

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



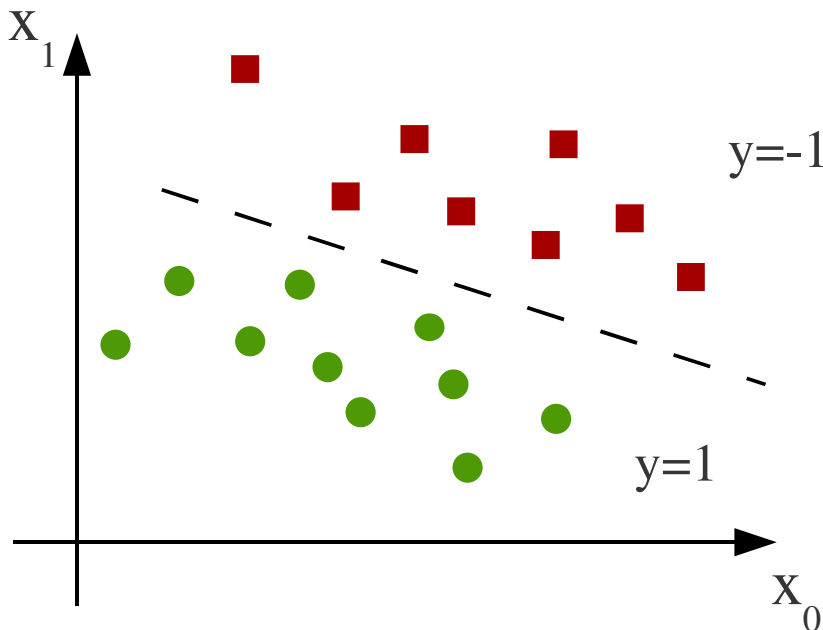
Machine Learning Tasks

Classification

$$D = \{s_i \equiv (x_i, y_i) \in \mathbb{R}^n \times Y, i = 1, \dots, m\} \Rightarrow \begin{cases} f: \mathbb{R}^n \rightarrow Y \\ f(x) \approx y \end{cases}$$

- s_i sample
- $\{x_i\}$ features
- y_i label

- **The goal:** finding a function that discriminates the two sets of data
- **No unique** solution



Example:

- 2 features, x_0 and x_1
- $Y \in \{-1, 1\}$

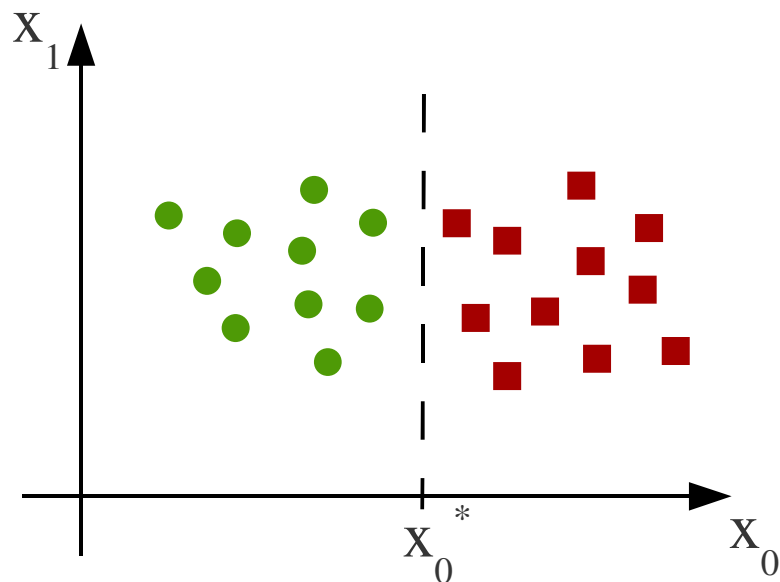
Machine Learning Tasks

Feature Selection

- In most classification models, identifying the more relevant features is as important as achieving high accuracy
- Very important in building predictors on high-throughput bioinformatics (mRNA and DNA arrays, proteomics, ...)

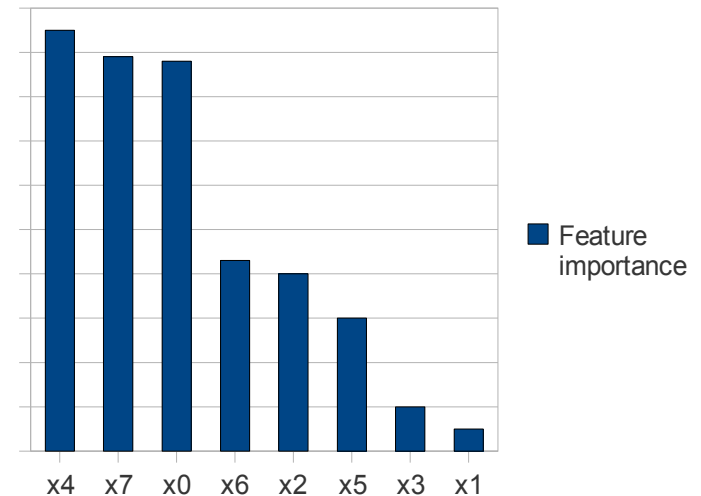
Example:

- 2 features
- feature x_0 is the most relevant



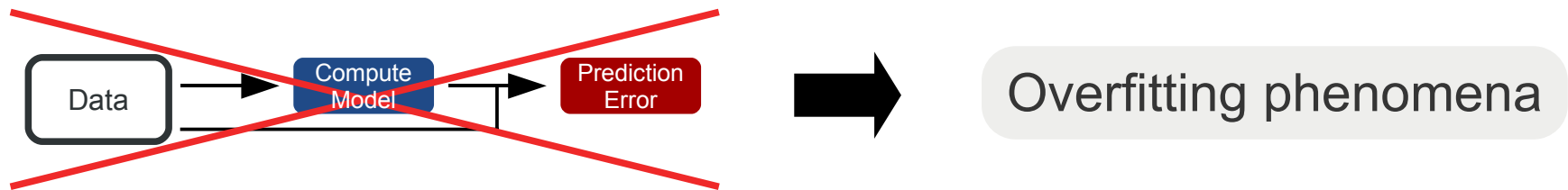
Example:

- 8 features
- feature x_4 , x_7 and x_0 are the most relevant

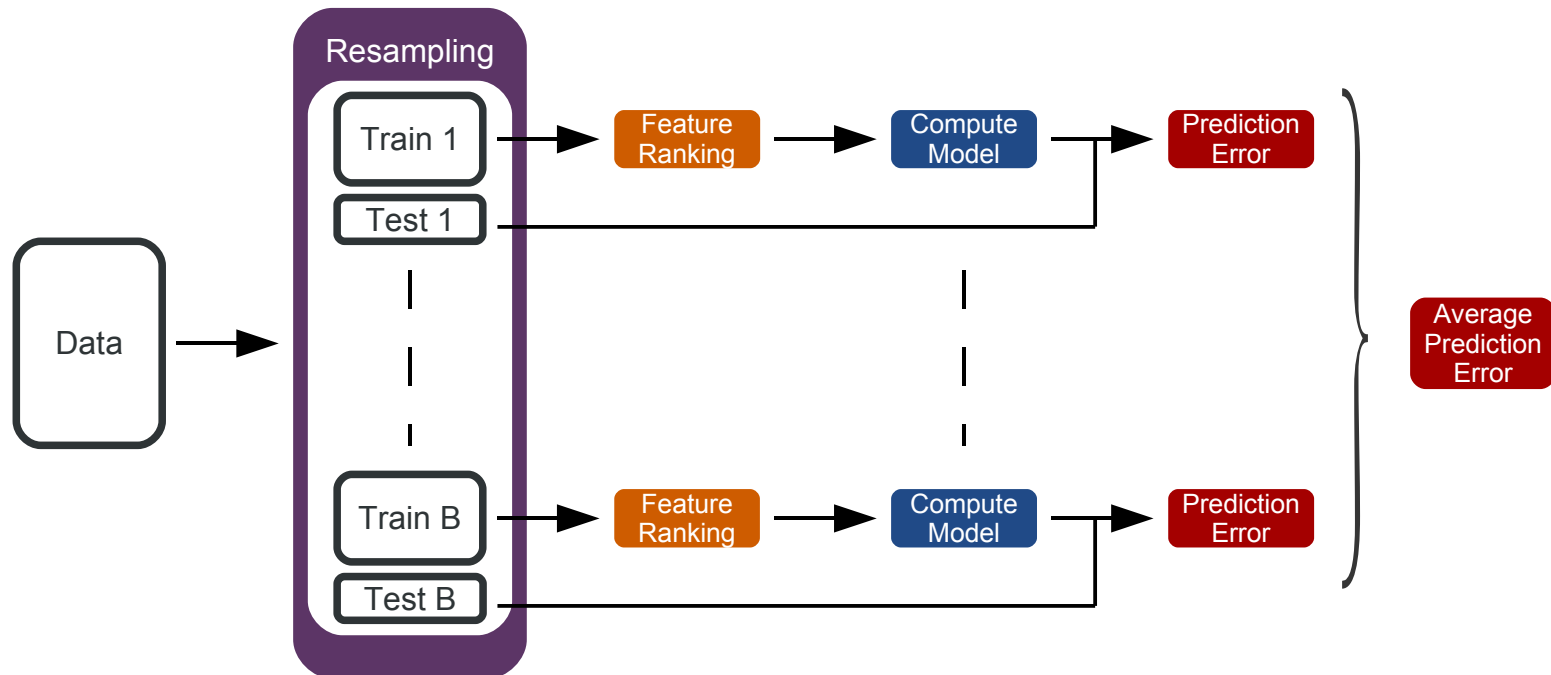


Machine Learning Tasks

Performance Evaluation



Obtaining **honest performance estimates** from a Machine Learning experiment usually requires fulfilling a complex pipeline of simpler tasks



mlpy



- Python/NumPy based package
- Implements different flavors of the machine learning functions required in:
 - Classification
 - Feature-ranking
 - Feature-lists-analysis
- Allows researchers to easily build customized combinations of complex pipelines

Main Features

- Suited for general-purpose machine learning tasks
- Elective application field: bioinformatics on high-throughput data
- High modularity
- Support of rapid prototyping of new algorithms
- Ease of use
- Open Source (GPLv3)

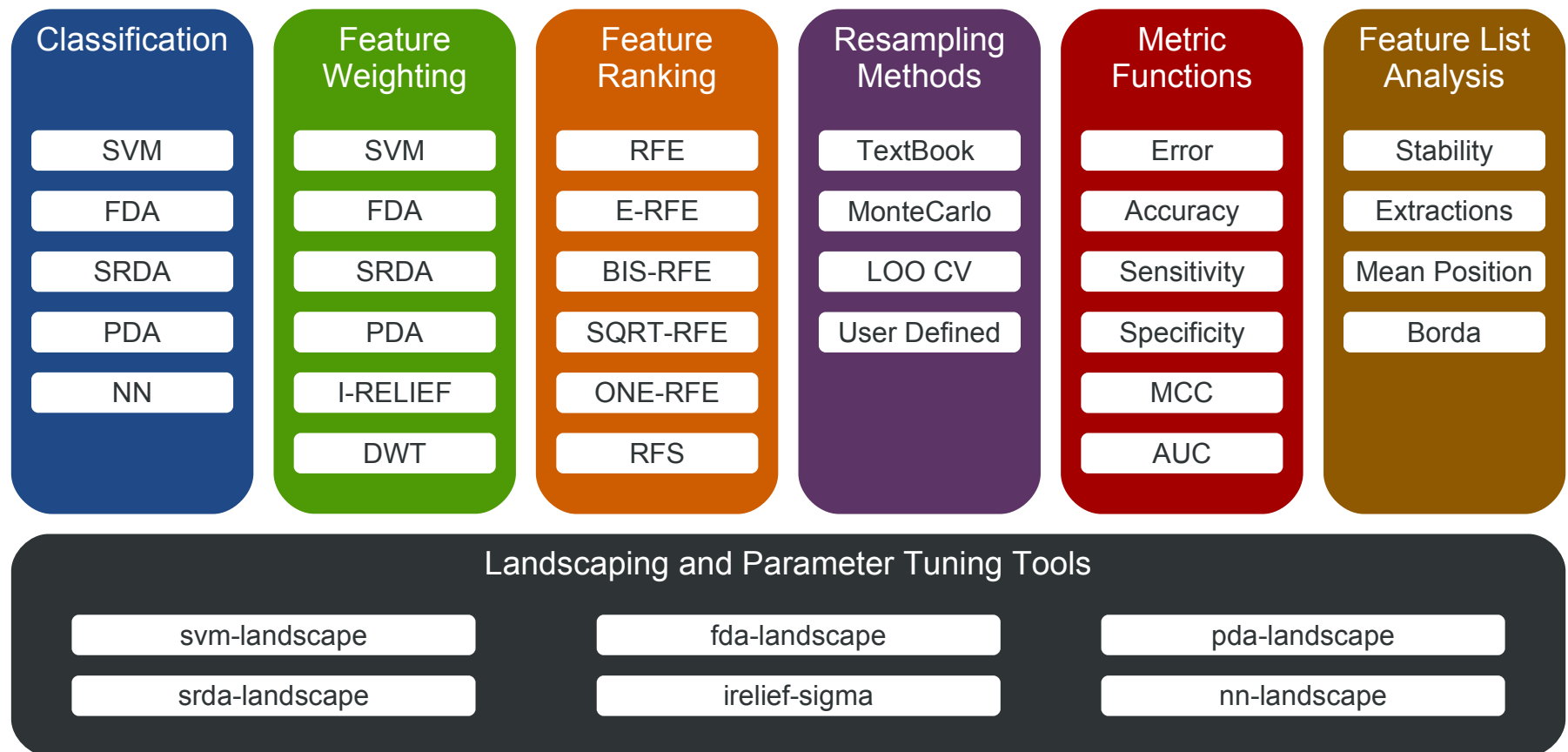
Main Features

- Computational efficiency and low memory use
 - intensive use of the NumPy module
 - parts with higher computational costs are implemented as internal C functions
- Source Code size: 464 KB
 - ~3000 lines of ANSI C code, ~2000 lines of Python code
- Multiplatform
 - Unix and Linux Systems
 - MS Windows Systems
- Requires:
 - Python ≥ 2.4
 - NumPy $\geq 1.0.3$



Overview

mlpy - version 1.2.7



Classification

Implemented Algorithms

- **Support Vector Machines** (SVMs) [Vapnik, 1995]
 - With Sequential Minimal Optimization (SMO) algorithm
 - Implemented in C
 - Kernels: Linear, Gaussian, Polynomial, Terminated Ramps [Merler and Jurman, 2006]
- **Nearest Neighbors** (NN) [Cover and Hart, 1967]
 - Implemented in C
- **Discriminant Analysis** (DA)
 - Fisher (KFDA) [Mika et al., 2001]
 - Penalized (PDA) [Ghosh, 2003]
 - Spectral Regression (SRDA) [Cai et al., 2008]

Classification

Usage

- 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 is stored in the internal `realpred` variable

Classification

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
>>>
>>> mysvm = svm(kernel = 'linear', C = 1)      # initialize svm
>>> mysvm.compute(x, y)                       # compute svm
1                                              # svm convergence
>>>
>>> p = array([4.0, 5.0, 6.0, 2.0])           # test point
>>> mysvm.predict(p)                          # predict svm model
-1
>>> mysvm.realpred                            # real-valued prediction
-0.5
```

Feature Weighting

Implemented Algorithms

- Classifier-derived methods:
 - **Support Vector Machines** (SVMs) for each kernel
 - **Discriminant Analysis** (DA)
 - Fisher (KFDA) – Cristianini method [Cristianini and Shawe-Taylor, 2006]
 - Spectral Regression (SRDA)
 - Penalized (PDA)
- Classifier-independent methods:
 - **Iterative RELIEF** (I-RELIEF) [Sun, 2007]
 - **Discrete Wavelet Transform** (DWT) [Subramani et al., 2006]

Feature Weighting

Usage and Example

- 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 [Guyon et al., 2002]
(RFE, ERFE [Furlanello et al., 2003], BISRFE, SQRTRFE, ONERFE)

Algorithm

Repeat

- Eliminate (a group of) the least relevant features
- Add them at the bottom of the ranked list
- Re-compute the feature weights on the reduced set of variables

Until all features are ordered

- **Recursive Forward Selection** family [Louw and Steel, 2006]
(RFS)

Feature Ranking

Usage and Example

- All ranking methods are included in a single class
- Rank by `compute(x, y, w)`, where `w` is a feature weighting method

```
>>> 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

- A few sampling procedures available:
 - **Textbook** (k-fold) cross validation
 - **Monte-Carlo** cross validation
 - **Leave-one-out** cross validation
 - **User-defined** train/test
- Stratification over labels is also available
- The functions return a list of tuples containing the train and the test 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 (AUC)**
- 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) [Jurman et al., 2008]
 - **Extraction** indicator
 - **Mean position** indicator
- An optimal list on top-k sublists can be retrieved (**Borda Count** [JC de Borda, 1781])

Landscaping and Parameters Tuning Tools

- The package includes executable scripts to be used off-the-shelf for landscaping and parameter tuning tasks:

svm-landscape

fda-landscape

nn-landscape

srda-landscape

pda-landscape

irelief-sigma

- User can choose the resampling method, range and number of steps
- Error, MCC and Canberra Distance are retrieved for each step

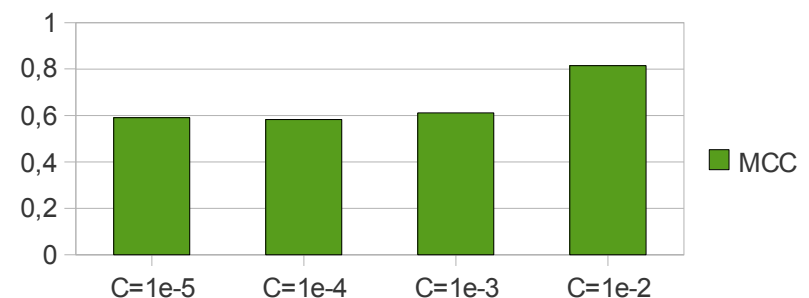
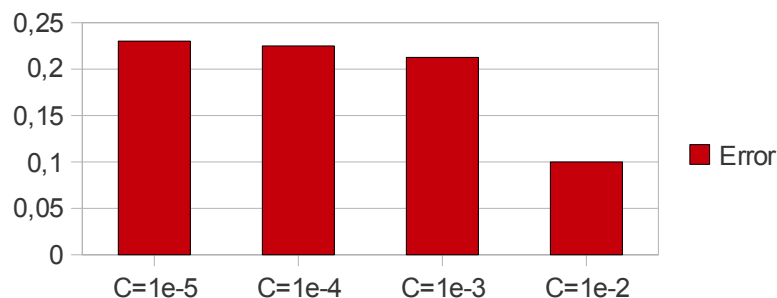
Landscaping Tools

Example

svm-landscape for SVM regularization parameter (C) tuning

- Stratified Monte-Carlo cross validation
- 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
```



Applications

- **mlpy** is the core of BioDCV (biodcv.fbk.eu) a distributed computing system for biomarker discovery
- **mlpy** is used by FBK-MPBA Research Unit for the MAQC-II project led by US Food and Drug Administration
- Runs on HPC facilities, Linux cluster at FBK and European Grid for E-scienceE (EGEE)
- **mlpy** is now used on datasets with dimension of **thousands of samples** and **millions of features**:
 - Copy Number Variation (CNVs)
 - Single Nucleotide Polymorphism (SNP)
 - Gene Expression (Microarray)
 - Proteomic (Mass Spectra)

Info

- **mlpy** is a project of:

Predictive Models for Biological and Environmental Data Analysis (MPBA) Research Unit (mpba.fbk.eu)
at **Fondazione Bruno Kessler (FBK)** (www.fbk.eu)

- Partially supported by AIRC-IFOM
- **mlpy** is free software. It is licensed under the GNU General Public License (GPL) version 3
- Homepage: mlpy.fbk.eu