UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

Data Analysis & Pattern Recognition

Validation Procedures. Leave-one-out cross-validation (LOOCV) and *k*-fold cross-validation

Francesc Pozo

Escola d'Enginyeria de Barcelona Est (EEBE)
Universitat Politècnica de Catalunya (UPC)

Master's Degree in Chemical Engineering
Master's Degree in Interdisciplinary and Innovative Engineering

Simple linear regression

- Simple linear regression is a very straightforward approach for predicting a quantitative response Y on the basis of a single predictor variable X. It assumes that there is approximately a linear relationship between X and Y. Mathematically, we can write this linear relationship as

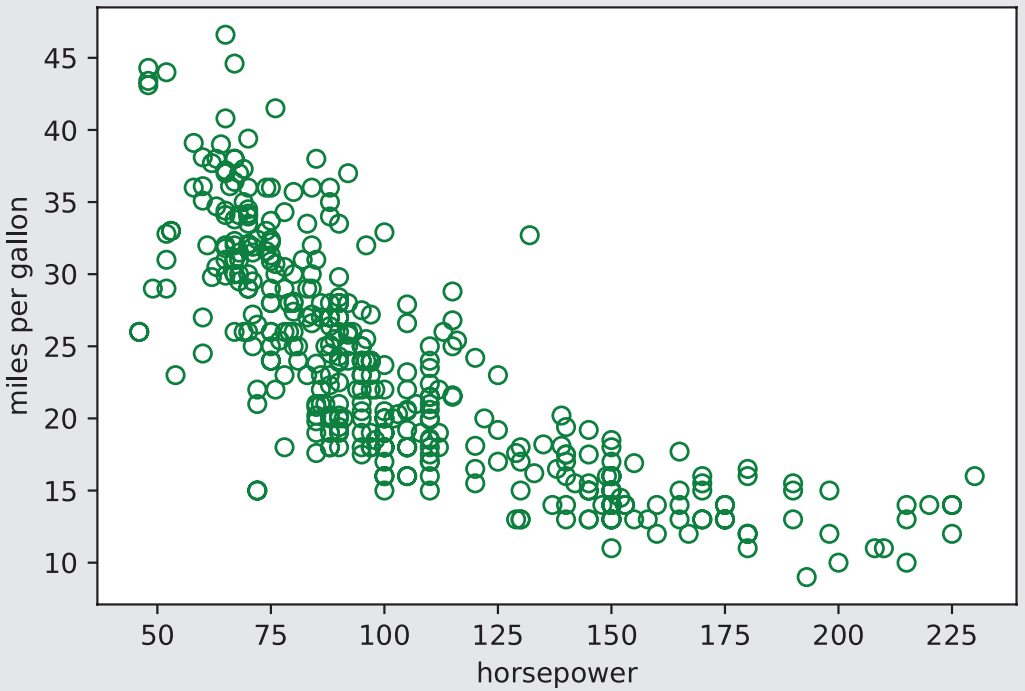
$$\boxed{Y \approx \beta_0 + \beta_1 X.}$$

Example

For example, X may represent `horsepower` and Y may represent `mpg` (miles per gallon) (with respect to the `Auto` data set). Then we can regress `mpg` onto `horsepower` by fitting the model

$$\boxed{\texttt{mpg} \approx \beta_0 + \beta_1 \times \texttt{horsepower}.}$$

Example



The `Auto` data set. For a number of cars, `mpg` and `horsepower` are shown. There is a pronounced relationship between `mpg` and `horsepower`, but it seems clear that this relationship is in fact non-linear: the data suggest a curved relationship.

Cross-validation

Python code

```
>>%matplotlib inline
>>import csv
>>import pandas as pd
>>import numpy as np
>>import matplotlib.pyplot as plt
>>filename = "Auto.csv"
>>df = pd.read_csv(filename)
>>moddf = df.dropna()
>>v = moddf.values
>>plt.xlabel('horsepower')
>>plt.ylabel('miles per gallon')
>>plt.scatter(v[:,3], v[:,0],s=40,facecolors='none', edgecolors=
    'g')
>>plt.savefig('scatterAuto.eps', dpi=300, bbox_inches='tight')
>>plt.show()
```

Non-linear relationships

- In some cases, the true relationship between the response and the predictors may be non-linear. Here we present a very simple way to directly extend the linear model to accomodate non-linear relationships, using polynomial regression.

$$\boxed{Y \approx \beta_0 + \sum_{i=1}^{N} \beta_i X^i.}$$

Example

For example, with respect to the `Auto` data set, we can regress `mpg` onto `horsepower` by fitting a quadratic model model

$$\boxed{\texttt{mpg} \approx \beta_0 + \beta_1 \times \texttt{horsepower} + \beta_2 \times \texttt{horsepower}^2.}$$

Measuring the quality of fit

- In order to evaluate the performance of a statistical learning method on a given data set, we need some way to measure how well its predictions actually match the observed data. In the regression setting, the most commonly-used measure is the mean squared error (MSE), given by

$$\boxed{\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}(x_i)\right)^2,}$$

where $\hat{f}(x_i)$ is the prediction that $\hat{f}$ gives for the ith observation.

Measuring the quality of fit

- When the MSE

$$\boxed{\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}(x_i)\right)^2}$$

  is computed using the training data, we refer to the training MSE.
- However, we are more interested in the accuracy of the predictions that we obtain when we apply our method to previously unseen test data. In this case, we refer to the test MSE (aka test error).

Training error versus test error

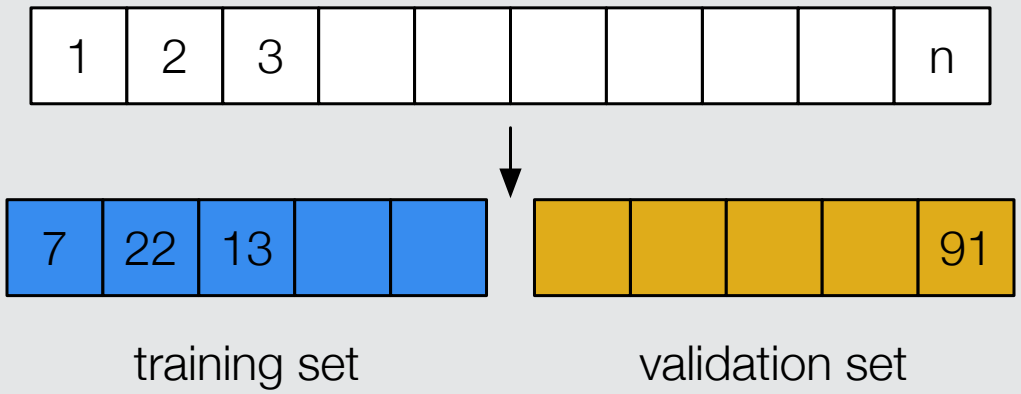
- The test error is the average error that results from using a statistical learning method to predict the response on a new observation, one that was not used in training the method.
- In contrast, the training error can be easily calculated by applying the statistical learning method to the observations used in its training.
- But the training error rate often is quite different from the test error rate, and in particular the former can dramatically underestimate the latter.

Validation-set approach

- Here we randomly divide the available set of samples into two parts: a training set and a validation or hold-out set.
- The model is fit on the training set, and the fitted model is used to predict the responses for the observations in the validation set.
- The resulting validation-set error provides an estimate of the test error. This is typically assessed using MSE in the case of a quantitative response and misclassification rate in the case of a qualitative (discrete) response.

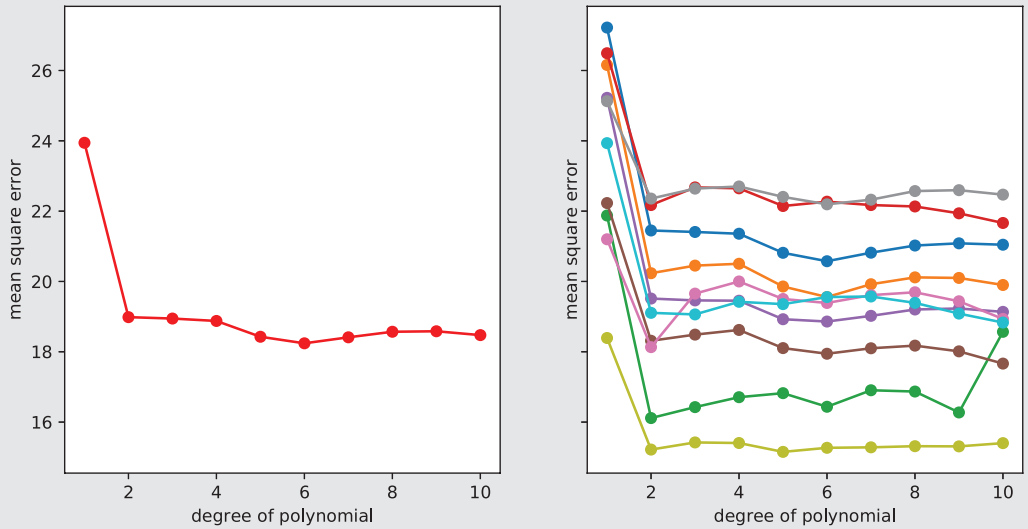
The validation process

- A random splitting into two halves: left part is training set, right part is validation set



Example

- The validation set approach was used on the `Auto` data set in order to estimate the test error that results from predicting `mpg` (miles per gallon) using polynomial functions of `horsepower` in a linear regression.
- We randomly split the 392 observations into two sets, a training set containing 196 of the data points, and a validation set containing the remaining 196 observations.



- *Left*: single split; *Right*: multiple splits.

Cross-validation

Python code

```
>>import numpy as np
>>import pandas as pd
>>import csv
>>from sklearn.linear_model import LinearRegression
>>from sklearn.preprocessing import PolynomialFeatures
>># loading data
>>filename = "Auto.csv"
>>df = pd.read_csv(filename)
>>moddf = df.dropna()
>>v = moddf.values
```

Cross-validation

Python code

```
>>x = v[:,3].reshape((-1,1)) #input data / regressor
>>y = v[:,0]
>>mse0 = list()
>>for deg in range(1,11):
      x_ = PolynomialFeatures(degree=deg, include_bias=False).
    fit_transform(x)
      model = LinearRegression().fit(x_, y)
      y_pred = model.predict(x_)
      mse0.append(np.sum((y-y_pred)**2)/len(y)) # MSE as a
    function of the degree of polynomial
```

Cross-validation

Python code

```
>>np.random.seed(3)
>>mse_all = np.zeros((10,10))
>>for resamp in range (0,10):
      nprc = np.random.choice(392,392,replace=False)
      xtrain = v[nprc[0:196],3].reshape((-1,1))
      ytrain = v[nprc[0:196],0]
      xtest = v[nprc[196:392],3].reshape((-1,1))
      ytest = v[nprc[196:392],0]
      mse = list()
      for deg in range(1,11):
          xtrain_ = PolynomialFeatures(degree=deg, include_bias=False).fit_transform(xtrain)
          xtest_ = PolynomialFeatures(degree=deg, include_bias=False).fit_transform(xtest)
          model = LinearRegression().fit(xtrain_, ytrain)
          y_pred = model.predict(xtest_)
          mse.append(np.sum((ytest-y_pred)**2)/len(ytrain)) # list of MSE as a function of the degree of poly
      mse_all[resamp,:] = mse
```

Python code

```
>>%matplotlib inline
>>import matplotlib.pyplot as plt
>>f, axes = plt.subplots(1, 2, figsize=(10, 5), sharey=True)
>>axes[0].plot(np.arange(1,11,1),mse0,'o-',color='r')
>>axes[0].set_ylabel('mean square error')
>>axes[0].set_xlabel('degree of polynomial')
>>for resamp in range(0,10):
      axes[1].plot(np.arange(1,11,1),mse_all[resamp,:],'o-')
>>axes[1].set_ylabel('mean square error')
>>axes[1].set_xlabel('degree of polynomial')
>>plt.savefig('mse2fold.eps', dpi=300, bbox_inches='tight')
>>plt.show()
```

Drawbacks of the validation set approach

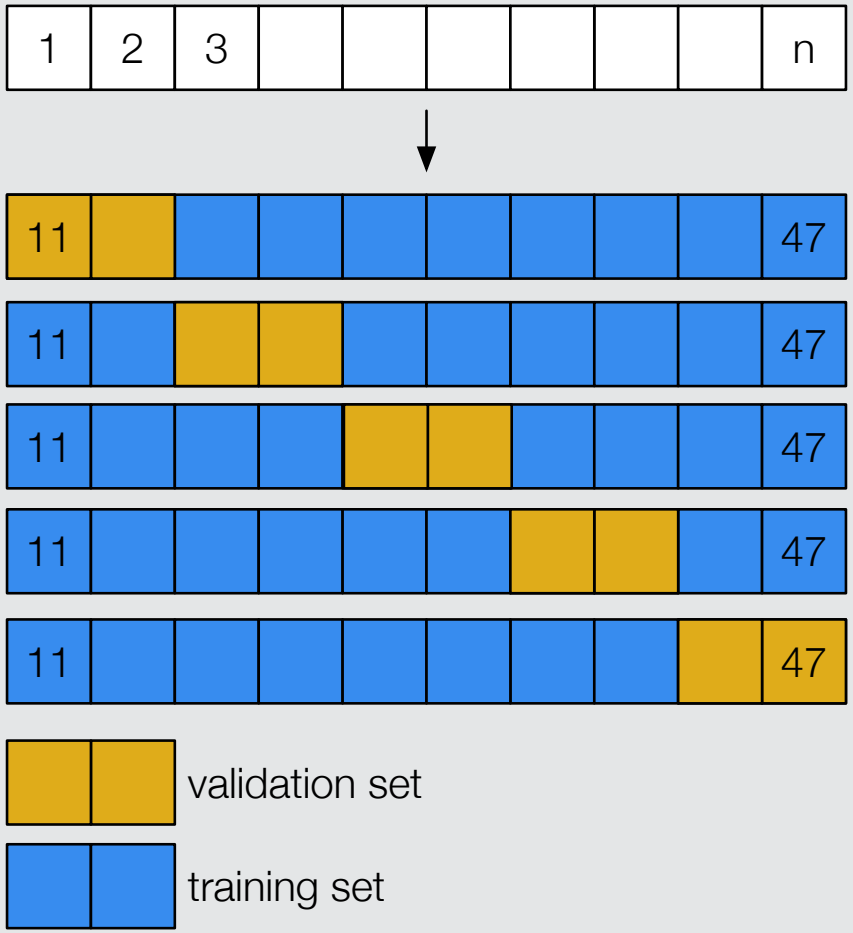
- The validation estimate of the test error can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set.
- In the validation approach, only a subset of the observations — those that are included in the training set rather than in the validation set— are used to fit the model.
- This suggests that the validation set error may tend to overestimate the test error for the model fit on the entire data set.

k-fold cross-validation

- Widely used approach for estimating test error.
- Estimates can be used to select best model, and to give an idea of the test error of the final chosen model.
- Idea is to randomly divide the data into k equal-sized parts. We leave out part k, fit the model to the other $k-1$*parts* (combined), and then obtain predictions for the left-out kth part.
- This is done in turn for each part $j = 1, 2, \ldots, k$, and then the results are combined.

k-fold cross-validation in detail

- A set of n observations is randomly split into k non-overlapping groups.



k-fold cross-validation in detail

- Let the k parts be $C_1, C_2, \ldots, C_k$, where C_j denotes the indices of the observations in part j. There are n_j observations in part k. If n is a multiple of k, then $n_j = n/k$.
- Compute

$$\boxed{CV_{(k)} = \sum_{j=1}^{k} \frac{n_j}{n} MSE_j,}$$

where

$$MSE_j = \sum_{i \in C_j} \frac{(y_i - \hat{y}_i)^2}{n_j},$$

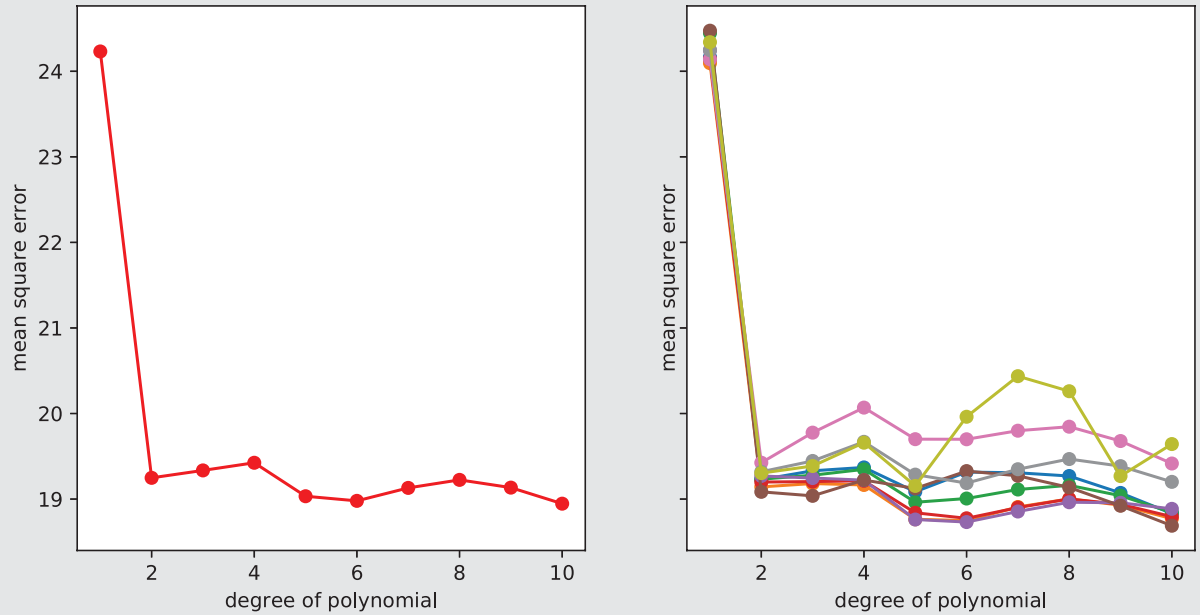
and $\hat{y}_i$ is the fit for observation i, obtained from the data with part j removed.

Leave-one-out cross-validation (LOOCV)

- Setting $k = n$ yields n-fold or leave-one-out cross-validation (LOOCV).
- LOOCV sometimes useful, but typically does not shake up the data enough. The estimates from each fold are highly correlated and hence their average can have high variance.
- A better choice is $k = 5$ or $k = 10$ (or a similar divisor of n).

Example

- Cross-validation was used on the `Auto` data set in order to estimate the test error that results from predicting `mpg` (miles per gallon) using polynomial functions of `horsepower`.
- *Left:* The LOOCV error curve. *Right:* 7-fold CV was run nine separate times, each with a different random split of the data into seven parts. The figure shows the nine slightly different CV error curves.



Cross-validation

Python code

```
>>mse_LOOCV = np.zeros((1,10))
>>mse_fold = np.zeros((392,10))
>>for fold in range(0,392):
      xtest = v[fold:fold+1,3].reshape((-1,1))
      ytest = v[fold:fold+1,0]
      aux_list = np.setdiff1d(np.arange(0,392,1),[fold])
      xtrain = v[aux_list,3].reshape((-1,1))
      ytrain = v[aux_list,0]
      mse = list()
      for deg in range(1,11):
          xtrain_ = PolynomialFeatures(degree=deg, include_bias=
    False).fit_transform(xtrain)
          xtest_ = PolynomialFeatures(degree=deg, include_bias=
    False).fit_transform(xtest)
          model = LinearRegression().fit(xtrain_, ytrain)
          y_pred = model.predict(xtest_)
          mse.append(np.sum((ytest-y_pred)**2)/len(ytest)) #
    list of MSE as a function of the degree of poly
      mse_fold[fold,:] = mse
>>mse_LOOCV = np.mean(mse_fold, axis=0)
```

Cross-validation

Python code

```
>># k—fold CV
>>k = 7
>>block = int(392/k)
>>np.random.seed(3)
>>mse_all2 = np.zeros((9,10)) # k—fold CV was run 9 separate times
>>for resamp in range (0,9):
      nprc = np.random.choice(392,392,replace=False)
      mse_fold = np.zeros((k,10))
      for fold in range(0,k):
          xtest = v[nprc[fold*block:(fold+1)*block],3].reshape((—1,1))
          ytest = v[nprc[fold*block:(fold+1)*block],0]
          aux_list = np.setdiff1d(nprc,nprc[fold*block:(fold+1)*block])
          xtrain = v[aux_list,3].reshape((—1,1))
          ytrain = v[aux_list,0]
          mse = list()
          for deg in range(1,11):
              xtrain_ = PolynomialFeatures(degree=deg, include_bias=False).fit_transform(xtrain)
              xtest_ = PolynomialFeatures(degree=deg, include_bias=False).fit_transform(xtest)
              model = LinearRegression().fit(xtrain_, ytrain)
              y_pred = model.predict(xtest_)
              mse.append(np.sum((ytest—y_pred)**2)/len(ytest)) # list of MSE as a function of the
      degree of poly
          mse_fold[fold,:] = mse
      mse_all2[resamp,:] = np.mean(mse_fold, axis = 0)
```

Python code

```
>>%matplotlib inline
>>import numpy as np
>>import matplotlib.pyplot as plt
>>f, axes = plt.subplots(1, 2, figsize=(10, 5), sharey=True)
>>axes[0].plot(np.arange(1,11,1),mse_LOOCV,'o-',color='red')
>>axes[0].set_ylabel('mean square error')
>>axes[0].set_xlabel('degree of polynomial')
>>for resamp in range(0,9):
      axes[1].plot(np.arange(1,11,1),mse_all2[resamp,:],'o-')
>>axes[1].set_ylabel('mean square error')
>>axes[1].set_xlabel('degree of polynomial')
>>plt.savefig('msekfold.eps', dpi=300, bbox_inches='tight')
>>plt.show()
```

Cross-validation on classifications problems

- We divide the data into k roughly equal-sized parts $C_1, C_2, \ldots, C_k$, where C_j denotes the indices of the observations in part j. There are n_j observations in part k. If n is a multiple of k, then $n_j = n/k$.
- Compute

$$\boxed{CV_{(k)} = \frac{1}{n} \sum_{j=1}^{k} \text{Err}_j,}$$

where

$$\text{Err}_j = \sum_{i \in C_j} \mathbb{I}(y_i \neq \hat{y}_i)$$

is the number of misclassified observations.