Intro to classification - kNN - 4

One should look for what is and not what he thinks should be. (Albert Einstein)

Module completion checklist

Objective	Complete
Perform cross-validation to better understand what the optimal model accuracy might be	
Uncover optimal hyperparameters with a grid search	

Finding optimal k

- **We have now:**
  - Run kNN on our training data
  - Predicted using the kNN model on our test data
  - Reviewed performance metrics for classification algorithms
  - Built a confusion matrix for the predicted model
- **We will now:**
  - Use cross-validation and re-run the model on the training data
  - Implement grid search to know what our optimal k value is
  - Evaluate the new predictions

Cross-validation: n-fold

- To quickly refresh our memory, here is how cross-validation works:
  i. Split the dataset into several subsets ("n" number of subsets) of equal size
  ii. **Use each subset as the test dataset** and **use the rest of the data as the training dataset**
  iii. Repeat the process for every subset you create

Test

Train

Data	x	y	z
1	...	...	...
2	...	...	...
3	...	...	...
4	...	...	...
5	...	...	...
6	...	...	...

Data	x	y	z
1	...	...	...
2	...	...	...
3	...	...	...
4	...	...	...
5	...	...	...
6	...	...	...

Data	x	y	z
1	...	...	...
2	...	...	...
3	...	...	...
4	...	...	...
5	...	...	...
6	...	...	...

- Cross-validation helps prevent overfitting by performing the model on multiple subsets

Cross-validation in Python

- We will be using two functions from `sklearn.model_selection` for cross-validation:
  - `cross_val_score`
  - `GridSearchCV`
- They are both model selection / optimization functions that use cross-validation and they are both part of the `sklearn.model_selection` package
  - `cross_val_score` will help us find the potential optimal model score
  - `GridSearchCV` will run through cross-validation multiple times for a range of given `k` and help us find the optimal value

cross_val_score for optimal accuracy

- Here is a little about `cross_val_score`

sklearn.model_selection.cross_val_score

`sklearn.model_selection.cross_val_score(estimator, X, y=None, groups=None, scoring=None, cv='warn', n_jobs=None, verbose=0, fit_params=None, pre_dispatch='2*n_jobs', error_score='raise-deprecating')` [source]

Evaluate a score by cross-validation

Read more in the User Guide.

Parameters:

estimator : *estimator object implementing 'fit'*
The object to use to fit the data.

X : *array-like*
The data to fit. Can be for example a list, or an array.

y : *array-like, optional, default: None*
The target variable to try to predict in the case of supervised learning.

groups : *array-like, with shape (n_samples,), optional*
Group labels for the samples used while splitting the dataset into train/test set.

scoring : *string, callable or None, optional, default: None*
A string (see model evaluation documentation) or a scorer callable object / function with signature `scorer(estimator, X, y)`.

cv : *int, cross-validation generator or an iterable, optional*
Determines the cross-validation splitting strategy. Possible inputs for cv are:

- None, to use the default 3-fold cross validation,
- integer, to specify the number of folds in a (Stratified)KFold,
- CV splitter,
- An iterable yielding (train, test) splits as arrays of indices.

For integer/None inputs, if the estimator is a classifier and `y` is either binary or multiclass, `StratifiedKFold` is used. In all other cases, `KFold` is used.

Refer User Guide for the various cross-validation strategies that can be used here.

Changed in version 0.20: `cv` default value if None will change from 3-fold to 5-fold in v0.22.

n_jobs : *int or None, optional (default=None)*
The number of CPUs to use to do the computation. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details.

verbose : *integer, optional*
The verbosity level.

fit_params : *dict, optional*
Parameters to pass to the fit method of the estimator.

Cross-validation pipeline for optimal accuracy

- We will use the `Pipeline()` function from `sklearn` so that we can follow the preprocessing order of splitting and then scaling before going into the model
- Read more about `Pipeline()` ***here***
- Note that here we use `StandardScaler()` for scaling vs `scale()` which we used before
- These two functions are doing exactly the same thing, but:
  - `scale(x)` is just a function, which transforms some data
  - `StandardScaler()` is a class supporting the Transformer API

```
# Create a pipeline of the scaler and Estimator
cv_pipeline = Pipeline([('transformer',  StandardScaler()), ('estimator', kNN)])
```

Cross-validation for optimal accuracy

- First, let's use `cross_val_score` to understand what our optimal accuracy should be
- We'll take the following steps:
  - using our `kNN_k` model, we train a model with a cross-validation reoccurrence of five times
  - we look at each cross-validation accuracy score
  - we then average the accuracy scores over the 5 times and we use that as our potential optimal model score

```
# Calculate cv scores
cv_scores = cross_val_score(cv_pipeline, X, y, cv = 5)
```

- Notice that we don't use our split data, the input is our entire `X` and `y`
- This is because the cross-validation occurs when the function holds out samples for us

Cross-validation for optimal accuracy

- Let's look at our output

```
# Print each cv score (accuracy) and average them.
print(cv_scores)
```

```
[0.94911937 0.94618395 0.9481409  0.95009785 0.9481409 ]
```

```
print("cv_scores mean:{}".format(np.mean(cv_scores)))
```

```
cv_scores mean:0.9483365949119374
```

```
mean = np.mean(cv_scores)
print("Optimal cv score is:", round(mean, 4))
```

```
Optimal cv score is: 0.9483
```

Module completion checklist

Objective	Complete
Perform cross-validation to better understand what the optimal model accuracy might be	✔
Uncover optimal hyperparameters with a grid search	

Parameters vs. hyperparameters

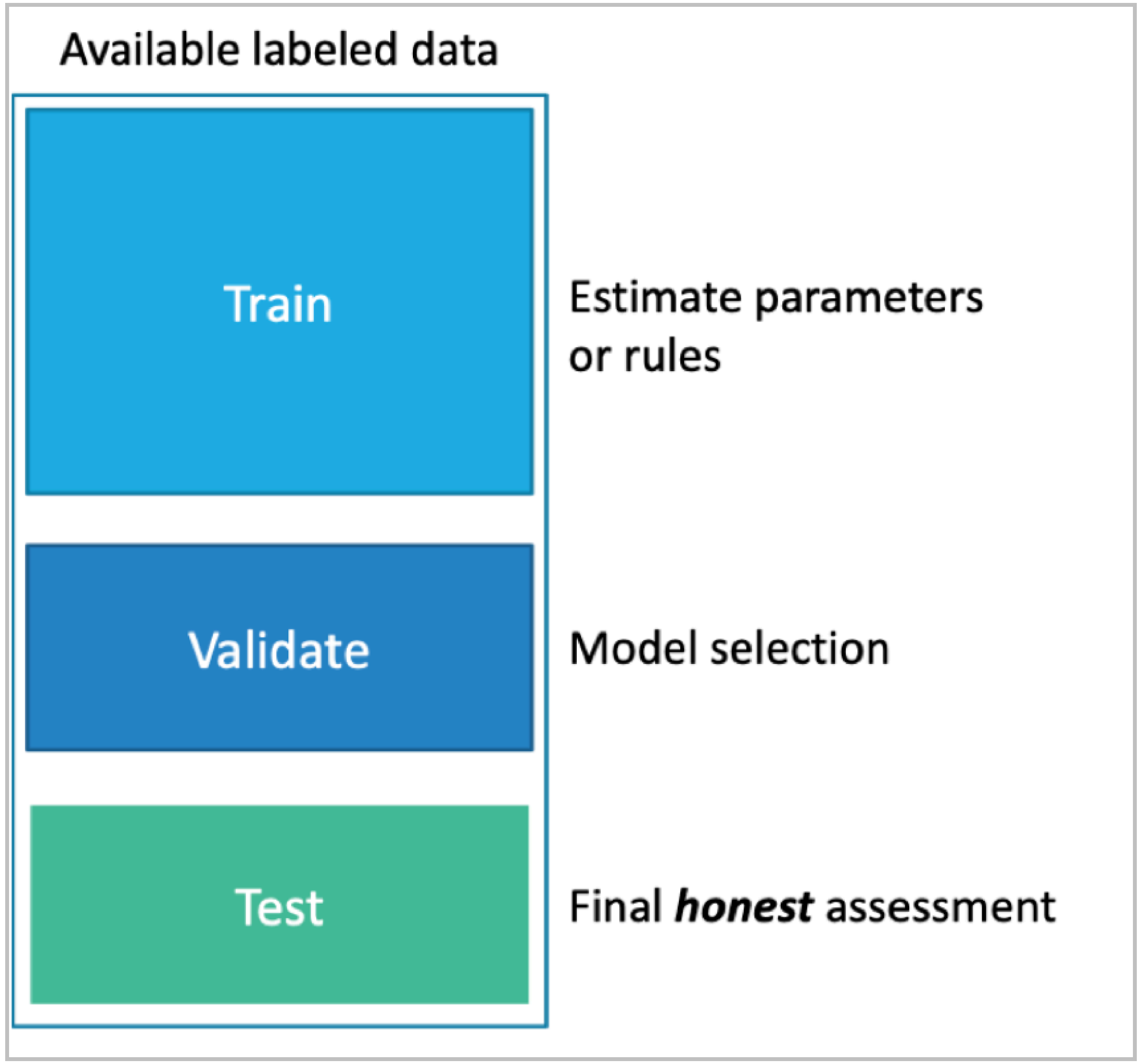
- **Parameters**
  - Parameters are derived from training data
  - Example: the weights of a predictor in a regression
  - They are learned by the algorithm from the data
- **Hyperparameters**
  - Hyperparameters are manually set before the training process
  - Example: k in kNN, number of trees in random forest, penalty in penalized regression
  - They are found using a **grid search**

What is a grid search?

- **Grid search helps us find the optimal hyperparameters**
    - Grid search allows us to search over a list of hyperparameter values to find the optimal hyperparameter
    - It is a brute force approach: it creates a model for every hyperparameter value in the list
    - We choose the hyperparameter value that created the model with the smallest error
- **Example kNN**
    - Grid search allows us to find the optimal `k`
    - We can use a grid search to search over k=1, k=2, k=3, k=4, etc.
    - Grid search creates models for every value of `k` in the list
    - We can choose the `k` that yields the smallest error

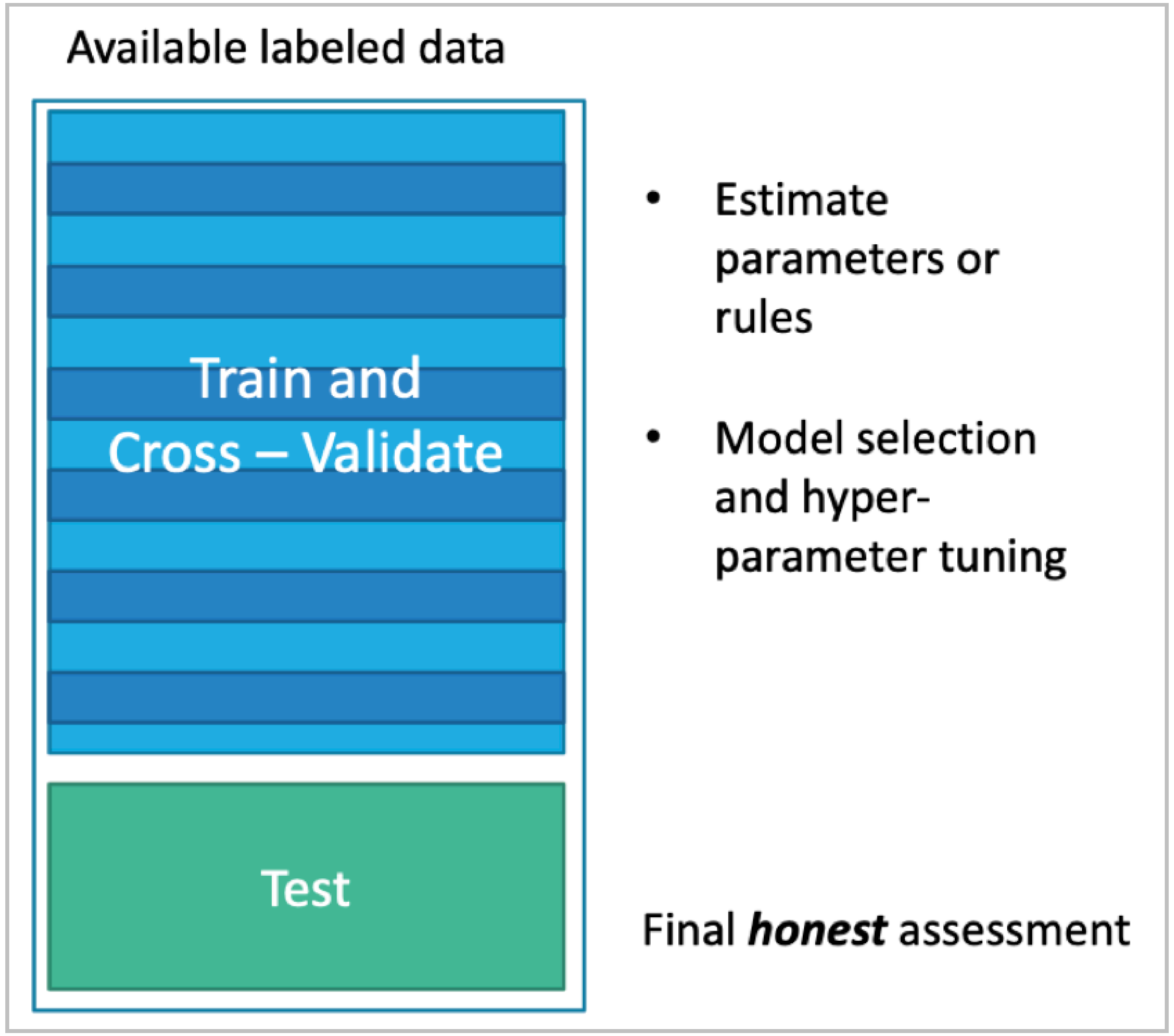
Finding the smallest error with grid search

- So far, we have used a **train - test split** to train and evaluate our models
- This worked because we only had parameters, but no hyperparameters
- Now that we also have hyperparameters in our models, we need a **train - validation - test split**
- The validation set allows us to compare the different models created by the grid search and choose the optimal hyperparameters



Using cross-validation with grid search

- Instead of using train - validation - test split, we can use **cross-validation**
- Cross-validation allows us to perform the split multiple times on the same dataset
- We have new train and validation sets for each fold `n`
- This leads to more accurate results, but is computationally intensive
- It is best suited for small datasets
- In this module, we will use cross-validation with our grid search



Keeping test data separate

- Whether you use **train - validation - test** or **cross-validation**, you always want to have a separate test set
- The test set must not be involved in the model training and selection process to allow for a **true assessment**
- Only a true assessment tells you how your model will perform on previously unseen data
- The errors on your train and validation sets will be lower than the errors you can expect on previously unseen data

Knowledge check

Module completion checklist

Objective	Complete
Perform cross-validation to better understand what the optimal model accuracy might be	✔
Uncover optimal hyperparameters with a grid search	✔

Congratulations on completing this module!

You are now ready to try Task 14 in the Exercise for this topic