high-performance Python package for predictive modeling

Davide Albanese, Stefano Merler, Giuseppe Jurman, Roberto Visintainer, Cesare Furlanello
FBK – MPBA Research Unit, Trento, Italy

NIPS Workshop on Machine Learning Open Source Software
12th December 2008
Main Issues
(in developing a Open Source ML library)

**Modularity**: setting up a correct methodological workflow requires fulfilling a complex pipeline of basic tasks

**Maintenance**: rapid prototyping of new algorithms allows keeping the library updated to state-of-the-art

**Reproducibility**: the experiments should be repeatable, so every single step should be exactly replicable

**Usability**: researchers should be able to build their own methodological pipeline

**Efficiency**: computing time and memory usage are relevant in most of ML tasks
Our Answer

Dynamic object-oriented programming language
- very clear, readable syntax
- portable
- stable and mature

Python module
- provides fast N-dimensional array manipulation
- basic linear algebra functions
- tools for integrating C/C++ code

Well established and popular programming language
- efficiency
- code portability
- code reusing
mlpy v1.2.7 - Overview

Computationally efficient with low memory use
- internal ANSI C99 functions
- intensive use of the NumPy module

Multiplatform
- Unix and GNU/Linux
- MS Windows
- Mac OS X

Compact
- Source Code size: 464 KB
- ~3000 lines of ANSI C99 code
- ~2000 lines of Python code

Requirements
- libc
- Python >= 2.4
- NumPy >= 1.0.3
mlpy v1.2.7 - Structure

Provides high level procedures that support the design of rich *Data Analysis Protocols (DAPs)* for **predictive classification** and **feature selection**

Elective application field: **bioinformatics** on **high-throughput data**
Classification

Implemented Algorithms

• **Support Vector Machines** [Vapnik, 95]
  – Sequential Minimal Optimization (SMO) algorithm
  – Implemented in C
  – Four Kernels: Linear, Gaussian, Polynomial, Terminated Ramps [Merler and Jurman, 06]

• **Nearest Neighbors** [Cover and Hart, 67]
  – Implemented in C

• **Discriminant Analysis**
  – Fisher (KFDA) [Mika et al., 01]
  – Penalized (PDA) [Ghosh, 03]
  – Spectral Regression (SRDA) [Cai et al., 08]
  – Diagonal Linear (DLDA – mlpy v1.2.8) [Pique-Regi, 06]

• **classifier(params)** for classifier initialization.

• **.compute(x, y)**
  the method for the training phase computing the model. x stores the data (samples x features) and y collects the corresponding labels.

• **.predict(p)**
  the method for the testing phase predicting the model on a test-set. Test points are stored in p.

• **.realpred**
  whenever possible it stores the real valued prediction.

• **.classifier__param**
  internal classifier parameters are accessible.
Feature Weighting

Implemented Algorithms

• directly within SVM classifiers:
  – for all implemented kernels

• directly with DA:
  – Fisher (KFDA) – Cristianini method [Cristianini and Shawe-Taylor, 06]
  – Spectral Regression (SRDA)
  – Penalized (PDA)
  – Diagonal Linear (DLDA – mlpy v1.2.8)

• Iterative RELIEF (I-RELIEF) [Sun, 07]

• Discrete Wavelet Transform (DWT) [Subramani et al., 06]
Feature Ranking

Implemented Algorithms

• Recursive Feature Elimination [Guyon et al., 02]
  – (Standard) RFE
  – Entropy-based RFE [Furlanello et al., 03]
  – Bisection RFE
  – Square-Root RFE

• Recursive/Sequential Forward Selection (R/S FS) [Louw and Steel, 06]

• One-step ranking

• `ranking(method, params)` for feature ranking initialization.

• `compute(x, y, w)` the method computing the feature ranking. \( w \) is the feature weighting method. It returns the list of the ranked features.
The ordered lists from the feature ranking experiments can be analyzed by:

- **canberra**\( \left( lists, k \right) \):
  - Canberra indicator on top-k positions [Jurman et al., 08]

- **canberraQ**\( \left( lists \right) \) (mlpy v1.2.8):
  - Canberra indicator on lists of different length

- **borda**\( \left( lists, k \right) \)
  - Extraction indicator
  - Mean position indicator
  - Optimal list on top-k sublists

JC de Borda, 1781
Metric functions

A set of different measure are available for the classifier performance assessment:

- **Error**
  - $\text{err} = (fp + fn) / ts$
  - $\text{errp} = fp / ap$
  - $\text{errn} = fn / an$

- **Accuracy**
  - $\text{acc} = (tp + tn) / ts$

- **Sensitivity and Specificity**
  - $\text{sens} = tp / ap$
  - $\text{spec} = tn / an$

- **Matthews Correlation Coefficient (MCC)**
  - $\text{MCC} = \frac{(tp \cdot tn) - (fp \cdot fn)}{\sqrt{(tp + fn)(tp + fp)(tn + fn)(tn + fp)}}$

- **Area Under the ROC Curve (AUC)**

Variability assessed by Bootstrap Confidence Intervals
Resampling Methods

A few sampling procedures available with focus on replicability:

- Textbook (k-fold) cross validation
- Monte-Carlo cross validation
- Leave-one-out cross validation
- User-defined train/test

Method\( (\text{params}) \) returns a list of tuples which contain the sample indexes for each replicate. For example:

\[
\text{training} \quad \text{test} \\
[((2, 4, 5, 6), [0, 1, 3]),
((0, 1, 5, 6), [2, 3, 4]),
((0, 1, 2, 3), [4, 5, 6]),
((1, 2, 3, 4), [0, 5, 6]),
((0, 2, 4, 6), [1, 3, 5]),
((0, 1, 2, 5), [3, 4, 6])
\]

StratMethod\( (\text{params}) \) the Strat prefix indicates that stratification over labels is available
Landscaping and Parameters Tuning Tools

The package includes executable scripts to be used off-the-shelf for landscaping and parameter tuning tasks. These scripts implement a basic DAP.

- **svm-landscape** (regularizer)
- **srda-landscape** (alpha parameter)
- **fda-landscape** (regularizer)
- **pda-landscape** (regressions steps)
- **nn-landscape**
- **irelief-sigma** (sigma parameter)

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

- mlpy is used by FBK-MPBA Research Unit for the MAQC-II project led by US FDA
- Runs on HPC facilities, Linux cluster at FBK and European Grid for E-sciencE (EGEE)
- mlpy is now used on datasets of $10^5$ samples and tested for up to $10^6$ features:
  - Copy Number Variation (CNVs)
  - Single Nucleotide Polymorphism (SNP)
  - Gene Expression (Microarray)
  - Proteomic (Mass Spectra)
- Partially supported by AIRC-IFOM
- Licensed under the GNU General Public License (GPL) version 3
- Homepage: https://mlpy.fbk.eu