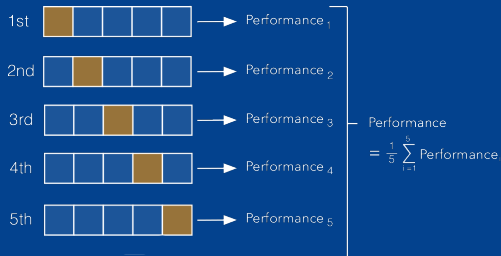




LESSON 3: End-to-end ML

CARSTEN EIE FRIGAARD

SPRING 2021



Agenda

End-to-end Machine Learning

1. Spørge-minutter
2. Admin
 - ▶ Afleveringer, grupper, etc.,
 - ▶ O1 Feedback Gruppe
 - ▶ f.eks. Grp01 + 02 + 03
 - ▶ Grp01 sender deres O1 til det to andre i gruppen!
3. General repetition af § 2,
 - ▶ kort intro til Stochastic Gradient Descent (SGD)
4. Algorithm and Model Selection,
 - ▶ model hyperparameters
 - ▶ k-fold cross validation.
5. Pipelines
 - ▶ Opgave: [L03/pipelines.ipynb](#)

CHAPTER 2

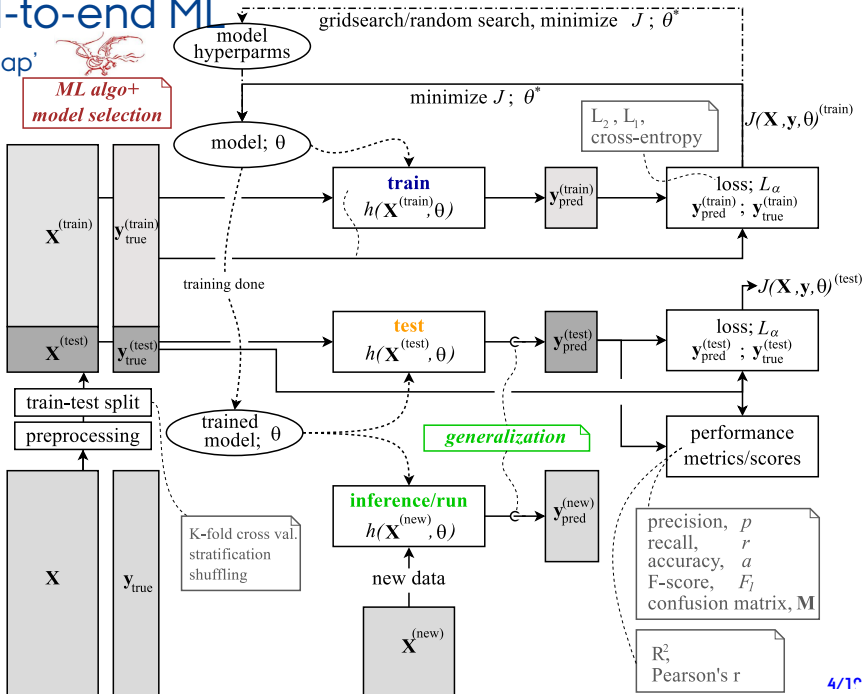
End-to-End Machine Learning Project

End-to-end ML

'The Map'



*ML algo+
model selection*





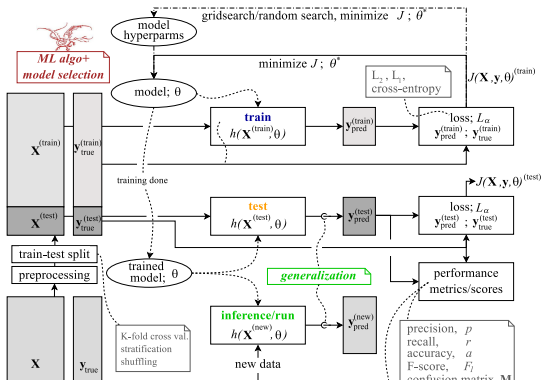
ML algo+
model selection

MACHINE LEARNING ALGORITHM SELECTION AND MODEL SELECTION

ML Algorithm Selection and Model Selection

Manually Choosing an Algorithm and Tuning a Model..

- ▶ algorithm selection
(choose a $h()$).
- ▶ model selection
(set hyperparameters on $h()$),
- ▶ model evaluation (train, test),
- ▶ **re-iteration and re-selection!**



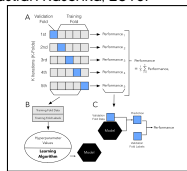
Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning

Sebastian Raschka
University of Wisconsin-Madison
Department of Statistics
November 2018
sraschka@stat.wisc.edu

Abstract

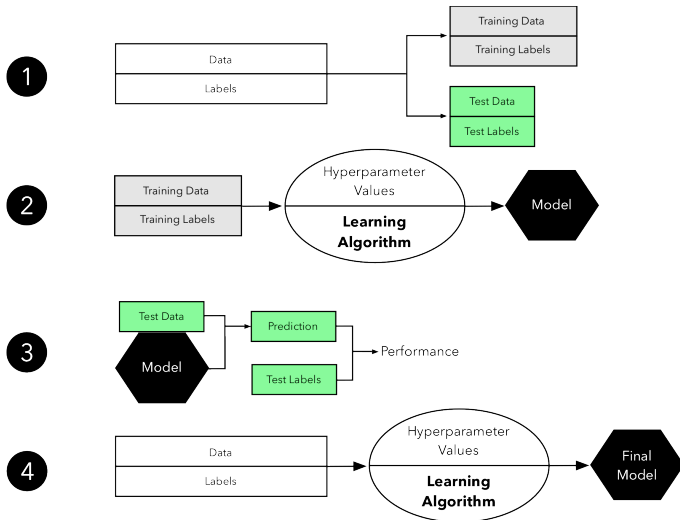
The correct use of model evaluation, model selection, and algorithm selection techniques is vital in academic machine learning research as well as in many industrial settings. This article reviews different techniques that can be used for each of these three subtasks and discusses the main advantages and disadvantages of each technique with references to theoretical and empirical studies. Further recommendations are given to encourage best yet feasible practices in research and applications of machine learning. Common methods such as the holdout method for model evaluation and selection are covered, which are not recommended when working with small datasets. Different flavors of the bootstrap technique are introduced for estimating the uncertainty of performance estimates, as an alternative to confidence intervals via normal approximation if bootstrapping is computationally feasible. Common cross-validation techniques such as leave-one-out cross-validation and k -fold cross-validation are reviewed, the bias-variance trade-off for choosing k is discussed, and practical tips for the optimal choice of k are given based on empirical evidence. Different statistical tests for algorithm comparisons are presented, and strategies for dealing with multiple comparisons such as omnibus tests and multiple comparison corrections are discussed. Finally, alternative methods for algorithm selection, such as the combined k -test 5×2 cross-validation and nested cross-validation, are recommended for comparing machine learning algorithms when datasets are small.

"Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning".
Sebastian Raschka. 2018.



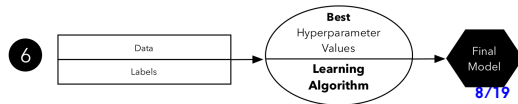
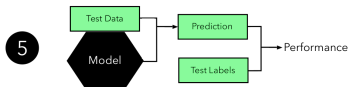
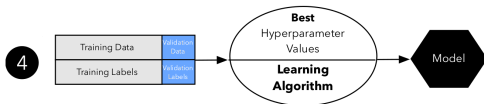
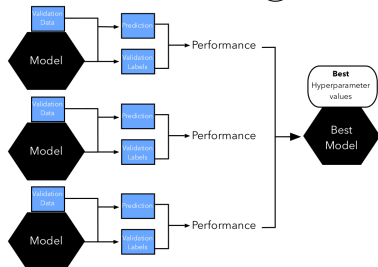
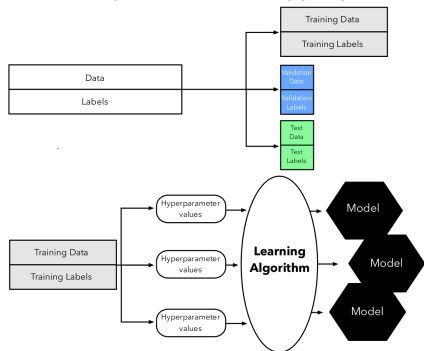
Model Evaluation

Simple Holdout Method (Train-Test Split)..



Model Evaluation and Selection

Three-way Holdout for Hyperparameter Tuning (Train-Validate-Test Split)...



Scikit-learn K-fold Demo..



Install User Guide API Examples More ▾

Go

Prev Up Next

scikit-learn 0.23.2

[Other versions](#)

Please [cite us](#) if you use the software.

`sklearn.model_selection.KFold`

Examples using

`sklearn.model_selection.KF`

`sklearn.model_selection.KFold`

```
class sklearn.model_selection.KFold(n_splits=5, *, shuffle=False,
                                     random_state=None)
```

[\[source\]](#)

K-Folds cross-validator

Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default).

Each fold is then used once as a validation while the k - 1 remaining folds form the training set.

Read more in the [User Guide](#).

Parameters: `n_splits` : *int*, *default=5*

Number of folds. Must be at least 2.

Changed in version 0.22: `n_splits` default value changed from 3 to 5.

`shuffle` : *bool*, *default=False*

Whether to shuffle the data before splitting into batches. Note that the samples within each split will not be shuffled.

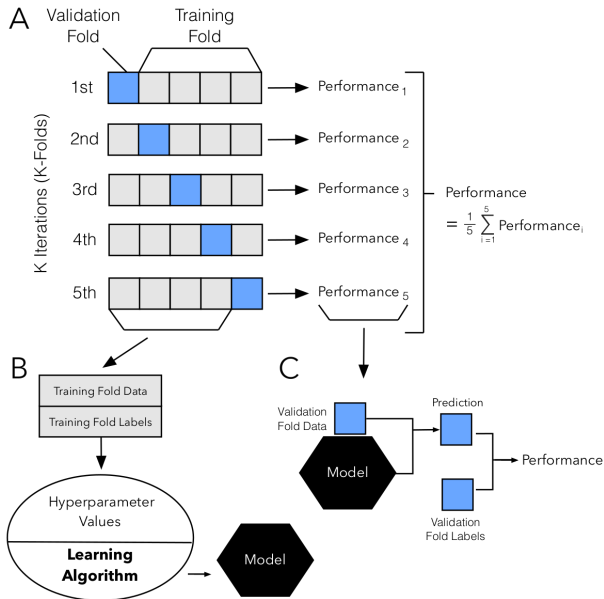
`random_state` : *int* or *RandomState* instance, *default=None*

Opgave:

- forklar Scikit's K-fold doc
- forklar koden L03/
Extra/k-fold_demo.ipynb

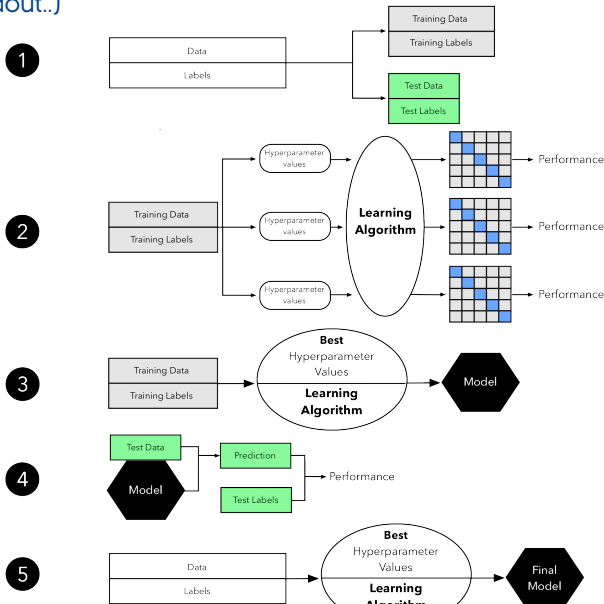
Model Evaluation

k-fold Cross-Validation Procedure, for *k*=5..



Model Evaluation and Selection

k-fold Cross-Validation for Hyperparameter Tuning (Somewhat Similar to Treeway Holdout..)



PIPELINES

Putting it all together in Python code...



Pipelines and Preprocessing of Data

Normalization via Scaling or Standardization

Why the need for preprocessing?

*Standardization of datasets is a **common requirement for many machine learning estimators** [..]; they might **behave badly** if the individual features do not more or less look like standard normally distributed data. [..]*

[<https://scikit-learn.org/stable/modules/preprocessing.html>]

Standardization of a feature vector \mathbf{x} , giving \mathbf{x}' mean zero, and standard deviation one

$$\mathbf{x}' = \frac{\mathbf{x} - \mu_{\mathbf{x}}}{\sigma_{\mathbf{x}}}$$

What kind of estimators needs preprocessing?

→ **Neural networks (NNs) in particular!**

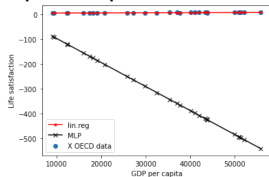
What is the difference between **Standardization** and **Scaling**?

Pipelines

Exercise: `pipelines.ipynb`: Revisit the OECD data in O1s for MLPs

Feature: GDP per capita feature in range 10K to 50K \$.

But MLP expects input in the range $[0;1]$ or perhaps $[-1;1]$.



```
1 # Manual scaling..
2 X_min = np.min(X)
3 X_max = np.max(X)
4 s = X_max - X_min
5
6 print(f"X_min={X_min:.0f}, X_max={X_max:.0f}, s={s:.0f}")
7
8 X_scaled = (X-X_min)/s
9 print(f"X_scaled.shape={X_scaled.shape}")
10 print(f"np.min(X_scaled)={np.min(X_scaled)}")
11 print(f"np.max(X_scaled)={np.max(X_scaled)}")
12
13 mlp.fit(X_scaled ,y.ravel())
14 y_pred_mlp = mlp.predict((M-X_min)/s)
15
16 plt.plot(m, y_pred_lin, "r")
17 plt.plot(m, y_pred_mlp, "k")
18
19 print(f"mpl.score={mlp.score(X_scaled, y.ravel()):0.2f}")
```

Prints:

```
X_min=9055,
X_max=55805, s=46750
X_scaled.shape=(29, 1)
np.min(X_scaled)=0.0
np.max(X_scaled)=1.0
mpl.score=0.70
```

Pipelines

OECD Data and MLPs: introducing a MinMaxScaler

```
1  # Now, do the same but via a pipeline..
2  from sklearn.preprocessing import MinMaxScaler
3
4  scaler = MinMaxScaler()
5  scaler.fit(X)
6  X_scaled = scaler.transform(X)
7  M_scaled = scaler.transform(M)
8
9  mlp.fit(X_scaled, y)
10 y_pred_mlp = mlp.predict(M_scaled)
11
12 print(f"mlp.score={mlp.score(X_scaled, y):0.2f}")
13
14 # PRINTS: mlp.score=0.71
```

Pipelines

OECD Data and MLPs: putting everything in a Full Pipeline

```
1  # Or even better, in a full pipeline..
2  from sklearn.pipeline import Pipeline
3
4  pipe = Pipeline( # indent pipeline as VHDL port mappings!
5      [
6          ('scaler', MinMaxScaler()),
7          ('mlp', mlp)
8      ]
9  )
10
11 pipe.fit(X, y)
12
13 print(f"pipe.score(..)={pipe.score(X, y):0.2f}")
14
15 # PRINTS: pipe.score(..)=0.68
```



Pipelines

Python code from Opgave: [L07/capacity_under_overfitting.ipynb](#)

```
1  from sklearn.pipeline import Pipeline
2  from sklearn.preprocessing import PolynomialFeatures
3  from sklearn.linear_model import LinearRegression
4  from sklearn.model_selection import cross_val_score
5  ..
6  polynomial_features = PolynomialFeatures(degree=degrees[i], ..
7  linear_regression = LinearRegression()
8
9  pipeline = Pipeline(
10     [
11         ("polynomial_features", polynomial_features),
12         ("linear_regression", linear_regression)
13     ]
14 )
15
16 pipeline.fit(X[:, np.newaxis], y)
17
18 scores = cross_val_score(
19     pipeline, X[:, np.newaxis], y,
20     scoring="neg_mean_squared_error", cv=10
21 )
22 score_mean = -scores.mean()
```

Pipelines and K-fold CV

Evaluate a score by cross-validation: `cross_val_score(..)`

 [Install](#) [User Guide](#) [API](#) [Examples](#) [More ▾](#)

[Prev](#) [Up](#) [Next](#)

scikit-learn 0.24.1
[Other versions](#)

Please [cite us](#) if you use the software.

[sklearn.model_selection.cross_val_score](#)
Examples using [sklearn.model_selection.cross_val_score](#)

sklearn.model_selection.cross_val_score

```
sklearn.model_selection.cross_val_score(estimator, X, y=None, *, groups=None, scoring=None, cv=None, n_jobs=None, verbose=0, fit_params=None, pre_dispatch='2*n_jobs', error_score=nan)
```

[\[source\]](#)

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 of shape (n_samples, n_features)
The data to fit. Can be for example a list, or an array.

y : array-like of shape (n_samples,) or (n_samples, n_outputs), default=None
The target variable to try to predict in the case of supervised learning.

groups : array-like of shape (n_samples,), default=None
Group labels for the samples used while splitting the dataset into train/test set. Only used in conjunction with a "Group" `cv` instance (e.g., [GroupKFold](#)).

scoring : str or callable, default=None
A str (see model evaluation documentation) or a scorer callable object / function with signature `scorer(estimator, X, y)` which should return only a single value.

Similar to [cross_validate](#) but only a single metric is permitted.

If None, the estimator's default scorer (if available) is used.

cv : int, cross-validation generator or an iterable, default=None

8/19