

# mlpy – Machine Learning Py

High-Performance Python/NumPy Based Package for Machine Learning

Davide Albanese, Stefano Merler, Giuseppe Jurman,  
Roberto Visintainer, Samantha Riccadonna,  
Silvano Paoli, Cesare Furlanello

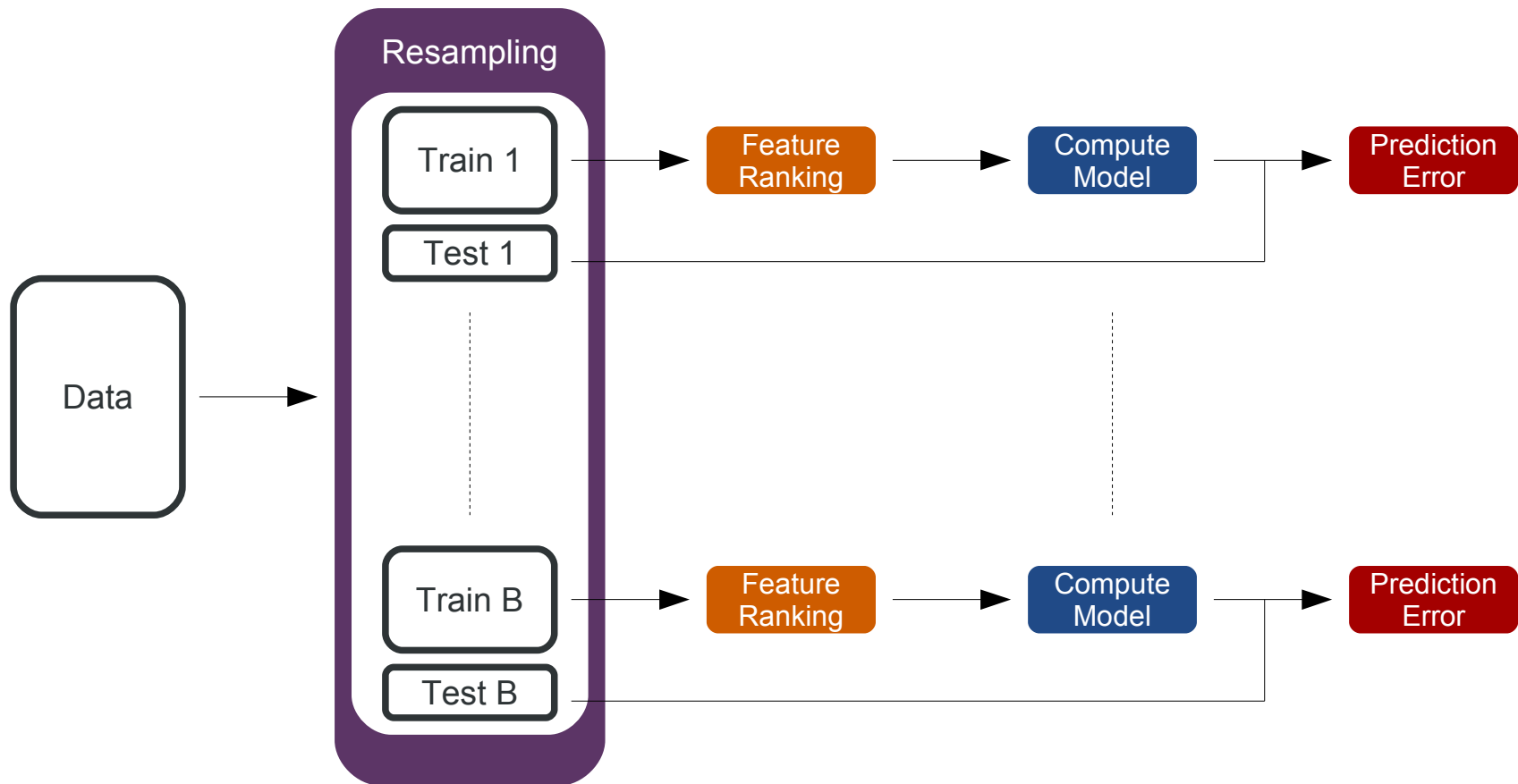
FBK-MPBA, Trento, Italy



# Introduction

- Obtaining honest performance estimates from a machine learning experiment usually requires fulfilling a complex pipeline of simpler tasks
- Those steps can be organized inside a Data Analysis Protocol (DAP) tailored by the researcher as suitable for the investigated problem, typically a predictive classification or regression task

# Introduction – Basic Example



- Resampling: to avoid incorrect estimates of the prediction error

# mlpy

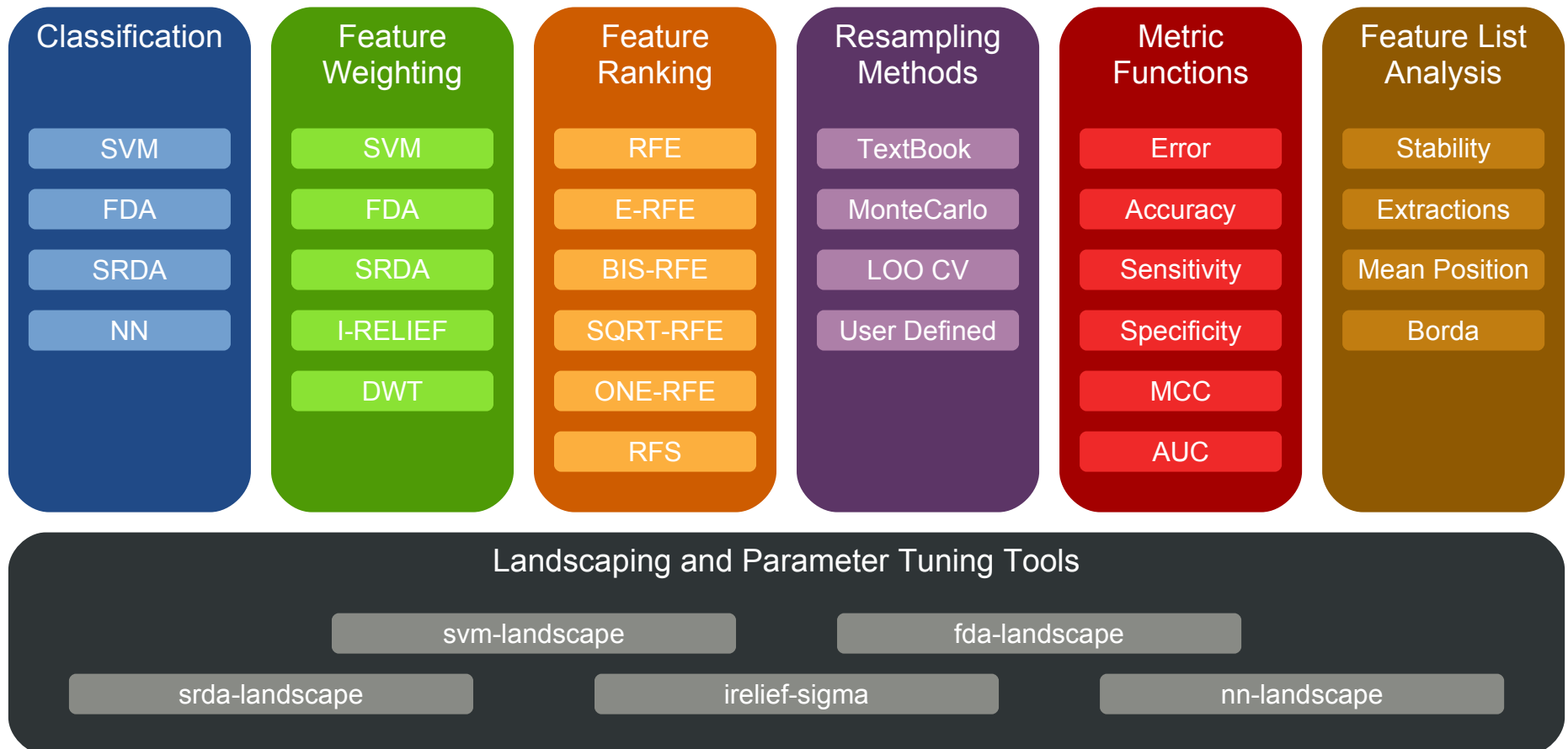
- Python/NumPy based package
- Open Source
- Implement different flavors of the machine learning functions required in:
  - Classification
  - Feature-ranking
  - Feature-lists-analysis

# mlpy - Features

- High modularity
- Ease of use
- Computationally efficient
  - intensive use of the NumPy module
  - parts with higher computational costs are implemented as internal C functions
- Suited for general-purpose machine learning tasks
- Its elective application field is bioinformatics and, in particular, the analysis of high-throughput data:
  - Genomics
  - Proteomics

# mlpy - Overview

- Main features can be divided into several groups according to their goals



# Classification

- **Support Vector Machines (SVMs)**
  - Implemented in C
  - Kernels:
    - Linear
    - Gaussian
    - Polynomial
    - Terminated Ramps
- **Discriminant Analysis (DA)**
  - Fisher (FDA)
  - Spectral Regression (SRDA)
- **Nearest Neighbors (NN)**
  - Implemented in C

# Classification

- Every classifier must be initialized with a specific set of parameters
- Two distinct methods are deployed for the training and the testing phases:
  - `compute(x, y)` : compute the model
  - `predict(p)` : predict model on a test-set
- Whenever possible, the real valued prediction can be obtained by the `realpred` variable





# Feature Weighting

- Feature weights coming directly from classifiers:
  - **Support Vector Machines** (SVMs) for each kernel
  - **Discriminant Analysis** (DA)
    - Fisher (FDA) – Cristianini method
    - Spectral Regression (SRDA)
- Classifier-independent methods:
  - **Iterative RELIEF** (I-RELIEF)
  - **Discrete Wavelet Transform** (DWT)

# Feature Weighting

- Every feature weighting method must be initialized with a specific set of parameters
- `weights(x, y)` method must be called to compute the feature weights

```
>>> from numpy import *
>>> from mlpy import *
>>>
>>> x = array([[1.0, 2.0, 3.0, 1.0],          # first sample
...           [1.0, 2.0, 3.0, 2.0],          # second sample
...           [1.0, 2.0, 3.0, 1.0]])         # third sample
>>> y = array([1, -1, 1])                    # classes
>>>
>>> myir = irelief(sigma = 1)                 # initialize irelief
>>> myir.weights(x, y)                       # compute the feature weights
array([ 0.,  0.,  0.,  1.]
```

# Feature Ranking

- The feature weights are used for selecting and ranking purposes inside one of the implemented schemes
  - **Recursive Feature Elimination** family (rfe, erfe, bisrfe, sqtrfe, onerfe)
  - **Recursive Forward Selection** family (rfs)
- A unique class for all the feature-ranking methods
- Compute with `compute(x, y, w)`, where `w` is a feature weighting method

# Feature Ranking - Example

```
>>> from numpy import *
>>> from mlpy import *
>>>
>>> x = array([[1.0, 2.0, 3.0, 1.0],           # first sample
...           [1.0, 2.0, 3.0, 2.0],           # second sample
...           [1.0, 2.0, 3.0, 1.0]])          # third sample
>>> y = array([1, -1, 1])                      # classes
>>>
>>> myrank = ranking(method = 'rfe')           # initialize ranking class
>>> mysvm = svm()                             # initialize svm class
>>> myrank.compute(x, y, mysvm)                # compute feature ranking
array([3, 2, 1, 0])    # the feature '3' is the most significant
```

# Resampling Methods

- The classification and feature ranking operations can be organized within a sampling procedure:
  - **Textbook** cross validation
  - **Monte-Carlo** cross validation
  - **Leave-one-out** cross validation
  - **User-defined** train/test
- Stratification over labels is also available
- The functions return the sample indexes

# Metric functions

- Performance assessment can be evaluated by a set of different measures (for binary classifiers):
  - **Error** (global, for positive and for negative samples)
  - **Accuracy**
  - **Sensitivity** and **Specificity**
  - **Matthews Correlation Coefficient**
  - **Area Under the ROC Curve**
- Inputs: true labels and predictions
- Variability assessed by Standard Deviation or Bootstrap Confidence Intervals (implemented in C)

# Feature List Analysis

- The ordered lists from the feature ranking experiments can be analyzed in terms of:
  - **Stability - Canberra** indicator on top-k positions (implemented in C)
  - **Extraction** indicator
  - **Mean position** indicator
- An optimal list on top-k sublists can be retrieved (Borda Count)



# Landscaping and Parameters Tuning Tools

- The package includes executable scripts to be used off-the-shelf for typical parameter tuning tasks such as SVM-kernel choice and optimization:
  - `svm-landscape`
  - `fda-landscape`
  - `nn-landscape`
  - `srda-landscape`
  - `irelief-sigma`
- User can choose the resampling method, range and number of steps
- Error and MCC are retrieved for each step

# Landscaping Tools - Example

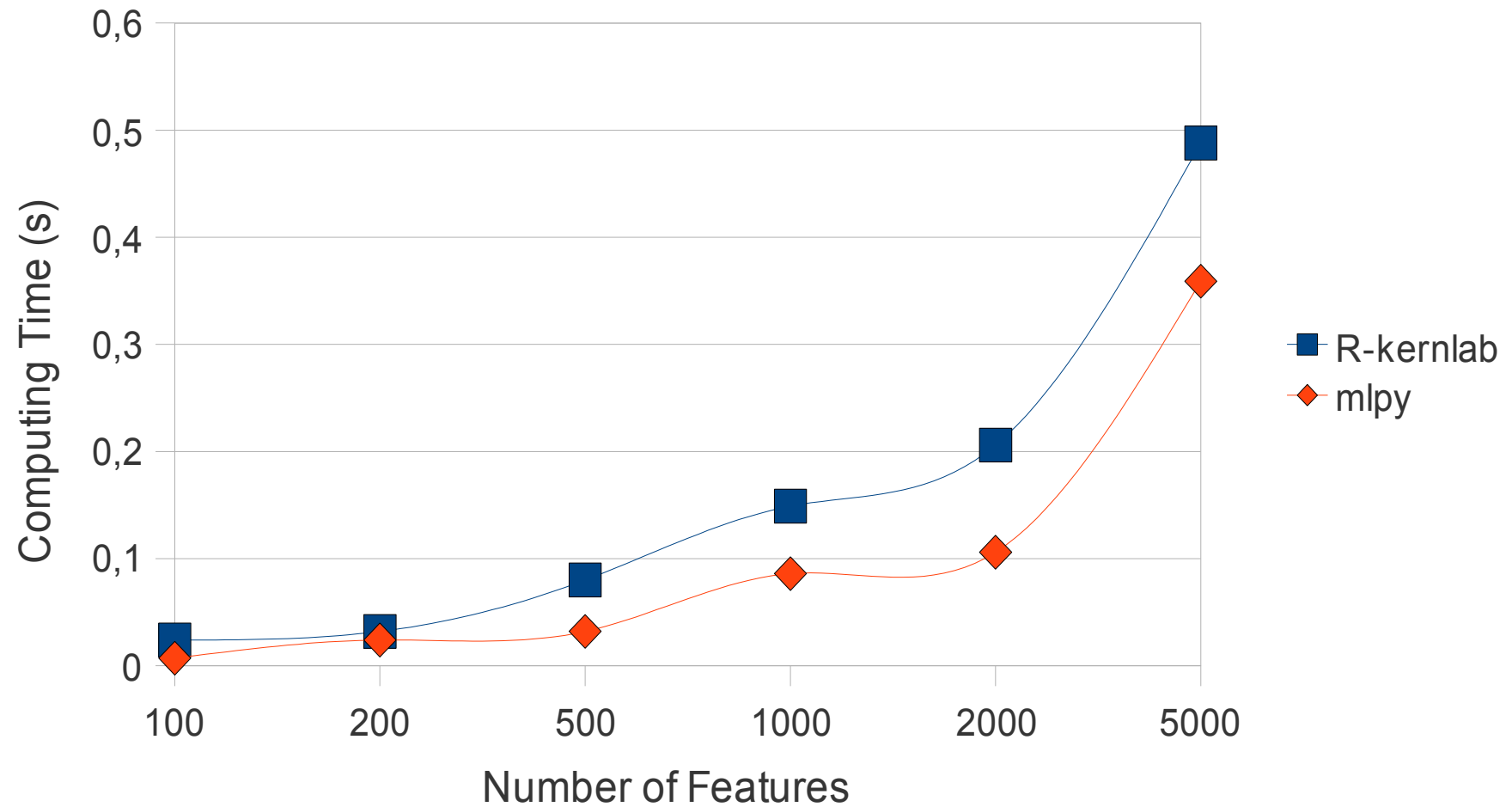
`svm-landscape` for SVM regularization parameter (C) tuning

- Stratified Monte-Carlo cross validation (4 sets, 10 train/test pairs)
- Standardize data
- 4 steps of C parameter

```
$ svm-landscape -d data.dat -c 4 10 -S -s -m -5 -M -2 -p 4
stratified monte carlo cv (4 sets, 10 pairs)
C 1.000000e-05: error 0.225000, mcc 0.590779
C 1.000000e-04: error 0.225000, mcc 0.583044
C 1.000000e-03: error 0.212500, mcc 0.610503
C 1.000000e-02: error 0.100000, mcc 0.814758
```

# R-kernlab VS mlpy

Linear SVM - 30 Samples



# Applications

- mlpy is the core of BioDCV ([biodcv.fbk.eu](http://biodcv.fbk.eu)) a distributed computing system for the complete validation of gene profiles
- mlpy is used by FBK-MPBA Research Unit for the MAQC-II project led by US Food and Drug Administration
- mlpy is now used on these datasets:
  - Copy Number Variation Data
  - Gene Expression Data (Microarray)
  - Proteomic Data (Mass Spectra)

Data can easily reach dimension of thousands of samples described up to one million of features

# Summary

- mlpy is a project of **Predictive Models for Biological and Environmental Data Analysis (MPBA)** Research Unit ([mpba.fbk.eu](http://mpba.fbk.eu)) at **Fondazione Bruno Kessler (FBK)** ([www.fbk.eu](http://www.fbk.eu))
- mlpy is free software. It is licensed under the GNU General Public License (GPL) version 3
- Homepage: [mlpy.fbk.eu](http://mlpy.fbk.eu)
- Email:
  - [albanese@fbk.eu](mailto:albanese@fbk.eu)
  - [jurman@fbk.eu](mailto:jurman@fbk.eu)
  - [visintainer@fbk.eu](mailto:visintainer@fbk.eu)
  - [furlan@fbk.eu](mailto:furlan@fbk.eu)