Cross Validation

Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Machine Learning for Economics and Finance

Bachelor in Economics

Marcel Weschke

27.05.2025

Learning Goals

At the end of this lecture, you should be able to:

- Understand why cross validation is important to assess the accuracy of a methods
- Analyze the difference between a standard validation set approach and k-fold cross validation
- Apply cross validation in Python

Book Chapter: 5

Motivation

- In supervised learning we are interested in finding a model that is able to produce accurate out-of-sample predictions
- **Problem**: while we can easily evaluate in-sample error rates, out-of sample errors for (unknown) test data are difficult to obtain
- For this we would need a large test dataset which is usually not available
- **Key question**: How can we accurately approximate out-of-sample test errors?

The Validation Set Approach

- In-sample errors provide a too optimistic estimate for out-of-sample errors
- **Idea**: Only use part of the data to train your model and use the other part to compute out-of-sample errors
- **Validation set**: Part of the data that is left out in the training process of the model
- **Validation error**: We use this model to compute out-of-sample prediction errors in the validation set. These validation errors provide an estimate of the unknown test error

Validation Set Approach in Practice

Split the data into:

- **Training set**: to train the model
- **Validation set**: to obtain estimate for out-of-sample errors based on which the best model is selected
- **Test set**: to compute out-of-sample errors for final model

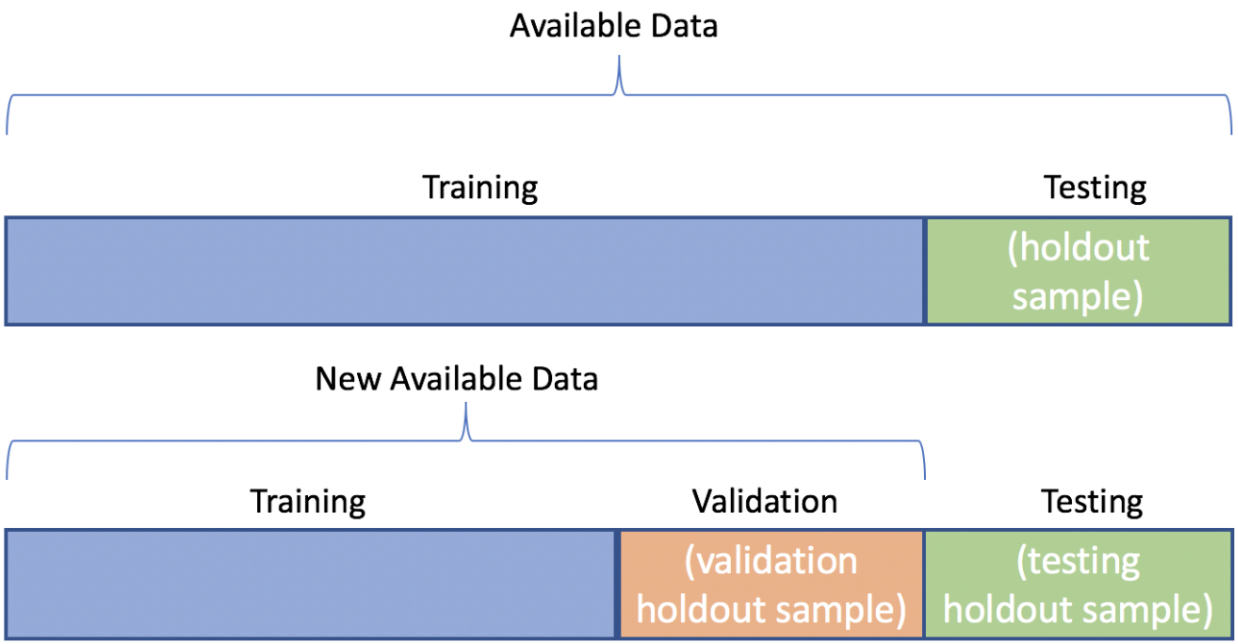


Figure from `https://datascience.stackexchange.com/questions/61467/clarification-on-train-test-and-val-and-how-to-use-implement-it`

Cross Validation in Practice

- Typical split of the data:
  - Training: 70%
  - Validation: 15%
  - Testing: 15%
- For very large data one might use up tp 90% for training
- For small datasets one might use smaller subsamples
- Important to make sure that training data is representative of validation and test data (non trivial, more on this later)

The Validation Set Approach in the *Auto* Data

Let's again look at the *Auto* data using the file `03_Auto_data_val_set.ipynb`.

Recall that we are interested in models of the form:

$$mpg = \beta_0 + \beta_1 horsepower + \beta_2 horsepower^2 + \ldots + \epsilon$$

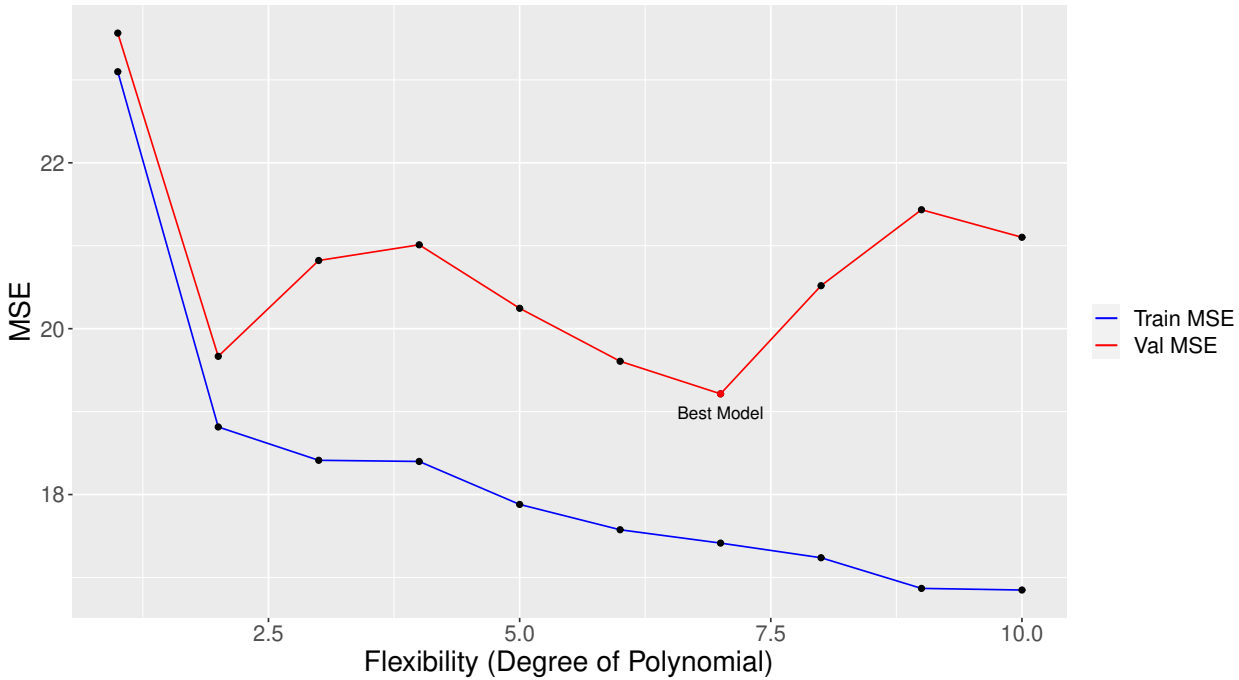
Question: Which model is the best (most accurate) to predict mpg ?

Splitting the Data

We randomly split the model into a training, validation and test set.

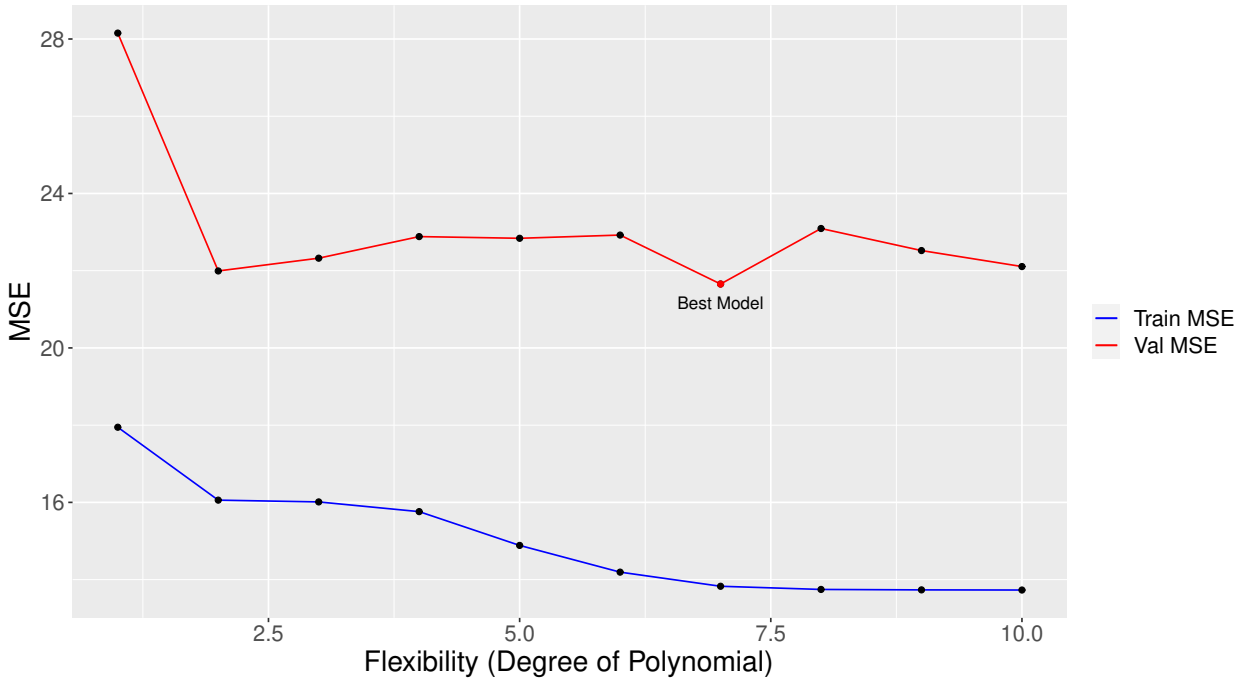
```
1  import numpy as np
2  from ISLP import load_data
3  Auto = load_data('Auto')  # Load data
4
5  n  = int(len(Auto)) # Number of observations in the dataset
6
7  # Set seed & shuffle data by row (randomize dataset)
8  np.random.seed(1)
9  Auto = Auto.sample(frac=1).reset_index(drop=True)
10
11 # Use (for example) 150 datappoints for training, 150 for validation and
   ↪  rest for testing
12 train_data = Auto.iloc[:150]   # training dataset
13 val_data = Auto.iloc[150:300]  # validation data
14 test_data = Auto.iloc[300:n]   # test dataset
```

Training and Validation Error for Different Models



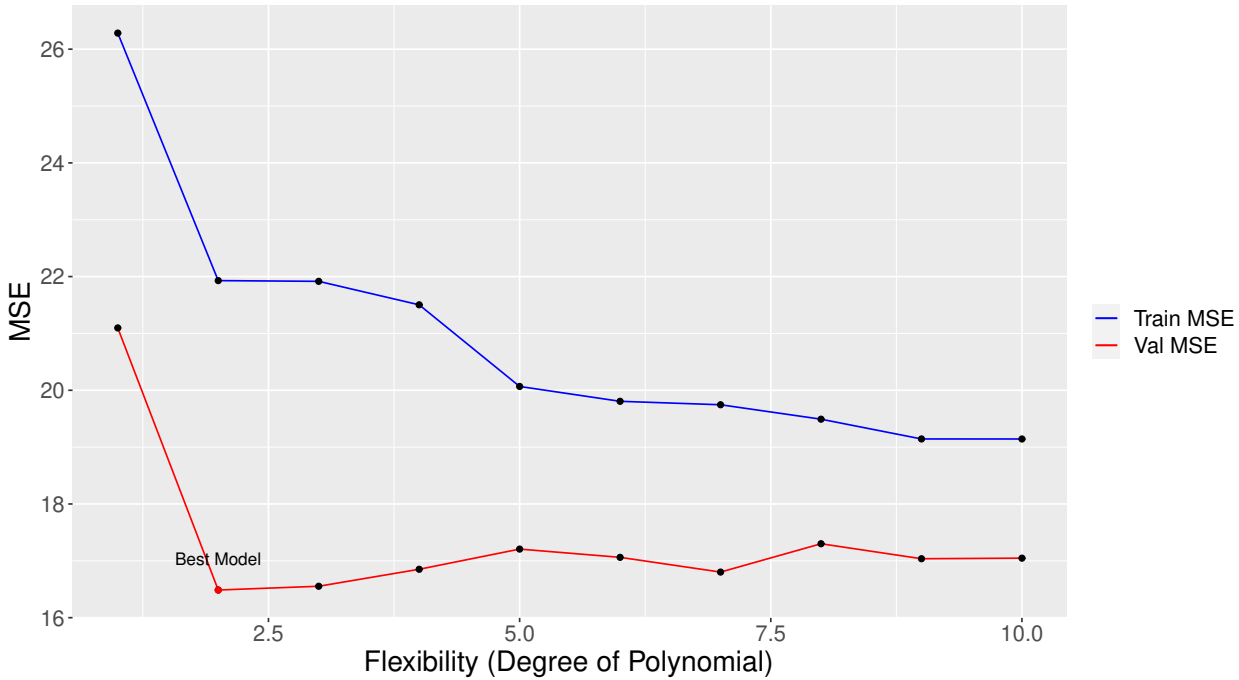

```
# Display Training, Validation and Test MSE of best model
print(f"{np.argmin(train_mse):.3f}, {np.argmin(val_mse):.3f},
↪  {testmse:.3f}")
> 17.41467 19.21416 19.06671
```

Same Figure but using different split (seed) of the data



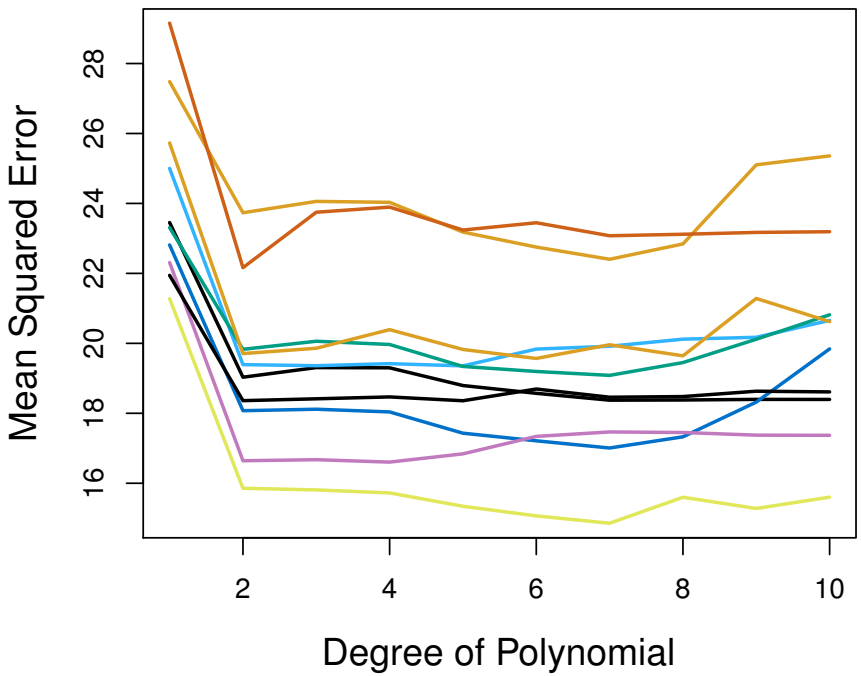

```
# Display Training, Validation and Test MSE of best model
print(f"{np.argmin(train_mse):.3f}, {np.argmin(val_mse):.3f},
↪  {testmse:.3f}")
> 13.82735 21.65498 22.79760
```

Same Figure but using different split (seed) of the data




```
1 # Display Training, Validation and Test MSE of best model
2 print(f"{np.argmin(train_mse):.3f}, {np.argmin(val_mse):.3f},
  ↪  {testmse:.3f}")
3 > 21.92878 16.48608 18.53450
```

Validation Error for Multiple Splits of the Data



The figure plots validation errors for different degrees of flexibility and for different splits of the data (different seeds)

$\Rightarrow$ Validation errors are highly dependent on how we split the data

Drawbacks of validation set approach

- The validation estimate of the test error can be highly variable
- Different splits of the data lead to different estimates of the test error rate.
- So the approach is unreliable if small changes in the split between training set and validation set have large effect on estimated errors
- This will especially be the case in small data samples and for highly flexible methods
- So how do we decide which model to use?
- **Possible solution**: k-fold cross validation

K-fold Cross-validation

- Widely used approach for estimating the test error
- Idea:
  - Randomly divide the training data into K equal-sized parts
  - Leave out part k
  - Fit the model to the other $K - 1$ parts (combined)
  - Compute validation errors for the left-out kth part
- This is done in turn for each part $k = 1, 2, ..., K$
- Obtain the final estimate for the test error by taking the average over the K validation errors

K-fold Cross-validation

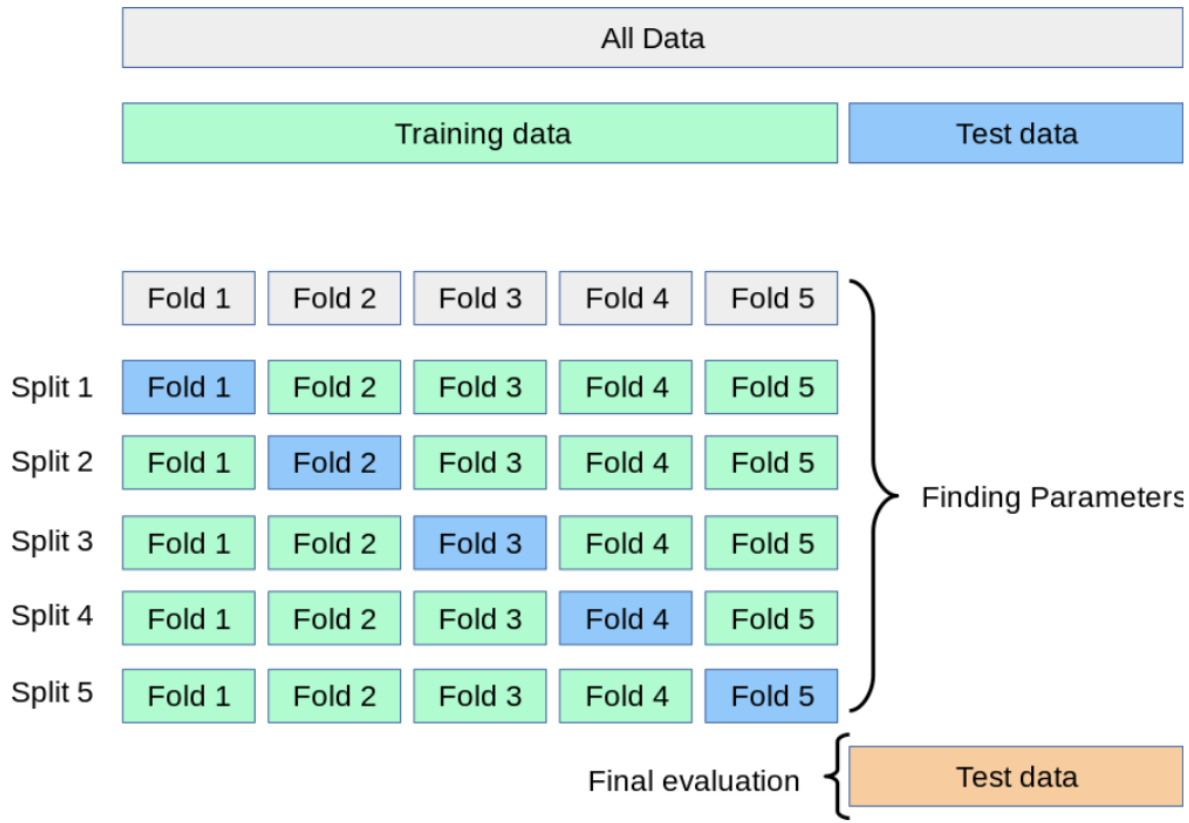


Figure from `https://scikit-learn.org/stable/modules/cross_validation.html`

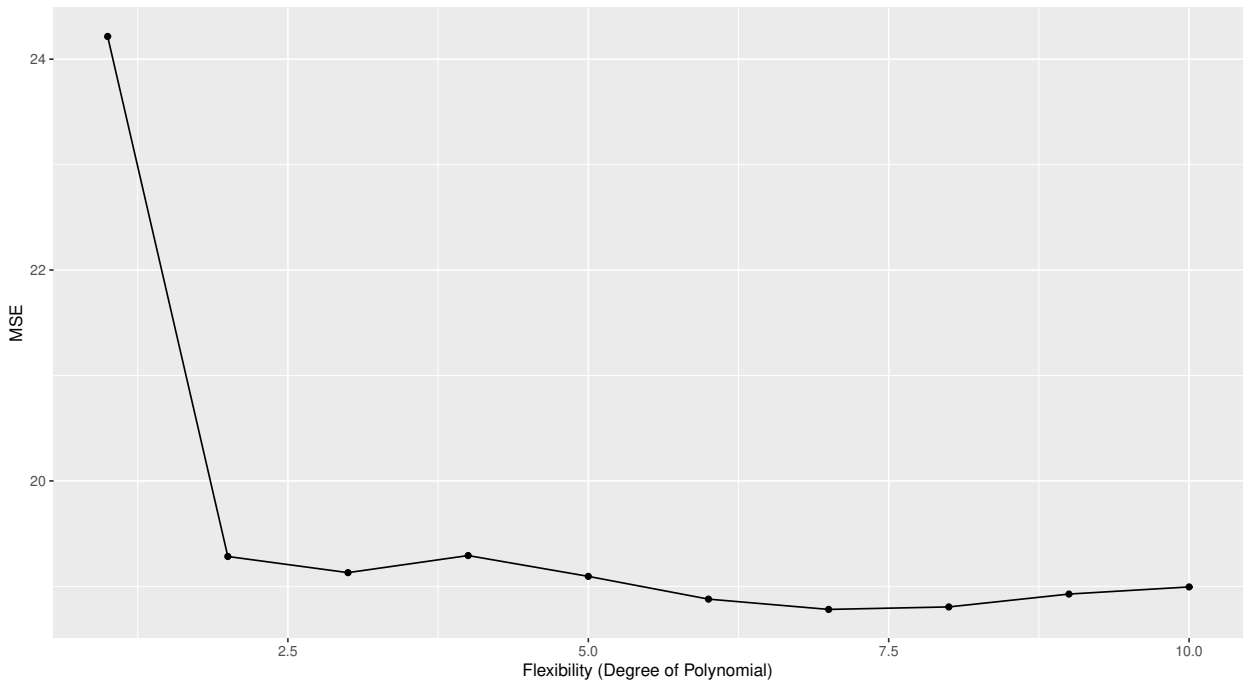
Python code: k-fold cross-validation

```
1  from sklearn.model_selection import cross_val_score
2  from sklearn.linear_model import LinearRegression
3  from ISLP import load_data
4  Auto = load_data('Auto')  # Load data
5
6  np.random.seed(2)  # Set seed
7
8  # Fit a specific model on training data
9  X = Auto[['horsepower']]
10 y = Auto['mpg']
11 model = LinearRegression().fit(X, y)
12
13 # Perform 10-fold cross-validation
14 folds = 10
15 cv_scores = cross_val_score(model, X, y, cv=folds,
   ↪  scoring='neg_mean_squared_error')
16 #print(f"cv_scores: cv_scores")
17
18 cv_error = -np.mean(cv_scores)
19 print(f"cv_error: {cv_error:.3f}")
20 > 24.21538
```

Python code: k-fold cross-validation

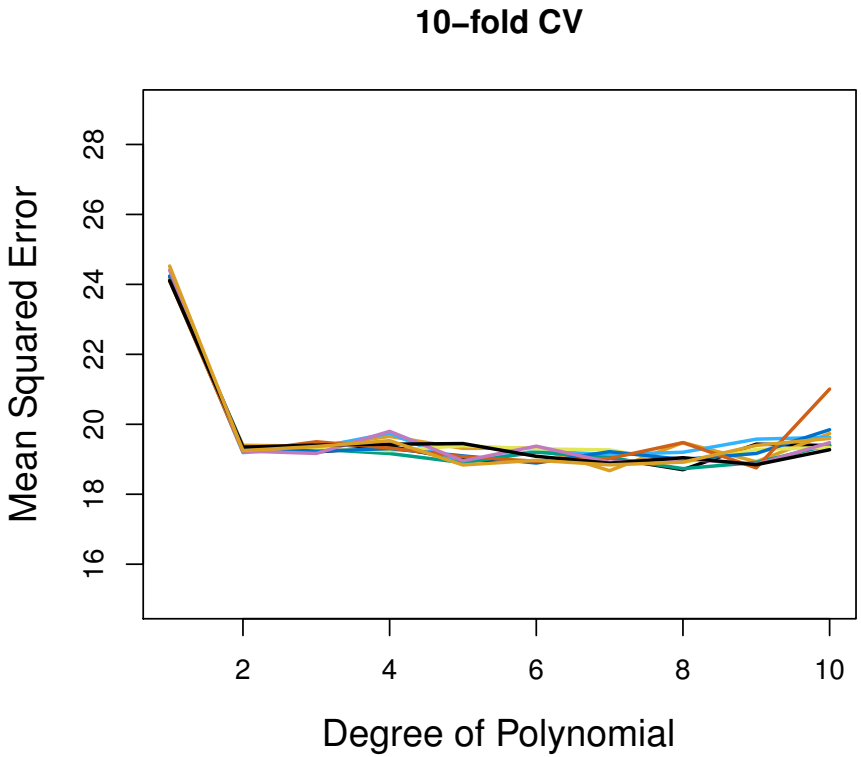
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

num_models = 10
cv_error2 = np.zeros(num_models)

# Loop through polynomial degrees and perform 10-fold cross-validation
 for i in range(1, num_models + 1):
 # Fit polynomial regression with degree i
 poly_features = PolynomialFeatures(degree=i)
 model = make_pipeline(poly_features, LinearRegression())

 # Perform 10-fold cross-validation
 pol_cv_scores = cross_val_score(model, X, y, cv=folds,
 ↪  scoring='neg_mean_squared_error')

 # Save MSE from k-fold CV for each model
 pol_cv_error[i - 1] = -np.mean(pol_cv_scores)

 print("pol_cv_error:", np.mean(pol_cv_error))
> 18.71538
```

K-fold CV for different models



The figure plots validation errors obtained from 10-fold cross validation

Multiple K-fold CV for the *Auto* Data



The figure plots validation errors obtained from multiple 10-fold cross validations

$\Rightarrow$ Variability of validation errors is much less dependent on how we split the data.

K-fold CV for Regression and Classification Problems

- K-fold cross validation can be used for both, regression and classification problems
- In regression we compute the average mean squared prediction error over the different folds
- In classification we compute the average error rate (or accuracy) over the different folds

Some remarks on K-Fold Cross-Validation

- Since each training set is only $(K-1)/K$ as big as the original training set, the estimates of prediction error will typically be biased upward
- Hence, once you have selected the best model using K-fold CV, use the hole sample (training + validation) to fit this model again.
- This model you can use to obtain out-of-sample predictions
- Typical split of the data: Use 70% for training (30% for testing) and 5- or 10-fold cross validation
- For very large datasets use up to 90% of the data for training

Cross-Validation for Time-Series Data

- With time-series data the ordering of the data should be considered
- For a fix split, use early sample for training, the sample afterwards for validation and the most recent data for testing
- k-fold CV can be used to validate a method (note, it does **not** account for the ordering of the data)
- Alternative approach: use rolling or recursive schemes:
  - Rolling: the training and validation samples are gradually shifted forward in time to include more recent data. The total number of time periods in each training and validation sample is held fixed
  - Recursive: similar to rolling except that it includes all historical data