mlpy Documentation

Release 3.5.0

Davide Albanese

Sep 12, 2025

CONTENTS

1 Install **3**
1.1 Download . . . 3
1.2 Installing on *nix from source . . . 3
1.3 Installing on Windows Xp/Vista/7 from binary installer . . . 4

2 Introduction **5**
2.1 Conventions . . . 5

3 Tutorial **7**
3.1 Tutorial 1 - Iris Dataset . . . 7

4 Linear Methods for Regression **11**
4.1 Ordinary Least Squares . . . 11
4.2 Ridge Regression . . . 12
4.3 Partial Least Squares . . . 13
4.4 Last Angle Regression (LARS) . . . 13
4.5 Elastic Net . . . 14

5 Linear Methods for Classification **17**
5.1 Linear Discriminant Analysis Classifier (LDAC) . . . 17
5.2 Basic Perceptron . . . 20
5.3 Elastic Net Classifier . . . 21
5.4 Logistic Regression . . . 23
5.5 Support Vector Classification . . . 23
5.6 Diagonal Linear Discriminant Analysis (DLDA) . . . 23
5.7 Golub Classifier . . . 24

6 Kernels **25**
6.1 Kernel Functions . . . 25
6.2 Kernel Classes . . . 25
6.3 Functions . . . 25
6.4 Centering in Feature Space . . . 26
6.5 Make a Custom Kernel . . . 27

7 Non Linear Methods for Regression **29**
7.1 Kernel Ridge Regression . . . 29
7.2 Support Vector Regression . . . 30

8 Non Linear Methods for Classification **31**
8.1 Parzen-based classifier . . . 31
8.2 Support Vector Classification . . . 32

8.3 Kernel Fisher Discriminant Classifier . . . 32
8.4 k-Nearest-Neighbor . . . 32
8.5 Classification Tree . . . 34
8.6 Maximum Likelihood Classifier . . . 35

9 Support Vector Machines (SVMs) 37
9.1 Support Vector Machines from [LIBSVM] . . . 37
9.2 Kernel Adatron . . . 39

10 Large Linear Classification from [LIBLINEAR] 43

11 Cluster Analysis 47
11.1 Hierarchical Clustering . . . 47
11.2 Memory-saving Hierarchical Clustering . . . 47
11.3 k-means . . . 47

12 Algorithms for Feature Weighting 49
12.1 Iterative RELIEF . . . 49

13 Feature Selection 51
13.1 Recursive Feature Elimination . . . 51

14 Dimensionality Reduction 53
14.1 Linear Discriminant Analysis (LDA) . . . 53
14.2 Spectral Regression Discriminant Analysis (SRDA) . . . 53
14.3 Kernel Fisher Discriminant Analysis (KFDA) . . . 53
14.4 Principal Component Analysis (PCA) . . . 55
14.5 Fast Principal Component Analysis (PCAFast) . . . 56
14.6 Kernel Principal Component Analysis (KPCA) . . . 57

15 Cross Validation 59
15.1 Leave-one-out and k-fold . . . 59
15.2 Random Subsampling (*aka MonteCarlo*) . . . 59
15.3 All Combinations . . . 59

16 Metrics 61
16.1 Classification . . . 61
16.2 Regression . . . 62

17 A Set of Statistical Functions 63

18 Canberra Distances and Stability Indicator of Ranked Lists 65
18.1 Canberra distance . . . 65
18.2 Canberra Distance with Location Parameter . . . 65
18.3 Canberra Stability Indicator . . . 65

19 Borda Count 67

20 Find Peaks 69

21 Dynamic Time Warping (DTW) 71
21.1 Standard DTW . . . 71
21.2 Subsequence DTW . . . 72

22 Longest Common Subsequence (LCS) 73
22.1 Standard LCS . . . 73

22.2 LCS for real series . . . 73

23 `mlpy.wavelet` - Wavelet Transform **75**
23.1 Padding . . . 75
23.2 Discrete Wavelet Transform . . . 75
23.3 Undecimated Wavelet Transform . . . 75
23.4 Continuous Wavelet Transform . . . 75

24 Short Guide to Centering and Scaling **77**

25 Indices and tables **79**

Bibliography **81**

Python Module Index **83**

Index **85**

Release
3.5

Date
Sep 12, 2025

Homepage
http://mlpy.sourceforge.net

Machine Learning PYthon (mlpy) is a high-performance Python library for predictive modeling.

This reference manual details functions, modules, and objects included in mlpy.

CHAPTER

ONE

INSTALL

1.1 Download

Download latest version for your OS from http://sourceforge.net/projects/mlpy/files/

1.2 Installing on *nix from source

On GNU/Linux, OSX and FreeBSD you need the following requirements:

- GCC
- Python >= 2.6 or 3.X
- NumPy >= 1.3.0 (with header files)
- SciPy >= 0.7.0
- GSL >= 1.11 (with header files)

From a terminal run:

```
$ python setup.py install
```

If you don't have root access, installing mlpy in a directory by specifying the `--prefix` argument. Then you need to set `PYTHONPATH`:

```
$ python setup.py install --prefix=/path/to/modules
$ export PYTHONPATH=$PYTHONPATH:/path/to/modules/lib/python{version}/site-packages
```

If the GSL header files or shared library are in non-standard locations on your system, use the `--include-dirs` and `--rpath` options to `build_ext`:

```
$ python setup.py build_ext --include-dirs=/path/to/header --rpath=/path/to/lib
$ python setup.py install
```

1.3 Installing on Windows Xp/Vista/7 from binary installer

Requirements:

- Python 2.6, 2.7, 3.1, 3.2 Windows installer (x86)
- NumPy >= 1.3.0 win32 installer
- SciPy >= 0.8.0 win32 installer

The GSL library is pre-compiled (by Visual Studio Express 2008) and included in mlpy.

Download and run the mlpy Windows installer (.exe).

CHAPTER

TWO

INTRODUCTION

2.1 Conventions

- x is a matrix $n \times p$ which represents a set of n samples in $\Re^p$.
- y is a vector n which represents the target values (integers in classification problems, floats in regression problems).

CHAPTER

THREE

TUTORIAL

If you are new in Python and NumPy see: http://docs.python.org/tutorial/ http://www.scipy.org/Tentative_NumPy_Tutorial and http://matplotlib.sourceforge.net/.

A learning problem usually considers a set of p-dimensional samples (observations) of data and tries to predict properties of unknown data.

3.1 Tutorial 1 - Iris Dataset

The well known Iris dataset represents 3 kinds of Iris flowers with 150 observations and 4 attributes: sepal length, sepal width, petal length and petal width.

A dimensionality reduction and learning tasks can be performed by the mlpy library with just a few number of commands.

Download `Iris dataset`

Load the modules:

```
>>> import numpy as np
>>> import mlpy
>>> import matplotlib.pyplot as plt # required for plotting
```

Load the Iris dataset:

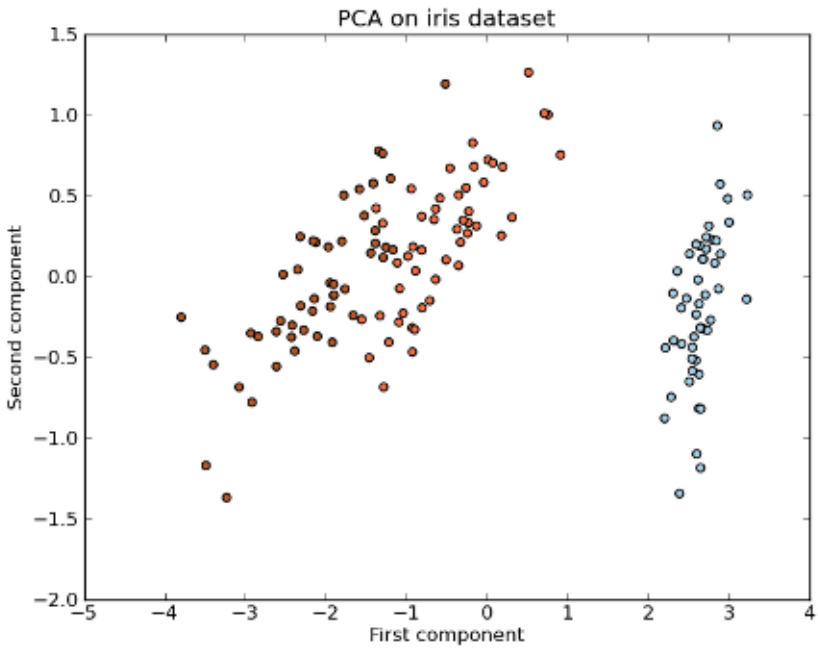
```
>>> iris = np.loadtxt('iris.csv', delimiter=',')
>>> x, y = iris[:, :4], iris[:, 4].astype(int) # x: (observations x attributes) matrix,␣
↪y: classes (1: setosa, 2: versicolor, 3: virginica)
>>> x.shape
(150, 4)
>>> y.shape
(150, )
```

Dimensionality reduction by Principal Component Analysis (PCA)

```
>>> pca = mlpy.PCA() # new PCA instance
>>> pca.learn(x) # learn from data
>>> z = pca.transform(x, k=2) # embed x into the k=2 dimensional subspace
>>> z.shape
(150, 2)
```

Plot the principal components:

```
>>> plt.set_cmap(plt.cm.Paired)
>>> fig1 = plt.figure(1)
>>> title = plt.title("PCA on iris dataset")
>>> plot = plt.scatter(z[:, 0], z[:, 1], c=y)
>>> labx = plt.xlabel("First component")
>>> laby = plt.ylabel("Second component")
>>> plt.show()
```



Learning by Kernel Support Vector Machines (SVMs) on principal components:

```
>>> linear_svm = mlpy.LibSvm(kernel_type='linear') # new linear SVM instance
>>> linear_svm.learn(z, y) # learn from principal components
```

For plotting purposes, we build the grid where we will compute the predictions (*zgrid*):

```
>>> xmin, xmax = z[:,0].min()-0.1, z[:,0].max()+0.1
>>> ymin, ymax = z[:,1].min()-0.1, z[:,1].max()+0.1
>>> xx, yy = np.meshgrid(np.arange(xmin, xmax, 0.01), np.arange(ymin, ymax, 0.01))
>>> zgrid = np.c_[xx.ravel(), yy.ravel()]
```

Now we perform the predictions on the grid. The *pred()* method returns the prediction for each point in zgrid:

```
>>> yp = linear_svm.pred(zgrid)
```

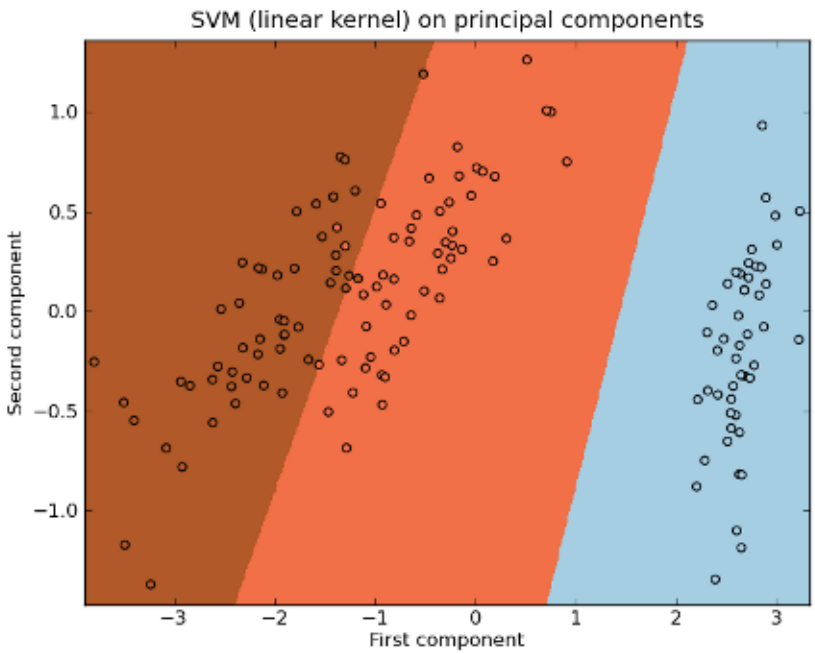
Plot the predictions:

```
>>> plt.set_cmap(plt.cm.Paired)
>>> fig2 = plt.figure(2)
>>> title = plt.title("SVM (linear kernel) on principal components")
>>> plot1 = plt.pcolormesh(xx, yy, yp.reshape(xx.shape))
>>> plot2 = plt.scatter(z[:, 0], z[:, 1], c=y)
```

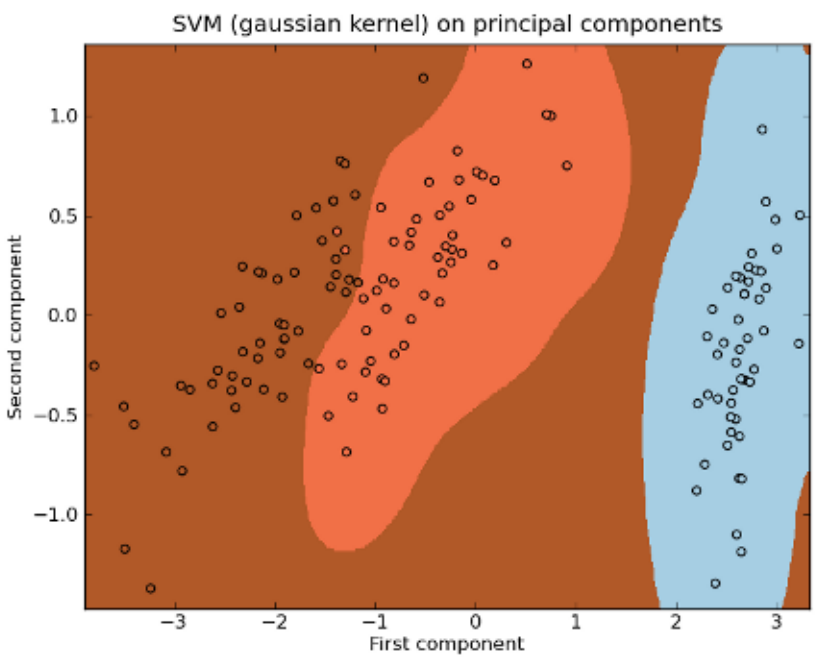
(continues on next page)

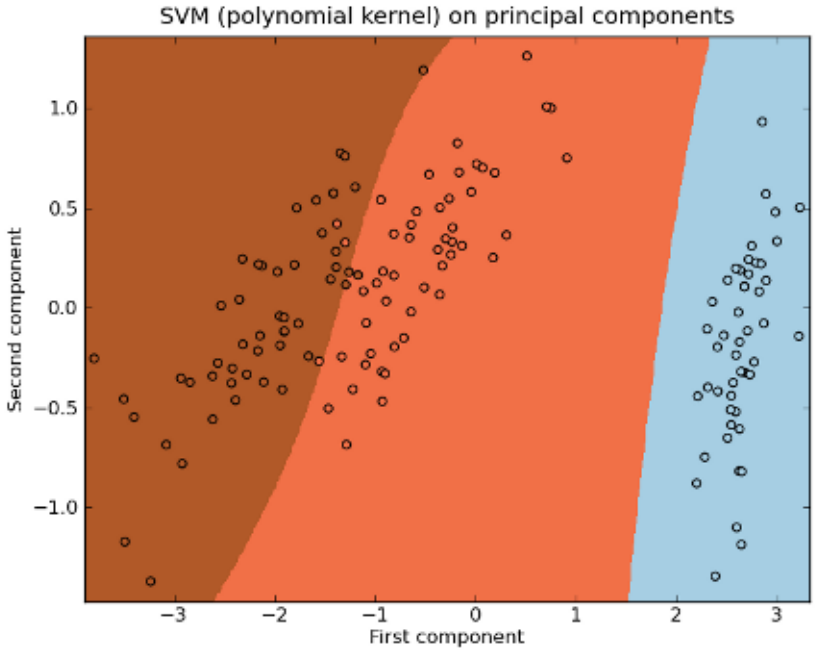
(continued from previous page)

```
>>> labx = plt.xlabel("First component")
>>> laby = plt.ylabel("Second component")
>>> limx = plt.xlim(xmin, xmax)
>>> limy = plt.ylim(ymin, ymax)
>>> plt.show()
```



We can try to use different kernels to obtain:




SVM (polynomial kernel) on principal components
Second component
First component
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3

CHAPTER

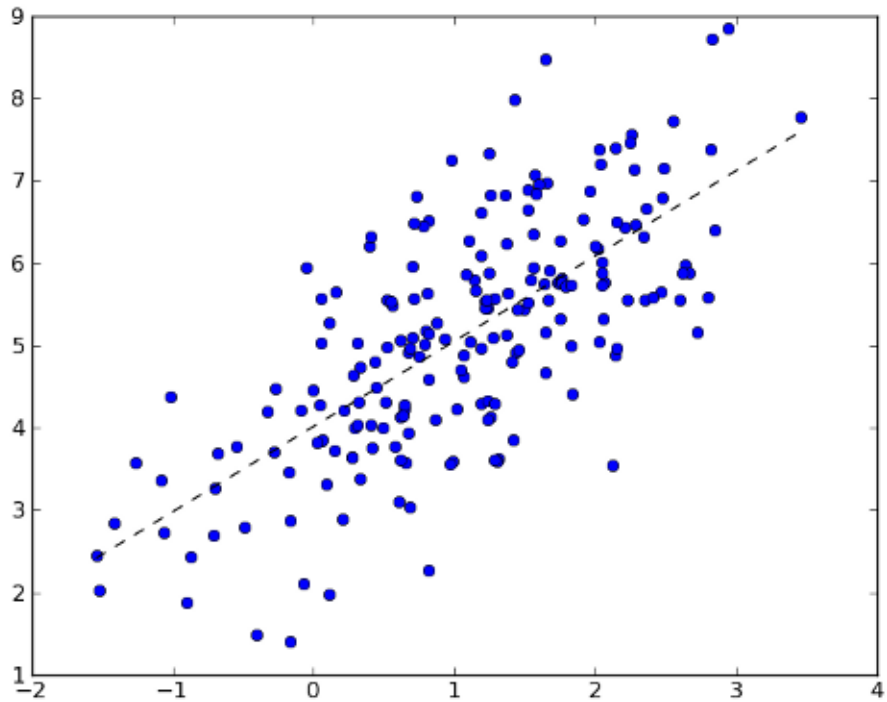
FOUR

LINEAR METHODS FOR REGRESSION

4.1 Ordinary Least Squares

Example:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> mean, cov, n = [1, 5], [[1,1],[1,2]], 200
>>> d = np.random.multivariate_normal(mean, cov, n)
>>> x, y = d[:, 0].reshape(-1, 1), d[:, 1]
>>> x.shape
(200, 1)
>>> ols = mlpy.OLS()
>>> ols.learn(x, y)
>>> xx = np.arange(np.min(x), np.max(x), 0.01).reshape(-1, 1)
>>> yy = ols.pred(xx)
>>> fig = plt.figure(1) # plot
>>> plot = plt.plot(x, y, 'o', xx, yy, '--k')
>>> plt.show()
```

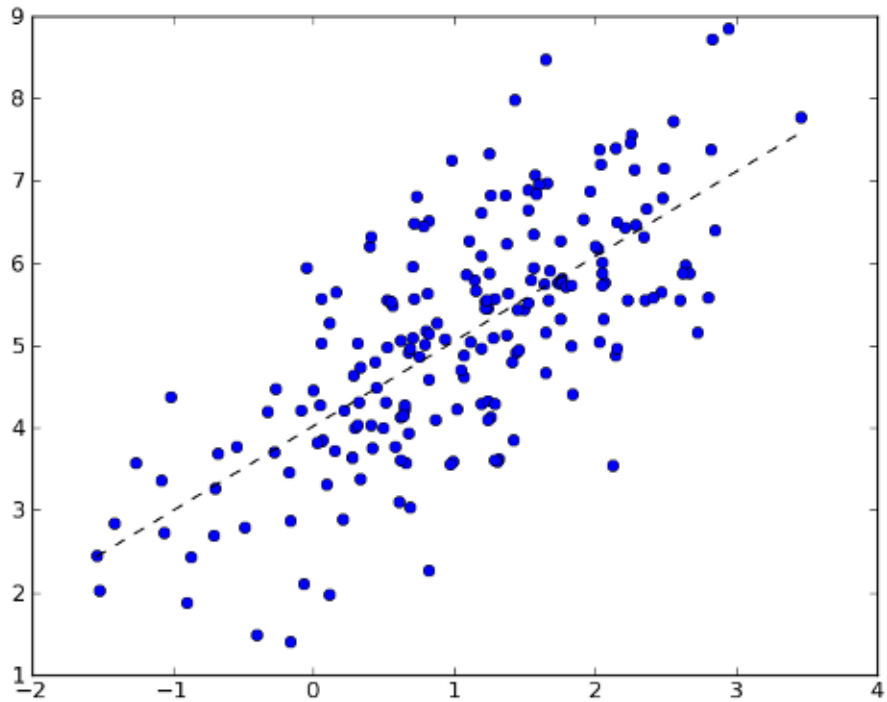


4.2 Ridge Regression

See [Hoerl70]. Ridge regression is also known as regularized least squares. It avoids overfitting by controlling the size of the model vector β, measured by its ℓ^2-norm.

Example:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> mean, cov, n = [1, 5], [[1,1],[1,2]], 200
>>> d = np.random.multivariate_normal(mean, cov, n)
>>> x, y = d[:, 0].reshape(-1, 1), d[:, 1]
>>> x.shape
(200, 1)
>>> ridge = mlpy.Ridge()
>>> ridge.learn(x, y)
>>> xx = np.arange(np.min(x), np.max(x), 0.01).reshape(-1, 1)
>>> yy = ridge.pred(xx)
>>> fig = plt.figure(1) # plot
>>> plot = plt.plot(x, y, 'o', xx, yy, '--k')
>>> plt.show()
```



4.3 Partial Least Squares

4.4 Last Angle Regression (LARS)

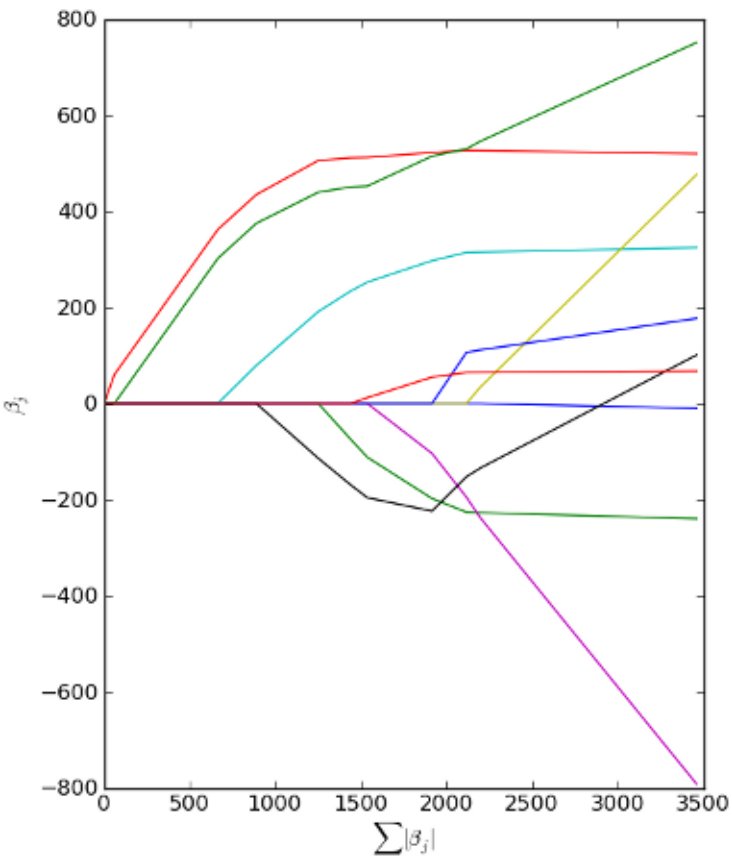
This example replicates the Figure 3 in [Efron04]. The diabetes data can be downloaded from http://www.stanford.edu/~hastie/Papers/LARS/diabetes.data

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> diabetes = np.loadtxt("diabetes.data", skiprows=1)
>>> x = diabetes[:, :-1]
>>> y = diabetes[:, -1]
>>> x -= np.mean(x, axis=0) # center x
>>> x /= np.sqrt(np.sum((x)**2, axis=0)) # normalize x
>>> y -= np.mean(y) # center y
>>> lars = mlpy.LARS()
>>> lars.learn(x, y)
>>> lars.steps() # number of steps performed
10
>>> lars.beta()
array([ -10.0098663 , -239.81564367,  519.84592005,  324.3846455 ,
       -792.17563855,  476.73902101,  101.04326794,  177.06323767,
        751.27369956,   67.62669218])
```

(continues on next page)

(continued from previous page)

```
>>> lars.beta0()
4.7406304540474682e-14
>>> est = lars.est() # returns all LARS estimates
>>> beta_sum = np.sum(np.abs(est), axis=1)
>>> fig = plt.figure(1)
>>> plot1 = plt.plot(beta_sum, est)
>>> xl = plt.xlabel(r'$\sum{|\beta_j|}$')
>>> yl = plt.ylabel(r'$\beta_j$')
>>> plt.show()
```



4.5 Elastic Net

Documentation and implementation is taken from http://web.mit.edu/lrosasco/www/contents/code/ENcode.html

Computes the coefficient vector which solves the elastic-net regularization problem

$$\min\{\|X\beta - Y\|^2 + \lambda(\|\beta\|_2^2 + \epsilon\|\beta\|_1\}$$

Elastic Net Regularization is an algorithm for learning and variable selection. It is based on a regularized least square procedure with a penalty which is the sum of an L1 penalty (like Lasso) and an L2 penalty (like ridge regression). The

first term enforces the sparsity of the solution, whereas the second term ensures democracy among groups of correlated variables. The second term has also a smoothing effect that stabilizes the obtained solution.

CHAPTER

FIVE

LINEAR METHODS FOR CLASSIFICATION

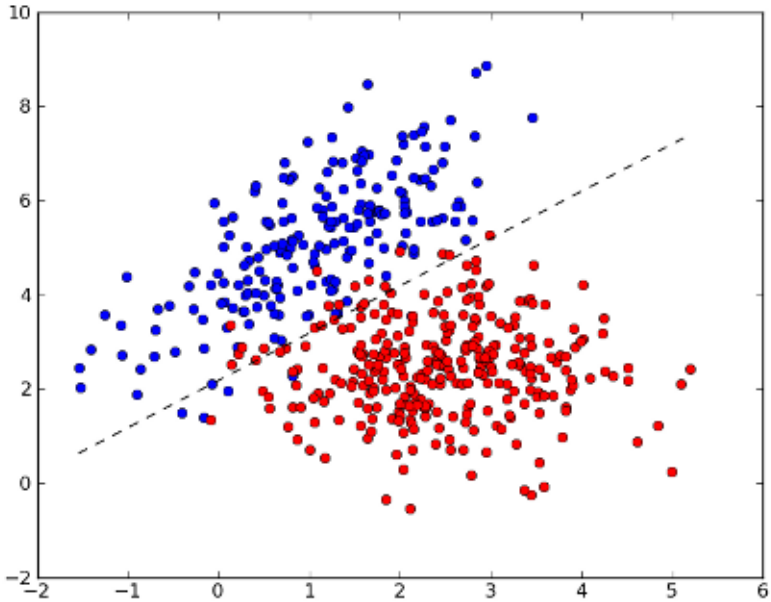
5.1 Linear Discriminant Analysis Classifier (LDAC)

See [Hastie09], page 106.

5.1.1 Examples

Binary classification:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> mean1, cov1, n1 = [1, 5], [[1,1],[1,2]], 200  # 200 samples of class 1
>>> x1 = np.random.multivariate_normal(mean1, cov1, n1)
>>> y1 = np.ones(n1, dtype=int)
>>> mean2, cov2, n2 = [2.5, 2.5], [[1,0],[0,1]], 300 # 300 samples of class -1
>>> x2 = np.random.multivariate_normal(mean2, cov2, n2)
>>> y2 = -np.ones(n2, dtype=int)
>>> x = np.concatenate((x1, x2), axis=0) # concatenate the samples
>>> y = np.concatenate((y1, y2))
>>> ldac = mlpy.LDAC()
>>> ldac.learn(x, y)
>>> w = ldac.w()
>>> w
array([ 2.5948979  -2.58553746])
>>> b = ldac.bias()
>>> b
5.63727441841
>>> xx = np.arange(np.min(x[:,0]), np.max(x[:,0]), 0.01)
>>> yy = - (w[0] * xx + b) / w[1] # separator line
>>> fig = plt.figure(1) # plot
>>> plot1 = plt.plot(x1[:, 0], x1[:, 1], 'ob', x2[:, 0], x2[:, 1], 'or')
>>> plot2 = plt.plot(xx, yy, '--k')
>>> plt.show()
```




```
>>> test = [[0, 2], [4, 2]] # test points
>>> ldac.pred(test)
array([-1, -1])
>>> ldac.labels()
array([-1,  1])
```

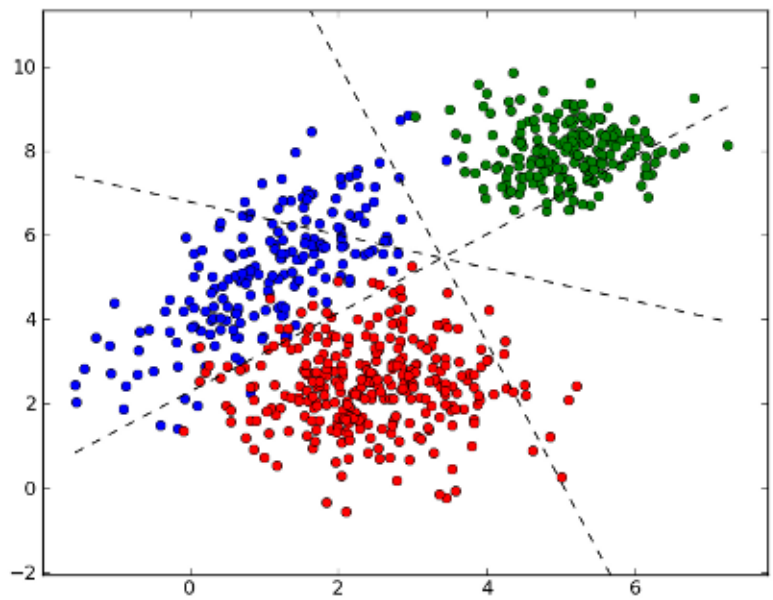
Multiclass classification:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> mean1, cov1, n1 = [1, 25], [[1,1],[1,2]], 200  # 200 samples of class 0
>>> x1 = np.random.multivariate_normal(mean1, cov1, n1)
>>> y1 = np.zeros(n1, dtype=int)
>>> mean2, cov2, n2 = [2.5, 22.5], [[1,0],[0,1]], 300 # 300 samples of class 1
>>> x2 = np.random.multivariate_normal(mean2, cov2, n2)
>>> y2 = np.ones(n2, dtype=int)
>>> mean3, cov3, n3 = [5, 28], [[0.5,0],[0,0.5]], 200 # 200 samples of class 2
>>> x3 = np.random.multivariate_normal(mean3, cov3, n3)
>>> y3 = 2 * np.ones(n3, dtype=int)
>>> x = np.concatenate((x1, x2, x3), axis=0) # concatenate the samples
>>> y = np.concatenate((y1, y2, y3))
>>> ldac = mlpy.LDAC()
>>> ldac.learn(x, y)
>>> w = ldac.w()
>>> w # w[i]: coefficients label ldac.labels()[i]
array([[-0.30949939  4.53041257]
       [ 2.52002288  1.50501818]
       [ 4.2499381   5.90569921]])
```

(continues on next page)

(continued from previous page)

```
>>> b = ldac.bias()
>>> b # b[i]: bias for label ldac.labels()[i]
array([-12.65129158  -5.7628039  -35.63605709])
>>> xx = np.arange(np.min(x[:,0]), np.max(x[:,0]), 0.01)
>>> yy1 = (xx* (w[1][0]-w[0][0]) + b[1] - b[0]) / (w[0][1]-w[1][1])
>>> yy2 = (xx* (w[2][0]-w[0][0]) + b[2] - b[0]) / (w[0][1]-w[2][1])
>>> yy3 = (xx* (w[2][0]-w[1][0]) + b[2] - b[1]) / (w[1][1]-w[2][1])
>>> fig = plt.figure(1) # plot
>>> plot1 = plt.plot(x1[:, 0], x1[:, 1], 'ob', x2[:, 0], x2[:, 1], 'or', x3[:, 0], x3[:,␣
↪1], 'og')
>>> plot2 = plt.plot(xx, yy1, '--k')
>>> plot3 = plt.plot(xx, yy2, '--k')
>>> plot4 = plt.plot(xx, yy3, '--k')
>>> plt.show()
```

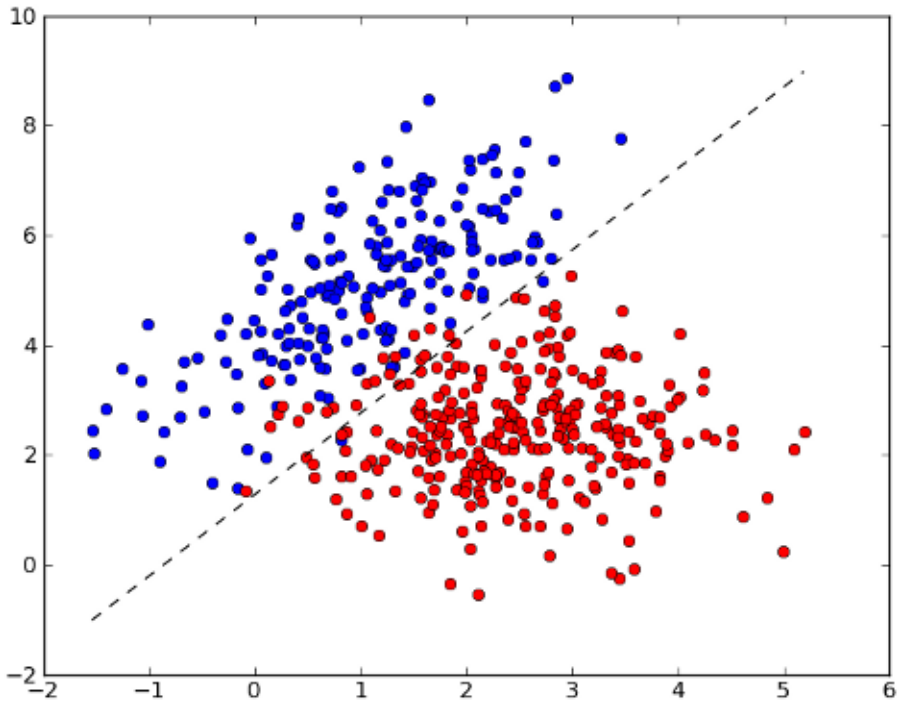



```
>>> test = [[6,7], [4, 2]] # test points
>>> ldac.pred(test)
array([2, 1])
>>> ldac.labels()
array([0, 1, 2])
```

5.2 Basic Perceptron

5.2.1 Examples

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> mean1, cov1, n1 = [1, 5], [[1,1],[1,2]], 200  # 200 samples of class 1
>>> x1 = np.random.multivariate_normal(mean1, cov1, n1)
>>> y1 = np.ones(n1, dtype=int)
>>> mean2, cov2, n2 = [2.5, 2.5], [[1,0],[0,1]], 300 # 300 samples of class -1
>>> x2 = np.random.multivariate_normal(mean2, cov2, n2)
>>> y2 = -np.ones(n2, dtype=int)
>>> x = np.concatenate((x1, x2), axis=0) # concatenate the samples
>>> y = np.concatenate((y1, y2))
>>> p = mlpy.Perceptron(alpha=0.1, thr=0.05, maxiters=100) # basic perceptron
>>> p.learn(x, y)
>>> w = p.w()
>>> w
array([-69.00185254,  46.49202132])
>>> b = p.bias()
>>> b
-59.600000000000001
>>> p.err()
0.050000000000000003
>>> p.iters()
46
>>> xx = np.arange(np.min(x[:,0]), np.max(x[:,0]), 0.01)
>>> yy = - (w[0] * xx + b) / w[1] # separator line
>>> fig = plt.figure(1) # plot
>>> plot1 = plt.plot(x1[:, 0], x1[:, 1], 'ob', x2[:, 0], x2[:, 1], 'or')
>>> plot2 = plt.plot(xx, yy, '--k')
>>> plt.show()
```




```
>>> test = [[0, 2], [4, 2]] # test points
>>> p.pred(test)
array([ 1, -1])
>>> p.labels()
array([-1,  1])
```

5.3 Elastic Net Classifier

See [Hastie09], Chapter 18, page 661.

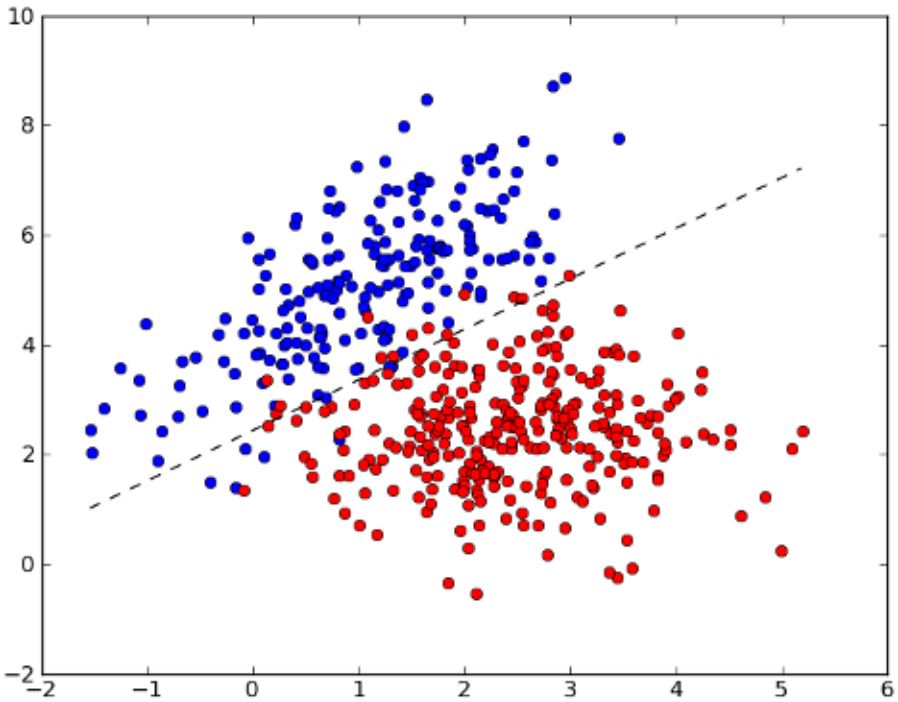
Example:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> mean1, cov1, n1 = [1, 5], [[1,1],[1,2]], 200  # 200 samples of class 1
>>> x1 = np.random.multivariate_normal(mean1, cov1, n1)
>>> y1 = np.ones(n1, dtype=int)
>>> mean2, cov2, n2 = [2.5, 2.5], [[1,0],[0,1]], 300 # 300 samples of class -1
>>> x2 = np.random.multivariate_normal(mean2, cov2, n2)
>>> y2 = -np.ones(n2, dtype=int)
>>> x = np.concatenate((x1, x2), axis=0) # concatenate the samples
>>> y = np.concatenate((y1, y2))
>>> en = mlpy.ElasticNetC(lmb=0.01, eps=0.001)
```

(continues on next page)

(continued from previous page)

```
>>> en.learn(x, y)
>>> w = en.w()
>>> w
array([-0.27733363,  0.30115026])
>>> b = en.bias()
>>> b
-0.73445916200332606
>>> en.iters()
1000
>>> xx = np.arange(np.min(x[:,0]), np.max(x[:,0]), 0.01)
>>> yy = - (w[0] * xx + b) / w[1] # separator line
>>> fig = plt.figure(1) # plot
>>> plot1 = plt.plot(x1[:, 0], x1[:, 1], 'ob', x2[:, 0], x2[:, 1], 'or')
>>> plot2 = plt.plot(xx, yy, '--k')
>>> plt.show()
```




```
>>> test = [[1, 4], [2, 2]] # test points
>>> en.pred(test)
array([ 1., -1.])
```

5.4 Logistic Regression

See *Large Linear Classification from [LIBLINEAR]*

5.5 Support Vector Classification

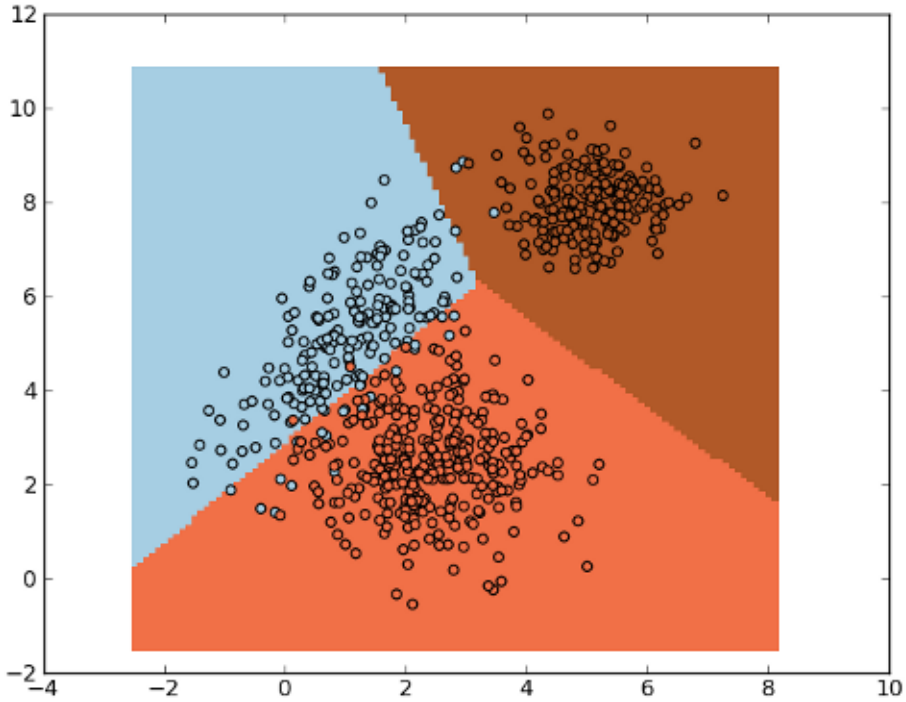
See *Large Linear Classification from [LIBLINEAR]*

5.6 Diagonal Linear Discriminant Analysis (DLDA)

See [Hastie09], page 651.

Example:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> mean1, cov1, n1 = [1, 5], [[1,1],[1,2]], 200  # 200 samples of class 0
>>> x1 = np.random.multivariate_normal(mean1, cov1, n1)
>>> y1 = np.zeros(n1, dtype=int)
>>> mean2, cov2, n2 = [2.5, 2.5], [[1,0],[0,1]], 300 # 300 samples of class 1
>>> x2 = np.random.multivariate_normal(mean2, cov2, n2)
>>> y2 = np.ones(n2, dtype=int)
>>> mean3, cov3, n3 = [5, 8], [[0.5,0],[0,0.5]], 200 # 200 samples of class 2
>>> x3 = np.random.multivariate_normal(mean3, cov3, n3)
>>> y3 = 2 * np.ones(n3, dtype=int)
>>> x = np.concatenate((x1, x2, x3), axis=0) # concatenate the samples
>>> y = np.concatenate((y1, y2, y3))
>>> da = mlpy.DLDA(delta=0.1)
>>> da.learn(x, y)
>>> xmin, xmax = x[:,0].min()-1, x[:,0].max()+1
>>> ymin, ymax = x[:,1].min()-1, x[:,1].max()+1
>>> xx, yy = np.meshgrid(np.arange(xmin, xmax, 0.1), np.arange(ymin, ymax, 0.1))
>>> xnew = np.c_[xx.ravel(), yy.ravel()]
>>> ynew = da.pred(xnew).reshape(xx.shape)
>>> fig = plt.figure(1)
>>> cmap = plt.set_cmap(plt.cm.Paired)
>>> plot1 = plt.pcolormesh(xx, yy, ynew)
>>> plot2 = plt.scatter(x[:,0], x[:,1], c=y)
>>> plt.show()
```



5.7 Golub Classifier

CHAPTER

SIX

KERNELS

6.1 Kernel Functions

A kernel is a function κ that for all $\mathbf{t}, \mathbf{x} \in X$ satisfies $\kappa(\mathbf{t}, \mathbf{x}) = \langle \Phi(\mathbf{t}), \Phi(\mathbf{x}) \rangle$, where Φ is a mapping from X to an (inner product) feature space F, $\Phi : \mathbf{t} \longmapsto \Phi(\mathbf{t}) \in F$.

The following functions take two array-like objects `t` (M, P) and `x` (N, P) and compute the (M, N) matrix $\mathbf{K^t}$ with entries

$$\mathbf{K^t}_{ij} = \kappa(\mathbf{t}_i, \mathbf{x}_j).$$

6.2 Kernel Classes

6.3 Functions

`mlpy.`**`kernel_linear`**(*t*, *x*)

Linear kernel, t_i' x_j.

`mlpy.`**`kernel_polynomial`**(*t*, *x*, *gamma=1.0*, *b=1.0*, *d=2.0*)

Polynomial kernel, (gamma t_i' x_j + b)^d.

`mlpy.`**`kernel_gaussian`**(*t*, *x*, *sigma=1.0*)

Gaussian kernel, exp(-||t_i - x_j||^2 / 2 * sigma^2).

`mlpy.`**`kernel_exponential`**(*t*, *x*, *sigma=1.0*)

Exponential kernel, exp(-||t_i - x_j|| / 2 * sigma^2).

`mlpy.`**`kernel_sigmoid`**(*t*, *x*, *gamma=1.0*, *b=1.0*)

Sigmoid kernel, tanh(gamma t_i' x_j + b).

Example:

```
>>> import mlpy
>>> x = [[5, 1, 3, 1], [7, 1, 11, 4], [0, 4, 2, 9]] # three training points
>>> K = mlpy.kernel_gaussian(x, x, sigma=10) # compute the kernel matrix K_ij = k(x_i, x_
↪j)
>>> K
array([[ 1.        ,  0.68045064,  0.60957091],
```

(continues on next page)

(continued from previous page)

```
       [ 0.68045064,  1.        ,  0.44043165],
       [ 0.60957091,  0.44043165,  1.        ]])
>>> t = [[8, 1, 5, 1], [7, 1, 11, 4]] # two test points
>>> Kt = mlpy.kernel_gaussian(t, x, sigma=10) # compute the test kernel matrix Kt_ij =
↪<Phi(t_i), Phi(x_j)> = k(t_i, x_j)
>>> Kt
array([[ 0.93706746,  0.7945336 ,  0.48190899],
       [ 0.68045064,  1.        ,  0.44043165]])
```

6.4 Centering in Feature Space

The centered kernel matrix $\tilde{\mathbf{K}}^{\mathbf{t}}$ is computed by:

$$\tilde{\mathbf{K}}^{\mathbf{t}}_{ij} = \left\langle \Phi(\mathbf{t}_i) - \frac{1}{N}\sum_{m=1}^{N}\Phi(\mathbf{x}_m), \Phi(\mathbf{x}_j) - \frac{1}{N}\sum_{n=1}^{N}\Phi(\mathbf{x}_n) \right\rangle.$$

We can express $\tilde{\mathbf{K}}^{\mathbf{t}}$ in terms of $\mathbf{K}^{\mathbf{t}}$ and $\mathbf{K}$:

$$\tilde{\mathbf{K}}^{\mathbf{t}}_{ij} = \mathbf{K}^{\mathbf{t}} - \mathbf{1}_N^T\mathbf{K} - \mathbf{K}^{\mathbf{t}}\mathbf{1}_N + \mathbf{1}_N^T\mathbf{K}\mathbf{1}_N$$

where $\mathbf{1}_N$ is the $N \times M$ matrix with all entries equal to $1/N$ and $\mathbf{K}$ is $\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$.

`mlpy.kernel_center(Kt, K)`

Centers the testing kernel matrix Kt respect the training kernel matrix K. If Kt = K (kernel_center(K, K), where K = k(x_i, x_j)), the function centers the kernel matrix K.

Parameters

Kt
[2d array_like object (M, N)] test kernel matrix Kt_ij = k(t_i, x_j). If Kt = K the function centers the kernel matrix K

K
[2d array_like object (N, N)] training kernel matrix K_ij = k(x_i, x_j)

Returns

Ktcentered
[2d numpy array (M, N)] centered version of Kt

Example:

```
>>> Kcentered = mlpy.kernel_center(K, K) # center K
>>> Kcentered
array([[ 0.19119746, -0.07197215, -0.11922531],
       [-0.07197215,  0.30395696, -0.23198481],
       [-0.11922531, -0.23198481,  0.35121011]])
>>> Ktcentered = mlpy.kernel_center(Kt, K) # center the test kernel matrix Kt respect to␣
↪K
```

(continues on next page)

(continued from previous page)

```
>>> Ktcentered
array([[ 0.15376875,  0.06761464, -0.22138339],
       [-0.07197215,  0.30395696, -0.23198481]])
```

6.5 Make a Custom Kernel

TODO

CHAPTER

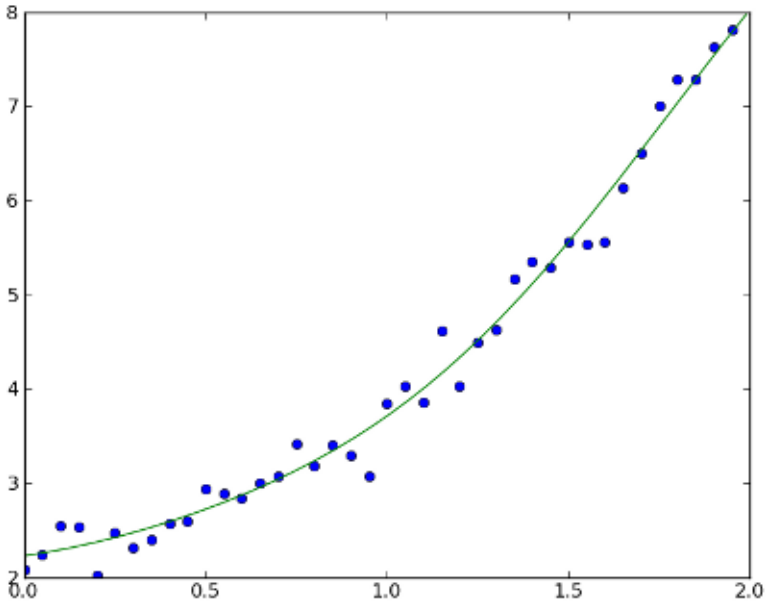
SEVEN

NON LINEAR METHODS FOR REGRESSION

7.1 Kernel Ridge Regression

Example:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> x = np.arange(0, 2, 0.05).reshape(-1, 1) # training points
>>> y = np.ravel(np.exp(x)) + np.random.normal(1, 0.2, x.shape[0]) # target values
>>> xt = np.arange(0, 2, 0.01).reshape(-1, 1) # testing points
>>> K = mlpy.kernel_gaussian(x, x, sigma=1) # training kernel matrix
>>> Kt = mlpy.kernel_gaussian(xt, x, sigma=1) # testing kernel matrix
>>> krr = KernelRidge(lmb=0.01)
>>> krr.learn(K, y)
>>> yt = krr.pred(Kt)
>>> fig = plt.figure(1)
>>> plot1 = plt.plot(x[:, 0], y, 'o')
>>> plot2 = plt.plot(xt[:, 0], yt)
>>> plt.show()
```



7.2 Support Vector Regression

See *Support Vector Machines (SVMs)*

CHAPTER

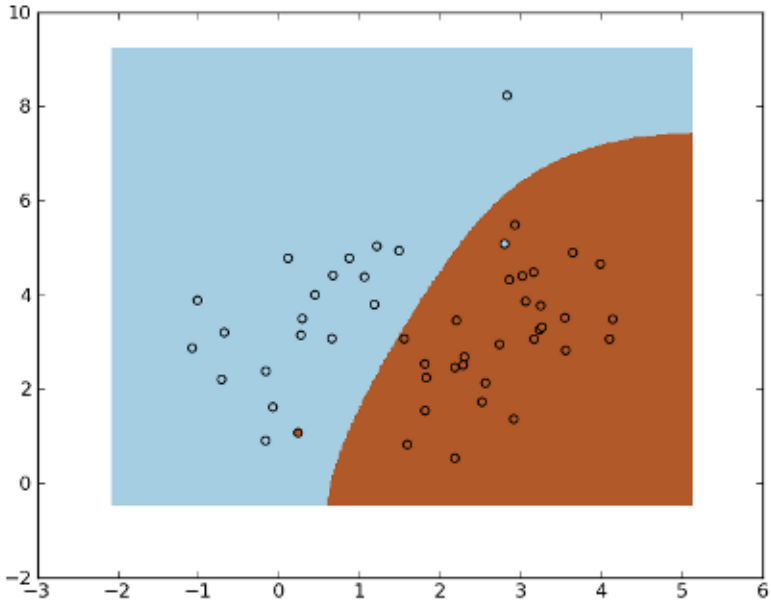
EIGHT

NON LINEAR METHODS FOR CLASSIFICATION

8.1 Parzen-based classifier

Example:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> mean1, cov1, n1 = [1, 4.5], [[1,1],[1,2]], 20  # 20 samples of class 1
>>> x1 = np.random.multivariate_normal(mean1, cov1, n1)
>>> y1 = np.ones(n1, dtype=int)
>>> mean2, cov2, n2 = [2.5, 2.5], [[1,1],[1,2]], 30 # 30 samples of class 2
>>> x2 = np.random.multivariate_normal(mean2, cov2, n2)
>>> y2 = 2 * np.ones(n2, dtype=int)
>>> x = np.concatenate((x1, x2), axis=0) # concatenate the samples
>>> y = np.concatenate((y1, y2))
>>> K = mlpy.kernel_gaussian(x, x, sigma=2) # kernel matrix
>>> parzen = mlpy.Parzen()
>>> parzen.learn(K, y)
>>> xmin, xmax = x[:,0].min()-1, x[:,0].max()+1
>>> ymin, ymax = x[:,1].min()-1, x[:,1].max()+1
>>> xx, yy = np.meshgrid(np.arange(xmin, xmax, 0.02), np.arange(ymin, ymax, 0.02))
>>> xt = np.c_[xx.ravel(), yy.ravel()] # test points
>>> Kt = mlpy.kernel_gaussian(xt, x, sigma=2) # test kernel matrix
>>> yt = parzen.pred(Kt).reshape(xx.shape)
>>> fig = plt.figure(1)
>>> cmap = plt.set_cmap(plt.cm.Paired)
>>> plot1 = plt.pcolormesh(xx, yy, yt)
>>> plot2 = plt.scatter(x[:,0], x[:,1], c=y)
>>> plt.show()
```



8.2 Support Vector Classification

See *Support Vector Machines (SVMs)*

8.3 Kernel Fisher Discriminant Classifier

8.4 k-Nearest-Neighbor

class mlpy.**KNN**(*k*)

k-Nearest Neighbor (euclidean distance)

Parameters

k

[int] number of nearest neighbors

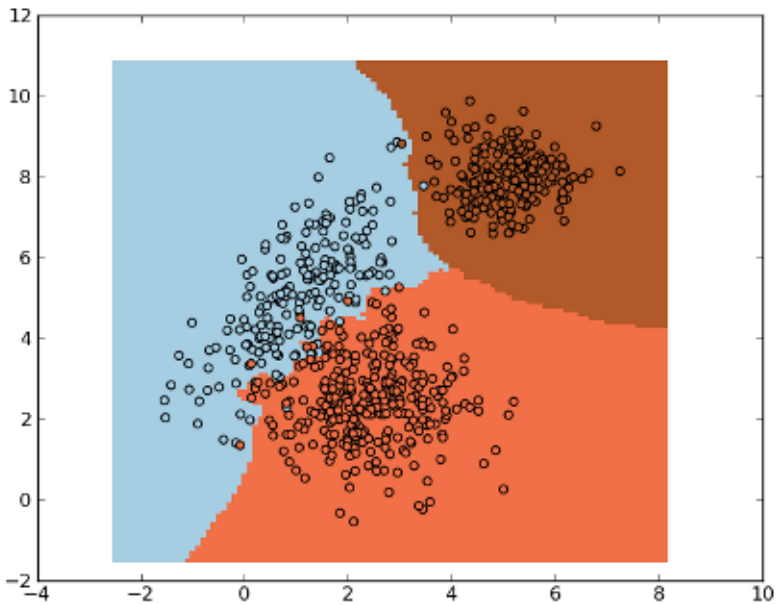
Example:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> mean1, cov1, n1 = [1, 5], [[1,1],[1,2]], 200  # 200 samples of class 1
>>> x1 = np.random.multivariate_normal(mean1, cov1, n1)
>>> y1 = np.ones(n1, dtype=int)
>>> mean2, cov2, n2 = [2.5, 2.5], [[1,0],[0,1]], 300 # 300 samples of class 2
>>> x2 = np.random.multivariate_normal(mean2, cov2, n2)
```

(continues on next page)

(continued from previous page)

```
>>> y2 = 2 * np.ones(n2, dtype=int)
>>> mean3, cov3, n3 = [5, 8], [[0.5,0],[0,0.5]], 200 # 200 samples of class 3
>>> x3 = np.random.multivariate_normal(mean3, cov3, n3)
>>> y3 = 3 * np.ones(n3, dtype=int)
>>> x = np.concatenate((x1, x2, x3), axis=0) # concatenate the samples
>>> y = np.concatenate((y1, y2, y3))
>>> knn = mlpy.KNN(k=3)
>>> knn.learn(x, y)
>>> xmin, xmax = x[:,0].min()-1, x[:,0].max()+1
>>> ymin, ymax = x[:,1].min()-1, x[:,1].max()+1
>>> xx, yy = np.meshgrid(np.arange(xmin, xmax, 0.1), np.arange(ymin, ymax, 0.1))
>>> xnew = np.c_[xx.ravel(), yy.ravel()]
>>> ynew = knn.pred(xnew).reshape(xx.shape)
>>> ynew[ynew == 0] = 1 # set the samples with no unique classification to 1
>>> fig = plt.figure(1)
>>> cmap = plt.set_cmap(plt.cm.Paired)
>>> plot1 = plt.pcolormesh(xx, yy, ynew)
>>> plot2 = plt.scatter(x[:,0], x[:,1], c=y)
>>> plt.show()
```



8.5 Classification Tree

class mlpy.**ClassTree**(*stumps=0*, *minsize=1*)

Classification Tree (gini index)

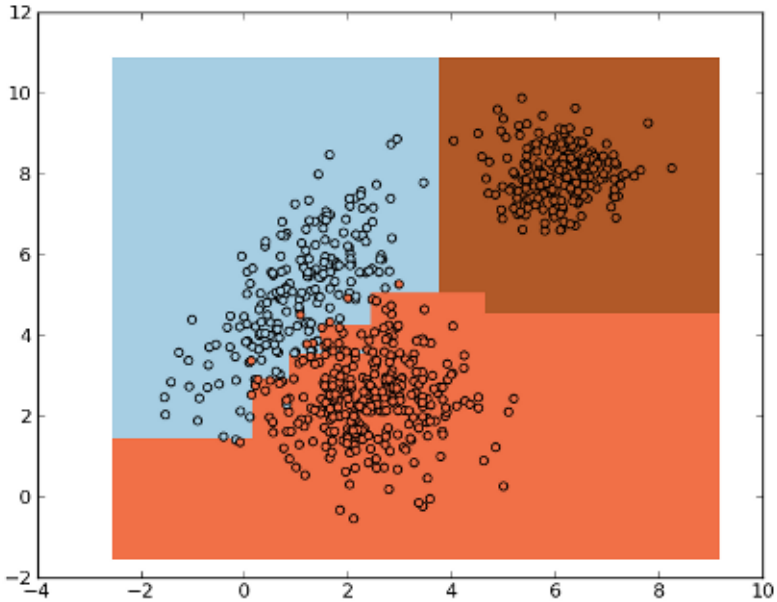
Parameters

stumps
[bool] True: compute single split or False: standard tree

minsize
[int (>=0)] minimum number of cases required to split a leaf

Example:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> mean1, cov1, n1 = [1, 5], [[1,1],[1,2]], 200  # 200 samples of class 1
>>> x1 = np.random.multivariate_normal(mean1, cov1, n1)
>>> y1 = np.ones(n1, dtype=int)
>>> mean2, cov2, n2 = [2.5, 2.5], [[1,0],[0,1]], 300 # 300 samples of class 2
>>> x2 = np.random.multivariate_normal(mean2, cov2, n2)
>>> y2 = 2 * np.ones(n2, dtype=int)
>>> mean3, cov3, n3 = [6, 8], [[0.5,0],[0,0.5]], 200 # 200 samples of class 3
>>> x3 = np.random.multivariate_normal(mean3, cov3, n3)
>>> y3 = 3 * np.ones(n3, dtype=int)
>>> x = np.concatenate((x1, x2, x3), axis=0) # concatenate the samples
>>> y = np.concatenate((y1, y2, y3))
>>> tree = mlpy.ClassTree(minsize=10)
>>> tree.learn(x, y)
>>> xmin, xmax = x[:,0].min()-1, x[:,0].max()+1
>>> ymin, ymax = x[:,1].min()-1, x[:,1].max()+1
>>> xx, yy = np.meshgrid(np.arange(xmin, xmax, 0.1), np.arange(ymin, ymax, 0.1))
>>> xnew = np.c_[xx.ravel(), yy.ravel()]
>>> ynew = tree.pred(xnew).reshape(xx.shape)
>>> ynew[ynew == 0] = 1 # set the samples with no unique classification to 1
>>> fig = plt.figure(1)
>>> cmap = plt.set_cmap(plt.cm.Paired)
>>> plot1 = plt.pcolormesh(xx, yy, ynew)
>>> plot2 = plt.scatter(x[:,0], x[:,1], c=y)
>>> plt.show()
```



8.6 Maximum Likelihood Classifier

class mlpy.**MaximumLikelihoodC**

Maximum Likelihood Classifier

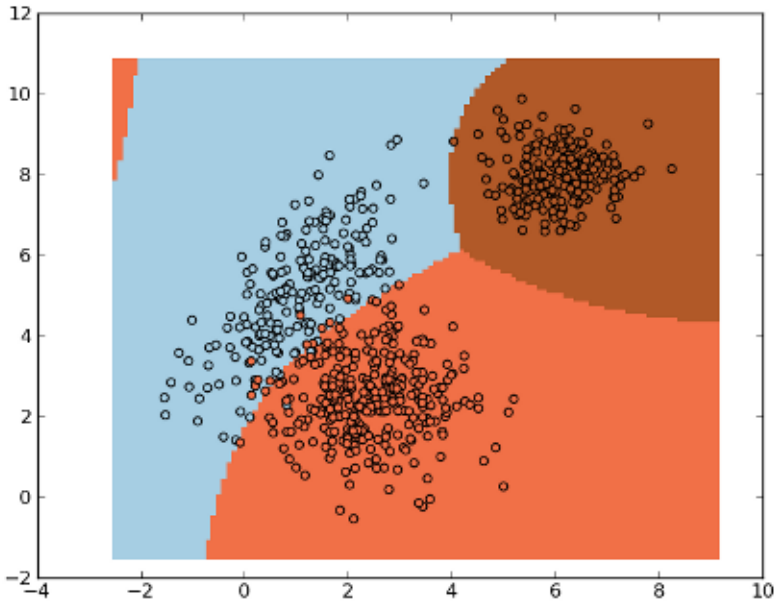
Example:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> mean1, cov1, n1 = [1, 5], [[1,1],[1,2]], 200  # 200 samples of class 1
>>> x1 = np.random.multivariate_normal(mean1, cov1, n1)
>>> y1 = np.ones(n1, dtype=int)
>>> mean2, cov2, n2 = [2.5, 2.5], [[1,0],[0,1]], 300 # 300 samples of class 2
>>> x2 = np.random.multivariate_normal(mean2, cov2, n2)
>>> y2 = 2 * np.ones(n2, dtype=int)
>>> mean3, cov3, n3 = [6, 8], [[0.5,0],[0,0.5]], 200 # 200 samples of class 3
>>> x3 = np.random.multivariate_normal(mean3, cov3, n3)
>>> y3 = 3 * np.ones(n3, dtype=int)
>>> x = np.concatenate((x1, x2, x3), axis=0) # concatenate the samples
>>> y = np.concatenate((y1, y2, y3))
>>> ml = mlpy.MaximumLikelihoodC()
>>> ml.learn(x, y)
>>> xmin, xmax = x[:,0].min()-1, x[:,0].max()+1
>>> ymin, ymax = x[:,1].min()-1, x[:,1].max()+1
>>> xx, yy = np.meshgrid(np.arange(xmin, xmax, 0.1), np.arange(ymin, ymax, 0.1))
>>> xnew = np.c_[xx.ravel(), yy.ravel()]
>>> ynew = ml.pred(xnew).reshape(xx.shape)
```

(continues on next page)

(continued from previous page)

```
>>> ynew[ynew == 0] = 1 # set the samples with no unique classification to 1
>>> fig = plt.figure(1)
>>> cmap = plt.set_cmap(plt.cm.Paired)
>>> plot1 = plt.pcolormesh(xx, yy, ynew)
>>> plot2 = plt.scatter(x[:,0], x[:,1], c=y)
>>> plt.show()
```



CHAPTER

NINE

SUPPORT VECTOR MACHINES (SVMS)

9.1 Support Vector Machines from [LIBSVM]

class mlpy.**LibSvm**(*svm_type='c_svc'*, *kernel_type='linear'*, *degree=3*, *gamma=0.001*, *coef0=0*, *C=1*, *nu=0.5*, *eps=0.001*, *p=0.1*, *cache_size=100*, *shrinking=True*, *probability=False*, *weight={}*)

LibSvm.

Parameters

svm_type
[string] SVM type, can be one of: 'c_svc', 'nu_svc', 'one_class', 'epsilon_svr', 'nu_svr'

kernel_type
[string] kernel type, can be one of: 'linear' (uT*v), 'poly' ((gamma*uT*v + coef0)^degree), 'rbf' (exp(-gamma*|u-v|^2)), 'sigmoid' (tanh(gamma*uT*v + coef0))

degree
[int (for 'poly' kernel_type)] degree in kernel

gamma
[float (for 'poly', 'rbf', 'sigmoid' kernel_type)] gamma in kernel (e.g. 1 / number of features)

coef0
[float (for 'poly', 'sigmoid' kernel_type)] coef0 in kernel

C
[float (for 'c_svc', 'epsilon_svr', 'nu_svr')] cost of constraints violation

nu
[float (for 'nu_svc', 'one_class', 'nu_svr')] nu parameter

eps
[float] stopping criterion, usually 0.00001 in nu-SVC, 0.001 in others

p
[float (for 'epsilon_svr')] p is the epsilon in epsilon-insensitive loss function of epsilon-SVM regression

cache_size
[float [MB]] size of the kernel cache, specified in megabytes

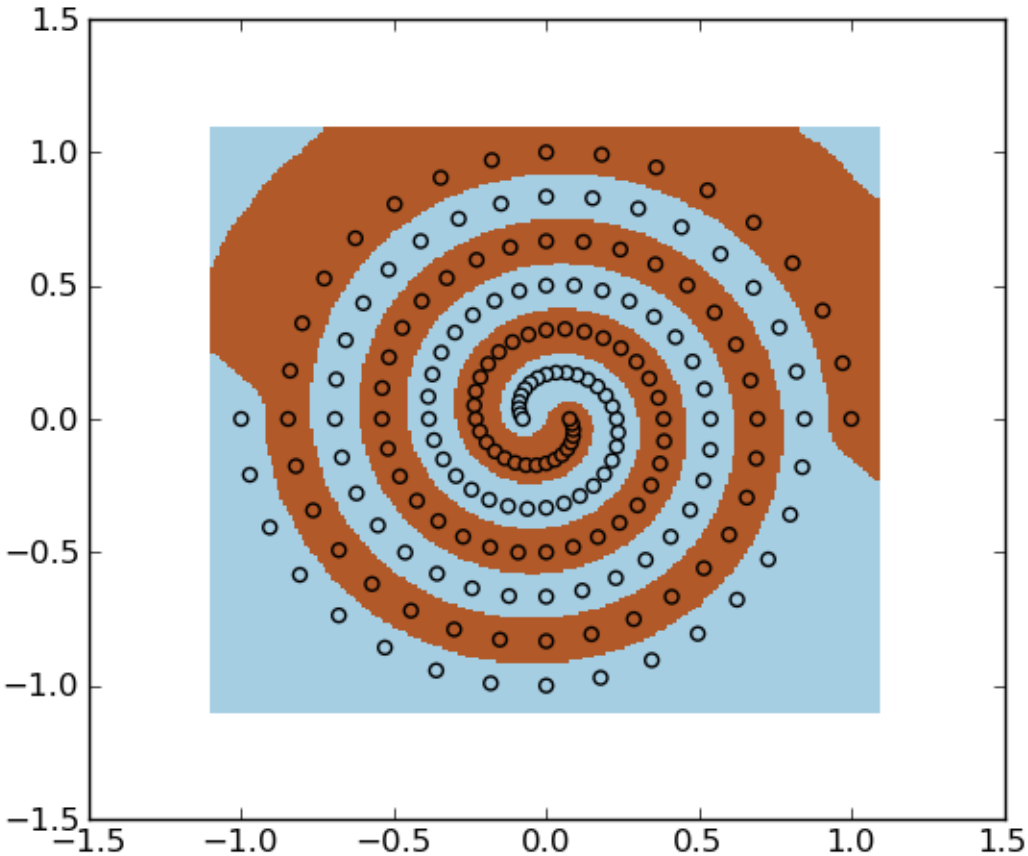
shrinking
[bool] use the shrinking heuristics

probability
[bool] predict probability estimates

weight
[dict] changes the penalty for some classes (if the weight for a class is not changed, it is set to 1). For example, to change penalty for classes 1 and 2 to 0.5 and 0.8 respectively set weight={1:0.5, 2:0.8}

Example on `spiral dataset`:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> f = np.loadtxt("spiral.data")
>>> x, y = f[:, :2], f[:, 2]
>>> svm = mlpy.LibSvm(svm_type='c_svc', kernel_type='rbf', gamma=100)
>>> svm.learn(x, y)
>>> xmin, xmax = x[:,0].min()-0.1, x[:,0].max()+0.1
>>> ymin, ymax = x[:,1].min()-0.1, x[:,1].max()+0.1
>>> xx, yy = np.meshgrid(np.arange(xmin, xmax, 0.01), np.arange(ymin, ymax, 0.01))
>>> xnew = np.c_[xx.ravel(), yy.ravel()]
>>> ynew = svm.pred(xnew).reshape(xx.shape)
>>> fig = plt.figure(1)
>>> plt.set_cmap(plt.cm.Paired)
>>> plt.pcolormesh(xx, yy, ynew)
>>> plt.scatter(x[:,0], x[:,1], c=y)
>>> plt.show()
```



9.2 Kernel Adatron

class mlpy.**KernelAdatron**(*C=1000*, *maxsteps=1000*, *eps=0.01*)

Kernel Adatron algorithm without-bias-term (binary classifier).

The algoritm handles a version of the 1-norm soft margin support vector machine. If C is very high the algoritm handles a version of the hard margin SVM.

Use positive definite kernels (such as Gaussian and Polynomial kernels)

Parameters

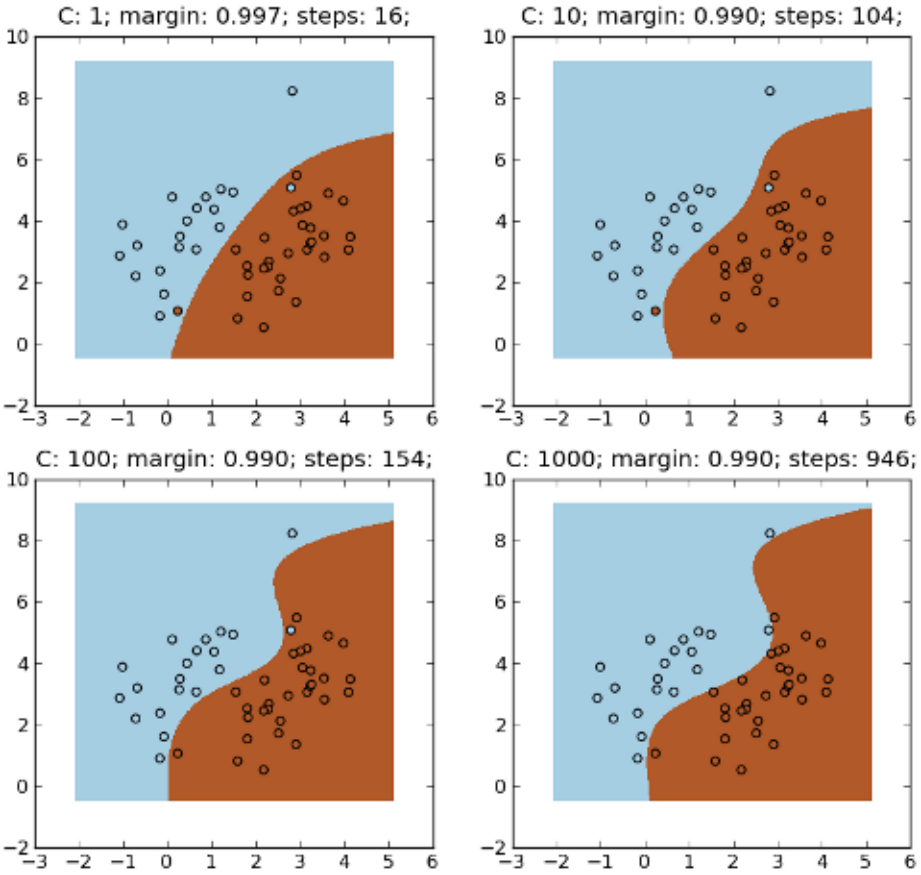
C
[float] upper bound on the value of alpha

maxsteps
[integer (> 0)] maximum number of steps

eps
[float (>=0)] the algoritm stops when abs(1 - margin) < eps

Example:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> mean1, cov1, n1 = [1, 4.5], [[1,1],[1,2]], 20  # 20 samples of class 1
>>> x1 = np.random.multivariate_normal(mean1, cov1, n1)
>>> y1 = np.ones(n1, dtype=int)
>>> mean2, cov2, n2 = [2.5, 2.5], [[1,1],[1,2]], 30 # 30 samples of class 2
>>> x2 = np.random.multivariate_normal(mean2, cov2, n2)
>>> y2 = 2 * np.ones(n2, dtype=int)
>>> x = np.concatenate((x1, x2), axis=0) # concatenate the samples
>>> y = np.concatenate((y1, y2))
>>> K = mlpy.kernel_gaussian(x, x, sigma=2) # kernel matrix
>>> xmin, xmax = x[:,0].min()-1, x[:,0].max()+1
>>> ymin, ymax = x[:,1].min()-1, x[:,1].max()+1
>>> xx, yy = np.meshgrid(np.arange(xmin, xmax, 0.02), np.arange(ymin, ymax, 0.02))
>>> xt = np.c_[xx.ravel(), yy.ravel()] # test points
>>> Kt = mlpy.kernel_gaussian(xt, x, sigma=2) # test kernel matrix
>>> fig = plt.figure(1)
>>> cmap = plt.set_cmap(plt.cm.Paired)
>>> for i, c in enumerate([1, 10, 100, 1000]):
...     ka = mlpy.KernelAdatron(C=c)
...     ax = plt.subplot(2, 2, i+1)
...     ka.learn(K, y)
...     ytest = ka.pred(Kt).reshape(xx.shape)
...     title = ax.set_title('C: %s; margin: %.3f; steps: %s;' % (c, ka.margin(), ka.
↪steps()))
...     plot1 = plt.pcolormesh(xx, yy, ytest)
...     plot2 = plt.scatter(x[:,0], x[:,1], c=y)
>>> plt.show()
```


C: 1; margin: 0.997; steps: 16;
C: 10; margin: 0.990; steps: 104;
C: 100; margin: 0.990; steps: 154;
C: 1000; margin: 0.990; steps: 946;

CHAPTER

TEN

LARGE LINEAR CLASSIFICATION FROM [LIBLINEAR]

Solvers:

- **l2r_lr**: L2-regularized logistic regression (primal)
- **l2r_l2loss_svc_dual**: L2-regularized L2-loss support vector classification (dual)
- **l2r_l2loss_svc**: L2-regularized L2-loss support vector classification (primal)
- **l2r_l1loss_svc_dual**: L2-regularized L1-loss support vector classification (dual)
- **mcsvm_cs**: multi-class support vector classification by Crammer and Singer
- **l1r_l2loss_svc**: L1-regularized L2-loss support vector classification
- **l1r_lr**: L1-regularized logistic regression
- **l2r_lr_dual**: L2-regularized logistic regression (dual)

class mlpy.**LibLinear**(*solver_type='l2r_lr'*, *C=1*, *eps=0.01*, *weight={}*)

LibLinear is a simple class for solving large-scale regularized linear classification. It currently supports L2-regularized logistic regression/L2-loss support vector classification/L1-loss support vector classification, and L1-regularized L2-loss support vector classification/ logistic regression.

Parameters

solver_type
[string] solver, can be one of 'l2r_lr', 'l2r_l2loss_svc_dual', 'l2r_l2loss_svc', 'l2r_l1loss_svc_dual', 'mcsvm_cs', 'l1r_l2loss_svc', 'l1r_lr', 'l2r_lr_dual'

C
[float] cost of constraints violation

eps
[float] stopping criterion

weight
[dict] changes the penalty for some classes (if the weight for a class is not changed, it is set to 1). For example, to change penalty for classes 1 and 2 to 0.5 and 0.8 respectively set weight={1:0.5, 2:0.8}

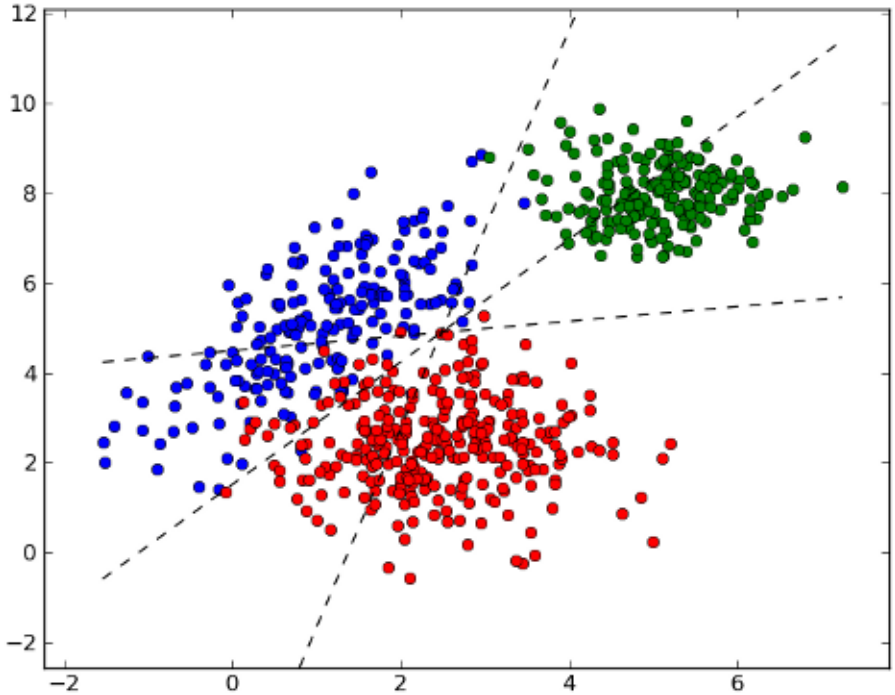
Example:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> mean1, cov1, n1 = [1, 5], [[1,1],[1,2]], 200  # 200 samples of class 0
>>> x1 = np.random.multivariate_normal(mean1, cov1, n1)
```

(continues on next page)

(continued from previous page)

```
>>> y1 = np.zeros(n1, dtype=int)
>>> mean2, cov2, n2 = [2.5, 2.5], [[1,0],[0,1]], 300 # 300 samples of class 1
>>> x2 = np.random.multivariate_normal(mean2, cov2, n2)
>>> y2 = np.ones(n2, dtype=int)
>>> mean3, cov3, n3 = [5, 8], [[0.5,0],[0,0.5]], 200 # 200 samples of class 2
>>> x3 = np.random.multivariate_normal(mean3, cov3, n3)
>>> y3 = 2 * np.ones(n3, dtype=int)
>>> x = np.concatenate((x1, x2, x3), axis=0) # concatenate the samples
>>> y = np.concatenate((y1, y2, y3))
>>> svm = mlpy.LibLinear(solver_type='l2r_l2loss_svc_dual', C=0.01)
>>> svm.learn(x, y)
>>> w = svm.w() # w[i]: coefficients for label svm.labels()[i]
>>> w
array([[-0.73225278,  0.33309388],
       [ 0.32295557, -0.44097029],
       [ 0.23192595,  0.11536679]])
>>> b = svm.bias() # b[i]: bias for label svm.labels()[i]
>>> b
array([-0.21631629,  0.96014472, -1.53933202])
>>> xx = np.arange(np.min(x[:,0]), np.max(x[:,0]), 0.01)
>>> yy1 = (xx* (w[1][0]-w[0][0]) + b[1] - b[0]) / (w[0][1]-w[1][1])
>>> yy2 = (xx* (w[2][0]-w[0][0]) + b[2] - b[0]) / (w[0][1]-w[2][1])
>>> yy3 = (xx* (w[2][0]-w[1][0]) + b[2] - b[1]) / (w[1][1]-w[2][1])
>>> fig = plt.figure(1) # plot
>>> plot1 = plt.plot(x1[:, 0], x1[:, 1], 'ob', x2[:, 0], x2[:, 1], 'or', x3[:, 0], x3[:,␣
↪1], 'og')
>>> plot2 = plt.plot(xx, yy1, '--k')
>>> plot3 = plt.plot(xx, yy2, '--k')
>>> plot4 = plt.plot(xx, yy3, '--k')
>>> plt.show()
```




```
>>> test = [[6,7], [4, 2]] # test points
>>> print svm.pred(test)
array([2, 1])
```

CHAPTER

ELEVEN

CLUSTER ANALYSIS

11.1 Hierarchical Clustering

Hierarchical Clustering algorithm derived from the R package 'amap' [Amap].

The condensed distance matrix *y* can be computed by `pdist()` function in **scipy** (<http://docs.scipy.org/doc/scipy/reference/spatial.distance.html>)

11.2 Memory-saving Hierarchical Clustering

Memory-saving Hierarchical Clustering derived from the R and Python package 'fastcluster' [fastcluster].

11.3 k-means

`mlpy.kmeans`(*x*, *k*, *plus=False*, *seed=0*)

k-means clustering.

Parameters

x
[2d array_like object (N, P)] data

k
[int (1<k<N)] number of clusters

plus
[bool] k-means++ algorithm for initialization

seed
[int] random seed for initialization

Returns

clusters, means, steps: 1d array, 2d array, int
cluster membership in 0,...,K-1, means (K,P), number of steps

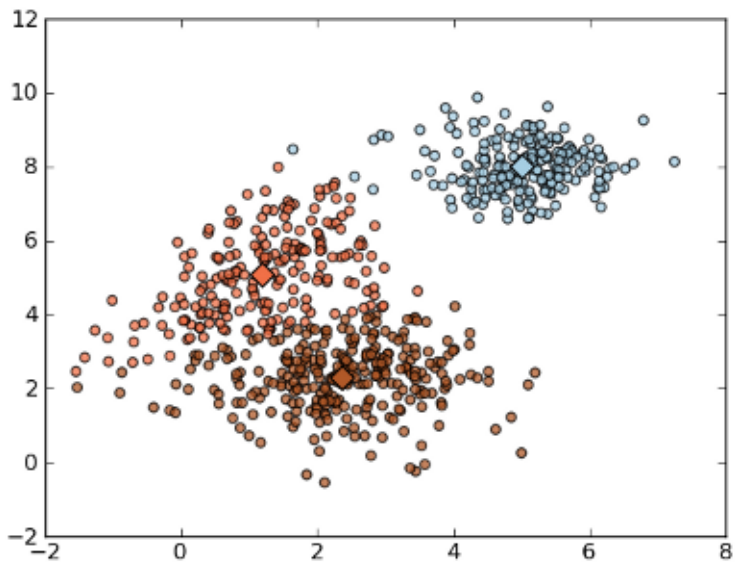
Example:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
```

(continues on next page)

(continued from previous page)

```
>>> np.random.seed(0)
>>> mean1, cov1, n1 = [1, 5], [[1,1],[1,2]], 200 # 200 points, mean=(1,5)
>>> x1 = np.random.multivariate_normal(mean1, cov1, n1)
>>> mean2, cov2, n2 = [2.5, 2.5], [[1,0],[0,1]], 300 # 300 points, mean=(2.5,2.5)
>>> x2 = np.random.multivariate_normal(mean2, cov2, n2)
>>> mean3, cov3, n3 = [5, 8], [[0.5,0],[0,0.5]], 200 # 200 points, mean=(5,8)
>>> x3 = np.random.multivariate_normal(mean3, cov3, n3)
>>> x = np.concatenate((x1, x2, x3), axis=0) # concatenate the samples
>>> cls, means, steps = mlpy.kmeans(x, k=3, plus=True)
>>> steps
13
>>> fig = plt.figure(1)
>>> plot1 = plt.scatter(x[:,0], x[:,1], c=cls, alpha=0.75)
>>> plot2 = plt.scatter(means[:,0], means[:,1], c=np.unique(cls), s=128, marker='d') #␣
↪plot the means
>>> plt.show()
```



CHAPTER

TWELVE

ALGORITHMS FOR FEATURE WEIGHTING

12.1 Iterative RELIEF

CHAPTER

THIRTEEN

FEATURE SELECTION

13.1 Recursive Feature Elimination

CHAPTER

FOURTEEN

DIMENSIONALITY REDUCTION

14.1 Linear Discriminant Analysis (LDA)

Example:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> mean1, cov1, n1 = [1, 4.5], [[1,1],[1,2]], 20  # 20 samples of class 1
>>> x1 = np.random.multivariate_normal(mean1, cov1, n1)
>>> y1 = np.ones(n1, dtype=int)
>>> mean2, cov2, n2 = [2.5, 2.5], [[1,1],[1,2]], 30 # 30 samples of class 2
>>> x2 = np.random.multivariate_normal(mean2, cov2, n2)
>>> y2 = 2 * np.ones(n2, dtype=int)
>>> x = np.concatenate((x1, x2), axis=0) # concatenate the samples
>>> y = np.concatenate((y1, y2))
>>> lda = mlpy.LDA()
>>> lda.learn(x, y) # compute the tranformation matrix
>>> z = lda.transform(x) # embedded x into the C-1 = 1 dimensional space
```

14.2 Spectral Regression Discriminant Analysis (SRDA)

14.3 Kernel Fisher Discriminant Analysis (KFDA)

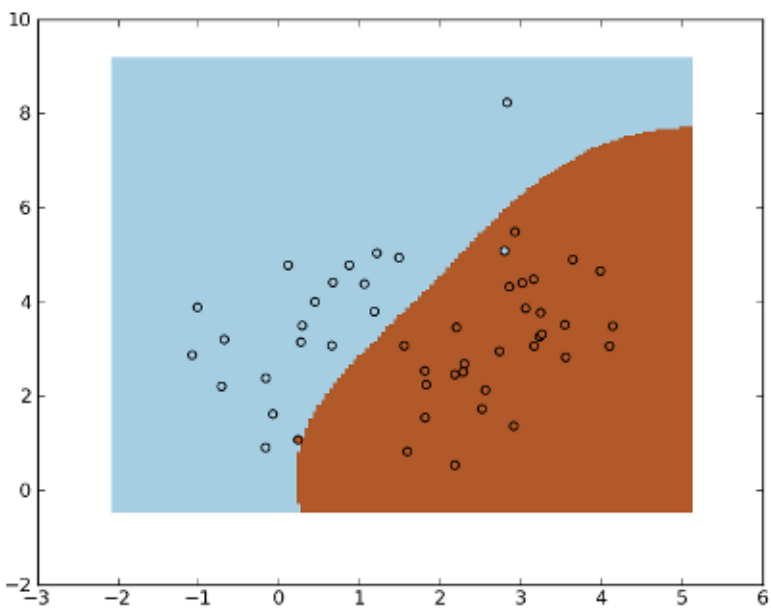
Example - KNN in kernel fisher space:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> mean1, cov1, n1 = [1, 4.5], [[1,1],[1,2]], 20  # 20 samples of class 1
>>> x1 = np.random.multivariate_normal(mean1, cov1, n1)
>>> y1 = np.ones(n1, dtype=int)
>>> mean2, cov2, n2 = [2.5, 2.5], [[1,1],[1,2]], 30 # 30 samples of class 2
>>> x2 = np.random.multivariate_normal(mean2, cov2, n2)
>>> y2 = 2 * np.ones(n2, dtype=int)
>>> x = np.concatenate((x1, x2), axis=0) # concatenate the samples
```

(continues on next page)

(continued from previous page)

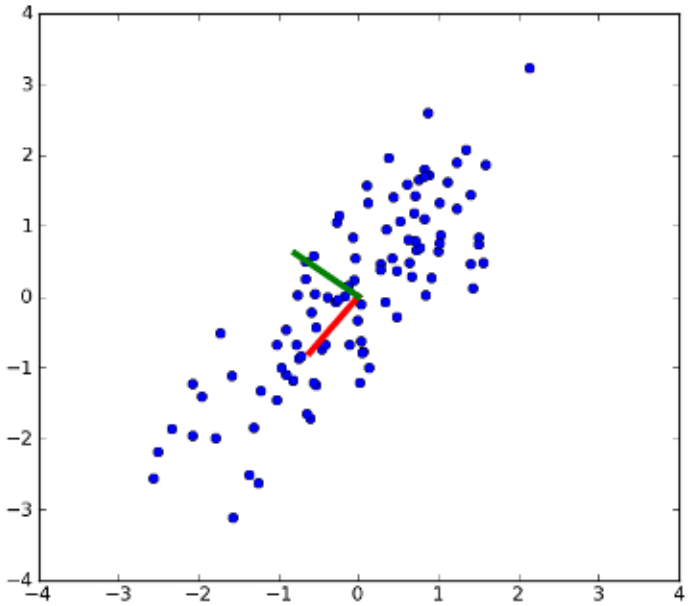
```
>>> y = np.concatenate((y1, y2))
>>> K = mlpy.kernel_gaussian(x, x, sigma=3) # compute the kernel matrix
>>> kfda = mlpy.KFDA(lmb=0.01)
>>> kfda.learn(K, y) # compute the tranformation vector
>>> z = kfda.transform(K) # embedded x into the kernel fisher space
>>> knn = mlpy.KNN(k=5)
>>> knn.learn(z, y) # learn KNN in the kernel fisher space
>>> xmin, xmax = x[:,0].min()-1, x[:,0].max()+1
>>> ymin, ymax = x[:,1].min()-1, x[:,1].max()+1
>>> xx, yy = np.meshgrid(np.arange(xmin, xmax, 0.05), np.arange(ymin, ymax, 0.05))
>>> xt = np.c_[xx.ravel(), yy.ravel()]
>>> Kt = mlpy.kernel_gaussian(xt, x, sigma=3) # compute the kernel matrix Kt
>>> zt = kfda.transform(Kt) # embedded xt into the kernel fisher space
>>> yt = KNN.pred(zt).reshape(xx.shape) # perform the KNN prediction in the kernel␣
↪fisher space
>>> fig = plt.figure(1)
>>> cmap = plt.set_cmap(plt.cm.Paired)
>>> plot1 = plt.pcolormesh(xx, yy, yt)
>>> plot2 = plt.scatter(x[:,0], x[:,1], c=y)
>>> plt.show()
```



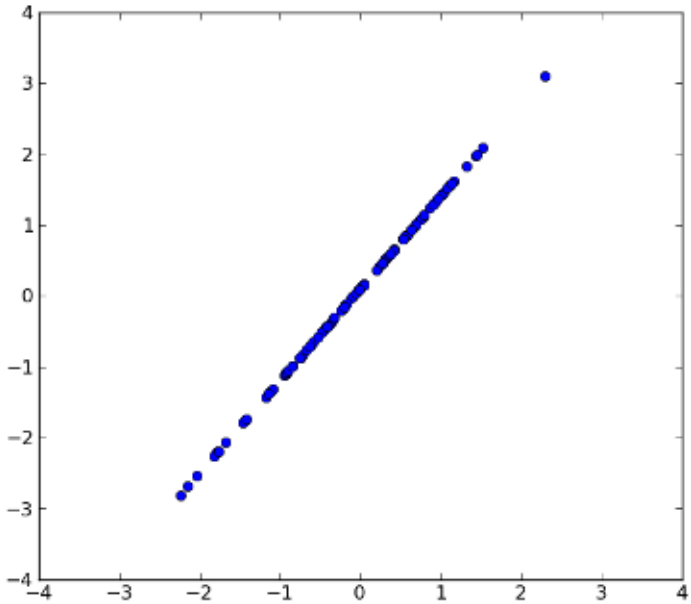
14.4 Principal Component Analysis (PCA)

Example:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> mean, cov, n = [0, 0], [[1,1],[1,1.5]], 100
>>> x = np.random.multivariate_normal(mean, cov, n)
>>> pca.learn(x)
>>> coeff = pca.coeff()
>>> fig = plt.figure(1) # plot
>>> plot1 = plt.plot(x[:, 0], x[:, 1], 'o')
>>> plot2 = plt.plot([0,coeff[0, 0]], [0, coeff[1, 0]], linewidth=4, color='r') # first␣
↪PC
>>> plot3 = plt.plot([0,coeff[0, 1]], [0, coeff[1, 1]], linewidth=4, color='g') # second␣
↪PC
>>> xx = plt.xlim(-4, 4)
>>> yy = plt.ylim(-4, 4)
>>> plt.show()
```




```
>>> z = pca.transform(x, k=1) # transform x using the first PC
>>> xnew = pca.transform_inv(z) # transform data back to its original space
>>> fig2 = plt.figure(2) # plot
>>> plot1 = plt.plot(xnew[:, 0], xnew[:, 1], 'o')
>>> xx = plt.xlim(-4, 4)
>>> yy = plt.ylim(-4, 4)
>>> plt.show()
```



14.5 Fast Principal Component Analysis (PCAFast)

Fast PCA implementation described in [Sharma07].

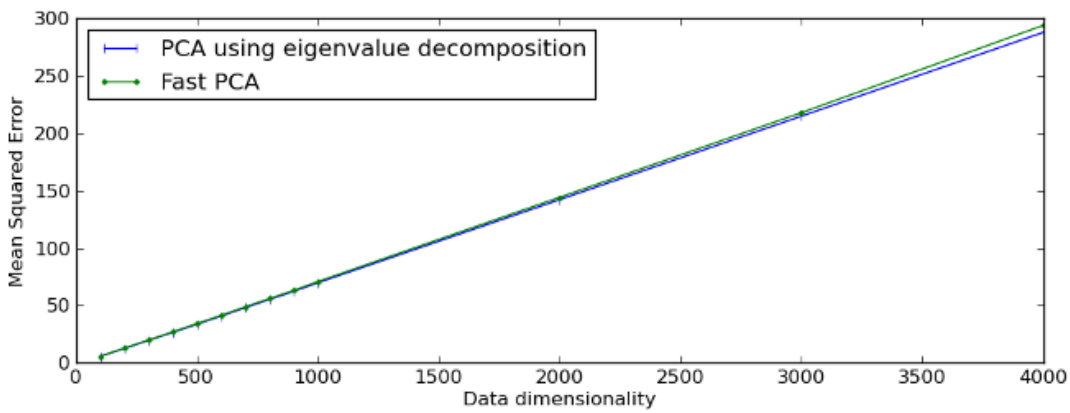
Example reproducing Figure 1 of [Sharma07]:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> h = 10 # dimension reduced to h=10
>>> n = 100 # number of samples
>>> d = np.array([100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 3000, 4000])
↪# number of dimensions
>>> mse_eig, mse_fast = np.zeros(len(d)), np.zeros(len(d))
>>> pca = mlpy.PCA(method='cov') # pca (eigenvalue decomposition)
>>> pca_fast=  mlpy.PCAFast(k=h) # fast pca
>>> for i in range(d.shape[0]):
...     x = np.random.rand(n, d[i])
...     pca.learn(x) # pca (eigenvalue decomposition)
...     y_eig = pca.transform(x, k=h) # reduced dimensional feature vectors
...     xhat_eig = pca.transform_inv(y_eig) # reconstructed vector
...     pca_fast.learn(x) # pca (eigenvalue decomposition)
...     y_fast = pca_fast.transform(x) # reduced dimensional feature vectors
...     xhat_fast = pca_fast.transform_inv(y_fast) # reconstructed vector
...     for j in range(n):
...         mse_eig[i] += np.sum((x[j] - xhat_eig[j])**2)
...         mse_fast[i] += np.sum((x[j] - xhat_fast[j])**2)
...     mse_eig[i] /= n
```

(continues on next page)

(continued from previous page)

```
...     mse_fast[i] /= n
...
>>> fig = plt.figure(1)
>>> plot1 = plt.plot(d, mse_eig, '|-b', label="PCA using eigenvalue decomposition")
>>> plot2 = plt.plot(d, mse_fast, '.-g', label="Fast PCA")
>>> leg = plt.legend(loc = 'best')
>>> xl = plt.xlabel("Data dimensionality")
>>> yl = plt.ylabel("Mean Squared Error")
>>> plt.show()
```



14.6 Kernel Principal Component Analysis (KPCA)

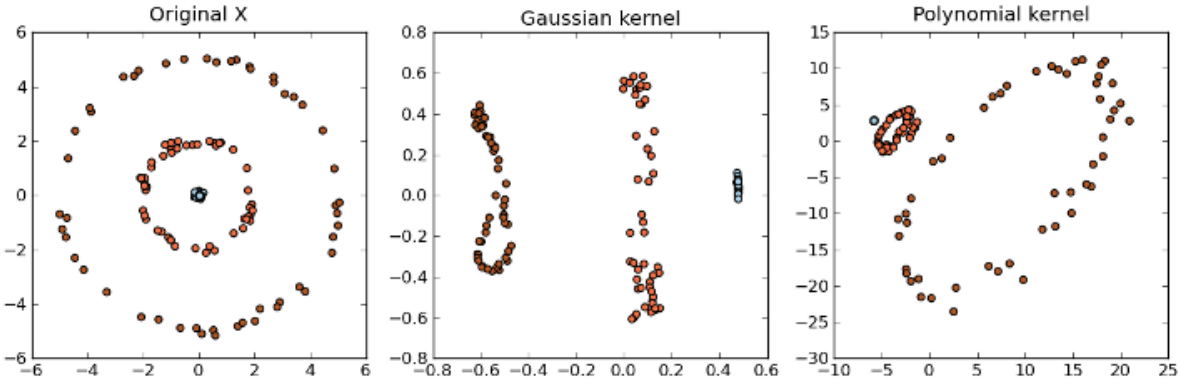
Example:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy
>>> np.random.seed(0)
>>> np.random.seed(0)
>>> x = np.zeros((150, 2))
>>> y = np.empty(150, dtype=int)
>>> theta = np.random.normal(0, np.pi, 50)
>>> r = np.random.normal(0, 0.1, 50)
>>> x[0:50, 0] = r * np.cos(theta)
>>> x[0:50, 1] = r * np.sin(theta)
>>> y[0:50] = 0
>>> theta = np.random.normal(0, np.pi, 50)
>>> r = np.random.normal(2, 0.1, 50)
>>> x[50:100, 0] = r * np.cos(theta)
>>> x[50:100, 1] = r * np.sin(theta)
>>> y[50:100] = 1
>>> theta = np.random.normal(0, np.pi, 50)
>>> r = np.random.normal(5, 0.1, 50)
>>> x[100:150, 0] = r * np.cos(theta)
>>> x[100:150, 1] = r * np.sin(theta)
```

(continues on next page)

(continued from previous page)

```
>>> y[100:150] = 2
>>> cmap = plt.set_cmap(plt.cm.Paired)
>>> gK = mlpy.kernel_gaussian(x, x, sigma=2) # gaussian kernel matrix
>>> pK = mlpy.kernel_polynomial(x, x, gamma=1.0, b=1.0, d=2.0) # polynomial kernel matrix
>>> gaussian_pca = mlpy.KPCA()
>>> polynomial_pca = mlpy.KPCA()
>>> gaussian_pca.learn(gK)
>>> polynomial_pca.learn(pK)
>>> gz = gaussian_pca.transform(gK, k=2)
>>> pz = polynomial_pca.transform(pK, k=2)
>>> fig = plt.figure(1)
>>> ax1 = plt.subplot(131)
>>> plot1 = plt.scatter(x[:, 0], x[:, 1], c=y)
>>> title1 = ax1.set_title('Original X')
>>> ax2 = plt.subplot(132)
>>> plot2 = plt.scatter(gz[:, 0], gz[:, 1], c=y)
>>> title2 = ax2.set_title('Gaussian kernel')
>>> ax3 = plt.subplot(133)
>>> plot3 = plt.scatter(pz[:, 0], pz[:, 1], c=y)
>>> title3 = ax3.set_title('Polynomial kernel')
>>> plt.show()
```



CHAPTER

FIFTEEN

CROSS VALIDATION

15.1 Leave-one-out and k-fold

15.2 Random Subsampling (*aka MonteCarlo*)

15.3 All Combinations

CHAPTER

SIXTEEN

METRICS

Compute metrics for assessing the performance of classification/regression models.

16.1 Classification

Examples:

```
>>> import mlpy
>>> t = [3,2,3,3,3,1,1,1]
>>> p = [3,2,1,3,3,2,1,1]
>>> mlpy.error(t, p)
0.25
>>> mlpy.accuracy(t, p)
0.75
```

16.1.1 Binary Classification Only

The Confusion Matrix:

Total Samples (ts)	Actual Positives (ap)	Actual Negatives (an)
Predicted Positives (pp)	True Positives (tp)	False Positives (fp)
Predicted Negatives (pn)	False Negatives (fn)	True Negatives (tn)

Examples:

```
>>> import mlpy
>>> t = [1, 1, 1,-1, 1,-1,-1,-1]
>>> p = [1,-1, 1, 1, 1,-1, 1,-1]
>>> mlpy.error_p(t, p)
0.25
>>> mlpy.error_n(t, p)
0.5
>>> mlpy.sensitivity(t, p)
0.75
>>> mlpy.specificity(t, p)
0.5
>>> mlpy.ppv(t, p)
0.59999999999999998
```

(continues on next page)

(continued from previous page)

```
>>> mlpy.npv(t, p)
0.66666666666666663
>>> mlpy.mcc(t, p)
0.2581988897471611
>>> p = [2.3,-0.4, 1.6, 0.6, 3.2,-4.9, 1.3,-0.3]
>>> mlpy.auc_wmw(t, p)
0.8125
>>> p = [2.3,0.4, 1.6, -0.6, 3.2,-4.9, -1.3,-0.3]
>>> mlpy.auc_wmw(t, p)
1.0
```

16.2 Regression

Example:

```
>>> import mlpy
>>> t = [2.4,0.4,1.2,-0.2,3.3,-4.9,-1.1,-0.1]
>>> p = [2.3,0.4,1.6,-0.6,3.2,-4.9,-1.3,-0.3]
>>> mlpy.mse(t, p)
0.052499999999999998
```

CHAPTER

SEVENTEEN

A SET OF STATISTICAL FUNCTIONS

CHAPTER

EIGHTEEN

CANBERRA DISTANCES AND STABILITY INDICATOR OF RANKED LISTS

18.1 Canberra distance

18.2 Canberra Distance with Location Parameter

See [Jurman08].

The function computes:

$$\sum_i \frac{|\min\{x_i+1, k+1\} - \min\{y_i+1, k+1\}|}{\min\{x_i+1, k+1\} + \min\{y_i+1, k+1\}}$$

18.3 Canberra Stability Indicator

See [Jurman08].

CHAPTER

NINETEEN

BORDA COUNT

CHAPTER

TWENTY

FIND PEAKS

CHAPTER

TWENTYONE

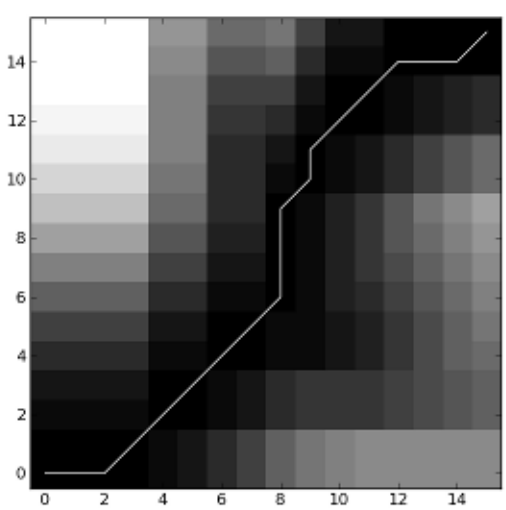
DYNAMIC TIME WARPING (DTW)

21.1 Standard DTW

Example

Reproducing the Fig. 2 example in [Salvador04].

```
>>> import mlpy
>>> import matplotlib.pyplot as plt
>>> import matplotlib.cm as cm
>>> x = [0,0,0,0,1,1,2,2,3,2,1,1,0,0,0,0]
>>> y = [0,0,1,1,2,2,3,3,3,3,2,2,1,1,0,0]
>>> dist, cost, path = mlpy.dtw_std(x, y, dist_only=False)
>>> dist
0.0
>>> fig = plt.figure(1)
>>> ax = fig.add_subplot(111)
>>> plot1 = plt.imshow(cost.T, origin='lower', cmap=cm.gray, interpolation='nearest')
>>> plot2 = plt.plot(path[0], path[1], 'w')
>>> xlim = ax.set_xlim((-0.5, cost.shape[0]-0.5))
>>> ylim = ax.set_ylim((-0.5, cost.shape[1]-0.5))
>>> plt.show()
```



21.2 Subsequence DTW

CHAPTER

TWENTYTWO

LONGEST COMMON SUBSEQUENCE (LCS)

22.1 Standard LCS

22.2 LCS for real series

CHAPTER

TWENTYTHREE

MLPY.WAVELET - WAVELET TRANSFORM

23.1 Padding

23.2 Discrete Wavelet Transform

Discrete Wavelet Transform based on the GSL DWT [Gsldwt].

For the forward transform, the output is the discrete wavelet transform $f_i \rightarrow w_{j,k}$ in a packed triangular storage layout, where j is the index of the level $j = 0 \ldots J-1$ and k is the index of the coefficient within each level, $k = 0 \ldots (2^j)-1$. The total number of levels is $J = \log_2(n)$. The output data has the following form,

$$(s_{-1,0}, d_{0,0}, d_{1,0}, d_{1,1}, d_{2,0}, \ldots, d_{j,k}, ..., d_{J-1,2^{J-1}-1})$$

where the first element is the smoothing coefficient $s_{-1,0}$, followed by the detail coefficients $d_{j,k}$ for each level j. The backward transform inverts these coefficients to obtain the original data.

Note: from GSL online manual (http://www.gnu.org/software/gsl/manual/)

23.3 Undecimated Wavelet Transform

Undecimated Wavelet Transform (also known as stationary wavelet transform, redundant wavelet transform, translation invariant wavelet transform, shift invariant wavelet transform or Maximal overlap wavelet transform) based on the "wavelets" R package.

23.4 Continuous Wavelet Transform

Continuous Wavelet Transform based on [Torrence98].

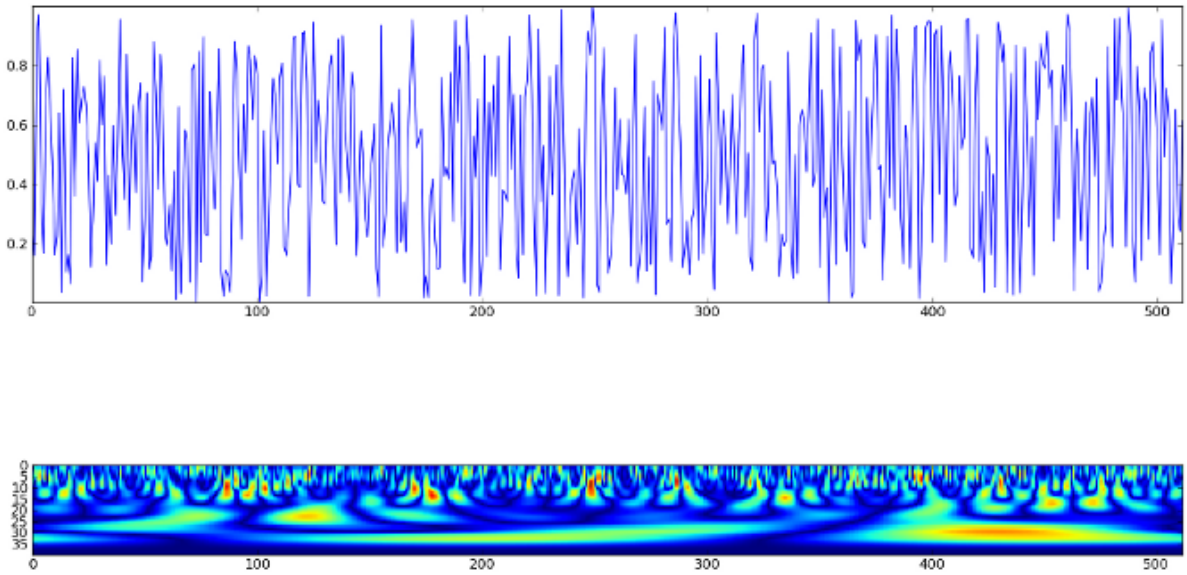
Example (requires matplotlib)

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import mlpy.wavelet as wave
>>> x = np.random.sample(512)
>>> scales = wave.autoscales(N=x.shape[0], dt=1, dj=0.25, wf='dog', p=2)
>>> X = wave.cwt(x=x, dt=1, scales=scales, wf='dog', p=2)
```

(continues on next page)

(continued from previous page)

```
>>> fig = plt.figure(1)
>>> ax1 = plt.subplot(2,1,1)
>>> p1 = ax1.plot(x)
>>> ax1.autoscale_view(tight=True)
>>> ax2 = plt.subplot(2,1,2)
>>> p2 = ax2.imshow(np.abs(X), interpolation='nearest')
>>> plt.show()
```



CHAPTER

TWENTYFOUR

SHORT GUIDE TO CENTERING AND SCALING

Centering:

1d array	`>>> x - np.mean(x)`
2d array along rows	`>>> x - np.mean(x, axis=1).reshape(-1, 1)`
2d array along cols	`>>> x - np.mean(x, axis=0)`

Unit length scaling (normalization). Elements are scaled to have and unit length ($\sum_{i=1}^{n} x_i^2 = 1$):

1d array	`>>> x / np.sqrt(np.sum((x)**2))`
2d array along rows	`>>> x / np.sqrt(np.sum((x)**2, axis=1)).` `↪reshape(-1, 1)`
2d array along cols	`>>> x / np.sqrt(np.sum((x)**2, axis=0))`

Standardization. Elements are scaled to have unit standard deviation. The standard deviation is computed using $n - 1$ instead of n (Bessel's correction).

1d array	`>>> x / np.std(x, ddof=1) # ddof=1: Bessel's correction`
2d array along rows	`>>> x / np.std(x, axis=1, ddof=1).reshape(-1, 1)`
2d array along cols	`>>> x / np.std(x, axis=0, ddof=1)`

CHAPTER

TWENTYFIVE

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[DeMol08] C De Mol, E De Vito and L Rosasco. Elastic Net Regularization in Learning Theory,CBCL paper #273/ CSAILTechnical Report #TR-2008-046, Massachusetts Institute of Technology, Cambridge, MA, July 24, 2008. arXiv:0807.3423 (to appear in the Journal of Complexity).

[Efron04] Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani. Least Angle Regression. Annals of Statistics, 2004, volume 32, pages 407-499.

[Hoerl70] A E Hoerl and R W Kennard. Ridge Regression: Biased Estimation for Nonorthogonal Problems. Technometrics. Vol. 12, No. 1, 1970, pp. 55–67.

[Hastie09] T Hastie, R Tibshirani, J Friedman. The Elements of Statistical Learning. Second Edition.

[Scolkopf98] Bernhard Scholkopf, Alexander Smola, and Klaus-Robert Muller. Nonlinear component analysis as a kernel eigenvalue problem. Neural Computation, 10(5):1299–1319, July 1998.

[LIBSVM] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

[Cristianini] N Cristianini and J Shawe-Taylor. An introduction to support vector machines. Cambridge University Press.

[Vapnik95] V Vapnik. The Nature of Statistical Learning Theory. Springer-Verlag, 1995.

[Friess] Friess, Cristianini, Campbell. The Kernel-Adatron Algorithm: a Fast and Simple Learning Procedure for Support Vector Machines.

[Kecman03] Kecman, Vogt, Huang. On the Equality of Kernel AdaTron and Sequential Minimal Optimization in Classification and Regression Tasks and Alike Algorithms for Kernel Machines. ESANN'2003 proceedings - European Symposium on Artificial Neural Networks, ISBN 2-930307-03-X, pp. 215-222.

[LIBLINEAR] Machine Learning Group at National Taiwan University. http://www.csie.ntu.edu.tw/~cjlin/liblinear/

[Amap] amap: Another Multidimensional Analysis Package, http://cran.r-project.org/web/packages/amap/index.html

[fastcluster] Fast hierarchical clustering routines for R and Python, http://cran.r-project.org/web/packages/fastcluster/index.html

[Sun07] Yijun Sun. Iterative RELIEF for Feature Weighting: Algorithms, Theories, and Applications. IEEE Trans. Pattern Anal. Mach. Intell. 29(6): 1035-1051, 2007.

[Cai08] D Cai, X He, J Han. SRDA: An Efficient Algorithm for Large-Scale Discriminant Analysis. Knowledge and Data Engineering, IEEE Transactions on Volume 20, Issue 1, Jan. 2008 Page(s):1 - 12.

[Sharma07] A Sharma, K K Paliwal. Fast principal component analysis using fixed-point algorithm. Pattern Recognition Letters 28 (2007) 1151–1155.

[Mika99] S Mika et al. Fisher Discriminant Analysis with Kernels. Neural Networks for Signal Processing IX, 1999. Proceedings of the 1999 IEEE Signal Processing Society Workshop.

[Scholkopf96] B Scholkopf, A Smola, KR Muller. Nonlinear Component Analysis as a Kernel EigenValue Problem

[Jurman08] G Jurman, S Riccadonna, R Visintainer and C Furlanello. Algebraic stability indicators for ranked lists in molecular profiling. Bioinformatics Vol. 24 no. 2 2008, pages 258–264.

[Salvador04] S Salvador and P Chan. FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space. 3rd Wkshp. on Mining Temporal and Sequential Data, ACM KDD '04, 2004.

[Torrence98] C Torrence and G P Compo. Practical Guide to Wavelet Analysis

[Gsldwt] Gnu Scientific Library, http://www.gnu.org/software/gsl/

PYTHON MODULE INDEX

m

mlpy, **??**
mlpy.wavelet, 75

INDEX

K

`kernel_center()` (*in module mlpy*), 26
`kernel_exponential()` (*in module mlpy*), 25
`kernel_gaussian()` (*in module mlpy*), 25
`kernel_linear()` (*in module mlpy*), 25
`kernel_polynomial()` (*in module mlpy*), 25
`kernel_sigmoid()` (*in module mlpy*), 25
`kmeans()` (*in module mlpy*), 47

M

`mlpy`
    `module`, 1
`mlpy.ClassTree` (*class in mlpy*), 34
`mlpy.KernelAdatron` (*class in mlpy*), 39
`mlpy.KNN` (*class in mlpy*), 32
`mlpy.LibLinear` (*class in mlpy*), 43
`mlpy.LibSvm` (*class in mlpy*), 37
`mlpy.MaximumLikelihoodC` (*class in mlpy*), 35
`mlpy.wavelet`
    `module`, 73
`module`
    `mlpy`, 1
    `mlpy.wavelet`, 73