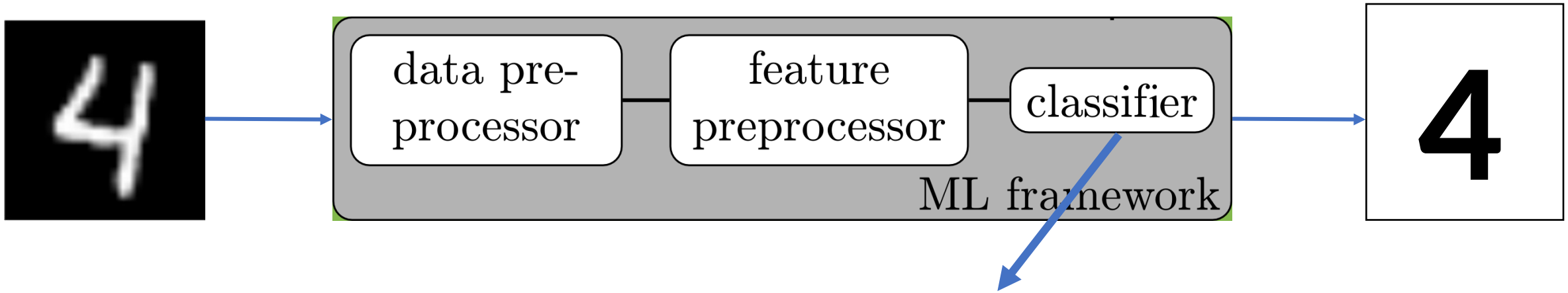
Meta-Knowledge Transfer Communication in Different Systems

Pavel Brazdil (University of Porto, INESC Tec, Portugal)
Jan N. van Rijn (Leiden University, the Netherlands)
Felix Mohr (Universidad de La Sabana, Colombia)
Henry Gouk (University of Edinburgh, Scotland)



Many design choices, hyperparameters, pre-processing methods, etc. The best decision differs per dataset

name	#λ
AdaBoost (AB)	4
Bernoulli naïve Bayes	2
decision tree (DT)	4
extreml. rand. trees	5
Gaussian naïve Bayes	-
gradient boosting (GB)	6
kNN	3
LDA	4
...	

Algorithm Selection Problem

- Canonical question: given dataset D, should we use a random forest or a neural network to maximize the final model's performance on a test set?
  - Binary variant, outdated
  - Many human intuition can be encoded and improved by using algorithmic reasoning
- Problem can be generalized to multi-class classification, ranking or regression problem
- Solutions considered
  - The algorithm selection framework (data characteristics, meta-features)
  - Utilizing earlier performance measures (active testing)
  - Learning curves

Hyperparameter Optimization Problem

- Canonical question: given dataset D and algorithm A, how should we set the hyperparameters of A to maximize the final model's performance on a test set?
- Concept of a search space, mixed hyperparameter types, hierarchy and conditions
- Concept of distance and assumption of smoothness

sklearn.svm.SVC

class sklearn.svm.**SVC**(*, *C=1.0*, *kernel='rbf'*, *degree=3*, *gamma='scale'*, *coef0=0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=-1*, *decision_function_shape='ovr'*, *break_ties=False*, *random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using `sklearn.svm.LinearSVC` or `sklearn.linear_model.SGDClassifier` instead, possibly after a `sklearn.kernel_approximation.Nystroem` transformer.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how `gamma`, `coef0` and `degree` affect each other, see the corresponding section in the narrative documentation: Kernel functions.

Read more in the User Guide.

Parameters:

C : *float, default=1.0*
Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default='rbf'*
Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

degree : *int, default=3*
Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

gamma : *{'scale', 'auto'} or float, default='scale'*
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

- if `gamma='scale'` (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features.

Changed in version 0.22: The default value of `gamma` changed from 'auto' to 'scale'.

coef0 : *float, default=0.0*
Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

sklearn.neural_network.MLPClassifier

class sklearn.neural_network.MLPClassifier(*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 classifier.

This model optimizes the log-loss function 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.

activation : *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*
Activation function for the hidden layer.

- 'identity', no-op activation, useful to implement linear bottleneck, returns f(x) = x
- 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- 'tanh', the hyperbolic tan function, returns f(x) = tanh(x).
- 'relu', the rectified linear unit function, returns f(x) = max(0, x)

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.

alpha : *float, default=0.0001*
L2 penalty (regularization term) parameter.

Combined Algorithm Selection and Hyperparameter Optimization Problem

- Combines the algorithm selection and hyperparameter optimization problem
- Given a search space consisting of algorithms and hyperparameters, find the algorithm and its hyperparameters that maximize the final model's performance
- Usually achieved by introducing a hierarchical search space, where the root node is the selection of algorithm, and the child nodes are the various options with their hyperparameters (there can be nested hierarchies)

Workflow Synthesis

- Various pre-processing components
  - Normalize data
  - PCA
  - Feature Selection
  - Feature Transformation
  - Many more!
- All with their own hyperparameters

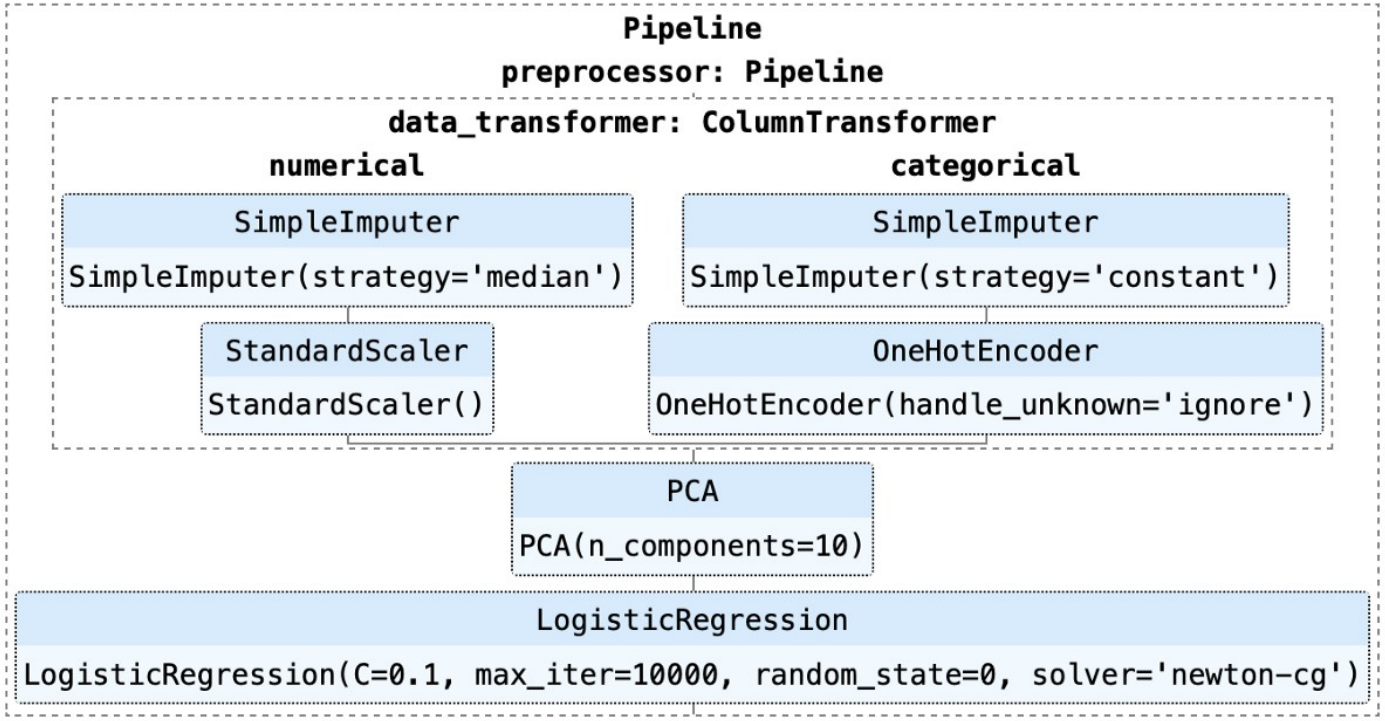


Figure from towardsdatascience.com

Transfer Learning for Few Shot Learning

- Traditional machine learning requires large amount of data, whereas humans only need few examples to learn a new concept
- Assumptions: humans are able to transfer experience from earlier learned tasks
- Solution types: Model Agnostic Meta-Learning (MAML), REPTILE

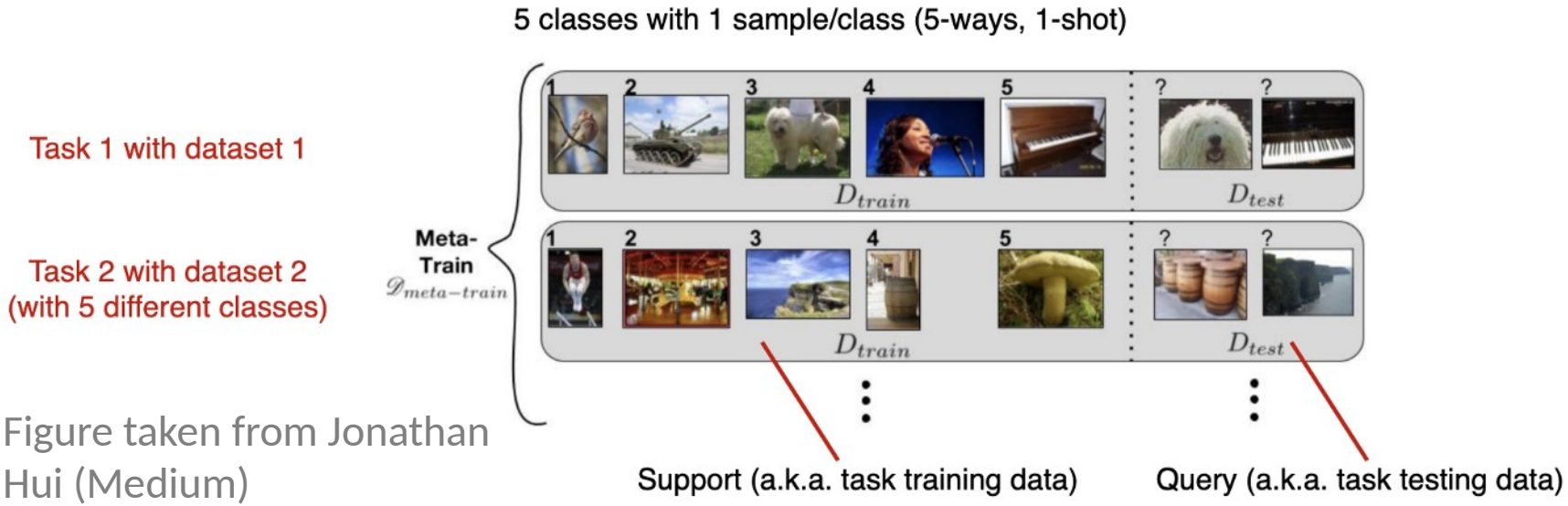


Figure taken from Jonathan Hui (Medium)

Pavel Brazdil
University of Porto

- Professor Emeritus
- PhD thesis on Metalearning 1981
- Algorithm selection
- Meta-features
- Active testing
- Learning curves

Jan N. van Rijn
Leiden University

- Assistant Professor
- OpenML.org
- Hyperparameter Importance (with Frank Hutter)
- Deep Learning / meta-learning
- AutoML for Neural Network Verification

Felix Mohr
Universidad de La Sabana

- Associate Professor
- MLPlan for Hyperparameter Optimization
- Naïve AutoML as a baseline for AutoML
- Learning Curves
- Metalearning competition

Henry Gouk
University of Edinburgh

- Research Fellow (eq. Assistant Professor)
- Deep Meta-learning
- Transfer learning
- Theory-driven machine learning

Combined Tutorial and Workshop

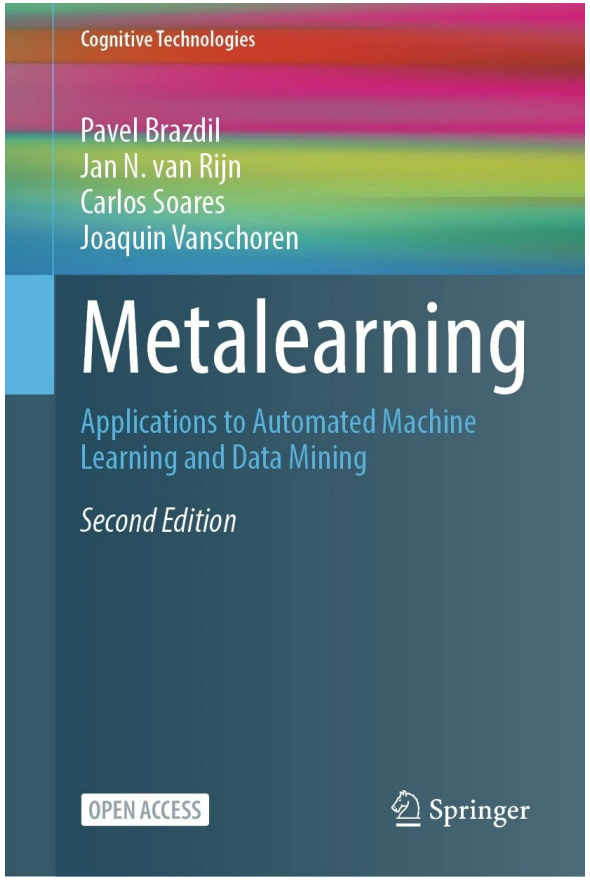
Program, slides and other material: https://metalearning-research.org/

- Tutorial on Metalearning
- Aimed at the book on Metalearning
- Various canonical techniques are being discussed
- http://metalearning-research.org/

- Workshop on meta-knowledge transfer and communication between systems
- Looks at the future of meta-learning
- Symbolic reasoning
- keynote speakers, Timothy Hospedales and Pascal Hitzler

Content of the Tutorial

(Mostly) based on the book: Metalearning: applications to Automated Machine Learning and Data Mining, 2nd edition (2022, Open Access)



- Introduction (JvR)
- Metalearning for Algorithm Selection (PB/FM)
- Meta-learning for Pipeline Optimization (FM)
- Short break (10m) & Q&A
- Few-shot learning (HG)
- Other developments and considerations (PB/JvR)
- Outlook (PB) & short Q&A