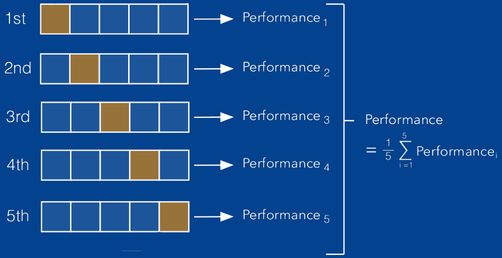
AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

LESSON 3: End-to-end ML

CARSTEN EIE FRIGAARD

SPRING 2021



Agenda

End-to-end Machine Learning

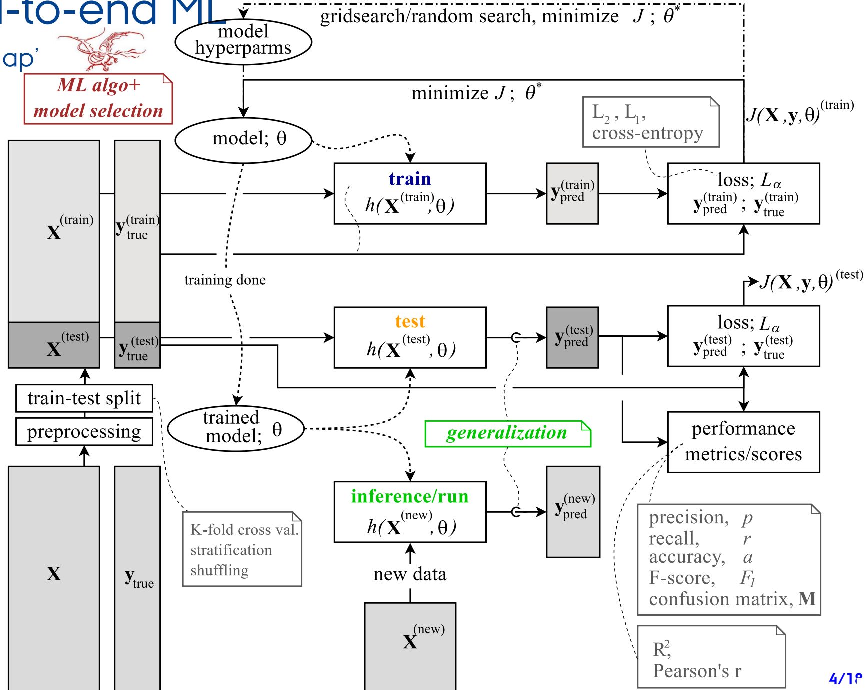
1. Spørge-minutter
2. Admin
   - Afleveringer, grupper, etc.,
   - O1 Feedback Gruppe
     - f.eks. Grp01 + 02 + 03
     - Grp01 sender deres O1 til det to andre i gruppen!
3. Generel repetition af § 2,
   - kort intro til Stochastic Gradient Descent (SGD)
4. Algorithm and Model Selection,
   - model 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
gridsearch/random search, minimize J ; θ^*
model hyperparms
minimize J ; θ^*
model; θ
$J(\mathbf{X},\mathbf{y},\theta)^{(train)}$
L_2, L_1, cross-entropy
$\mathbf{X}^{(train)}$
$\mathbf{y}_{true}^{(train)}$
train
$h(\mathbf{X}^{(train)},\theta)$
$\mathbf{y}_{pred}^{(train)}$
loss; L_α
$\mathbf{y}_{pred}^{(train)}$; $\mathbf{y}_{true}^{(train)}$
training done
$J(\mathbf{X},\mathbf{y},\theta)^{(test)}$
$\mathbf{X}^{(test)}$
$\mathbf{y}_{true}^{(test)}$
test
$h(\mathbf{X}^{(test)},\theta)$
$\mathbf{y}_{pred}^{(test)}$
loss; L_α
$\mathbf{y}_{pred}^{(test)}$; $\mathbf{y}_{true}^{(test)}$
train-test split
preprocessing
trained model; θ
generalization
performance metrics/scores
$\mathbf{X}$
$\mathbf{y}_{true}$
K-fold cross val.
stratification
shuffling
inference/run
$h(\mathbf{X}^{(new)},\theta)$
$\mathbf{y}_{pred}^{(new)}$
new data
$\mathbf{X}^{(new)}$
precision, p
recall, r
accuracy, a
F-score, F_1
confusion matrix, $\mathbf{M}$
R^2,
Pearson's r

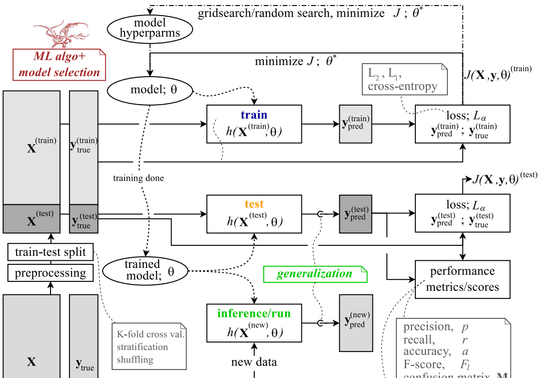
ML algo+
model selection

MACHINE LEARNING ALGORITHM SELECTION AND MODEL SELECTION

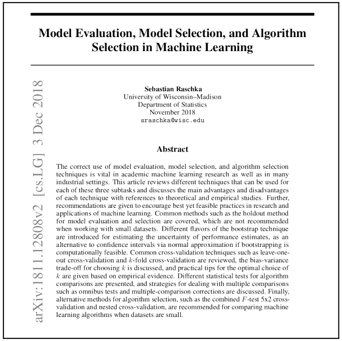
ML Algorithm Selection and Model Selection

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!



ML algo+ model selection
model hyperparms
gridsearch/random search, minimize J ; θ^*
minimize J ; θ^*
L_2, L_1, cross-entropy
$J(\mathbf{X},\mathbf{y},\theta)^{(train)}$
model; θ
train
$h(\mathbf{X}^{(train)},\theta)$
$\mathbf{y}^{(train)}_{pred}$
loss; L_α
$\mathbf{y}^{(train)}_{pred}$; $\mathbf{y}^{(train)}_{true}$
$\mathbf{X}^{(train)}$
$\mathbf{y}^{(train)}_{true}$
training done
$J(\mathbf{X},\mathbf{y},\theta)^{(test)}$
test
$h(\mathbf{X}^{(test)},\theta)$
$\mathbf{y}^{(test)}_{pred}$
loss; L_α
$\mathbf{y}^{(test)}_{pred}$; $\mathbf{y}^{(test)}_{true}$
$\mathbf{X}^{(test)}$
$\mathbf{y}^{(test)}_{true}$
train-test split
preprocessing
trained model; θ
generalization
performance metrics/scores
inference/run
$h(\mathbf{X}^{(new)},\theta)$
$\mathbf{y}^{(new)}_{pred}$
new data
K-fold cross val.
stratification
shuffling
$\mathbf{X}$
$\mathbf{y}_{true}$
precision, p
recall, r
accuracy, a
F-score, F_1
confusion matrix, $\mathbf{M}$



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

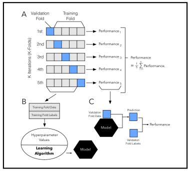
Sebastian Raschka
University of Wisconsin–Madison
Department of Statistics
November 2018
sraschka@wisc.edu

arXiv:1811.12808v2 [cs.LG] 3 Dec 2018

Abstract

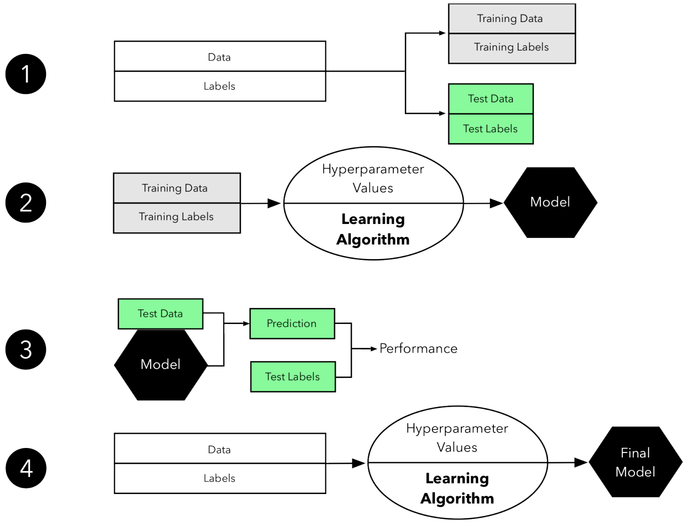
The correct use of model evaluation, model selection, and algorithm selection techniques is vital in academic machine learning research as well as in many industrial settings. This article reviews different techniques that can be used for each of these three subtasks and discusses the main advantages 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 F-test 5x2 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.



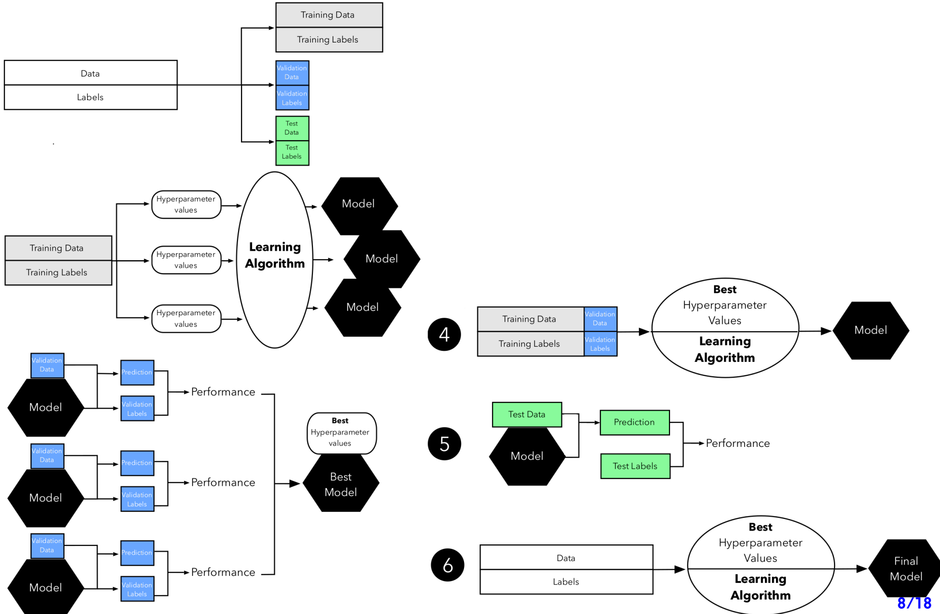
Model Evaluation

Simple Holdout Method (Train-Test Split)..

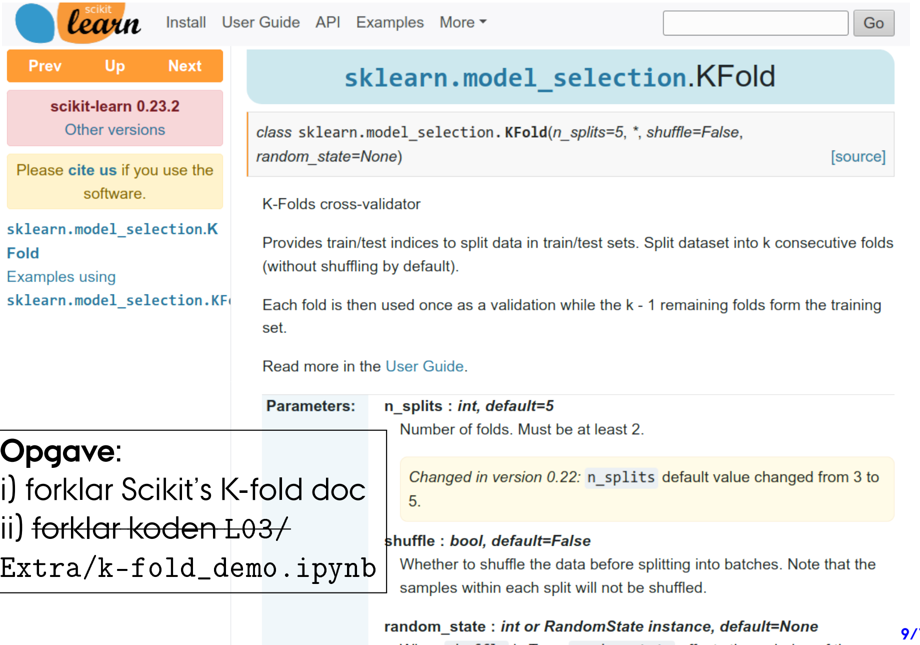


Model Evaluation and Selection

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



Scikit-learn K-fold Demo..



scikit-learn 0.23.2
Other versions

Please **cite us** if you use the software.

`sklearn.model_selection.K` **Fold**
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.

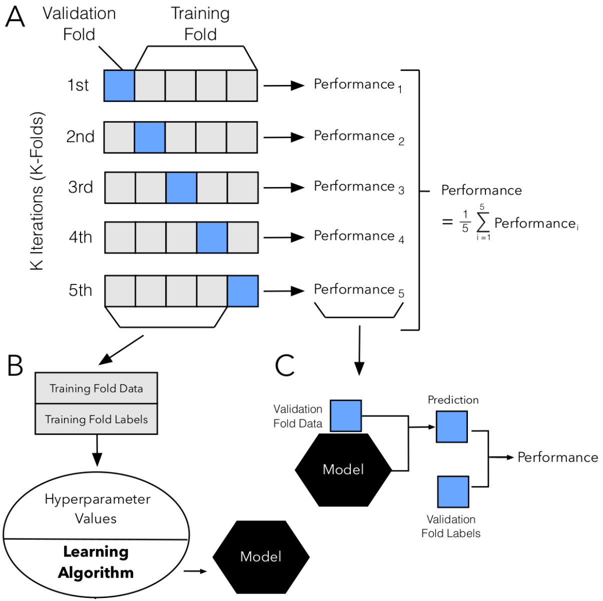
shuffle : ***bool, default=False***
Whether to shuffle the data before splitting into batches. Note that the samples within each split will not be shuffled.

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

Opgave:
i) forklar Scikit's K-fold doc
ii) ~~forklar koden `L03/`~~
`Extra/k-fold_demo.ipynb`

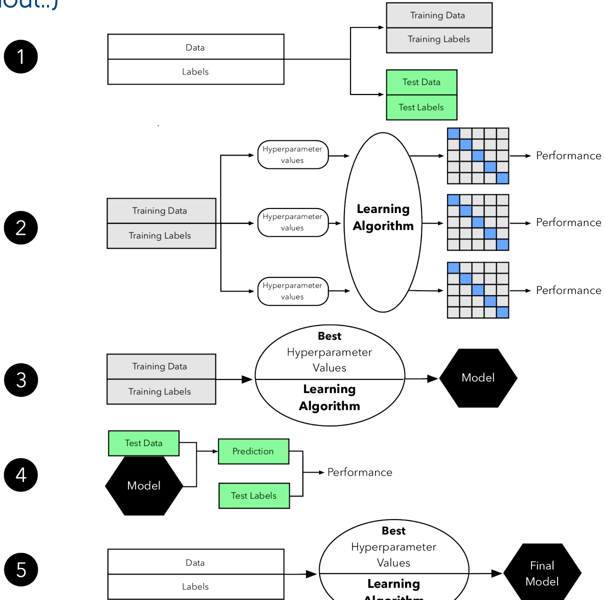
Model Evaluation

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

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

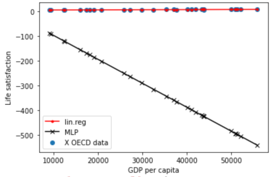
What kind of estimators needs preprocessing?
$\rightarrow$ **Neural networks (NNs) in particular!**

What is the difference between Standardization and Scaling?

Pipelines

Exercise: `pipelines.ipynb`: Revisit the OECD data in 01s 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].



```python
# Manual scaling..
X_min = np.min(X)
X_max = np.max(X)
s = X_max - X_min

print(f"X_min={X_min:.0f},  X_max={X_max:.0f},  s={s:.0f}")

X_scaled = (X-X_min)/s
print(f"X_scaled.shape={X_scaled.shape}")
print(f"np.min(X_scaled)={np.min(X_scaled)}")
print(f"np.max(X_scaled)={np.max(X_scaled)}")

mlp.fit(X_scaled ,y.ravel())
y_pred_mlp = mlp.predict((M-X_min)/s)

plt.plot(m, y_pred_lin, "r")
plt.plot(m, y_pred_mlp, "k")

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`

```python
# Now, do the same but via a pipeline..
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
M_scaled = scaler.transform(M)

mlp.fit(X_scaled, y)
y_pred_mlp = mlp.predict(M_scaled)

print(f"mpl.score={mlp.score(X_scaled, y):0.2f}")

# PRINTS: mpl.score=0.71
```

Pipelines

OECD Data and MLPs: putting everything in a Full Pipeline

```
# Or even better, in a full pipeline..
from sklearn.pipeline import Pipeline

pipe = Pipeline( # indent pipeline as VHDL port mappings!
  [
    ('scaler', MinMaxScaler()),
    ('mlp', mlp)
  ]
)

pipe.fit(X, y)

print(f"pipe.score(..)={pipe.score(X, y):0.2f}")

# PRINTS: pipe.score(..)=0.68
```

Pipelines

Python code from Opgave: `L07/capacity_under_overfitting.ipynb`

```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
..
polynomial_features = PolynomialFeatures(degree=degrees[i], ..
linear_regression = LinearRegression()

pipeline = Pipeline(
  [
    ("polynomial_features", polynomial_features),
    ("linear_regression", linear_regression)
  ]
)

pipeline.fit(X[:, np.newaxis], y)

scores = cross_val_score(
    pipeline, X[:, np.newaxis], y,
    scoring="neg_mean_squared_error", cv=10
  )
score_mean = -scores.mean()
```