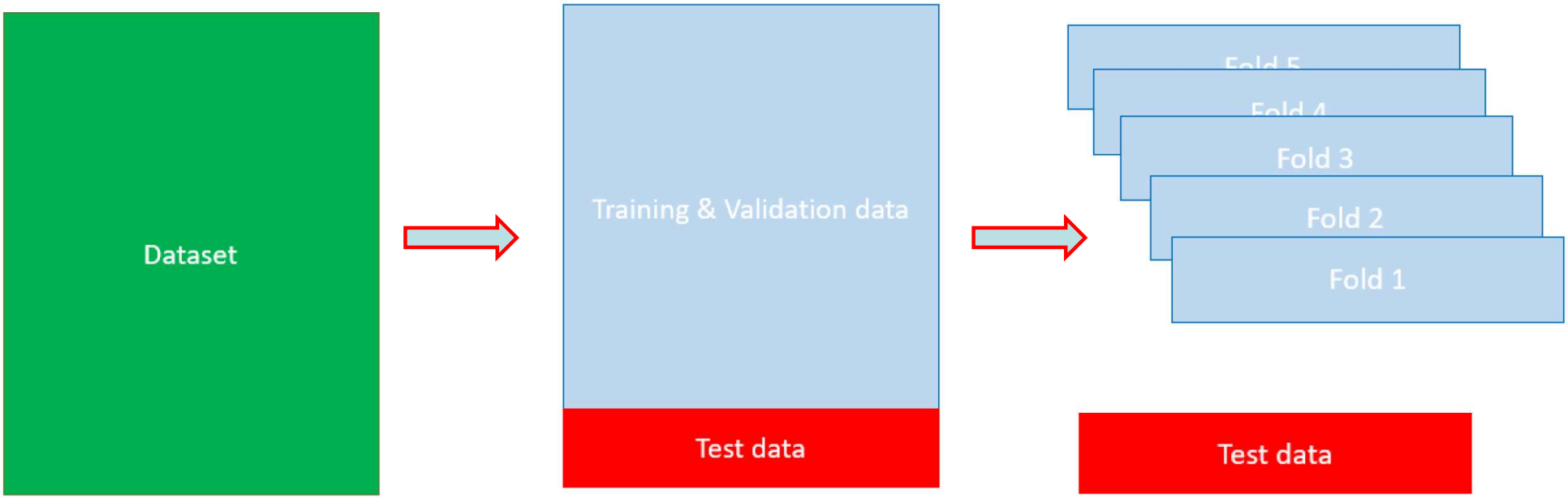
Another option is to use cross validation.

Idea: we repeat the training & validation "process" several times. Each time we use a different validation dataset (which is a subset of the dataset).

The hyperparameters are selected by taking into account the performance of the model on different validation splits

Cross-validation

We can split the dataset in the following way:



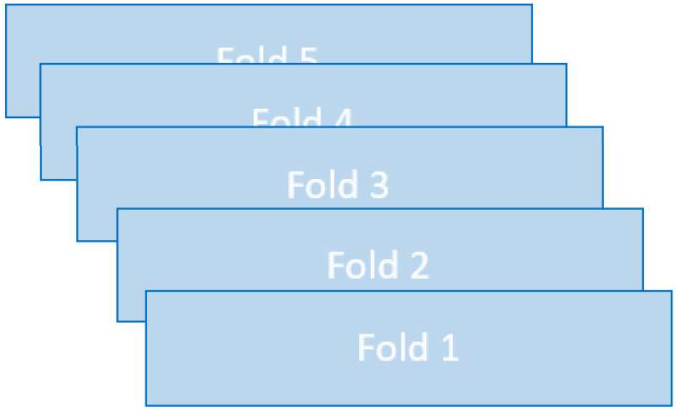
Note: at this point we haven't specified which part is our training data or which part is our validation data, but instead we have J subsets of the data called Fold 1, Fold 2, $\cdots$, Fold J.

The validation dataset will be specified at a later stage.

Cross-validation

A single train-validation-test split: for smaller data sets we have a small validation set, and performance depends crucially upon the split.

Idea: we repeat the training & validation "process" several times. Each time we use a different validation dataset (which is a subset of the dataset).



Case 1. We use Fold 1 as validation data and the other 4 folds as training data.

Case 2. We use Fold 2 as validation data and the other 4 folds as training data.

⋮

Case 5. We use Fold 5 as validation data and the other 4 folds as training data.

Note: a fold can be validation data in one case but training data in another case.

The cross-validation approach

Suppose that we have a hyper-parameter λ.

For a given value λ_q. The validation error associated with $\lambda = \lambda_q$ can be computed in the following way:

Use Fold 2, 3, . . . , 5 (training data) to train the model $\hat{\phi}^{(1)}_{\lambda_q}$ with hyper-parameter $\lambda = \lambda_q$.

Use Fold 1 (val data) to compute validation error $V_1(\lambda_q)$ of $\hat{\phi}^{(1)}_{\lambda_q}$.

Use Fold 1, 3, . . . , 5 (training data) to train the model $\hat{\phi}^{(2)}_{\lambda_q}$ with hyper-parameter $\lambda = \lambda_q$.

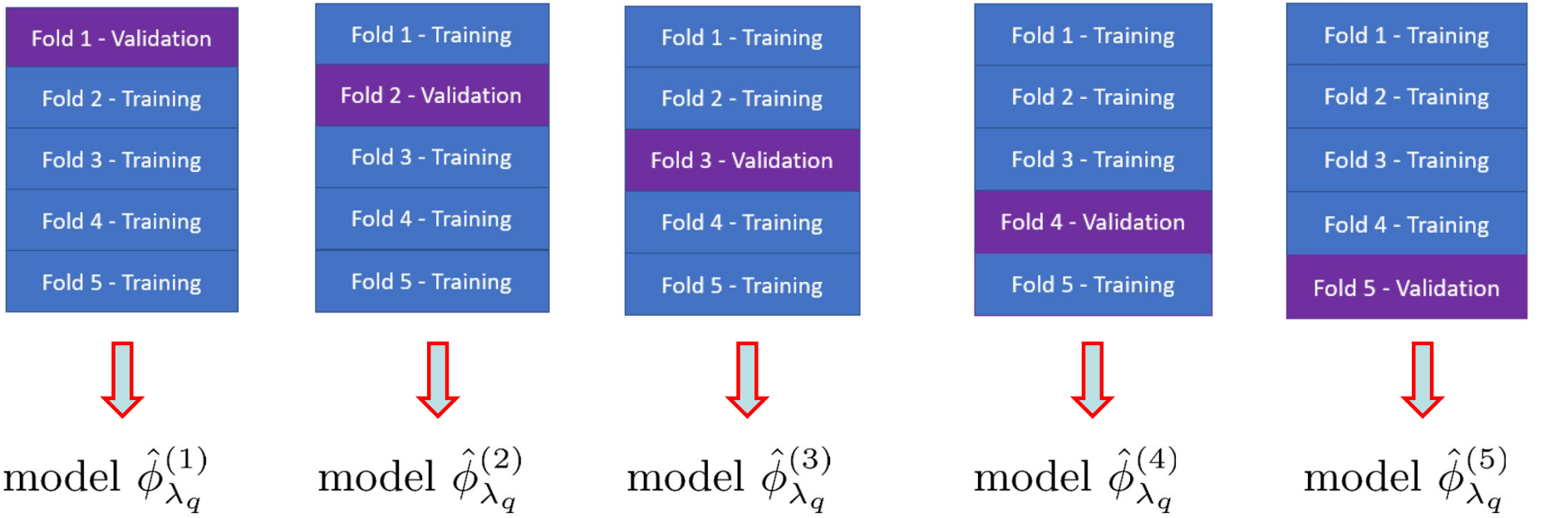
Use Fold 2 (val data) to compute validation error $V_2(\lambda_q)$ of $\hat{\phi}^{(2)}_{\lambda_q}$.

$\vdots$

The cross-validation approach

Suppose that we have a hyper-parameter λ.

For a given value λ_q. The validation error associated with $\lambda = \lambda_q$ can be computed in the following way:



model $\hat{\phi}^{(1)}_{\lambda_q}$ val error $V_1(\lambda_q)$ | model $\hat{\phi}^{(2)}_{\lambda_q}$ val error $V_2(\lambda_q)$ | model $\hat{\phi}^{(3)}_{\lambda_q}$ val error $V_3(\lambda_q)$ | model $\hat{\phi}^{(4)}_{\lambda_q}$ val error $V_4(\lambda_q)$ | model $\hat{\phi}^{(5)}_{\lambda_q}$ val error $V_5(\lambda_q)$

The final validation error $\overline{V}(\lambda_q)$ is computed by averaging $V_1(\lambda_q), \cdots, V_5(\lambda_q)$.

Note: to compute the final validation error requires training multiple models (associated with the different split)

The cross-validation approach

Suppose that we have a hyper-parameter λ.

For a given value λ_q. The validation error associated with $\lambda = \lambda_q$ can be computed in the following way:

model $\hat{\phi}_{\lambda_q}^{(1)}$	model $\hat{\phi}_{\lambda_q}^{(2)}$	model $\hat{\phi}_{\lambda_q}^{(3)}$	model $\hat{\phi}_{\lambda_q}^{(4)}$	model $\hat{\phi}_{\lambda_q}^{(5)}$
val error $V_1(\lambda_q)$	val error $V_2(\lambda_q)$	val error $V_3(\lambda_q)$	val error $V_4(\lambda_q)$	val error $V_5(\lambda_q)$

The final validation error $\overline{V}(\lambda_q)$ is computed by averaging $V_1(\lambda_q), \cdots, V_5(\lambda_q)$.

Note: to compute the final validation error requires training multiple models (associated with the different split)

We have reused training data for validation (each fold used exactly once as val data).

This procedure is often referred to as "k fold cross-validation".

Note: Here "k" refers to the number of folds (in this case $k = 5$) - not the number of neighbours (in the knn method)

Hyper-parameter selection

Suppose that we have a hyper-parameter λ.

For a given value λ_q. The validation error associated with $\lambda = \lambda_q$ can be computed in the following way:

model $\hat{\phi}^{(1)}_{\lambda_q}$	model $\hat{\phi}^{(2)}_{\lambda_q}$	model $\hat{\phi}^{(3)}_{\lambda_q}$	model $\hat{\phi}^{(4)}_{\lambda_q}$	model $\hat{\phi}^{(5)}_{\lambda_q}$
val error $V_1(\lambda_q)$	val error $V_2(\lambda_q)$	val error $V_3(\lambda_q)$	val error $V_4(\lambda_q)$	val error $V_5(\lambda_q)$

The final validation error $V(\lambda_q)$ is computed by averaging $V_1(\lambda_q), \cdots, V_5(\lambda_q)$.

Hyper-parameter selection:

Consider a list of possible values of λ: $\{\lambda_1, \cdots, \lambda_Q\}$.

For each λ_q $(q = 1, 2, \cdots, Q)$, compute the average validation error $\overline{V}(\lambda_q)$ using k-fold cross validation (this requires training multiple models associated with λ_q).

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

Hyper-parameter selection

Hyper-parameter selection:

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

For $q = 1, \cdots, Q$,

J means the number of folds!

For $j = 1, \cdots, J$,

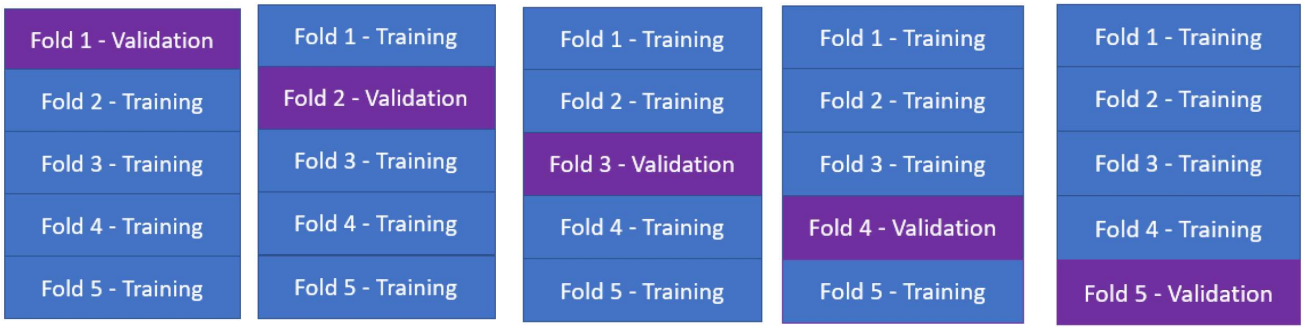
Train the model $\hat{\phi}_{\lambda_q}^{(j)}$ with hyper-parameter $\lambda = \lambda_q$ using all folds except for j.

Compute the validation error $V_j(\lambda_q)$ of the model on the j-th fold.

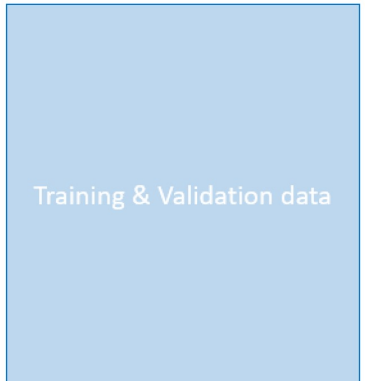
Compute the average validation error $\overline{V}(\lambda_q) := \frac{1}{J} \sum_{j=1}^{J} V_j(\lambda_q)$

Select the hyper-parameter $\hat{\lambda} \in \{\lambda_1, \cdots, \lambda_Q\}$ to minimise $\overline{V}(\hat{\lambda})$.

The cross-validation approach



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



Use the combined data to retrain the model with the selected hyperparameter.

Use the test data to test the final model.

The cross-validation approach in R

The cross-validation approach in R

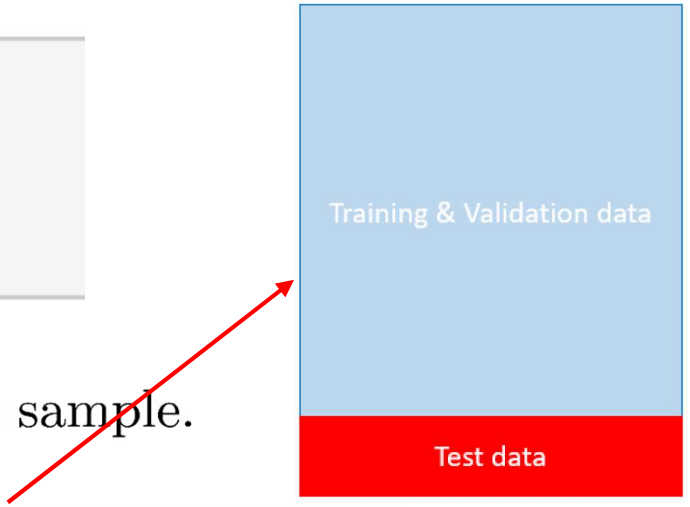
Suppose we have a data frame called "data". We take 25% of our data as test data.

```
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.

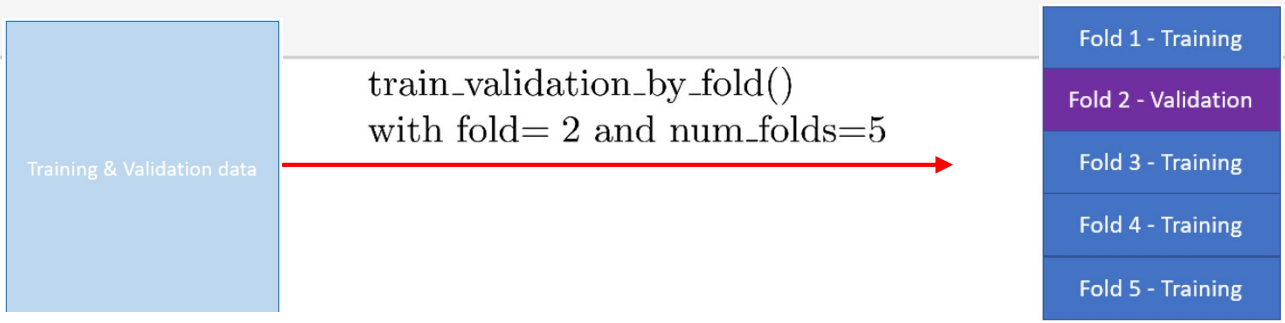



```
test_data<-data%>%filter(row_number()%in%test_inds)
train_validation_data<-data%>%filter(!row_number()%in%test_inds)
```

The cross-validation approach in R

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

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

```
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)

}
```

Train the model $\hat{\phi}_{\lambda_q}^{(j)}$ with hyper-parameter $\lambda = \lambda_q$ using all folds except for j.

Compute the validation error $V_j(\lambda_q)$ of the model on the j-th fold.

Train the model $\hat{\phi}_{\lambda_q}^{(j)}$ with hyper-parameter $\lambda = \lambda_q$ using all folds except for j.

Compute the validation error $V_j(\lambda_q)$ of the model on the j-th fold.

The cross-validation approach in R

We specify the number of folds and a selection of possible values of k (i.e., the numbers of neighbours)

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

$j = 1, \cdots, J,$

$\{\lambda_1, \cdots, \lambda_Q\}$

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

```
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))
```

For $q = 1, \cdots, Q,$

For $j = 1, \cdots, J,$

Train the model $\hat{\phi}^{(j)}_{\lambda_q}$ with hyper-parameter $\lambda = \lambda_q$ using all folds except for j.

Compute the validation error $V_j(\lambda_q)$ of the model on the j-th fold.

Compute the average validation error $\overline{V}(\lambda_q) := \frac{1}{J}\sum_{j=1}^{J} V_j(\lambda_q)$

The cross-validation approach in R

Find the hyper-parameter which minimises the validation error.

Select the hyper-parameter $\hat{\lambda} \in \{\lambda_1, \cdots, \lambda_Q\}$ to minimise $\overline{V}(\hat{\lambda})$.

```
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")
```

Use the combined data to retrain the model with the selected hyperparameter.

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)
```

Test data

Use the test data to test the final model.

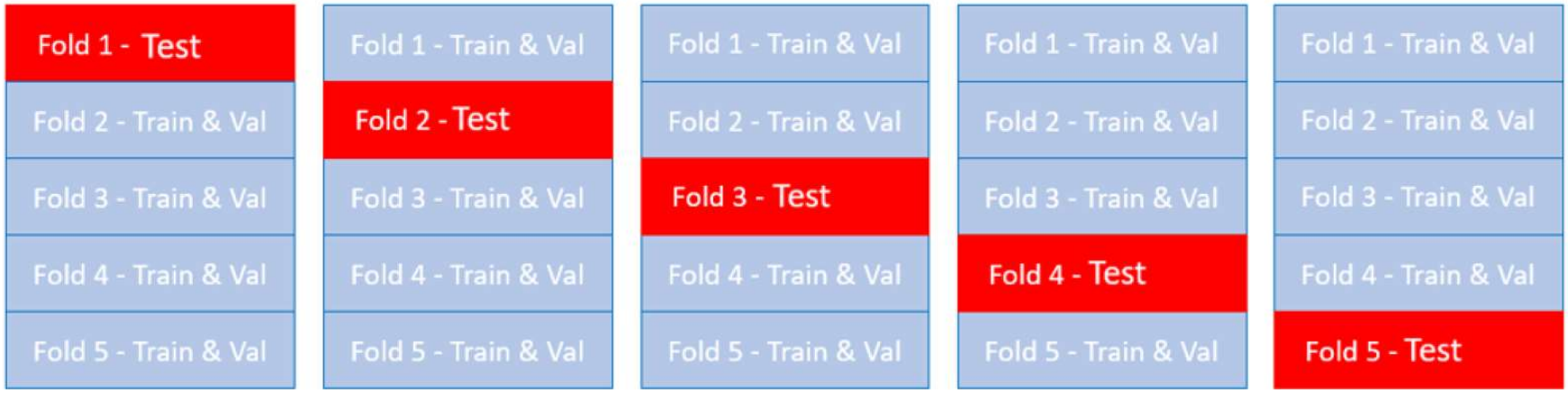
The cross-validation approach in R

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

```
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)
}
```

Cross-validation for test error

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



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

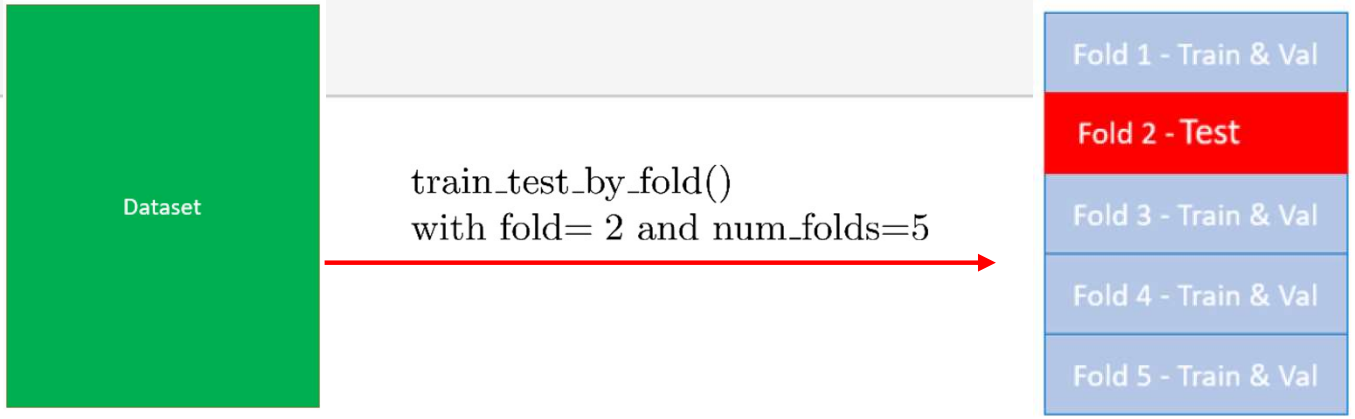
This can be computationally expensive since it requires another outer loop through the data.

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

"fold" specifies the fold used as test data.

```
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)
}
```

This is the function that performs k fold cross-validation and selects the optimal k given the train & val set

The cross-validation approach in R

Finally, we define a procedure for estimating the average 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 the average 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?

If our hyperparameter places too much emphasis on regularisation, under-fitting will occur:

- Both the training and the test errors will be large.

If our hyperparameter places too little emphasis on regularisation over-fitting will occur

- The training error will be small but the gap between the test and the training 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 hyperparameter selection.

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

University of BRISTOL

Thanks for listening!

Dr. Rihuan Ke

rihuan.ke@bristol.ac.uk

Statistical Computing and Empirical Methods
Unit EMATM0061, MSc Data Science