

MogaML

Generated by Doxygen 1.9.6



<b>1 File Index</b>	<b>1</b>
1.1 File List	1
<b>2 File Documentation</b>	<b>3</b>
2.1 src/dimreduction.c File Reference	3
2.1.1 Function Documentation	3
2.1.1.1 ChiSquared_Reduce()	3
2.1.1.2 PCA_Reduce()	4
2.2 src/main.c File Reference	4
2.2.1 Function Documentation	4
2.2.1.1 main()	4
2.3 src/matrix.c File Reference	4
2.3.1 Macro Definition Documentation	6
2.3.1.1 ERROR	6
2.3.2 Function Documentation	6
2.3.2.1 Matrix_add()	6
2.3.2.2 Matrix_approx_equal()	7
2.3.2.3 Matrix_clone()	7
2.3.2.4 Matrix_col()	8
2.3.2.5 Matrix_det()	8
2.3.2.6 Matrix_display()	9
2.3.2.7 Matrix_equal()	9
2.3.2.8 Matrix_free()	9
2.3.2.9 Matrix_frobenius_norm()	10
2.3.2.10 Matrix_from_array()	10
2.3.2.11 Matrix_identity()	10
2.3.2.12 Matrix_inverse()	11
2.3.2.13 Matrix_minor()	11
2.3.2.14 Matrix_multiply()	12
2.3.2.15 Matrix_norm()	12
2.3.2.16 Matrix_reset()	13
2.3.2.17 Matrix_row()	13
2.3.2.18 Matrix_scale()	13
2.3.2.19 Matrix_slice_rows()	14
2.3.2.20 Matrix_solve()	14
2.3.2.21 Matrix_sub()	15
2.3.2.22 Matrix_submatrix()	15
2.3.2.23 Matrix_svd()	16
2.3.2.24 Matrix_trace()	16
2.3.2.25 Matrix_transpose()	17
2.3.2.26 Matrix_zeros()	17
2.3.2.27 Vector_max()	17

2.3.2.28 Vector_max_index()	18
2.3.2.29 Vector_norm()	18
2.3.3 Variable Documentation	19
2.3.3.1 MATRIX_MAX_ITER	19
2.3.3.2 MATRIX_TOLERANCE	19
2.4 src/neuralnetwork.c File Reference	19
2.4.1 Function Documentation	19
2.4.1.1 NN_backward_pass()	19
2.4.1.2 NN_create_layer()	20
2.4.1.3 NN_create_network()	20
2.4.1.4 NN_create_neuron()	20
2.4.1.5 NN_forward_pass()	20
2.4.1.6 NN_get_sigmoid()	20
2.5 src/supervised.c File Reference	20
2.5.1 Macro Definition Documentation	22
2.5.1.1 ERROR	22
2.5.2 Function Documentation	22
2.5.2.1 GaussianNBC()	22
2.5.2.2 GaussianNBC_predict()	23
2.5.2.3 KNN_append_data()	23
2.5.2.4 KNN_free()	24
2.5.2.5 KNN_predict()	24
2.5.2.6 KNNClassifier()	24
2.5.2.7 KNNRegressor()	25
2.5.2.8 LassoRegression()	25
2.5.2.9 LassoRegression_compute_gradient()	26
2.5.2.10 LassoRegression_compute_mse()	26
2.5.2.11 LassoRegression_default_loss()	27
2.5.2.12 LassoRegression_exact_optimum()	27
2.5.2.13 LinearRegression()	28
2.5.2.14 LinearRegression_compute_gradient()	28
2.5.2.15 LinearRegression_compute_mse()	29
2.5.2.16 LinearRegression_default_loss()	29
2.5.2.17 LinearRegression_exact_optimum()	29
2.5.2.18 LinearRegression_free()	30
2.5.2.19 LinearRegression_predict()	30
2.5.2.20 LinearRegression_set_loss()	30
2.5.2.21 LinearRegression_set_mode()	31
2.5.2.22 LinearRegression_train()	31
2.5.2.23 LogisticRegression()	31
2.5.2.24 LogisticRegression_predict()	32
2.5.2.25 LogisticRegression_train()	32

2.5.2.26 RidgeRegression()	33
2.5.2.27 RidgeRegression_compute_gradient()	33
2.5.2.28 RidgeRegression_compute_mse()	34
2.5.2.29 RidgeRegression_default_loss()	34
2.5.2.30 RidgeRegression_exact_optimum()	34
2.5.2.31 Supervised_find_nearest_centroid()	35
2.5.2.32 Supervised_read_csv()	35
2.6 src/unsupervised.c File Reference	36
2.6.1 Macro Definition Documentation	36
2.6.1.1 EPOCHS	36
2.6.1.2 ERROR	37
2.6.1.3 LEARNING_RATE	37
2.6.1.4 MINIBATCH_SIZE	37
2.6.1.5 TOLERANCE	37
2.6.2 Function Documentation	37
2.6.2.1 KMeans()	37
2.6.2.2 KMeans_free()	38
2.6.2.3 KMeans_predict()	38
2.6.2.4 KMeans_train()	38
2.6.2.5 Unsupervised_read_csv()	39
<b>Index</b>	<b>41</b>



# Chapter 1

## File Index

### 1.1 File List

Here is a list of all files with brief descriptions:

src/ <a href="#">dimreduction.c</a> . . . . .	3
src/ <a href="#">main.c</a> . . . . .	4
src/ <a href="#">matrix.c</a> . . . . .	4
src/ <a href="#">neuralnetwork.c</a> . . . . .	19
src/ <a href="#">supervised.c</a> . . . . .	20
src/ <a href="#">unsupervised.c</a> . . . . .	36





## Chapter 2

# File Documentation

### 2.1 src/dimreduction.c File Reference

```
#include "dimreduction.h"
```

#### Functions

- Matrix [PCA\\_Reduce](#) (const Matrix \*X, unsigned int target\_rank)  
*Computes a low-rank approximation of a matrix using Principle Component Analysis.*
- Matrix [ChiSquared\\_Reduce](#) (const Matrix \*X, const Matrix \*y, unsigned int target\_features)  
*Selects the most relevant input features using the chi-squared statistical test.*

#### 2.1.1 Function Documentation

##### 2.1.1.1 ChiSquared\_Reduce()

```
Matrix ChiSquared_Reduce (
    const Matrix * X,
    const Matrix * y,
    unsigned int target_features )
```

Selects the most relevant input features using the chi-squared statistical test.

#### Parameters

<i>X</i>	original input data
<i>y</i>	output labels
<i>target_features</i>	Number of features to select

**Returns**

Matrix with target\_features input features

**2.1.1.2 PCA\_Reduce()**

```
Matrix PCA_Reduce (
    const Matrix * X,
    unsigned int target_rank )
```

Computes a low-rank approximation of a matrix using Principle Component Analysis.

**Parameters**

<i>X</i>	the Matrix
<i>target_rank</i>	rank of the output Matrix

**Returns**

Matrix of rank target\_rank

**2.2 src/main.c File Reference**

```
#include <stdio.h>
#include "matrix.h"
```

**Functions**

- int [main](#) ()

**2.2.1 Function Documentation****2.2.1.1 main()**

```
int main ( )
```

**2.3 src/matrix.c File Reference**

```
#include "matrix.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

## Macros

- `#define ERROR(fmt, ...)`  
*Macro to output message to error stream.*

## Functions

- void `Matrix_free` (Matrix mat)  
*Frees the memory allocated for the Matrix.*
- void `Matrix_reset` (Matrix \*mat)  
*Sets all entries of the Matrix to 0.*
- void `Matrix_display` (const Matrix \*mat)  
*Prints the contents of the mAtrix to stdout, formatted with newlines.*
- bool `Matrix_equal` (const Matrix \*mat1, const Matrix \*mat2)  
*Checks if two Matrices are pairwise (exactly) equal.*
- bool `Matrix_approx_equal` (const Matrix \*mat1, const Matrix \*mat2, double tolerance)  
*Checks if two Matrices are pairwise equal within a given tolerance.*
- double `Matrix_det` (const Matrix \*mat)  
*Computes the determinant of a square Matrix.*
- double `Matrix_trace` (const Matrix \*mat)  
*Computes the sum of the diagonal entries of a square Matrix.*
- double `Matrix_norm` (const Matrix \*mat)  
*Computes the l2-norm of a Matrix.*
- double `Matrix_frobenius_norm` (const Matrix \*mat)  
*Computes the frobenius norm of a Matrix.*
- double `Vector_norm` (const Matrix \*mat, unsigned int k)  
*Computes the l-k norm of a Vector.*
- double `Vector_max` (const Matrix \*mat)  
*Computes the largest value in a row vector.*
- int `Vector_max_index` (const Matrix \*mat)  
*Computes the index of the largest value in a row vector.*
- Matrix `Matrix_zeros` (int rows, int cols)  
*Initialises new Matrix with entries set to 0.*
- Matrix `Matrix_identity` (int size)  
*Initialises an Identity matrix (square matrix with diagonal entries 1)*
- Matrix `Matrix_from_array` (int rows, int cols, double \*data)  
*Initialises a Matrix given a 2D array of elements.*
- Matrix `Matrix_scale` (double c, const Matrix \*mat)  
*Scales a matrix by a constant factor.*
- Matrix `Matrix_add` (const Matrix \*mat1, const Matrix \*mat2)  
*Sums the entries in two Matrices element-wise.*
- Matrix `Matrix_sub` (const Matrix \*mat1, const Matrix \*mat2)  
*Subtracts the entries in two Matrices element-wise.*
- Matrix `Matrix_multiply` (const Matrix \*mat1, const Matrix \*mat2)  
*Multiplies two Matrices.*
- Matrix `Matrix_minor` (const Matrix \*mat, int row, int col)  
*Computes the minor of a Matrix by removing a row and a column.*
- Matrix `Matrix_row` (const Matrix \*mat, int row\_index)  
*Get a single row from a matrix.*
- Matrix `Matrix_col` (const Matrix \*mat, int col\_index)

- Get a single column from a matrix.*

• Matrix [Matrix\\_slice\\_rows](#) (const Matrix \*mat, int start, int end)

*Get rows within the specified range.*
- Matrix [Matrix\\_submatrix](#) (const Matrix \*mat, int start\_row, int end\_row, int start\_col, int end\_col)

*Get a Matrix that contains a subset of the rows and columns.*
- Matrix [Matrix\\_transpose](#) (const Matrix \*mat)

*Flip the rows and columns of a Matrix.*
- Matrix [Matrix\\_clone](#) (const Matrix \*mat)

*Create a new matrix with identical entries.*
- Matrix [Matrix\\_inverse](#) (const Matrix \*mat)

*Compute the inverse of a square Matrix.*
- Matrix [Matrix\\_solve](#) (const Matrix \*A, const Matrix \*b)

*Return the solution to the equation  $Ax=b$ .*
- SVDResult [Matrix\\_svd](#) (const Matrix \*mat)

*Compute the Singular Value Decomposition (SVD) of a Matrix.*

## Variables

- const int [MATRIX\\_MAX\\_ITER](#) = 1000

*Maximum number of iterations for any numerical method on Matrices.*
- const double [MATRIX\\_TOLERANCE](#) = 1e-6

*Maximum margin of error for floating point arithmetic on Matrices.*

## 2.3.1 Macro Definition Documentation

### 2.3.1.1 ERROR

```
#define ERROR(
    fmt,
    ... )
```

**Value:**

```
do { \
    fprintf(stderr, fmt, ##__VA_ARGS__); \
} while (0)
```

Macro to output message to error stream.

## 2.3.2 Function Documentation

### 2.3.2.1 Matrix\_add()

```
Matrix Matrix_add (
    const Matrix * mat1,
    const Matrix * mat2 )
```

Sums the entries in two Matrices element-wise.

**Parameters**

<i>mat1</i>	first Matrix
<i>mat2</i>	second Matrix

**Returns**

Result of the Matrix addition

**Exceptions**

<i>Exception</i>	when the Matrices have different dimensions
------------------	---

**2.3.2.2 Matrix\_approx\_equal()**

```
bool Matrix_approx_equal (
    const Matrix * mat1,
    const Matrix * mat2,
    double tolerance )
```

Checks if two Matrices are pairwise equal within a given tolerance.

**Parameters**

<i>mat1</i>	first Matrix
<i>mat2</i>	second Matrix
<i>tolerance</i>	Maximum difference between each element

**Returns**

true if the matrices have the same dimension, and the entries are within the tolerance of each other  
false otherwise

**2.3.2.3 Matrix\_clone()**

```
Matrix Matrix_clone (
    const Matrix * mat )
```

Create a new matrix with identical entries.

**Parameters**

<i>mat</i>	originial Matrix
------------	------------------

**Returns**

Identical Matrix

**2.3.2.4 Matrix\_col()**

```
Matrix Matrix_col (
    const Matrix * mat,
    int col_index )
```

Get a single column from a matrix.

**Parameters**

<i>mat</i>	original Matrix
<i>col</i>	Index of column to fetch

**Returns**

Column vector (in Matrix form) containing the desired column

**Exceptions**

<i>Exception</i>	requires $0 \leq \text{col} < \text{Matrix.cols}$
------------------	---

**2.3.2.5 Matrix\_det()**

```
double Matrix_det (
    const Matrix * mat )
```

Computes the determinant of a square Matrix.

**Parameters**

<i>mat</i>	the Matrix
------------	------------

**Returns**

double Value of the determinant

**Exceptions**

<i>Exception</i>	for non-square Matrices
------------------	-------------------------

### 2.3.2.6 Matrix\_display()

```
void Matrix_display (
    const Matrix * mat )
```

Prints the contents of the mAtrix to stdout, formatted with newlines.

#### Parameters

<i>mat</i>	Matrix to be displayed
------------	------------------------

### 2.3.2.7 Matrix\_equal()

```
bool Matrix_equal (
    const Matrix * mat1,
    const Matrix * mat2 )
```

Checks if two Matrices are pairwise (exactly) equal.

#### Parameters

<i>mat1</i>	first Matrix
<i>mat2</i>	second Matrix

#### Returns

true if the matrices have the same dimension, and have equal entries  
false otherwise

### 2.3.2.8 Matrix\_free()

```
void Matrix_free (
    Matrix mat )
```

Frees the memory allocated for the Matrix.

#### Parameters

<i>mat</i>	Matrix to be freed
------------	--------------------

**Note**

Subsequent accessing of Matrix data causes segfault

**2.3.2.9 Matrix\_frobenius\_norm()**

```
double Matrix_frobenius_norm (
    const Matrix * mat )
```

Computes the frobenius norm of a Matrix.

**Parameters**

<i>mat</i>	the Matrix
------------	------------

**Returns**

double value of the frobenius norm

**2.3.2.10 Matrix\_from\_array()**

```
Matrix Matrix_from_array (
    int rows,
    int cols,
    double * data )
```

Initialises a Matrix given a 2D array of elements.

**Parameters**

<i>rows</i>	Number of rows in the Matrix
<i>cols</i>	Number of columns in the Matrix
<i>data</i>	2D Array, containing the rows of the Matrix

**Returns**

Matrix

**2.3.2.11 Matrix\_identity()**

```
Matrix Matrix_identity (
    int size )
```

Initialises an Identity matrix (square matrix with diagonal entries 1)



## Parameters

<i>size</i>	Number of rows and columns
-------------	----------------------------

## Returns

Matrix

**2.3.2.12 Matrix\_inverse()**

```
Matrix Matrix_inverse (
    const Matrix * mat )
```

Compute the inverse of a square Matrix.

## Parameters

<i>mat</i>	original Matrix
------------	-----------------

## Returns

Matrix

## Exceptions

<i>Exception</i>	for non-square Matrices, and for singular Matrices
------------------	--

**2.3.2.13 Matrix\_minor()**

```
Matrix Matrix_minor (
    const Matrix * mat,
    int row,
    int col )
```

Computes the minor of a Matrix by removing a row and a column.

## Parameters

<i>mat</i>	original Matrix
<i>row</i>	Index of row to remove
<i>col</i>	Index of column to remove

**Returns**

Remaining Matrix minor

**Exceptions**

<i>Exception</i>	requires $0 \leq \text{row} < \text{Matrix.rows}$ && $0 \leq \text{col} < \text{Matrix.cols}$
------------------	---

**2.3.2.14 Matrix\_multiply()**

```
Matrix Matrix_multiply (
    const Matrix * mat1,
    const Matrix * mat2 )
```

Multiplies two Matrices.

**Parameters**

<i>mat1</i>	first Matrix
<i>mat2</i>	second Matrix

**Returns**

Result of Matrix multiplication

**Exceptions**

<i>Exception</i>	requires $\text{mat1->cols} == \text{mat2->rows}$
------------------	---

**2.3.2.15 Matrix\_norm()**

```
double Matrix_norm (
    const Matrix * mat )
```

Computes the l2-norm of a Matrix.

**Parameters**

<i>mat</i>	the Matrix
------------	------------

**Returns**

double value of the l2-norm

### 2.3.2.16 Matrix\_reset()

```
void Matrix_reset (
    Matrix * mat )
```

Sets all entries of the Matrix to 0.

#### Parameters

<i>mat</i>	Matrix to be reset
------------	--------------------

### 2.3.2.17 Matrix\_row()

```
Matrix Matrix_row (
    const Matrix * mat,
    int row_index )
```

Get a single row from a matrix.

#### Parameters

<i>mat</i>	original Matrix
<i>row</i>	Index of row to fetch

#### Returns

Row vector (in Matrix form) containing the desired row

#### Exceptions

<i>Exception</i>	requires $0 \leq \text{row} < \text{Matrix.rows}$
------------------	---

### 2.3.2.18 Matrix\_scale()

```
Matrix Matrix_scale (
    double c,
    const Matrix * mat )
```

Scales a matrix by a constant factor.

#### Parameters

<i>c</i>	Factor
<i>mat</i>	original Matrix

**Returns**

New Matrix

**2.3.2.19 Matrix\_slice\_rows()**

```
Matrix Matrix_slice_rows (
    const Matrix * mat,
    int start,
    int end )
```

Get rows within the specified range.

**Parameters**

<i>mat</i>	Original matrix
<i>start</i>	Index to start fetching from (inclusive)
<i>end</i>	Index to stop fetching (exclusive)

**Returns**

Sub-matrix containing specified rows

**Exceptions**

<i>Exception</i>	requires $0 \leq \text{start} < \text{end} \leq \text{matrix.rows}$
------------------	---

**2.3.2.20 Matrix\_solve()**

```
Matrix Matrix_solve (
    const Matrix * A,
    const Matrix * b )
```

Return the solution to the equation  $Ax=b$ .

**Parameters**

<i>A</i>	Matrix A
<i>b</i>	column Vector b (embedded as a Matrix)

**Returns**

Column vector x as a Matrix, the solution to the equation

## Exceptions

<i>Exception</i>	for incompatible dimensions, or singular Matrices A
------------------	---

**2.3.2.21 Matrix\_sub()**

```
Matrix Matrix_sub (
    const Matrix * mat1,
    const Matrix * mat2 )
```

Subtracts the entries in two Matrices element-wise.

## Parameters

<i>mat1</i>	first Matrix
<i>mat2</i>	second Matrix

## Returns

Result of the Matrix subtraction

## Exceptions

<i>Exception</i>	when the Matrices have different dimensions
------------------	---

**2.3.2.22 Matrix\_submatrix()**

```
Matrix Matrix_submatrix (
    const Matrix * mat,
    int start_row,
    int end_row,
    int start_col,
    int end_col )
```

Get a Matrix that contains a subset of the rows and columns.

## Parameters

<i>mat</i>	original Matrix
<i>start_row</i>	Index to start fetching rows from (inclusive)
<i>end_row</i>	Index to stop fetching rows from (exclusive)
<i>start_col</i>	Index to start fetching cols from (inclusive)
<i>end_col</i>	Index to stop fetching cols from (exclusive)

**Returns**

Matrix containing the specified rows and columns

**Exceptions**

<i>Exception</i>	requires $0 \leq \text{start\_row} < \text{end\_row} \leq \text{mat->rows}$ && $0 \leq \text{start\_col} < \text{end\_col} \leq \text{mat->cols}$
------------------	---

**2.3.2.23 Matrix\_svd()**

```
SVDResult Matrix_svd (
    const Matrix * mat )
```

Compute the Singular Value Decomposition (SVD) of a Matrix.

**Parameters**

<i>mat</i>	Matrix
------------	--------

**Returns**

SVDResult containing orthonormal Matrices U, V, and diagonal Matrix Sigma with entries in decreasing order

**2.3.2.24 Matrix\_trace()**

```
double Matrix_trace (
    const Matrix * mat )
```

Computes the sum of the diagonal entries of a square Matrix.

**Parameters**

<i>mat</i>	the Matrix
------------	------------

**Returns**

double Value of the trace

**Exceptions**

<i>Exception</i>	for non-square Matrices
------------------	-------------------------

### 2.3.2.25 Matrix\_transpose()

```
Matrix Matrix_transpose (
    const Matrix * mat )
```

Flip the rows and columns of a Matrix.

#### Parameters

<i>mat</i>	original matrix
------------	-----------------

#### Returns

Transposed Matrix

### 2.3.2.26 Matrix\_zeros()

```
Matrix Matrix_zeros (
    int rows,
    int cols )
```

Initialises new Matrix with entries set to 0.

#### Parameters

<i>rows</i>	Number of rows in the Matrix
<i>cols</i>	Number of columns in the Matrix

#### Returns

Matrix

### 2.3.2.27 Vector\_max()

```
double Vector_max (
    const Matrix * mat )
```

Computes the largest value in a row vector.

#### Parameters

<i>mat</i>	the Vector (embedded in a Matrix)
------------	-----------------------------------

**Returns**

double Value of the largest element in the Vector

**Exceptions**

<i>Exception</i>	for Matrices that are not row vectors (Must have dimension (Nx1))
------------------	---

**2.3.2.28 Vector\_max\_index()**

```
int Vector_max_index (
    const Matrix * mat )
```

Computes the index of the largest value in a row vector.

**Parameters**

<i>mat</i>	the Vector (embedded in a Matrix)
------------	-----------------------------------

**Returns**

int Index of the largest element in the Vector

**Exceptions**

<i>Exception</i>	for Matrices that are not row vectors (Must have dimension (Nx1))
------------------	---

**2.3.2.29 Vector\_norm()**

```
double Vector_norm (
    const Matrix * mat,
    unsigned int k )
```

Computes the l-k norm of a Vector.

**Parameters**

<i>mat</i>	the Vector (embedded in a Matrix)
<i>k</i>	Non-negative integer to adjust norm type

**Returns**

double l-k norm



### 2.3.3 Variable Documentation

#### 2.3.3.1 MATRIX\_MAX\_ITER

```
const int MATRIX_MAX_ITER = 1000
```

Maximum number of iterations for any numerical method on Matrices.

#### 2.3.3.2 MATRIX\_TOLERANCE

```
const double MATRIX_TOLERANCE = 1e-6
```

Maximum margin of error for floating point arithmetic on Matrices.

## 2.4 src/neuralnetwork.c File Reference

```
#include "neuralnetwork.h"
```

### Functions

- ActivationFunction [NN\\_get\\_sigmoid](#) ()
- Neuron [NN\\_create\\_neuron](#) (int input\_size, ActivationFunction activation)
- Layer [NN\\_create\\_layer](#) (int num\_neurons, int input\_size, ActivationFunction activation)
- NeuralNetwork [NN\\_create\\_network](#) (int \*layer\_sizes, int num\_layers, ActivationFunction activation)
- Matrix [NN\\_forward\\_pass](#) (NeuralNetwork \*network, const Matrix \*input)
- void [NN\\_backward\\_pass](#) (NeuralNetwork \*network, const Matrix \*input, const Matrix \*expected\_output, double learning\_rate)

### 2.4.1 Function Documentation

#### 2.4.1.1 NN\_backward\_pass()

```
void NN_backward_pass (
    NeuralNetwork * network,
    const Matrix * input,
    const Matrix * expected_output,
    double learning_rate )
```

#### 2.4.1.2 NN\_create\_layer()

```
Layer NN_create_layer (
    int num_neurons,
    int input_size,
    ActivationFunction activation )
```

#### 2.4.1.3 NN\_create\_network()

```
NeuralNetwork NN_create_network (
    int * layer_sizes,
    int num_layers,
    ActivationFunction activation )
```

#### 2.4.1.4 NN\_create\_neuron()

```
Neuron NN_create_neuron (
    int input_size,
    ActivationFunction activation )
```

#### 2.4.1.5 NN\_forward\_pass()

```
Matrix NN_forward_pass (
    NeuralNetwork * network,
    const Matrix * input )
```

#### 2.4.1.6 NN\_get\_sigmoid()

```
ActivationFunction NN_get_sigmoid ( )
```

## 2.5 src/supervised.c File Reference

```
#include "supervised.h"
```

### Macros

- `#define ERROR(fmt, ...)`  
*Macro to output error message to stdout.*

## Functions

- int [Supervised\\_find\\_nearest\\_centroid](#) (const double \*point, const Matrix \*centroids)  
*Find the nearest centroid to a given point (embedded as a double array)*
- LabelledData [Supervised\\_read\\_csv](#) (const char \*filename)  
*Load Comma Separated Values (CSV) into a LabelledData structure, assuming the last element is the label.*
- LinearRegressionModel [LinearRegression](#) (const Matrix \*X, const Matrix \*y)  
*Construct a Linear Regression model with no regularisation.*
- LinearRegressionModel [RidgeRegression](#) (const Matrix \*X, const Matrix \*y, double lambda)  
*Construct a Linear Regression model with Ridge Regularisation.*
- LinearRegressionModel [LassoRegression](#) (const Matrix \*X, const Matrix \*y, double lambda)  
*Construct a Linear Regression model with Lasso Regularisation.*
- void [LinearRegression\\_train](#) (LinearRegressionModel \*model)  
*Train a linear regression model.*
- void [LinearRegression\\_set\\_mode](#) (LinearRegressionModel \*model, enum ComputationMode mode)  
*Set the computation mode of a Linear Regression Model.*
- void [LinearRegression\\_free](#) (LinearRegressionModel model)  
*Free allocated memory for a LinearRegressionModel.*
- void [LinearRegression\\_set\\_loss](#) (LinearRegressionModel \*model, LossFunction loss\_function)  
*Change the loss function.*
- LossFunction [LinearRegression\\_default\\_loss](#) ()  
*Mean Squared Error (MSE) LossFunction with no regularisation.*
- LossFunction [RidgeRegression\\_default\\_loss](#) ()  
*Mean Squared Error (MSE) LossFunction with Ridge regularisation (hyper-parameter lambda = model->hyper\_params[0][0])*
- LossFunction [LassoRegression\\_default\\_loss](#) ()  
*Mean Squared Error (MSE) LossFunction with Lasso regularisation (hyper-parameter lambda = model->hyper\_params[0][0])*
- Matrix [LinearRegression\\_predict](#) (const LinearRegressionModel \*model, const Matrix \*x\_new)  
*Calculate the Linear Regression Estimate for new data.*
- double [LinearRegression\\_compute\\_mse](#) (const Matrix \*X, const Matrix \*y, const Matrix \*params, const Matrix \*hyper\_params)  
*Compute Mean Squared Estimate (MSE)*
- double [RidgeRegression\\_compute\\_mse](#) (const Matrix \*X, const Matrix \*y, const Matrix \*params, const Matrix \*hyper\_params)  
*Compute Mean Squared Estimate (MSE) with Ridge Regularisation.*
- double [LassoRegression\\_compute\\_mse](#) (const Matrix \*X, const Matrix \*y, const Matrix \*params, const Matrix \*hyper\_params)  
*Compute Mean Squared Estimate (MSE) with Lasso Regularisation.*
- Matrix [LinearRegression\\_exact\\_optimum](#) (const Matrix \*X, const Matrix \*y, const Matrix \*hyper\_params)  
*Computes the global optimum algebraically for Linear Regression with no regularisation.*
- Matrix [RidgeRegression\\_exact\\_optimum](#) (const Matrix \*X, const Matrix \*y, const Matrix \*hyper\_params)  
*Computes the global optimum algebraically for Linear Regression with Ridge regularisation.*
- Matrix [LassoRegression\\_exact\\_optimum](#) (const Matrix \*X, const Matrix \*y, const Matrix \*hyper\_params)  
*Placeholder function, throws exception since exact optimum can't be computed for Lasso Regression.*
- Matrix [LinearRegression\\_compute\\_gradient](#) (const Matrix \*X, const Matrix \*y, const Matrix \*params, const Matrix \*hyper\_params)  
*Computes the gradient of the MSE Loss Function.*
- Matrix [RidgeRegression\\_compute\\_gradient](#) (const Matrix \*X, const Matrix \*y, const Matrix \*params, const Matrix \*hyper\_params)  
*Compute a sub-gradient of the MSE Loss Function with Ridge regularisation.*

- Matrix [LassoRegression\\_compute\\_gradient](#) (const Matrix \*X, const Matrix \*y, const Matrix \*params, const Matrix \*hyper\_params)  
*Computes a sub-gradient of the MSE Loss Function with Lasso regularisation.*
- KNNModel [KNNClassifier](#) (unsigned int k, const Matrix \*X, const Matrix \*y)  
*Constructs a KNN model using the majority-vote prediction.*
- KNNModel [KNNRegressor](#) (unsigned int k, const Matrix \*X, const Matrix \*y)  
*Constructs a KNN model using the majority-vote prediction.*
- void [KNN\\_free](#) (KNNModel model)  
*Free memory allocated by KNN Model.*
- void [KNN\\_append\\_data](#) (KNNModel \*model, const Matrix \*X\_new, const Matrix \*y\_new)  
*Add data to existing KNN model.*
- Matrix [KNN\\_predict](#) (const KNNModel \*model, const Matrix \*x\_new)  
*Predict the labels of unseen data using the KNN algorithm.*
- LogisticRegressionModel [LogisticRegression](#) (const Matrix \*X, const Matrix \*y)  
*Construct a Logistic Regression Model.*
- void [LogisticRegression\\_train](#) (LogisticRegressionModel \*model)  
*Train a Logistic Regression model.*
- Matrix [LogisticRegression\\_predict](#) (const LogisticRegressionModel \*model, const Matrix \*X\_new)  
*Predict the labels of unseen data.*
- GaussianNBCModel [GaussianNBC](#) (const Matrix \*X, const Matrix \*y)  
*Create a Naive Bayes Classifier (NBC) model.*
- Matrix [GaussianNBC\\_predict](#) (const GaussianNBCModel \*model, const Matrix \*X\_new)  
*Use the classifier to predict the labels of unseen data.*

## 2.5.1 Macro Definition Documentation

### 2.5.1.1 ERROR

```
#define ERROR(  
    fmt,  
    ... )
```

**Value:**

```
do { \
    fprintf(stderr, fmt, ##__VA_ARGS__); \
} while (0)
```

Macro to output error message to stdout.

## 2.5.2 Function Documentation

### 2.5.2.1 GaussianNBC()

```
GaussianNBCModel GaussianNBC (  
    const Matrix * X,  
    const Matrix * y )
```

Create a Naive Bayes Classifier (NBC) model.

**Parameters**

<i>X</i>	Input features
<i>y</i>	Input labels

**Returns**

GaussianNBCModel

**2.5.2.2 GaussianNBC\_predict()**

```
Matrix GaussianNBC_predict (
    const GaussianNBCModel * model,
    const Matrix * X_new )
```

Use the classifier to predict the labels of unseen data.

**Parameters**

<i>model</i>	Model to be used
<i>X_new</i>	Unseen data

**Returns**

Matrix Predicted labels in Vector form

**2.5.2.3 KNN\_append\_data()**

```
void KNN_append_data (
    KNNModel * model,
    const Matrix * X_new,
    const Matrix * y_new )
```

Add data to existing KNN model.

**Parameters**

<i>model</i>	Existing model
<i>X</i>	New input features
<i>y</i>	New input labels

### 2.5.2.4 KNN\_free()

```
void KNN_free (
    KNNModel model )
```

Free memory allocated by KNN Model.

#### Parameters

<i>model</i>	
--------------	--

### 2.5.2.5 KNN\_predict()

```
Matrix KNN_predict (
    const KNNModel * model,
    const Matrix * x_new )
```

Predict the labels of unseen data using the KNN algorithm.

#### Parameters

<i>model</i>	
<i>x_new</i>	

#### Returns

Matrix

### 2.5.2.6 KNNClassifier()

```
KNNModel KNNClassifier (
    unsigned int k,
    const Matrix * X,
    const Matrix * y )
```

Constructs a KNN model using the majority-vote prediction.

#### Parameters

<i>k</i>	Number of neighbours used by the model
<i>X</i>	Input features
<i>y</i>	Input labels

**Returns**

KNNModel

**2.5.2.7 KNNRegressor()**

```
KNNModel KNNRegressor (
    unsigned int k,
    const Matrix * X,
    const Matrix * y )
```

Constructs a KNN model using the majority-vote prediction.

**Parameters**

<i>k</i>	Number of neighbours used by the model
<i>X</i>	Input features
<i>y</i>	Input labels

**Returns**

KNNModel

**2.5.2.8 LassoRegression()**

```
LinearRegressionModel LassoRegression (
    const Matrix * X,
    const Matrix * y,
    double lambda )
```

Construct a Linear Regression model with Lasso Regularisation.

**Parameters**

<i>X</i>	input data in a Matrix
<i>y</i>	output labels in a Vector (embedded as a Matrix)
<i>lambda</i>	Hyper-parameter for Ridge loss

**Returns**

LinearRegressionModel with default computation mode set to BATCH

**Exceptions**

<i>Exception</i>	when $X \rightarrow \text{rows} \neq y \rightarrow \text{rows}$
------------------	---

**Note**

No algebraic solution exists for lasso regression  
 Input Matrix is padded with a 1-column for the bias term

**2.5.2.9 LassoRegression\_compute\_gradient()**

```
Matrix LassoRegression_compute_gradient (
    const Matrix * X,
    const Matrix * y,
    const Matrix * params,
    const Matrix * hyper_params )
```

Computes a sub-gradient of the MSE Loss Function with Lasso regularisation.

**Parameters**

<i>X</i>	Input features
<i>y</i>	Input labels
<i>params</i>	Current parameters
<i>hyper_params</i>	Contains lambda in data[0][0]

**Returns**

Matrix derivative of the loss function wrt the parameters

**2.5.2.10 LassoRegression\_compute\_mse()**

```
double LassoRegression_compute_mse (
    const Matrix * X,
    const Matrix * y,
    const Matrix * params,
    const Matrix * hyper_params )
```

Compute Mean Squared Estimate (MSE) with Lasso Regularisation.

**Parameters**

<i>X</i>	Matrix with input data
<i>y</i>	Vector (embedded in a Matrix) with correct labels
<i>params</i>	Weights and bias used to make prediction
<i>hyper_params</i>	Contains lambda in data[0][0]



**Returns**

double Mean Squared Error

**2.5.2.11 LassoRegression\_default\_loss()**

```
LossFunction LassoRegression_default_loss ( )
```

Mean Squared Error (MSE) LossFunction with Lasso regularisation (hyper-parameter lambda = model->hyper\_params[0][0])

**Returns**

LossFunction

**Note**

Cannot be trained with computation mode set to ALGEBRAIC

**2.5.2.12 LassoRegression\_exact\_optimum()**

```
Matrix LassoRegression_exact_optimum (
    const Matrix * X,
    const Matrix * y,
    const Matrix * hyper_params )
```

Placeholder function, throws exception since exact optimum can't be computed for Lasso Regression.

**Parameters**

<i>X</i>	Input features
<i>y</i>	Input labels
<i>hyper_params</i>	Contains lambda in data[0][0]

**Returns**

Matrix Global optimum value of params

**Exceptions**

<i>Exception</i>	
------------------	--

### 2.5.2.13 LinearRegression()

```
LinearRegressionModel LinearRegression (
    const Matrix * X,
    const Matrix * y )
```

Construct a Linear Regression model with no regularisation.

#### Parameters

<i>X</i>	input data in a Matrix
<i>y</i>	output labels in a Vector (embedded as a Matrix)

#### Returns

LinearRegressionModel with default computation mode set to ALGEBRAIC

#### Exceptions

<i>Exception</i>	when $X \rightarrow \text{rows} \neq y \rightarrow \text{rows}$
------------------	---

#### Note

Input Matrix is padded with a 1-column for the bias term

### 2.5.2.14 LinearRegression\_compute\_gradient()

```
Matrix LinearRegression_compute_gradient (
    const Matrix * X,
    const Matrix * y,
    const Matrix * params,
    const Matrix * hyper_params )
```

Computes the gradient of the MSE Loss Function.

#### Parameters

<i>X</i>	Input features
<i>y</i>	Input labels
<i>params</i>	Current parameters
<i>hyper_params</i>	Unneeded

#### Returns

Matrix derivative of the loss function wrt the parameters

### 2.5.2.15 LinearRegression\_compute\_mse()

```
double LinearRegression_compute_mse (
    const Matrix * X,
    const Matrix * y,
    const Matrix * params,
    const Matrix * hyper_params )
```

Compute Mean Squared Estimate (MSE)

#### Parameters

<i>X</i>	Matrix with input data
<i>y</i>	Vector (embedded in a Matrix) with correct labels
<i>params</i>	Weights and bias used to make prediction
<i>hyper_params</i>	Unneeded

#### Returns

double Mean Squared Error

### 2.5.2.16 LinearRegression\_default\_loss()

```
LossFunction LinearRegression_default_loss ( )
```

Mean Squared Error (MSE) LossFunction with no regularisation.

#### Returns

LossFunction

### 2.5.2.17 LinearRegression\_exact\_optimum()

```
Matrix LinearRegression_exact_optimum (
    const Matrix * X,
    const Matrix * y,
    const Matrix * hyper_params )
```

Computes the global optimum algebraically for Linear Regression with no regularisation.

#### Parameters

<i>X</i>	Input features
<i>y</i>	Input labels
<i>hyper_params</i>	Unneeded

**Returns**

Matrix Global optimum value of params

**2.5.2.18 LinearRegression\_free()**

```
void LinearRegression_free (
    LinearRegressionModel model )
```

Free allocated memory for a LinearRegressionModel.

**Parameters**

<i>model</i>	Model to be freed
--------------	-------------------

**Note**

Further calls to model may segfault

**2.5.2.19 LinearRegression\_predict()**

```
Matrix LinearRegression_predict (
    const LinearRegressionModel * model,
    const Matrix * x_new )
```

Calculate the Linear Regression Estimate for new data.

**Parameters**

<i>model</i>	Linear Regression Model to predict with
<i>x_new</i>	Matrix containing the new (unpadded) data as rows

**Returns**

Vector (embedded as Matrix) containing the predicted labels

**2.5.2.20 LinearRegression\_set\_loss()**

```
void LinearRegression_set_loss (
    LinearRegressionModel * model,
    LossFunction loss_function )
```

Change the loss function.

**Parameters**

<i>model</i>	model to be changed
<i>loss_function</i>	New loss function

**Note**

model needs to be retrained before predictions are made

**2.5.2.21 LinearRegression\_set\_mode()**

```
void LinearRegression_set_mode (
    LinearRegressionModel * model,
    enum ComputationMode mode )
```

Set the computation mode of a Linear Regression Model.

**Parameters**

<i>model</i>	The model to be changed
<i>mode</i>	Desired computation mode

**Note**

Some LossFunctions do not have an algebraic solution, and may throw exceptions when trained with computation mode ALGEBRAIC

**2.5.2.22 LinearRegression\_train()**

```
void LinearRegression_train (
    LinearRegressionModel * model )
```

Train a linear regression model.

**Parameters**

<i>model</i>	Model to be trained
--------------	---------------------

**2.5.2.23 LogisticRegression()**

```
LogisticRegressionModel LogisticRegression (
```

```
const Matrix * X,
const Matrix * y )
```

Construct a Logistic Regression Model.

#### Parameters

<i>X</i>	Input features
<i>y</i>	Input labels

#### Returns

LogisticRegressionModel

### 2.5.2.24 LogisticRegression\_predict()

```
Matrix LogisticRegression_predict (
    const LogisticRegressionModel * model,
    const Matrix * X_new )
```

Predict the labels of unseen data.

#### Parameters

<i>model</i>	Model to be used
<i>X_new</i>	Data to be predicted

#### Returns

Vector (embedded in a Matrix) containing the predicted labels

#### Exceptions

<i>Exception</i>	requires model to be trained
------------------	------------------------------

### 2.5.2.25 LogisticRegression\_train()

```
void LogisticRegression_train (
    LogisticRegressionModel * model )
```

Train a Logistic Regression model.

#### Parameters

<i>model</i>	to be trained
--------------	---------------

### 2.5.2.26 RidgeRegression()

```
LinearRegressionModel RidgeRegression (
    const Matrix * X,
    const Matrix * y,
    double lambda )
```

Construct a Linear Regression model with Ridge Regularisation.

#### Parameters

<i>X</i>	input data in a Matrix
<i>y</i>	output labels in a Vector (embedded as a Matrix)
<i>lambda</i>	Hyper-parameter for Ridge loss

#### Returns

LinearRegressionModel with default computation mode set to ALGEBRAIC

#### Exceptions

<i>Exception</i>	when $X \rightarrow \text{rows} \neq y \rightarrow \text{rows}$
------------------	---

#### Note

Input Matrix is padded with a 1-column for the bias term

### 2.5.2.27 RidgeRegression\_compute\_gradient()

```
Matrix RidgeRegression_compute_gradient (
    const Matrix * X,
    const Matrix * y,
    const Matrix * params,
    const Matrix * hyper_params )
```

Compute a sub-gradient of the MSE Loss Function with Ridge regularisation.

#### Parameters

<i>X</i>	Input features
<i>y</i>	Input labels
<i>params</i>	Current parameters
<i>hyper_params</i>	Contains lambda in data[0][0]

**Returns**

Matrix derivative of the loss function wrt the parameters

**2.5.2.28 RidgeRegression\_compute\_mse()**

```
double RidgeRegression_compute_mse (
    const Matrix * X,
    const Matrix * y,
    const Matrix * params,
    const Matrix * hyper_params )
```

Compute Mean Squared Estimate (MSE) with Ridge Regularisation.

**Parameters**

<i>X</i>	Matrix with input data
<i>y</i>	Vector (embedded in a Matrix) with correct labels
<i>params</i>	Weights and bias used to make prediction
<i>hyper_params</i>	Contains lambda in data[0][0]

**Returns**

double Mean Squared Error

**2.5.2.29 RidgeRegression\_default\_loss()**

```
LossFunction RidgeRegression_default_loss ( )
```

Mean Squared Error (MSE) LossFunction with Ridge regularisation (hyper-parameter lambda = model->hyper\_params[0][0])

**Returns**

LossFunction

**2.5.2.30 RidgeRegression\_exact\_optimum()**

```
Matrix RidgeRegression_exact_optimum (
    const Matrix * X,
    const Matrix * y,
    const Matrix * hyper_params )
```

Computes the global optimum algebraically for Linear Regression with Ridge regularisation.



## Parameters

<i>X</i>	Input features
<i>y</i>	Input labels
<i>hyper_params</i>	Contains lambda in data[0][0]

## Returns

Matrix Global optimum value of params

**2.5.2.31 Supervised\_find\_nearest\_centroid()**

```
int Supervised_find_nearest_centroid (
    const double * point,
    const Matrix * centroids )
```

Find the nearest centroid to a given point (embedded as a double array)

## Parameters

<i>point</i>	Array of values representing point in euclidean space
<i>centroids</i>	matrix containing centroid coordinates as rows

## Returns

index of closes centroid

**2.5.2.32 Supervised\_read\_csv()**

```
LabelledData Supervised_read_csv (
    const char * filename )
```

Load Comma Separated Values (CSV) into a LabelledData structure, assuming the last element is the label.

## Parameters

<i>filename</i>	
-----------------	--

## Returns

LabelledData parsed data

### Exceptions

<i>Exception</i>	file must exist, and be structured appropriately
------------------	--

## 2.6 src/unsupervised.c File Reference

```
#include "unsupervised.h"
```

### Macros

- #define [LEARNING\\_RATE](#) 0.01
- #define [EPOCHS](#) 1000
- #define [MINIBATCH\\_SIZE](#) 32
- #define [TOLERANCE](#) 1e-6
- #define [ERROR](#)(fmt, ...)

### Functions

- Matrix [Unsupervised\\_read\\_csv](#) (const char \*filename)  
*Reads a Matrix from a Comma Separated Value (CSV) file.*
- KMeansModel [KMeans](#) (unsigned int k, const Matrix \*X)  
*Construct a KMeansModel with a set of data.*
- void [KMeans\\_train](#) (KMeansModel \*model)  
*Train the model using Lloyd's algorithm.*
- Matrix [KMeans\\_predict](#) (const KMeansModel \*model, const Matrix \*X\_new)  
*Predict the cluster of unseen data using a KMeansModel.*
- void [KMeans\\_free](#) (KMeansModel \*model)  
*Free memory allocated to KMeans Model.*

### 2.6.1 Macro Definition Documentation

#### 2.6.1.1 EPOCHS

```
#define EPOCHS 1000
```

### 2.6.1.2 ERROR

```
#define ERROR(  
    fmt,  
    ... )
```

Value:

```
do { \  
    fprintf(stderr, fmt, ##__VA_ARGS__); \  
} while (0)
```

### 2.6.1.3 LEARNING\_RATE

```
#define LEARNING_RATE 0.01
```

### 2.6.1.4 MINIBATCH\_SIZE

```
#define MINIBATCH_SIZE 32
```

### 2.6.1.5 TOLERANCE

```
#define TOLERANCE 1e-6
```

## 2.6.2 Function Documentation

### 2.6.2.1 KMeans()

```
KMeansModel KMeans (  
    unsigned int k,  
    const Matrix * X )
```

Construct a KMeansModel with a set of data.

Parameters

<i>k</i>	Number of clusters
<i>X</i>	Data

**Returns**

KMeansModel

**2.6.2.2 KMeans\_free()**

```
void KMeans_free (
    KMeansModel * model )
```

Free memory allocated to KMeans Model.

**Parameters**

<i>model</i>	Model to be freed
--------------	-------------------

**Note**

Subsequent accesses to the model may segfault

**2.6.2.3 KMeans\_predict()**

```
Matrix KMeans_predict (
    const KMeansModel * model,
    const Matrix * X_new )
```

Predict the cluster of unseen data using a KMeansModel.

**Parameters**

<i>model</i>	
<i>X_new</i>	

**Returns**

Matrix Predicted clusters

**2.6.2.4 KMeans\_train()**

```
void KMeans_train (
    KMeansModel * model )
```

Train the model using lloyd's algorithm.

## Parameters

<i>model</i>	Model to be trained
--------------	---------------------

### 2.6.2.5 Unsupervised\_read\_csv()

```
Matrix Unsupervised_read_csv (  
    const char * filename )
```

Reads a Matrix from a Comma Separated Value (CSV) file.

## Parameters

<i>filename</i>	
-----------------	--

## Returns

Matrix read data



# Index

- ChiSquared\_Reduce
  - dimreduction.c, [3](#)
- dimreduction.c
  - ChiSquared\_Reduce, [3](#)
  - PCA\_Reduce, [4](#)
- EPOCHS
  - unsupervised.c, [36](#)
- ERROR
  - matrix.c, [6](#)
  - supervised.c, [22](#)
  - unsupervised.c, [36](#)
- GaussianNBC
  - supervised.c, [22](#)
- GaussianNBC\_predict
  - supervised.c, [23](#)
- KMeans
  - unsupervised.c, [37](#)
- KMeans\_free
  - unsupervised.c, [38](#)
- KMeans\_predict
  - unsupervised.c, [38](#)
- KMeans\_train
  - unsupervised.c, [38](#)
- KNN\_append\_data
  - supervised.c, [23](#)
- KNN\_free
  - supervised.c, [23](#)
- KNN\_predict
  - supervised.c, [24](#)
- KNNClassifier
  - supervised.c, [24](#)
- KNNRegressor
  - supervised.c, [25](#)
- LassoRegression
  - supervised.c, [25](#)
- LassoRegression\_compute\_gradient
  - supervised.c, [26](#)
- LassoRegression\_compute\_mse
  - supervised.c, [26](#)
- LassoRegression\_default\_loss
  - supervised.c, [27](#)
- LassoRegression\_exact\_optimum
  - supervised.c, [27](#)
- LEARNING\_RATE
  - unsupervised.c, [37](#)
- LinearRegression
  - supervised.c, [27](#)
- LinearRegression\_compute\_gradient
  - supervised.c, [28](#)
- LinearRegression\_compute\_mse
  - supervised.c, [28](#)
- LinearRegression\_default\_loss
  - supervised.c, [29](#)
- LinearRegression\_exact\_optimum
  - supervised.c, [29](#)
- LinearRegression\_free
  - supervised.c, [30](#)
- LinearRegression\_predict
  - supervised.c, [30](#)
- LinearRegression\_set\_loss
  - supervised.c, [30](#)
- LinearRegression\_set\_mode
  - supervised.c, [31](#)
- LinearRegression\_train
  - supervised.c, [31](#)
- LogisticRegression
  - supervised.c, [31](#)
- LogisticRegression\_predict
  - supervised.c, [32](#)
- LogisticRegression\_train
  - supervised.c, [32](#)
- main
  - main.c, [4](#)
- main.c
  - main, [4](#)
- matrix.c
  - ERROR, [6](#)
  - Matrix\_add, [6](#)
  - Matrix\_approx\_equal, [7](#)
  - Matrix\_clone, [7](#)
  - Matrix\_col, [8](#)
  - Matrix\_det, [8](#)
  - Matrix\_display, [9](#)
  - Matrix\_equal, [9](#)
  - Matrix\_free, [9](#)
  - Matrix\_frobenius\_norm, [10](#)
  - Matrix\_from\_array, [10](#)
  - Matrix\_identity, [10](#)
  - Matrix\_inverse, [11](#)
  - MATRIX\_MAX\_ITER, [19](#)
  - Matrix\_minor, [11](#)
  - Matrix\_multiply, [12](#)
  - Matrix\_norm, [12](#)
  - Matrix\_reset, [12](#)
  - Matrix\_row, [13](#)

- Matrix\_scale, 13
- Matrix\_slice\_rows, 14
- Matrix\_solve, 14
- Matrix\_sub, 15
- Matrix\_submatrix, 15
- Matrix\_svd, 16
- MATRIX\_TOLERANCE, 19
- Matrix\_trace, 16
- Matrix\_transpose, 16
- Matrix\_zeros, 17
- Vector\_max, 17
- Vector\_max\_index, 18
- Vector\_norm, 18
- Matrix\_add
  - matrix.c, 6
- Matrix\_approx\_equal
  - matrix.c, 7
- Matrix\_clone
  - matrix.c, 7
- Matrix\_col
  - matrix.c, 8
- Matrix\_det
  - matrix.c, 8
- Matrix\_display
  - matrix.c, 9
- Matrix\_equal
  - matrix.c, 9
- Matrix\_free
  - matrix.c, 9
- Matrix\_frobenius\_norm
  - matrix.c, 10
- Matrix\_from\_array
  - matrix.c, 10
- Matrix\_identity
  - matrix.c, 10
- Matrix\_inverse
  - matrix.c, 11
- MATRIX\_MAX\_ITER
  - matrix.c, 19
- Matrix\_minor
  - matrix.c, 11
- Matrix\_multiply
  - matrix.c, 12
- Matrix\_norm
  - matrix.c, 12
- Matrix\_reset
  - matrix.c, 12
- Matrix\_row
  - matrix.c, 13
- Matrix\_scale
  - matrix.c, 13
- Matrix\_slice\_rows
  - matrix.c, 14
- Matrix\_solve
  - matrix.c, 14
- Matrix\_sub
  - matrix.c, 15
- Matrix\_submatrix
  - matrix.c, 15
- Matrix\_svd
  - matrix.c, 16
- MATRIX\_TOLERANCE
  - matrix.c, 19
- Matrix\_trace
  - matrix.c, 16
- Matrix\_transpose
  - matrix.c, 16
- Matrix\_zeros
  - matrix.c, 17
- MINIBATCH\_SIZE
  - unsupervised.c, 37
- neuralnetwork.c
  - NN\_backward\_pass, 19
  - NN\_create\_layer, 19
  - NN\_create\_network, 20
  - NN\_create\_neuron, 20
  - NN\_forward\_pass, 20
  - NN\_get\_sigmoid, 20
- NN\_backward\_pass
  - neuralnetwork.c, 19
- NN\_create\_layer
  - neuralnetwork.c, 19
- NN\_create\_network
  - neuralnetwork.c, 20
- NN\_create\_neuron
  - neuralnetwork.c, 20
- NN\_forward\_pass
  - neuralnetwork.c, 20
- NN\_get\_sigmoid
  - neuralnetwork.c, 20
- PCA\_Reduce
  - dimreduction.c, 4
- RidgeRegression
  - supervised.c, 33
- RidgeRegression\_compute\_gradient
  - supervised.c, 33
- RidgeRegression\_compute\_mse
  - supervised.c, 34
- RidgeRegression\_default\_loss
  - supervised.c, 34
- RidgeRegression\_exact\_optimum
  - supervised.c, 34
- src/dimreduction.c, 3
- src/main.c, 4
- src/matrix.c, 4
- src/neuralnetwork.c, 19
- src/supervised.c, 20
- src/unsupervised.c, 36
- supervised.c
  - ERROR, 22
  - GaussianNBC, 22
  - GaussianNBC\_predict, 23
  - KNN\_append\_data, 23



- KNN\_free, [23](#)
- KNN\_predict, [24](#)
- KNNClassifier, [24](#)
- KNNRegressor, [25](#)
- LassoRegression, [25](#)
- LassoRegression\_compute\_gradient, [26](#)
- LassoRegression\_compute\_mse, [26](#)
- LassoRegression\_default\_loss, [27](#)
- LassoRegression\_exact\_optimum, [27](#)
- LinearRegression, [27](#)
- LinearRegression\_compute\_gradient, [28](#)
- LinearRegression\_compute\_mse, [28](#)
- LinearRegression\_default\_loss, [29](#)
- LinearRegression\_exact\_optimum, [29](#)
- LinearRegression\_free, [30](#)
- LinearRegression\_predict, [30](#)
- LinearRegression\_set\_loss, [30](#)
- LinearRegression\_set\_mode, [31](#)
- LinearRegression\_train, [31](#)
- LogisticRegression, [31](#)
- LogisticRegression\_predict, [32](#)
- LogisticRegression\_train, [32](#)
- RidgeRegression, [33](#)
- RidgeRegression\_compute\_gradient, [33](#)
- RidgeRegression\_compute\_mse, [34](#)
- RidgeRegression\_default\_loss, [34](#)
- RidgeRegression\_exact\_optimum, [34](#)
- Supervised\_find\_nearest\_centroid, [35](#)
- Supervised\_read\_csv, [35](#)
- Supervised\_find\_nearest\_centroid
  - supervised.c, [35](#)
- Supervised\_read\_csv
  - supervised.c, [35](#)
- TOLERANCE
  - unsupervised.c, [37](#)
- unsupervised.c
  - EPOCHS, [36](#)
  - ERROR, [36](#)
  - KMeans, [37](#)
  - KMeans\_free, [38](#)
  - KMeans\_predict, [38](#)
  - KMeans\_train, [38](#)
  - LEARNING\_RATE, [37](#)
  - MINIBATCH\_SIZE, [37](#)
  - TOLERANCE, [37](#)
  - Unsupervised\_read\_csv, [39](#)
- Unsupervised\_read\_csv
  - unsupervised.c, [39](#)
- Vector\_max
  - matrix.c, [17](#)
- Vector\_max\_index
  - matrix.c, [18](#)
- Vector\_norm
  - matrix.c, [18](#)