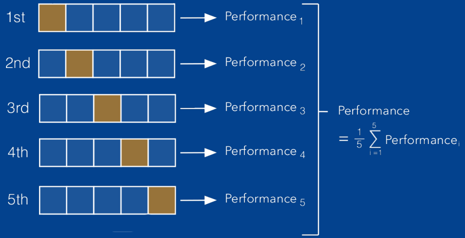
AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

LESSON 4: Data analyse

CARSTEN EIE FRIGAARD

AUTUMN 2021



Agenda

Data analyse

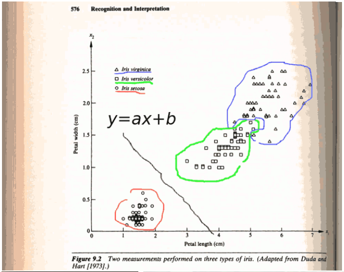
1. Datasæt og Jeres O4 projekt.
   - Opgave: `L04/dataanalyse.ipynb`
2. Statestik og visualisering.
3. Pipelines
   - Opgave: `L04/pipelines.ipynb`
4. Algorithm and Model Selection,
   - model hyperparameters
   - k-fold cross validation,
   - ekstra materiale: `L04/Extra/k-fold_demo.ipynb`

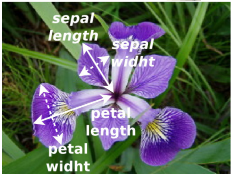
'Demo' datasæt

MNIST, Iris og Moon

Iris:

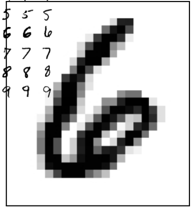
Sepal/petal længde/bredde,
Mr. Fisher, 1936,
"Anderson's Iris data set"
`sklearn.datasets.load_iris(..)`





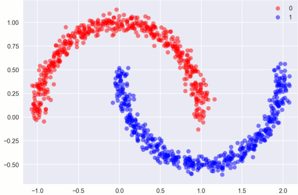
MNIST:

Håndskrevne tal,
preprocesseret, centrerede,
`sklearn.datasets.fetch_openml('mnist_784'..`



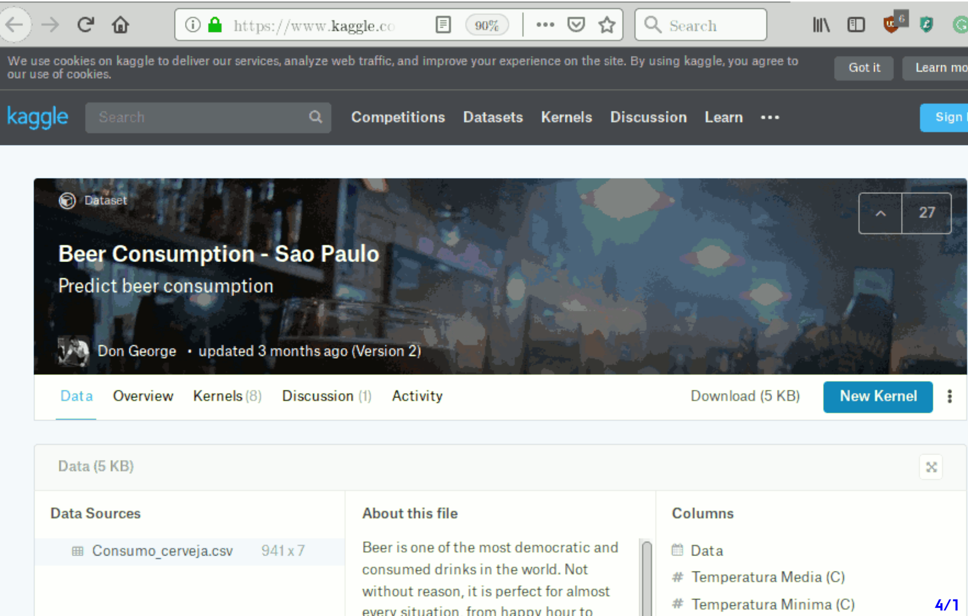
Moon:

'XOR' lign.,
non-linear decision boundary,
`sklearn.datasets.make_moons(..)`



'Dit' datasæt

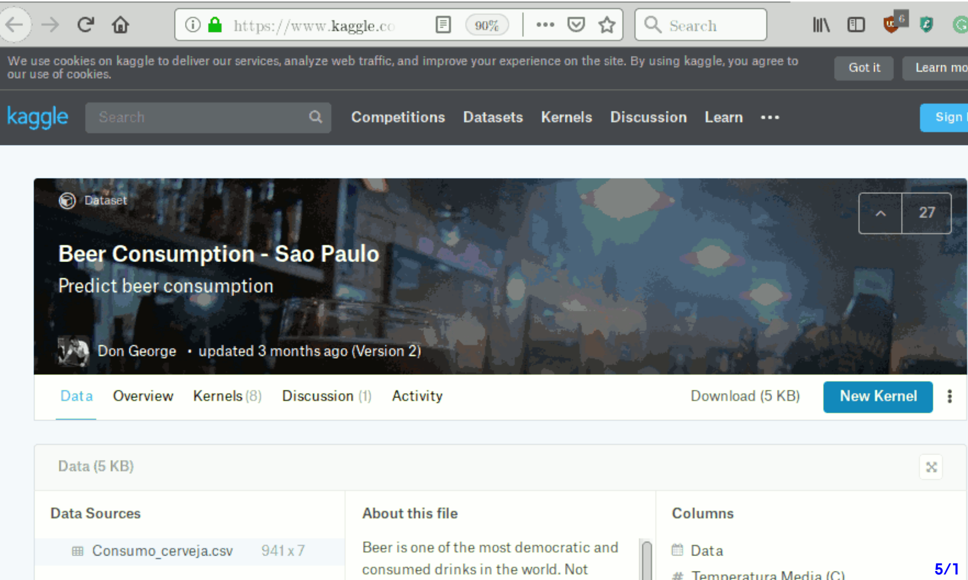
Fra `https://www.kaggle.com...`



Opg. L04 Beskrivelse af eget slutprojekt.pdf

Dit datasæt fra f.eks. `https://www.kaggle.com...`

(brug min login: user=cef@ase.au.dk, password=test123)



Opg. L04 Beskrivelse af eget slutprojekt.pdf

..eller UCI `https://archive.ics.uci.edu/ml/index.php`..

UCI

About Citation Policy Donate a Data Set Contac

Search

Repository Web

Google

Machine Learning Repository

Center for Machine Learning and Intelligent Systems

View ALL Data Set

Welcome to the UC Irvine Machine Learning Repository!

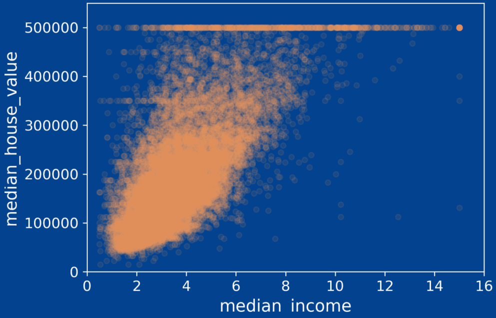
We currently maintain 557 data sets as a service to the machine learning community. You may **view all data sets** through our searchable interface. For a general overview of the Repository, please visit our About page. For information about citing data sets in publications, please read our citation policy. If you wish to donate a data set, please consult our donation policy. For any other questions, feel free to contact the Repository librarians.

Supported By: NATIONAL SCIENCE FOUNDATION In Collaboration With: Rexa.info • Research • People • Connections

Latest News:	**Newest Data Sets:**	**Most Popular Data Sets (hits since 2007):**
09-24-2018: Welcome to the new Repository admins Dheeru Dua and Efi Karra Taniskidou!	**07-22-2020:** UCI Facebook Large Page-Page Network	**3521507:** Iris
04-04-2013: Welcome to the new Repository admins Kevin	**07-17-2020:** UCI Amphibians	**1917226:** Adult
	07-12-2020: UCI Early stage diabetes risk prediction dataset.	

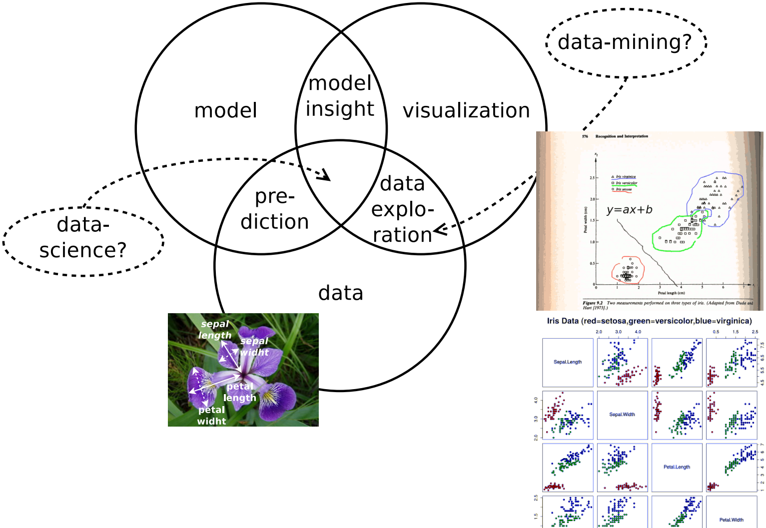
STATISTICS AND VISUALIZATION

μ, σ and being a Data Science Expert..



Machine learning baggrund

Machine learning: data science ekspert (fra L01)



Mean and Variance

The mean and variance (and hence the standard deviation) for a random variable X can for a population of N samples be estimated as

$$
\begin{aligned}
E[X] &= \tfrac{1}{N}\textstyle\sum_{i=1}^{N} X_i && = \mu_X \\
V[X] &= E[(X_i - E[X])(X_i - E[X])] \\
&= E[X_i X_i] - E[X]^2 \\
&= \left(\tfrac{1}{N}\textstyle\sum_{i=1}^{N} X_i X_i\right) - \mu_X^2 && = \sigma_X^2 \\
\sigma &= \sqrt{V}
\end{aligned}
$$

Notice that the $1/N$ factor most often appears as $1/(N-1)$ for the variance estimation.

When using the factor $1/(N-1)$, $\hat{V}$ is said to be the best unbiased estimator, when it is $1/N$ it will be biased, both assuming an underlying normal distribution.

Auto- and Cross-covariance

Now let's goto full matrix notation for the covariance. For a data matrix $\mathbf{X}$

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_d^{(2)} \\ \vdots & & & \\ x_1^{(n)} & x_2^{(n)} & \cdots & x_d^{(n)} \end{bmatrix}$$

the previous one-dimensional random variable X can now be seen as a d-dimensional vector $\mathbf{x}^{(i)}$, that is one of the data rows in the full data matrix

$$X_i \to \mathbf{x}^{(i)} = \begin{bmatrix} x_1^{(i)} & x_2^{(i)} & \cdots & x_d^{(i)} \end{bmatrix}^T$$

Auto- and Cross-covariance

The (biased) auto–covariance matrix can be estimated as

$$
\begin{aligned}
E[\mathbf{X}] &= \mu_{\mathbf{X}} \\
&= \tfrac{1}{n} \textstyle\sum_{i=1}^{n} \mathbf{x}^{(i)}
\end{aligned}
$$

$$
\begin{aligned}
\Sigma(\mathbf{X}) &= \mathrm{cov}(\mathbf{X}, \mathbf{X}) \\
&= \tfrac{1}{n} \textstyle\sum_{i=1}^{n} \mathbf{x}^{(i)} \mathbf{x}^{(i)T} - \mu_{\mathbf{X}} \mu_{\mathbf{X}}^{T}
\end{aligned}
$$

with implicit $1/N$ factor here (inside the first $E[\cdot]$), so the definition is a biased covariance. You may opt to use the factor $1/(N-1)$ instead.

For yet another data matrix $\mathbf{Z}$ the (biased) cross-covariance matrix is given by

$$
\begin{aligned}
\Sigma(\mathbf{X}, \mathbf{Z}) &= \mathrm{cov}(\mathbf{X}, \mathbf{Z}) \\
&= \tfrac{1}{n} \textstyle\sum_{i=1}^{n} \mathbf{x}^{(i)} \mathbf{z}^{(i)T} - \mu_{\mathbf{X}} \mu_{\mathbf{Z}}^{T}
\end{aligned}
$$

Covariance Matrix, binary classifier (SKIP slide!)

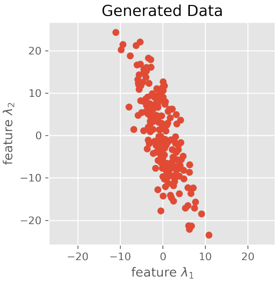
For a dataset, **X**; features cat and dog; classifying cat/dog

Covariance matrix (dim = features x features)

$$\Sigma(\mathbf{X}) = \begin{bmatrix} \sigma_{\text{cat,cat}} & \sigma_{\text{cat,dog}} \\ \sigma_{\text{dog,cat}} & \sigma_{\text{dog,dog}} \end{bmatrix} = \begin{array}{c|cc} & \text{cat} & \text{dog} \\ \hline \text{cat} & \sigma^2_{\text{cat}} & \sigma_{\text{cat,dog}} \\ \text{dog} & \sigma_{\text{dog,cat}} & \sigma^2_{\text{dog}} \end{array}$$

co-variance

$$\sigma_{\text{cat,dog}} = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)}_{\text{cat}} - \mu_{\text{cat}})(x^{(i)}_{\text{dog}} - \mu_{\text{dog}})$$



Example $\Sigma(\mathbf{X}) = \begin{bmatrix} 13.2 & -28.8 \\ -28.8 & 93.3 \end{bmatrix}$

Confusion matrix, cats only, cat/non-cat
(dim = classes x classes)

$$\mathbf{M} = \begin{bmatrix} \text{TP} & \text{FP} \\ \text{FN} & \text{TN} \end{bmatrix} = \begin{array}{c|cc} & \multicolumn{2}{c}{\textit{actual}} \\ & \text{cat} & \text{non-cat} \\ \hline \text{cat} & \text{T, cat} & \text{F, cat} \\ \text{dog} & \text{F, dog} & \text{T, non-cat} \end{array}$$

Looking for Correlations

..using the auto-covariance matrix,

	Grade 3	Grade 4	Grade 5	Grade 6	Grade 7	Grade 8
Grade 3	1.0000	0.7598	0.7199	0.6940	0.6869	0.6432
Grade 4	0.7598	1.004	0.7975	0.7675	0.7574	0.7189
Grade 5	0.7198	0.7975	0.9933	0.7813	0.7639	0.7218
Grade 6	0.6940	0.7675	0.7813	0.9899	0.7958	0.7579
Grade 7	0.6869	0.7574	0.7639	0.7958	0.9820	0.7884
Grade 8	0.6432	0.7189	0.7218	0.7579	0.7884	0.9826

..for regression: the standard correlation coefficient, Pearson's r,

Looking for Correlations

Since the dataset is not too large, you can easily compute the *standard correlation coefficient* (also called *Pearson's r*) between every pair of attributes using the `corr()` method:

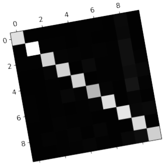
```
corr_matrix = housing.corr()
```

Now let's look at how much each attribute correlates with the median house value:

```
>>> corr_matrix["median_house_value"].sort_values(ascending=False)
median_house_value     1.000000
median_income          0.687170
total_rooms            0.135231
housing_median_age     0.114220
households             0.064702
total_bedrooms         0.047865
population            -0.026699
longitude             -0.047279
latitude              -0.142826
Name: median_house_value, dtype: float64
```

The correlation coefficient ranges from –1 to 1. When it is close to 1, it means that there is a strong positive correlation; for example, the median house value tends to go up when the median income goes up. When the coefficient is close to –1, it means that there is a strong negative correlation; you can see a small negative correlation

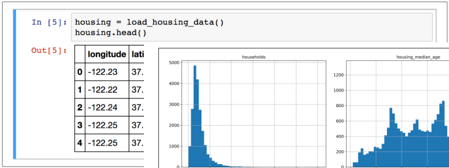
..or for classification: the confusion matrix



Data Science: Visualization

Take a Quick Look at the Data Structure

Let's take a look at the top five rows using the DataFrame's `head()` method (see Figure 2-5).



```
In [5]: housing = load_housing_data()
        housing.head()
```

Out[5]:

	longitude	lati
0	-122.23	37.
1	-122.22	37.
2	-122.24	37.
3	-122.25	37.
4	-122.25	37.

Figure 2-5. Top five rows in

Figure 2-8. A histogram fo

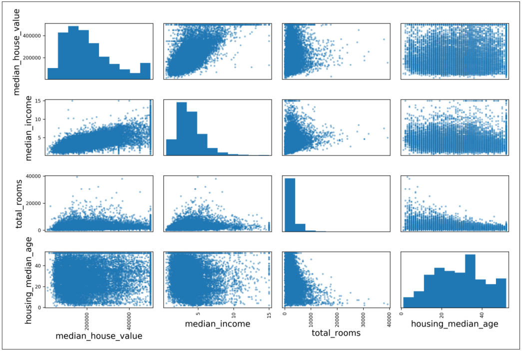


Figure 2-15. Scatter matrix

Data Science: Visualization

..why so many sales at `median_house_value` $= 350$ K\$ and 500 K\$??

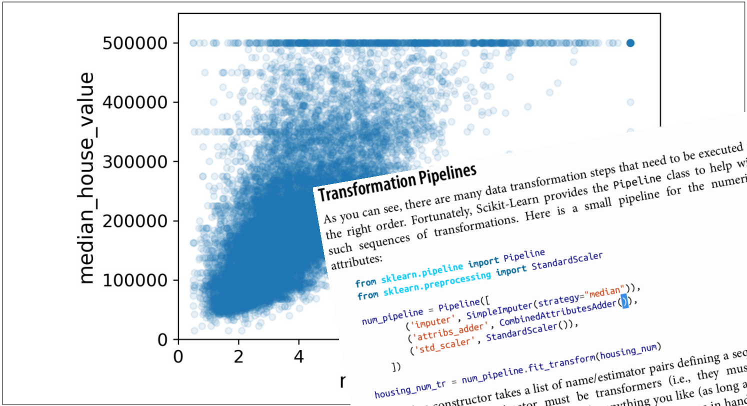


Figure 2-16. Median income versus media

..fixing data via pipelines..

Transformation Pipelines

As you can see, there are many data transformation steps that need to be executed in the right order. Fortunately, Scikit-Learn provides the `Pipeline` class to help with such sequences of transformations. Here is a small pipeline for the numerical attributes:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

The `Pipeline` constructor takes a list of name/estimator pairs defining a sequence of steps. All but the last estimator must be transformers (i.e., they must have a `fit_transform()` method). The names can be anything you like (as long as they are unique and don't contain double underscores "__"): they will come in handy later for hyperparameter tuning.

When you call the pipeline's `fit()` method, it calls `fit_transform()` sequentially on all transformers, passing the output of each call as the parameter to the next call, until it reaches the final estimator, for which it just calls the `fit()` method.

Prepare the Data for Machine Learning Algorithms | 73

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?

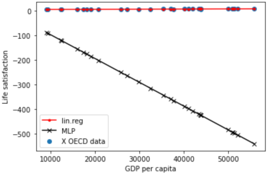
$\rightarrow$ **Neural networks (NNs) in particular!**

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].




```
# 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`

```
# 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

```python
# 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

```
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()
```

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.cr`

`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.

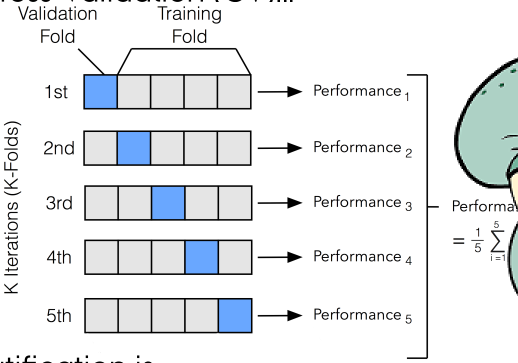
cv : ***int, cross-validation generator or an iterable, default=None***

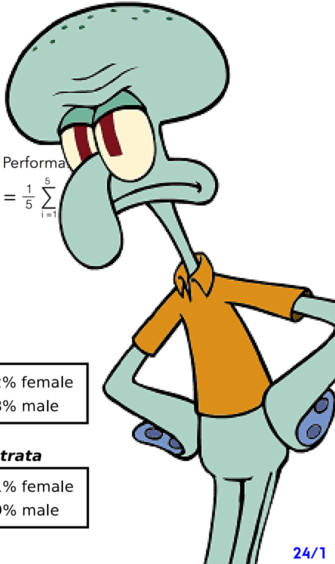
ALGORITHM SELECTION AND MODEL SELECTION

And k-fold CV

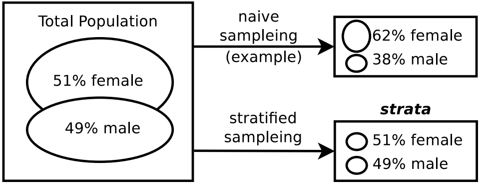
Forbered data: cross-validation, stratification

K-fold cross-validation (CV)...





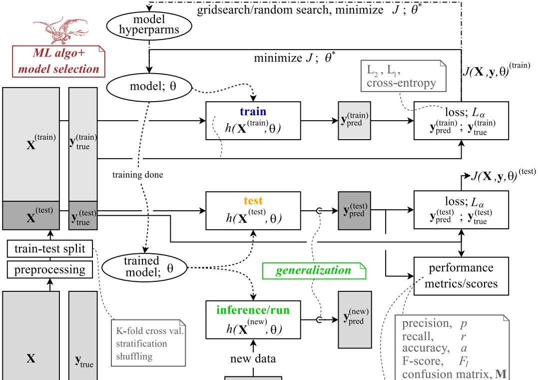
and stratification is...

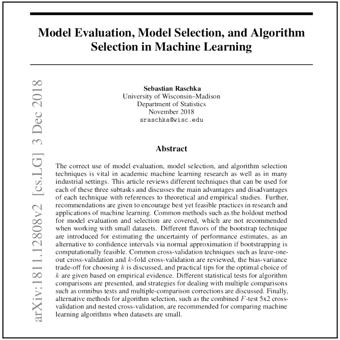


ML Algorithm Selection and Model Selection

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

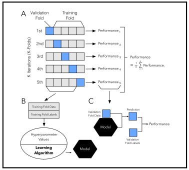
arXiv:1811.12808v2 [cs.LG] 3 Dec 2018

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

Abstract

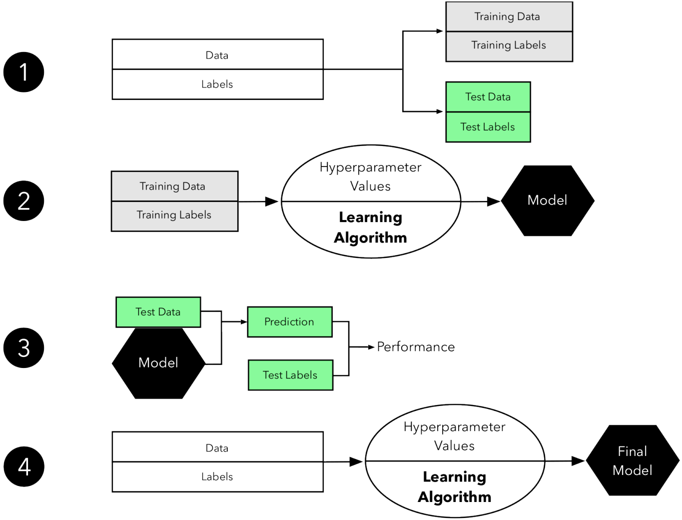
The correct use of model evaluation, model selection, and algorithm selection techniques is vital in academic machine learning research as well as in many industrial settings. This article reviews different techniques that can be used for each of these three subtasks and discusses the main advantages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



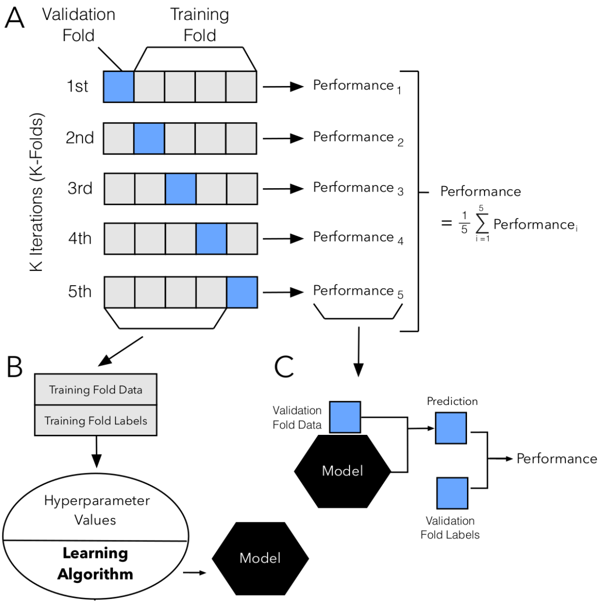
Model Evaluation

Simple Holdout Method (Train-Test Split)..



Model Evaluation

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



Scikit-learn K-fold Demo..

Install User Guide API Examples More

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.KFold

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 `L04/Extra/k-fold_demo.ipynb`

Code Review: k-fold Cross Validation

p.89, [HOML]

Measuring Accuracy Using Cross-Validation

A good way to evaluate a model is to use cross-validation, just as you did in Chapter 2.

Implementing Cross-Validation

Occasionally you will need more control over the cross-validation process than what Scikit-Learn provides off-the-shelf. In these cases, you can implement cross-validation yourself; it is actually fairly straightforward. The following code does roughly the same thing as Scikit-Learn's `cross_val_score()` function, and prints the same result:

```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone

skfolds = StratifiedKFold(n_splits=3, random_state=42)

for train_index, test_index in skfolds.split(X_train, y_train_5):
    clone_clf = clone(sgd_clf)
    X_train_folds = X_train[train_index]
    y_train_folds = y_train_5[train_index]
    X_test_fold = X_train[test_index]
    y_test_fold = y_train_5[test_index]

    clone_clf.fit(X_train_folds, y_train_folds)
    y_pred = clone_clf.predict(X_test_fold)
    n_correct = sum(y_pred == y_test_fold)
    print(n_correct / len(y_pred))  # prints 0.9502, 0.96565 and 0.96495
```

The `StratifiedKFold` class performs stratified sampling (as explained in Chapter 2) to produce folds that contain a representative ratio of each class. At each iteration the code creates a clone of the classifier, trains that clone on the training folds, and makes predictions on the test fold. Then it counts the number of correct predictions and outputs the ratio of correct predictions.

Code Review: k-fold Cross Validation

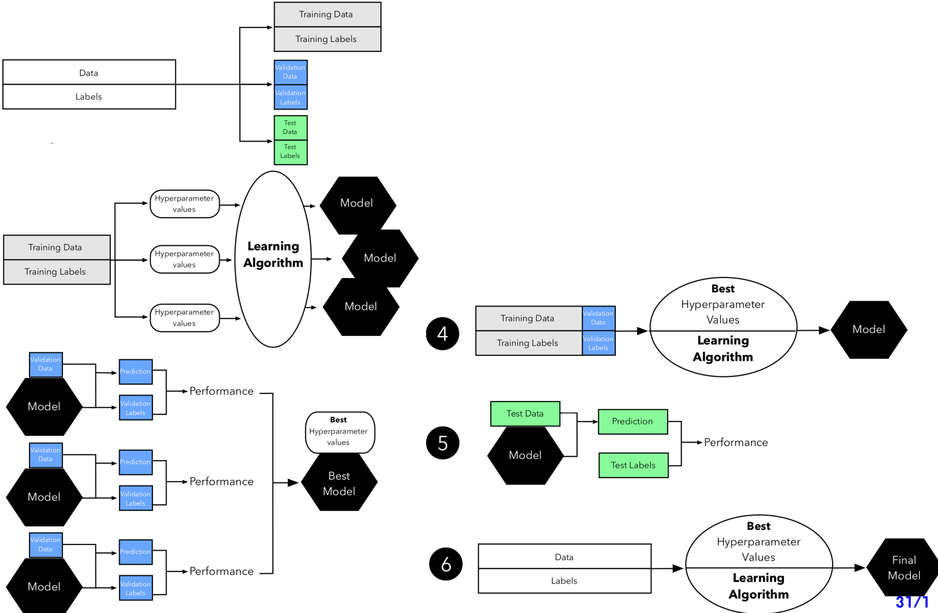
p.89, [HOML], with some code cleanup..

```python
from sklearn.model_selection import KFold [..]

def MyKFoldSplit(clf, X, y, kfolds=3, debug=True):
    def PrintVarInfo(varname, var): [..]

    skfolds = KFold
       (n_splits=kfolds, random_state=42, shuffle=True)

    for train_index, val_index in skfolds.split(X, y):
        clone_clf = clone(clf)

        X_train_folds = X[train_index]
        y_train_folds = y[train_index]
        X_val_fold    = X[val_index]
        y_val_fold    = y[val_index]

        clone_clf.fit(X_train_folds, y_train_folds)
        y_pred = clone_clf.predict(X_val_fold)
        PrintScores(y_val_fold, y_pred, i)
        i += 1

print("K-fold demo..")
MyKFoldSplit(sgd_clf,
    X_train, y_train_5, 3)
```

```
58 print("OK")

K-fold demo..
MyKFoldSplit(clf, X, y, kfolds=3)..                                        =(60000,
  type(X)            =<class 'numpy.ndarray'>,  X.shape                    =(60000,
  type(y)            =<class 'numpy.ndarray'>,  y.shape                    =(40000,
  type(train_index)  =<class 'numpy.ndarray'>,  train_index.shape          =(20000,
  type(val_index)    =<class 'numpy.ndarray'>,  val_index.shape
  FOLD  0: accuracy=0.97, precision=0.94, recall=0.70, F1=0.80
  FOLD  1: accuracy=0.95, precision=0.67, recall=0.89, F1=0.76
  FOLD  2: accuracy=0.97, precision=0.89, recall=0.73, F1=0.80
```

Model Evaluation and Selection

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



Model Evaluation and Selection

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