Evaluation and Resampling

Elsa Ferreira Gomes

2023/2024

Resampling Methods

Resampling methods are an indispensable tool and involve repeatedly drawing samples from a training set and refitting a model of interest on each sample in order to obtain additional information about the fitted model.
For example: To estimate the variability of a linear regression fit

- we can repeatedly draw diferent samples from the training data,
- fit a linear regression to each new sample
- and then examine the extent to which the resulting fits difer.

Resampling

Such an approach may allow us to obtain information that would not be available from fitting the model only once using the original training sample.

Resampling approaches can be computationally expensive

- they involve fitting the same machine learning method multiple times using diferent subsets of the training data.
- due to recent advances in computing power, the computational requirements of resampling methods generally are not prohibitive.

Two of the most commonly used resampling methods

- Cross-validation
- Bootstrap.

Model Evaluation

- We want to **estimate** how well the model performs with **unknown** cases
  - makes good predictions
  - makes good diagnoses
  - assigns correctly an email to a folder

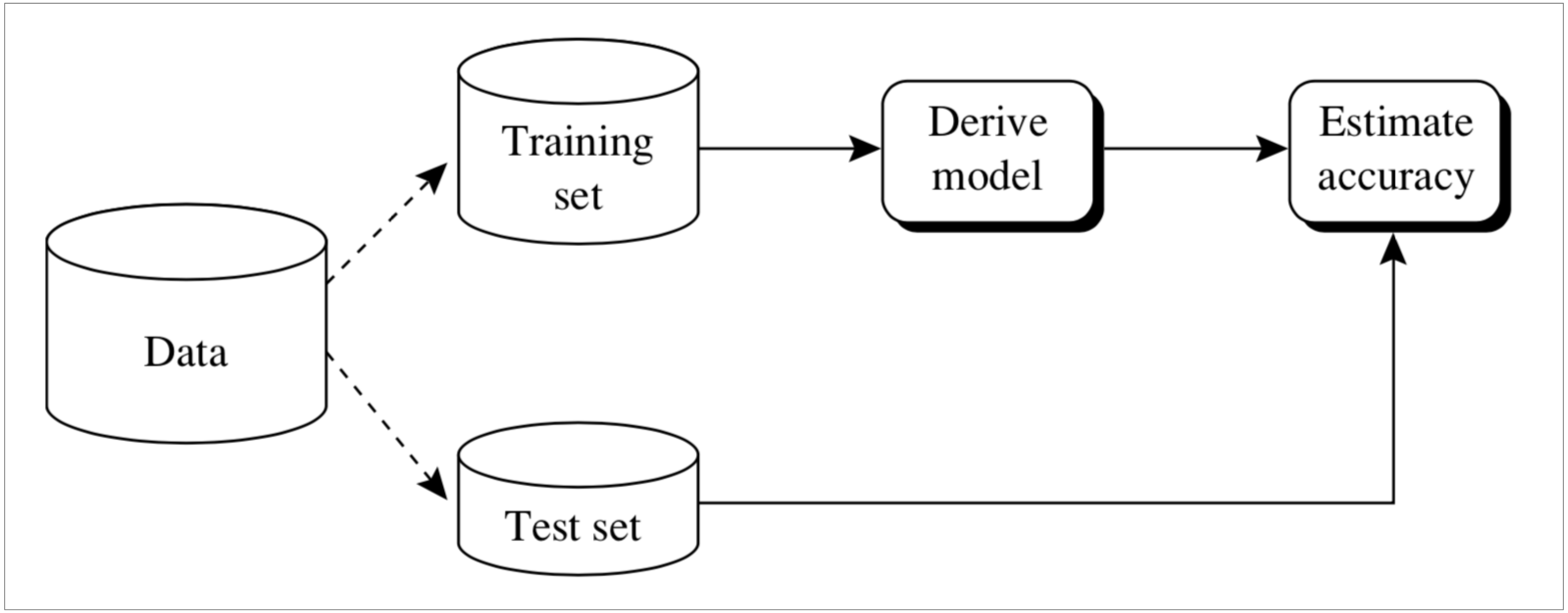
Model Evaluation: model selection

- We have, e.g, a classification problem
    - Is Logist Regression **better** than Linear Discriminant Analysys?
- **Model Selection**
    - Obtain a model with each one (on the same data)
    - Identify the model that performs better (on the same data)

Training Error vs Test error

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

Estimating evaluation metrics



- **Option 1**: Estimate accuracy on the training examples
    - Estimate tends to be **optimistic**
    - Very bad choice
- **Option 2**: Isolate a subset for testing (**holdout**)
    - Estimate is more **reliable**
    - Estimate tends to be **pessimistic**
    - Depends on **sampling**
    - Depends on the **sizes** of the train and test set

Metrics for Evaluating Classifier Performance

- How to **assess** a classifier?
    - many metrics
- Most popular
    - Accuracy
    - Recall
    - Precision
    - F1
    - Sensitivity
    - Specificity

Recall

$$Recall = \frac{TP}{P}$$

- A.k.a.
  - **True Positive Rate**
  - **Sensitivity**
    - How sensitive is the model to the positive class?

Precision

- Are all our decisions good?
  - If $Precision = 1$ we only say right things

$$Precision = \frac{TP}{TP + FP}$$

- **Note the following**
  - Management wants to get more clients and asks for a model with higher **recall**.
  - Data scientists say: "we risk getting more bad clients too"
  - When Recall increases Precision **tends** to go down
  - and vice-versa

Accuracy

- **Accuracy**
    - the test asks $Total$ questions
    - each question is equally important
    - the model gets $Right$ questions right
    - the proportion of right answers

$$Accuracy = \frac{Right}{Total}$$

- **Error**
    - $Error = 1 - Accuracy$

The confusion matrix

- My credit decision model has an accuracy of 64% (or 0.64, we can say either way)
- How good is it on each class?
- The **Confusion Matrix**
    - where are the errors?
    - we can see that 24 loans are wrongly given
    - and 12 are wrongly denied

classified as->	**loan**	**no loan**
loan	43	12
no loan	24	21

Binary classification

- When we are learning a **concept** of interest
  - e.g. a good credit client, the presence of a disease
- We have
  - **positive examples**
  - **negative examples**
- Depending on how a model classifies a test case

classified as	loan	no loan
loan	True Positives	False Negatives
no loan	False Positives	True Negatives

Redefining Accuracy

$$Accuracy = \frac{TP + TN}{P + N}$$

- In other words
  - The main diagonal divided by the sum of all the matrix

F_1

- How to combine recall and precision?
    - calculate the **harmonic mean**

$$F_1 = \frac{2 \times Recall \times Precision}{Recall + Precision}$$

- F_1 is
    - also known as **F-score**
    - low if **either** recall or precision are low
    - equal to Recall and Precision if $Recall = Precision$
    - generalised by F_β

Specificity

- Are we excluding all the bad clients?
    - If $Specificity = 1$ we identify all bad clients (and hopefully some good ones)

$$Specificity = \frac{TN}{N}$$

- A.k.a.
    - **True Negative Rate**
    - Used in medical applications
        - negatives are patients without the disease

An example

- Loans are the core business of loan companies/banks. The main profit comes directly from the loan’s interest. The loan companies grant a loan after an intensive process of verification and validation. However, they still don’t have assurance if the applicant is able to repay the loan with no difficulties.

classified as	loan	no loan
loan	43	12
no loan	32	21

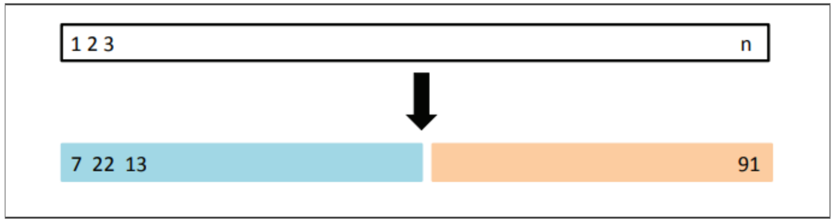
- **Accuracy** is $(43+21)/(32+12+43+21)=0.59$
- **Recall** ('loan' as positive) is $43/(43+12)=0.78$
    - the bank identifies 78% of the good clients, but 22% are missed
- **Precision** is $43/(43+32)=0.57$
    - 57% of the loans given would fail
- **Specificity** is $21/(21+32)=0.40$
    - the bank only detects 40% of the 'bad' clients
- **F1** is $2\times 0.78\times 0.57/(0.78+0.57)=0.66$
    - there is a relatively good balance of recall and precision

Holdout

- **Holdout** evaluation method
    - separate data set in **train** and **test**
- use train to **learn** the model, and test to **assess** it

In [1]:
```
from IPython.display import Image
Image("cv_5_1.png")
```

Out[1]:



Dealing with assessment variance

- A problem with holdout is **variance** of the evaluation estimates
    - Accuracy is also a **random variable**
    - We do not know its **true value** or **distribution**
- Solutions:
    - Testing with **more data**
        - reduces variance
        - if there is more data
    - **Repeating** the train-test cycle
        - reduces variance
        - enables **studying the distribution** of the measure (e.g. accuracy)
        - but we need a different sample for each iteration

K-fold Cross-validation

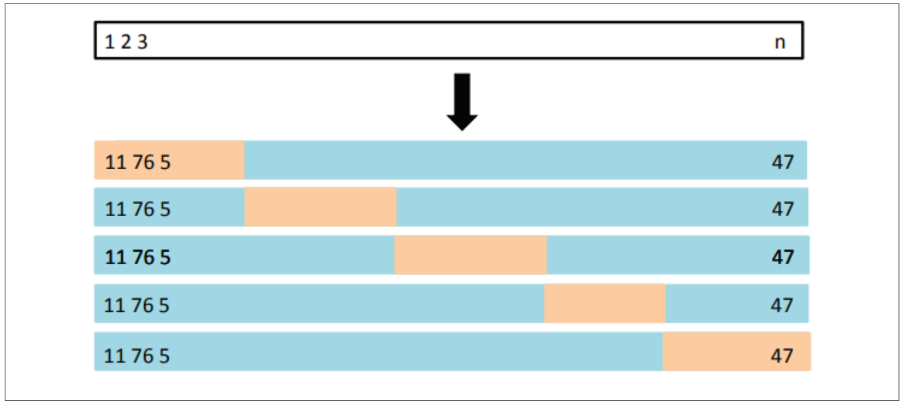
- Widely used approach for estimating test error.
- Estimates can be used to select best model, and to give an idea of the test error of the final chosen model.
- Idea:
    - Randomly divide the data into K equal-sized parts.
    - We leave out part k, fit the model to the other $K-1$ parts
    - Obtain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

Repeating train-test: k-fold Cross-validation

- In each iteration $i \in 1. \dots, k$
  - use fold i for testing
  - use the other $k-1$ folds for training
  - obtain Acc_i
- Study the distribution of the Acc_i

In [2]:
```
from IPython.display import Image
Image("cv_5_5.png")
```

Out[2]:



Repeating train-test: k-fold Cross-validation

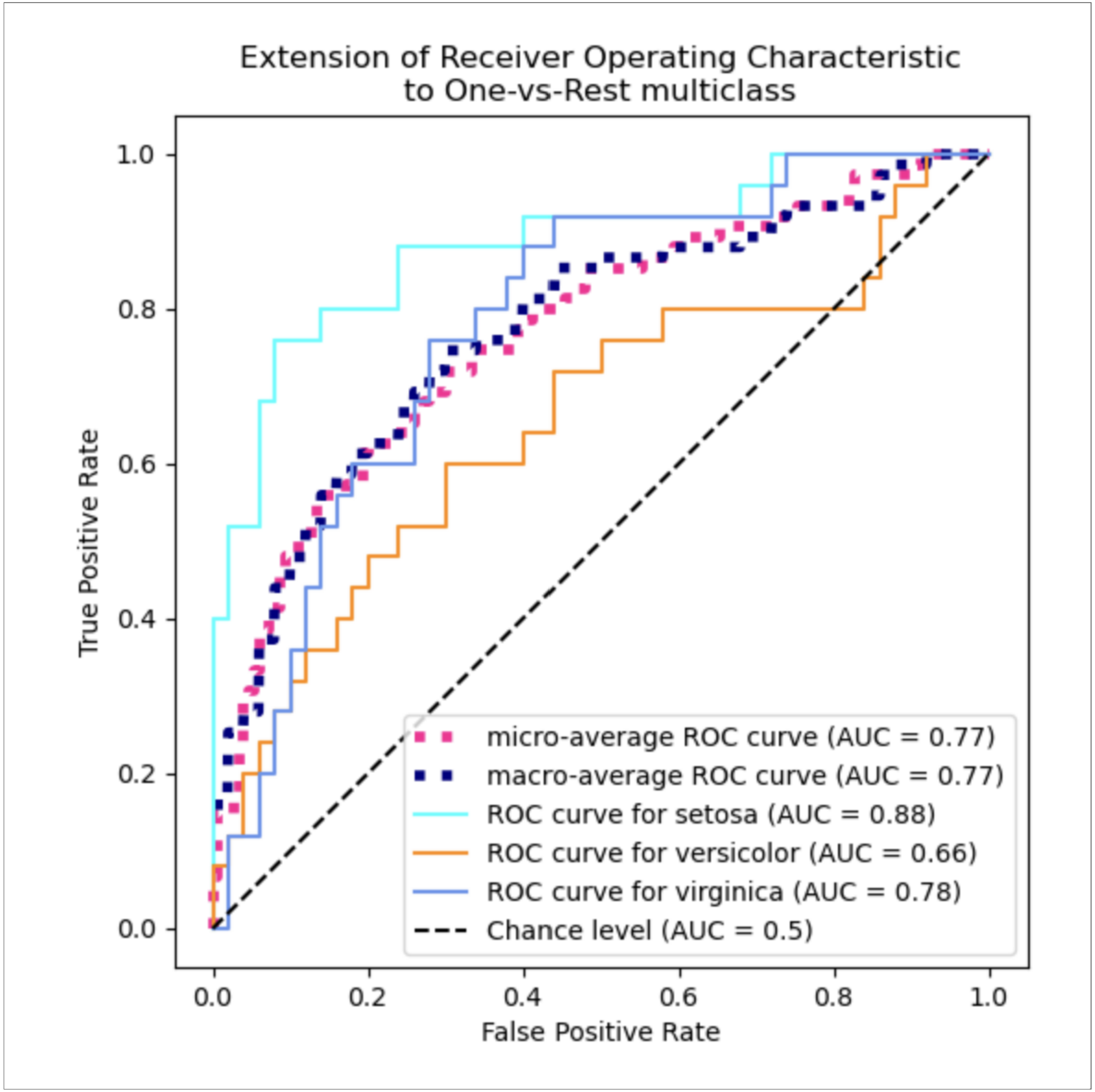
- Each sample is used **exactly once**
- Test sets are **independent**
    - Training sets are not
- What if we have a **small class**?
    - **stratified cross validation** to avoid under representation
- If we have **very few examples**?
    - **leave one out** cross validation
    - also leave p out for other (small) values of p
- **How many** folds should we use?
    - typical: 10 fold cross validation (10 fold CV)
    - (Demsar 2006) 2 times 5 fold CV
- **Shufling** before splitting may be a good idea

Leave-One-Out Cross-Validation (LOOC)

- Leave-one-out
  - If the sample is very small we can make $K = N$
  - We train with **all examples but one** in each iteration
  - Leave-one-out still estimates Expected Test Error

ROC curves

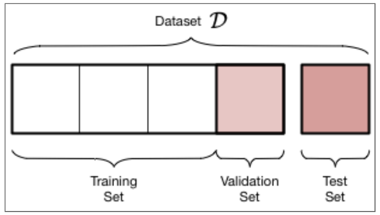
- We can compare the performance of classifiers on the whole spectrum of misclassification costs
- **Receiver operating characteristic curves**
    - plot relation between TPR and FPR
    - the **AUC**, area under the curve, is an assessment measure


Extension of Receiver Operating Characteristic
to One-vs-Rest multiclass
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
micro-average ROC curve (AUC = 0.77)
macro-average ROC curve (AUC = 0.77)
ROC curve for setosa (AUC = 0.88)
ROC curve for versicolor (AUC = 0.66)
ROC curve for virginica (AUC = 0.78)
Chance level (AUC = 0.5)

(**https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html**)

Validation, Test and deployment

- We can use cross-validation for **tuning**
  - hyperparamater tuning, pre-processing decisions, ...
- In after tuning (and other decisions) use a **new test set**
- The **internal test set** is the **validation set** (or sets)



Validation, Test and deployment

- **Why** do we need **validation and test ?**
    - using the test set to improve results leads to overly **optimistic** results
        - it is like **using the future to make predictions**
        - or **knowing the exam questions** when you study
        - but we do it in the lab as long as comparisons are fair
    - the test set should be for **testing only**
- Which model we use in **deployment** ?
    - We use the approach and hyperparameters that had best test results
    - We can **then** use the whole data to train the model
        - if data is scarce. We can use less data too

Measuring statistical significance

- We compare two algorithms **A** and **B**
    - **A** has 0.8832 accuracy
    - **B** has 0.8845 accuracy
- Is this difference **important**?
    - **Statistical significance**
        - the difference occurs most of the time
        - how likely is it to observe this difference or larger?
    - **Usefulness**
        - st. significant does not imply useful
        - does it save more lifes with fewer secondary effects?

Measuring statistical significance

- Use **Hypothesis testing**
- 10 fold CV example with t-test
  - obtain two samples of the accuracies: Acc_A and Acc_B
  - each sample has size 10
  - calculate the means $mean_A$ and $mean_B$
  - we assume that the accuracy values follow a **t-distribution** with $k-1$ degrees of freedom
  - we can use a paired **t-test**
  - H_0 **or Null hypothesis** is that the difference of means is zero
  - the **paired t-test** checks if we can reject H_0
  - the test **assumes independence** of the samples (not true)

More on statistical significance tests

- t-test with cross validation should be avoided
  - the independence of the values does not exist
  - it gives some information though
- if we have enough data to promote independence
  - t-test is acceptable
- To compare two algorithms on multiple datasets (e.g. 30)
  - cross-validate on each dataset
  - choose a **level of significance** (typically 1% to 5%)
  - if assumptions hold use a parametric test (**t-test**)
  - if not, use non-parametric **wilcoxon signed rank** test
- To compare many algorithms on multiple datasets
  - use **Friedman** test and post-hoc **Nemenyi** with **critical distances**
- Be **careful with multiple comparisons**
  - sometimes we have to **adjust** the p-values

In [3]:
```
# %load ../standard_import.txt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn.linear_model as skl_lm
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split, LeaveOneOut, KFold, cross_val_score
from sklearn.preprocessing import PolynomialFeatures

%matplotlib inline
#plt.style.use('seaborn-white')
```

Example: Auto dataset

Compare linear vs higher-order polynomial terms in a linear regression

```python
In [4]: df1 = pd.read_csv('Auto.csv', na_values='?').dropna()

df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 392 entries, 0 to 396
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int64  
 2   displacement  392 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        392 non-null    int64  
 5   acceleration  392 non-null    float64
 6   year          392 non-null    int64  
 7   origin        392 non-null    int64  
 8   name          392 non-null    object 
dtypes: float64(4), int64(4), object(1)
memory usage: 30.6+ KB
```

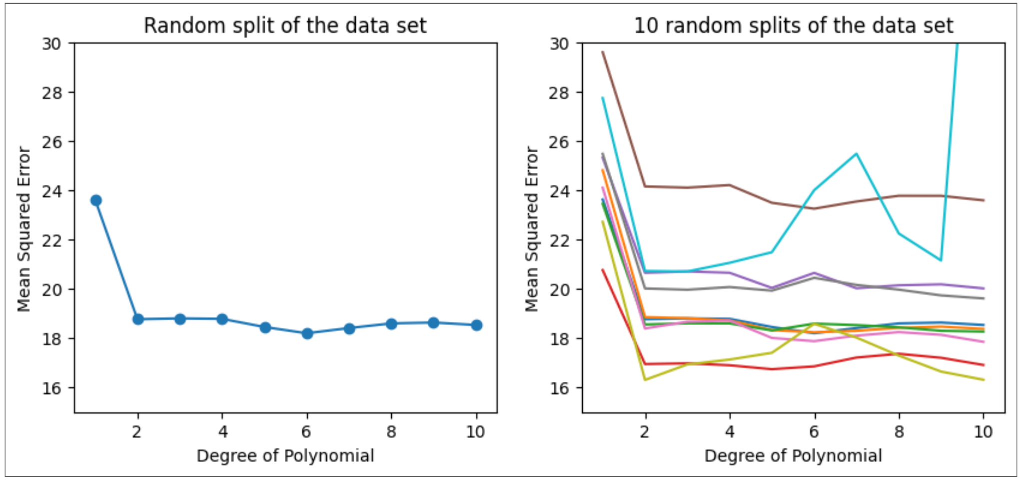
Cross-Validation

Validation Set Approach

- We randomly split the 392 observations into two sets:
  - training set containing 196 of the data points
  - validation set containing the remaining 196 observations.

Using Polynomial feature generation in scikit-learn
http://scikit-learn.org/dev/modules/preprocessing.html#generating-polynomial-features

```
In [5]: t_prop = 0.5
        p_order = np.arange(1,11)
r_state = np.arange(0,10)

X, Y = np.meshgrid(p_order, r_state, indexing='ij')
Z = np.zeros((p_order.size,r_state.size))

regr = skl_lm.LinearRegression()

# Generate 10 random splits of the dataset
for (i,j),v in np.ndenumerate(Z):
    poly = PolynomialFeatures(int(X[i,j]))
    X_poly = poly.fit_transform(df1.horsepower.values.reshape(-1,1))

    X_train, X_test, y_train, y_test = train_test_split(X_poly, df1.mpg.ravel(),
                                                        test_size=t_prop, random_state=Y[i,j])

    regr.fit(X_train, y_train)
    pred = regr.predict(X_test)
    Z[i,j]= mean_squared_error(y_test, pred)

fig, (ax1, ax2) = plt.subplots(1,2, figsize=(10,4))

# Left plot (first split)
ax1.plot(X.T[0],Z.T[0], '-o')
ax1.set_title('Random split of the data set')

# Right plot (all splits)
ax2.plot(X,Z)
ax2.set_title('10 random splits of the data set')

for ax in fig.axes:
    ax.set_ylabel('Mean Squared Error')
    ax.set_ylim(15,30)
    ax.set_xlabel('Degree of Polynomial')
    ax.set_xlim(0.5,10.5)
    ax.set_xticks(range(2,11,2));
```



Random split of the data set
10 random splits of the data set
Mean Squared Error
Degree of Polynomial
30
28
26
24
22
20
18
16
2
4
6
8
10

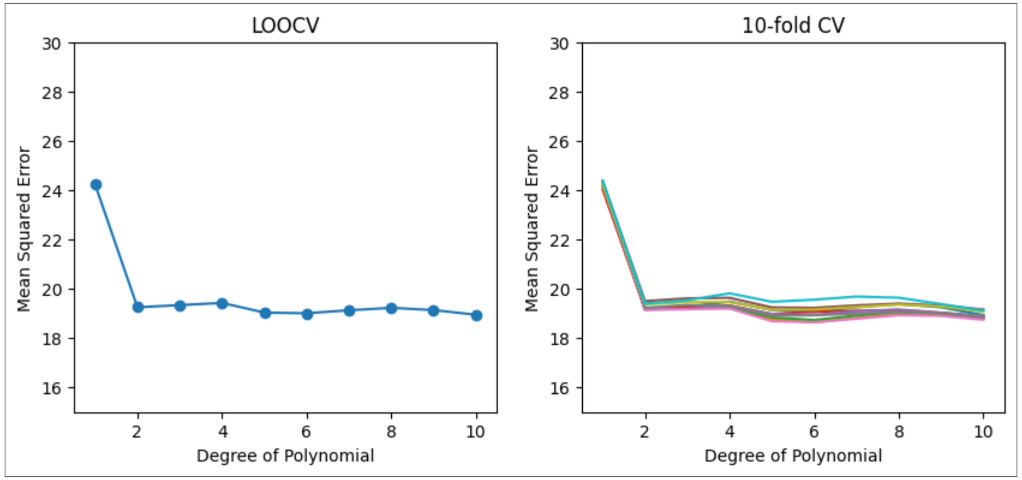
Leave-One-Out Cross-Validation (LOOC)

- Leave-one-out
    - If the sample is very small we can make $K = N$
    - We train with **all examples but one** in each iteration
    - Leave-one-out still estimates Expected Test Error

In [6]:
```
p_order = np.arange(1,11)
r_state = np.arange(0,10)

# LeaveOneOut CV
regr = skl_lm.LinearRegression()
loo = LeaveOneOut()
loo.get_n_splits(df1)
scores = list()

for i in p_order:
    poly = PolynomialFeatures(i)
    X_poly = poly.fit_transform(df1.horsepower.values.reshape(-1,1))
    score = cross_val_score(regr, X_poly, df1.mpg, cv=loo, scoring='neg_mean_squared_error').mean()
    scores.append(score)
```

In [7]:
```
# k-fold CV
folds = 10
elements = len(df1.index)

X, Y = np.meshgrid(p_order, r_state, indexing='ij')
Z = np.zeros((p_order.size,r_state.size))

regr = skl_lm.LinearRegression()

for (i,j),v in np.ndenumerate(Z):
    poly = PolynomialFeatures(X[i,j])
    X_poly = poly.fit_transform(df1.horsepower.values.reshape(-1,1))

    kf_10 = KFold(n_splits=folds, random_state=Y[i,j], shuffle=True)
    #kf_10 = KFold(n_splits=folds, random_state=None, shuffle=False)

    Z[i,j] = cross_val_score(regr, X_poly, df1.mpg, cv=kf_10, scoring='neg_mean_squared_error').mean()
```

In [8]:
```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(10,4))

# Note: cross_val_score() method return negative values for the scores.
# https://github.com/scikit-learn/scikit-learn/issues/2439

# Left plot
ax1.plot(p_order, np.array(scores)*-1, '-o')
ax1.set_title('LOOCV')

# Right plot
ax2.plot(X,Z*-1)
ax2.set_title('10-fold CV')

for ax in fig.axes:
    ax.set_ylabel('Mean Squared Error')
    ax.set_ylim(15,30)
    ax.set_xlabel('Degree of Polynomial')
    ax.set_xlim(0.5,10.5)
    ax.set_xticks(range(2,11,2));
```



Bootstrap Method

- Is a powerful tool that can be used to quantify the uncertainty associated with a given estimator or statistical learning method.
- It is not the same as the term “bootstrap” used in computer science meaning to “boot” a computer from a set of core instructions, though the derivation is similar.

Repeating train-test: Bootstrapping

- **Bootstrapping** samples the data set with replacement
    - k bootstrap subsamples from the **same** data
        - same size as data, can have **repeated examples**
    - k test sets with the examples left out in each bootstrap
        - on average 63.2% of the data set
    - obtain an accuracy estimate for each subsample
    - calculate average and variance of the k estimates

References

- Han, Kamber & Pei, Data Mining Concepts and Techniques, Morgan Kaufman.
- Jake VanderPlas, Data Science Handbook, O'Reilly
- Janez Demsar, Statistical Comparisons of Classifiers over Multiple Data Sets, JMLR, 2006.