Problem Based Learning 2018
Introduction to Machine Learning with Python

L. Richter
Department of Computer Science
Technische Universität München

Monday, Jun 25th
Overview

1. Motivation

2. Libraries
   - Numpy
   - Pandas
   - Scikit-learn

3. Workflows
Python is a general purpose language → easy integration with other software
comprehensive machine learning library available
large and vivid support community
supports high performance computations as well as interactive data exploration
Useful Libraries:

**Numpy**  Underlying high-performance data structures and associated functions  

(SciPy) Collection of mathematical algorithms and convenience function built on Numpy  

**Pandas**  High level data interface for preprocessing and manipulation  

**Scikit-learn**  Machine learning library
Numpy

- [http://www.numpy.org](http://www.numpy.org)
- `import numpy as np`
- Provides high-performance data structure: `ndarray`
- Convenience function to generate and handle `ndarray` objects
- Sometimes `array` is used as alias for `ndarray`
Numpy Array Properties

- `numpy.array` ≠ `array.array`
- **homogeneous** multidimensional array, i.e. all items have the same type
- if the input data has different types is upcasted to the least general type matching all present types
- the individual items are indexed via tuples of positive integers
**ndarray Attributes**

- **ndim**: number of dimensions (axes)
- **shape**: the dimensions of the array, i.e. the number of elements on each axis
- **size**: the total number of elements of the array
- **dtype**: an object describing the type of the elements, e.g. int8, int16, int32, int64, float32, float64, ...
- **itemsize**: the memory footprint of each element in bytes
- **(data)**: the buffer containing the actual elements
Creation

- conversion from other Python structures e.g., lists, tuples
- intrinsic numpy array by array creation objects e.g., arange, ones, zeros, etc.
- creating arrays from raw bytes through the use of strings or buffers
- use of special library functions e.g., random
**ndarray Code Snippets**

- `np.array([1,2,3,4])`
- `np.array([(1.5,2,3), (4,5,6)])`
- `np.arange([start, ]stop, [step, ]dtype=None)`, if step is specified start must also be given
- `np.reshape(shape)`, the shape must be compatible with the number of elements
- `np.arange(12).reshape(4,3)`
- special value: -1, the value of this axis is calculated using size and the other dimension(s)
- `np.ravel(x)` flattens array `x` to one dimension (eq. to `reshape(-1)`)
```python
>>> a = np.arange(12).reshape(4,3)
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])

>>> a.reshape(1,-1)
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11]])

>>> a.reshape(12).shape
(12,)

>>> a.reshape(-1,).shape
(12,)

>>> a.reshape(-1,)
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```
- arithmetic operators on arrays apply *elementwise* and return a *new* array
- use the *dot* method for matrix multiplication
- the $+=$, $\ast=$, ... operators act in place, modifying the existing array
- stacking:
Indexing, Slicing, Iterating

- index which elements you are interested in: \([\text{incl\_start:excl\_end:step\_size}]\)
- if \(\text{step\_size} < 0\) then you iterate backwards
- in a multidimensional array you can specify one index per axis, separated by comma
- missing indices are considered complete slices (:) 
- you can indicate missing indices with dots ...
Pandas

- “Numpy arrays on steroids”
- spreadsheet-like interface
- allows to set your own row and column labels
- heterogeneously-typed columns
- **goal**: becoming the most powerful and flexible open source data analysis / manipulation tool available in any language
Series: 1D labeled homogeneously-typed array
DataFrame: general 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed column
Panel: 3D data structure, but now deprecated
you can create a `pandas.DataFrame` in a number of ways:

- dict of 1D ndarrays, lists, dicts or Series
- 2D numpy.ndarray
- Structured or record ndarray
- another DataFrame
- from file, e.g. in csv format
Element Access

- similar to R data frame
- `df['col']` selects a column
- `df['col1','col2']` selects the rows col1 and col2
- `df[['col1','col2']]` selects columns col1 and col2
- `.loc[row_indexer, column_indexer]` is the typical label based access operator
- `.iloc[row_indexer, column_indexer]` is the typical integer based access operator
- supports slicing with `:`
Useful Attributes

- **columns**: access the column labels
- **items**: access the row labels
- **head(), tail()**: show the first or last 5 lines of the DataFrame
- **values**: returns the raw values as *ndarray*
- **sum(), mean(), info(), describe()**: much more sophisticated method for filtering, aggregation, joining, etc.
Scikit-learn

- machine learning library
- on top of numpy and scipy
- found in package sklearn
Structure

- not uniform!
- most of the useful functions are grouped in sub-folders
- check out the documentation for the explanation and the required import statements

http://scikit-learn.org/stable/
Typical Steps

- import and clean your data using Pandas
- convert nominal features into numerical ones with one hot encoding using the `get_dummies()` DataFrame method
- recode the class label if necessary
- export the feature for the machine learning step as 2D ndarray
- export the labels as 1D ndarray
Typical Steps II

- split your data into training and test set
- fit the classifier with the training set
- predict on the test set
- calculate the performance