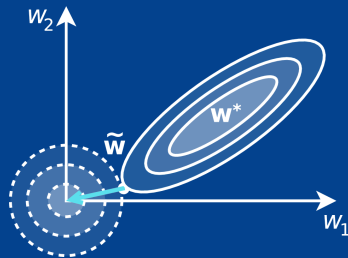




Lesson 8: Regularization, Optimization and Searching

CARSTEN EIE FRIGAARD

AUTUMN 2020

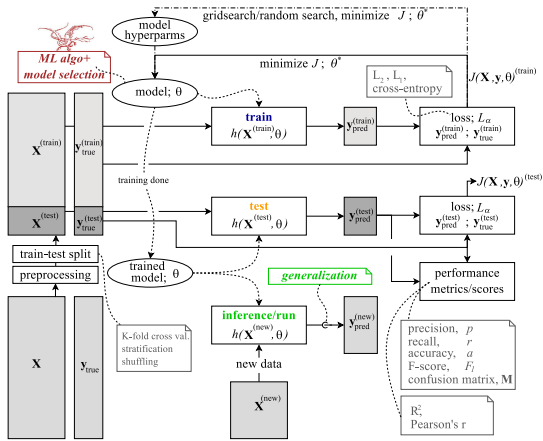


L08: Agenda

- ▶ **ML Algorithm and Model Selection:**
Three-way Holdout for Hyperparameter Tuning,
and k -fold CV revisited.
- ▶ **Regularization and Optimization:**
Regulizers,
Exercise: [L09/regulizers.ipynb](#)
Optimizers (no-exe).
- ▶ **Searching:**
Gridsearch,
Randomsearch,
Exercise: [L09/gridsearch.ipynb](#)

Manually Choosing an Algorithm and Tuning a Model..

- ▶ algorithm selection.
- ▶ model selection,
- ▶ model evaluation,
- ▶ re-iteration and re-selection!



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

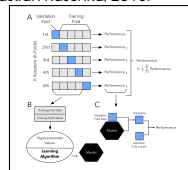
Sebastian Raschka
University of Wisconsin-Madison
Department of Statistics
November 2018
araschka@wisc.edu

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 task, with references to theoretical and empirical studies. Further recommendations are given to encourage best yet feasible practices in research and industry. The most commonly used techniques for model evaluation and model selection are covered, which are not recommended when working with small datasets. Different flavors of the bootstrap technique are discussed, and the advantages and disadvantages of cross-validation as an alternative to confidence intervals via normal approximation if bootstrapping is computationally expensive. Common cross-validation techniques such as leave-one-out, k-fold, and repeated k-fold are compared. The advantages and disadvantages of 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 are discussed. Finally, several alternative methods for algorithm selection, such as the combined F_1 -test/SVM classifier and nested cross-validation, are recommended for comparing machine

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

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



ML Algorithm Selection and Model Selection

Manually Choosing an Algorithm and Tuning a Model..

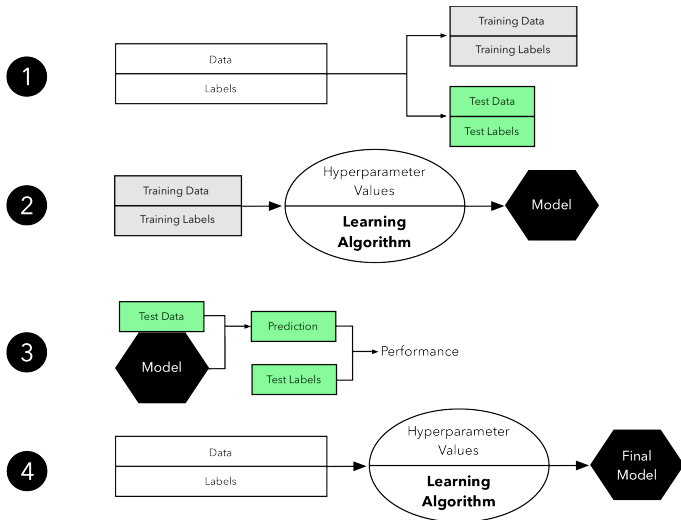
- ▶ algorithm selection:
 - ▶ choose an algo that '*fits*' the problem, ...perhaps via searching??
- ▶ model selection:
 - ▶ looking for 'optimal' model capacity,
 - ▶ tuning model **hyperparameters**..
 - ▶ model weights **regularizers**..(for NN's)
 - ▶ gradient descent **optimizers**..(for NN's)
- ▶ model evaluation:
 - ▶ the performance metric score function,
 - ▶ how do you evaluate **generalization** performance?
 - ▶ holdout method (train-test split) and *k*-fold CV,
 - ▶ three-way split (train-validate-test split)..
- ▶ **re-iteration and re-selection!**



NOTE: Model selection: ~ selection the best capacity/hyperparameter for a given model—NOT choosing the ML algo/model itself!

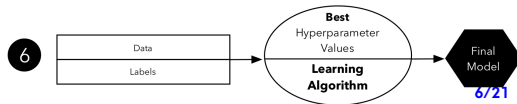
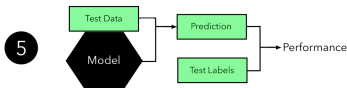
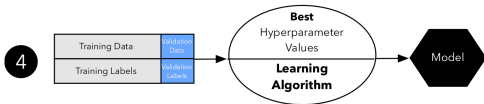
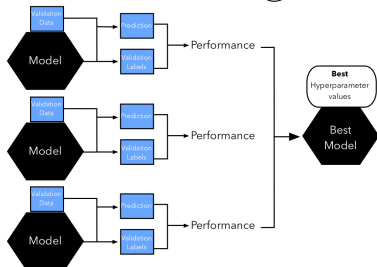
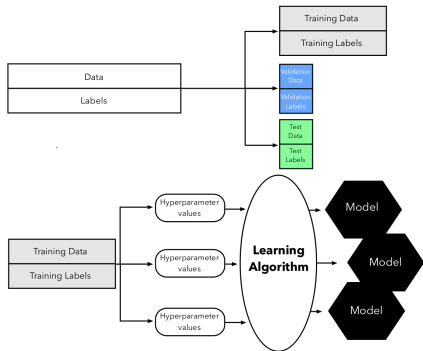
Model Evaluation

Simple Holdout Method (Train-Test Split)..



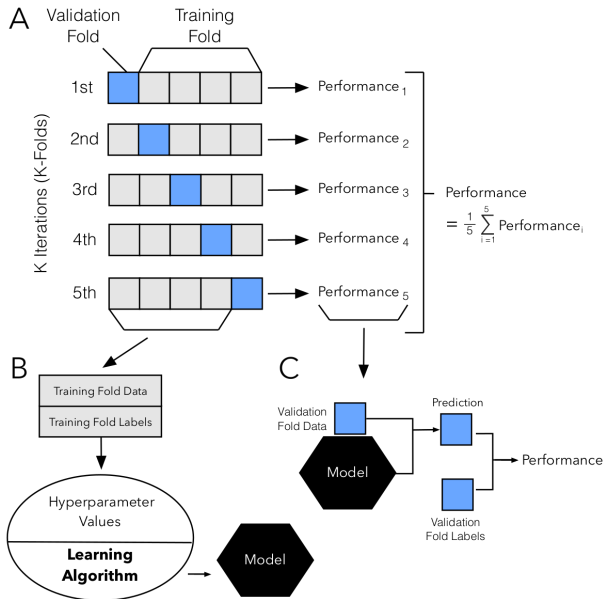
Model Evaluation and Selection

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



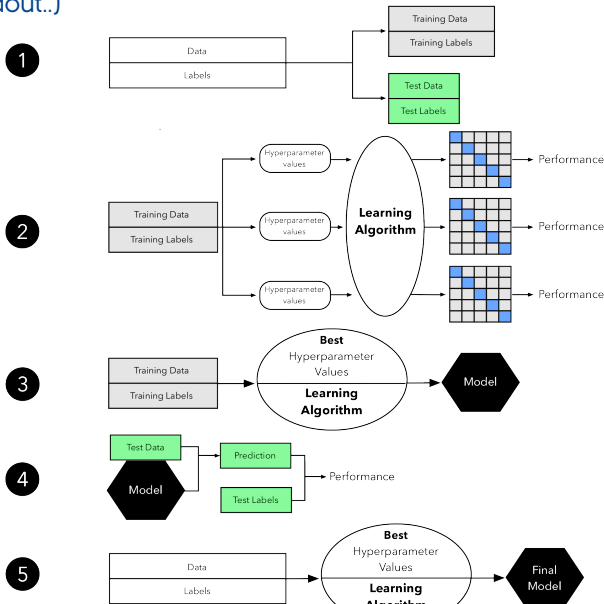
Model Evaluation

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



Model Evaluation and Selection

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



Regularization

Adding a Penalty to the Cost Function

For a linear regressor, our cost function was

$$J(\mathbf{X}, \mathbf{y}; \mathbf{w}) = ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 \propto \text{MSE}(\mathbf{X}, \mathbf{y}; \mathbf{w})$$

But now enters a **penalty factor**, Ω , that scaled with α adds extra cost to J ,

$$\tilde{J}(\mathbf{X}, \mathbf{y}; \mathbf{w}) = ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 + \alpha\Omega(\mathbf{w})$$

so this becomes **a-tug-of-war** between the two terms in \tilde{J} .

The effect of the added penalty is to:

- ▶ put a **constraint** on the norm of the weights, \mathbf{w} , disallowing 'em to grow wildly,
- ▶ leading to **reduced overfitting**, disabling the model to learn the background noise in the data.

\mathcal{L}_2 Regularization

Ridge Penalization

Aka Weight Decay, aka Tikhonov regularization

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \mathbf{w}^\top \mathbf{w}$$

$$\tilde{J}_{\text{ridge}}(\mathbf{X}, \mathbf{y}; \mathbf{w}) = J(\mathbf{X}, \mathbf{y}; \mathbf{w}) + \alpha \mathbf{w}^\top \mathbf{w}$$

with $\mathbf{w} = [w_1 \ w_2 \ \cdots \ w_n]^\top$ without the bias element w_0 in the regularizer term, Ω , and recalling the Euclidean norm

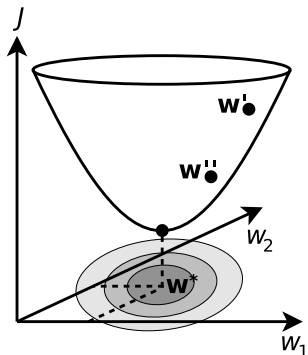
$$\mathcal{L}_2^2 : \|\mathbf{x}\|_2^2 = \mathbf{x}^\top \mathbf{x}$$

and give-or-take some additional $1/2$ or $1/n$ constant, that we do not care about.

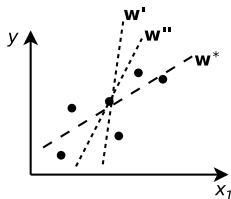
\mathcal{L}_2 Regularization

Ridge Penalization

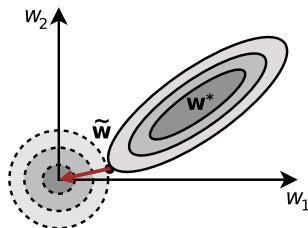
A graphical view for a linear regressor



3D: ideal convex loss in $J - \mathbf{w}$ space.



1D featurespace



2D: flat $w_2 - w_1$ view with some feature scaling.

The tug-of-war: what happens with $\tilde{\mathbf{w}}$,
if \mathbf{w}^* is far from the origin $[w_1, w_2] = (0, 0)$?

\mathcal{L}_1 Regularization

Lasso penalization

Now, just replace the \mathcal{L}_2 with \mathcal{L}_1 and we have the Lasso regularizer

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_1$$

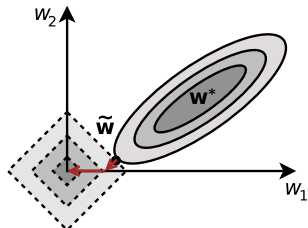
$$\tilde{J}_{\text{lasso}}(\mathbf{X}, \mathbf{y}; \mathbf{w}) = J(\mathbf{X}, \mathbf{y}; \mathbf{w}) + \alpha \|\mathbf{w}\|_1$$

with the Manhattan norm

$$\mathcal{L}_1 : \|\mathbf{x}\|_1 = \sum_{i=1}^n \text{abs}(x_i)$$

and the \mathcal{L}_1 penalty tends to drive weights to zero:

- ▶ automatic feature selection,
- ▶ outputs a sparse model,
- ▶ i.e few nonzero w 's.



\mathcal{L}_1 and \mathcal{L}_2 Regularization

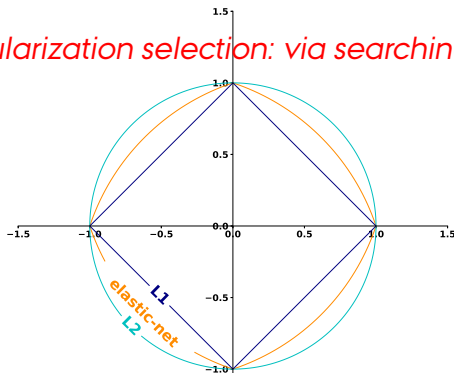
Elastic-net Penalization

And finally a combination of the two: an Elastic-net regularizer

$$\Omega(\mathbf{w}) = \beta \|\mathbf{w}\|_1 + (1 - \beta) \|\mathbf{w}\|_2^2$$

$$\tilde{J}_{\text{elastic}}(\mathbf{X}, \mathbf{y}; \mathbf{w}) = J(\mathbf{X}, \mathbf{y}; \mathbf{w}) + \alpha (\beta \|\mathbf{w}\|_1 + (1 - \beta) \|\mathbf{w}\|_2^2)$$

Regularization selection: via searching..



Optimizers

Momentum Optimization

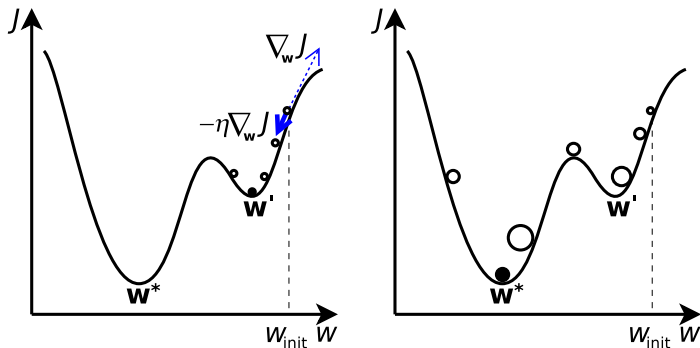
Normal GD algo

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} J$$

but now with added (physical) momentum

$$\mathbf{m} \leftarrow \beta \mathbf{m} - \eta \nabla_{\mathbf{w}} J$$

$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{m}$$



Optimizer selection: via searching..

Optimizers

Or solvers in Scikit-learn..

sklearn.neural_network.MLPRegressor

```
class sklearn.neural_network.MLPRegressor(hidden_layer_sizes=(100, ), activation='relu', *, solver='adam', alpha=0.0001,
batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True,
random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True,
early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10,
max_fun=15000)
```

[\[source\]](#)

Multi-layer Perceptron regressor.

This model optimizes the squared-loss using LBFGS or stochastic gradient descent.

New in version 0.18.

Parameters: **hidden_layer_sizes** : *tuple, length = n_layers - 2, default=(100,)*

The ith element represents the number of neurons in the ith hidden layer.

solver : {'lbfgs', 'sgd', 'adam'}, default='adam'

The solver for weight optimization.

- 'lbfgs' is an optimizer in the family of quasi-Newton methods.
- 'sgd' refers to stochastic gradient descent.
- 'adam' refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba

Note: The default solver 'adam' works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, 'lbfgs' can converge faster and perform better.

activation : {'identity', 'logistic', 'tanh', 'relu'}, default='relu'

Activation function for the hidden layer.

Optimizers

Or **optimizers** in Keras..



About Keras

Getting started

Developer guides

Keras API reference

Models API

Layers API

Callbacks API

Data preprocessing

Optimizers

Metrics

Losses

Built-in small datasets

Keras Applications

Utilities

Code examples

Why choose Keras?

» [Keras API reference](#) / Optimizers

Optimizers

Usage with `compile()` & `fit()`

An optimizer is one of the two arguments required for compiling a Keras model:

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential()
model.add(layers.Dense(64, kernel_initializer='uniform', input_shape=(10,)))
model.add(layers.Activation('softmax'))

opt = keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=opt)
```

You can either instantiate an optimizer before passing it to `model.compile()`, as in the above example, or you can pass it by its string identifier. In the latter case, the default parameters for the optimizer will be used.

```
# pass optimizer by name: default parameters will be used
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

Available optimizers

- SGD
- RMSprop
- Adam
- Adadelta
- Adagrad
- Adamax
- Nadam
- Ftrl

Optimizers

- ▷ Usage with `compile()` & `fit()`
- ▷ Usage in a custom training loop
- ▷ Learning rate decay scheduling
- ▷ Available optimizers
- ▷ Core Optimizer API
 - apply_gradients method
 - weights property
 - get_weights method
 - set_weights method

ML Models (or ML algorithms)

Models encountered so far

Some classifiers and regressors..

```
sklearn.neighbors.KNeighborsRegressor  
sklearn.linear_model.LinearRegression  
sklearn.linear_model.SGDClassifier  
sklearn.linear_model.SGDRegressor
```

Perhaps...

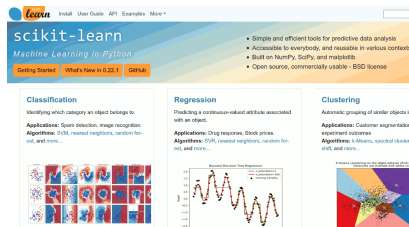
```
sklearn.naive_bayes.GaussianNB  
sklearn.naive_bayes.MultinomialNB
```

Not really or not in depth

```
sklearn.linear_model.Perceptron  
sklearn.linear_model.LogisticRegression  
sklearn.svm.SVC  
sklearn.svm.SVR  
sklearn.neural_network.MLPClassifier  
sklearn.neural_network.MLPRegressor
```

Or even more exotic models like..

- ▶ supervised ensemble: AdaBoost, Bagging, DecisionTree, RandomForest,...
- ▶ semi-supervised: ??
- ▶ unsupervised: K-means, manifolds, restricted Boltzmann machines,...
- ▶ clustering: K-means



The screenshot shows the scikit-learn website. At the top, there's a navigation bar with links: Install, User Guide, API, Examples, and More. Below this is the scikit-learn logo and the tagline "Machine Learning in Python". There are three buttons: "Getting Started", "What's New in 0.22.1", and "Github". To the right, there are four bullet points: "Simple and efficient tools for predictive data analysis", "Accessible to everybody and reusable in various contexts", "Built on NumPy, SciPy, and matplotlib", and "Open source, commercially usable - BSD license". Below the navigation bar, there are three main sections: "Classification" (identifying which category an object belongs to, with applications like spam detection and image recognition), "Regression" (predicting a continuous-valued attribute, with applications like drug responses and stock prices), and "Clustering" (automatic grouping of similar objects, with applications like customer segmentation and experiment outcomes). Each section has a small image or plot illustrating the concept.



ML Algorithm + Model Selection via Searching

What ML algorithm to choose?

- ▶ manual:
algorithm characteristics, \mathcal{O} complexity, etc.
browsing through Scikit-learn documentation,
...and also based on data assumptions.
- ▶ semi-automatic:
brute-force model search, and fun with python!

```
1 models = {  
2     SVC(gamma="scale"),  
3     SGDClassifier(tol=1e-3, eta0=0.1),  
4     GaussianNB()  
5 }  
6  
7 for i in models:  
8     i.fit(X_train, y_train)  
9     y_pred_test = i.predict(X_test)  
10    p = precision_score(y_test, y_pred_test, average='micro')  
11    print(f'{type(i).__name__:13s}: precision={p:0.2f}')
```

prints..

| | |
|----------------|--------|
| GaussianNB: | p=1.00 |
| SGDClassifier: | p=0.93 |
| SVC: | p=0.98 |

NOTE: Python set = {a, b}
Python dictionary = {a: x, b: y}

Model Selection via Grid Search

The hyperparameter-set for SGD linear regressor

```
1 class sklearn.linear_model.SGDRegressor(  
2     loss      = 'squared_loss', penalty      = 'l2',  
3     alpha     = 0.0001,               l1_ratio   = 0.15,  
4     tol       = None,                 shuffle    = True,  
5     verbose   = 0,                   epsilon    = 0.1,  
6     eta0      = 0.01,                power_t    = 0.25,  
7     n_iter_no_change=5,               warm_start = False,  
8     fit_intercept = True,             max_iter   = None,  
9     average     = False,             n_iter     = None,  
10    random_state  = None,             learning_rate='invscaling',  
11    early_stopping=False,             validation_fraction=0.1  
12 )
```



Search best hyperparameters in a (smaller) set, say

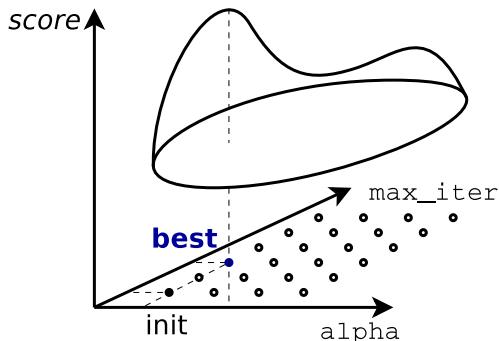
```
1 model = SGDClassifier()  
2 tuning_parameters = {  
3     'alpha': [ 0.001, 0.01, 0.1],  
4     'max_iter': [1, 10, 100, 100],  
5     'learning_rate': ('constant', 'optimal', 'invscaling', 'adaptive')  
6 }  
7 ..  
8 grid_tuned = GridSearchCV(model, tuning_parameters, ..
```

Model Selection via Grid Search

How to select 'best' set of hyperparameter—using brute force?

Gridsearch seen in 3D for the two hyperspace dimensions:

- ▶ $\alpha \in [1, 2, 3, \dots]$ (NOTE: linear range for this plot only,
- ▶ $\text{max_iter} \in [1, 2, 3, \dots]$ should be 1, 10, 100 or similar.)



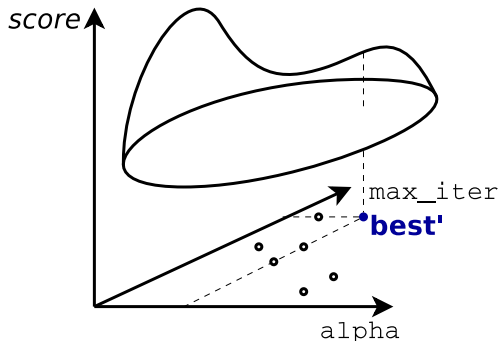
- ▶ why score and not J on z-axis?
- ▶ and what if there are many hyperparameters and many combinations? → Zzzzzzz!

Model Selection via Randomized Search

How to select 'best' set of hyperparameters—faster than brute force?

Replace `GridSearchCV()` with

```
RandomizedSearchCV(n_iter=100,...)
```



- ▶ faster, but will not yield the (sub) optimal score maximum,
- ▶ ...but does it matter in a huge hyperparameter search-space?