MogaML

Generated by Doxygen 1.9.6

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

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

X	original input data
y	output labels
target_features	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

X	the Matrix
target_rank	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

mat1	first Matrix
mat2	second Matrix

Returns

Result of the Matrix addition

Exceptions

Exception	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

mat1	first Matrix
mat2	second Matrix
tolerance	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

mat	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

mat	original Matrix
col	Index of column to fetch

Returns

Column vector (in Matrix form) containing the desired column

Exceptions

Exception	requires 0 <= col < Matrix.cols

2.3.2.5 Matrix_det()

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

Computes the determinant of a square Matrix.

Parameters

mat	the Matrix

Returns

double Value of the determinant

Exceptions

Exception	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

mat	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

mat1	first Matrix
mat2	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

mat	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

mat	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

rows	Number of rows in the Matrix
cols	Number of columns in the Matrix
data	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

size	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

mat	original Matrix

Returns

Matrix

Exceptions

Exception	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

mat	original Matrix
row	Index of row to remove
col	Index of column to remove

Returns

Remaining Matrix minor

Exceptions

Exception	requires 0 <= row < Matrix.rows && 0 <= col < Matrix.cols

2.3.2.14 Matrix_multiply()

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

Multiplies two Matrices.

Parameters

mat1	first Matrix
mat2	second Matrix

Returns

Result of Matrix multiplication

Exceptions

Exception	requires mat1->cols == mat2->rows

2.3.2.15 Matrix_norm()

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

Computes the l2-norm of a Matrix.

Parameters

mat	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

mat	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

mat	original Matrix
row	Index of row to fetch

Returns

Row vector (in Matrix form) containing the desired row

Exceptions

Exception	requires 0 <= row < Matrix.rows

2.3.2.18 Matrix_scale()

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

Scales a matrix by a constant factor.

Parameters

c	Factor
mat	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

mat	Original matrix
start	Index to start fetching from (inclusive)
end	Index to stop fetching (exclusive)

Returns

Sub-matrix containing specified rows

Exceptions

Exception	requires 0 <= start < end <= 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

A	Matrix A
b	column Vector b (embedded as a Matrix)

Returns

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

Exceptions

Exception	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

mat1	first Matrix
mat2	second Matrix

Returns

Result of the Matrix subtraction

Exceptions

Exception	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

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

Returns

Matrix containing the specified rows and columns

Exceptions

Exception	requires 0 <= start_row < end_row <= mat->rows && 0 <= start_col < end_col <= mat->cols

2.3.2.23 Matrix_svd()

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

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

Parameters

mat	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

mat	the Matrix

Returns

double Value of the trace

Exceptions

Exception	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

mat	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

rows	Number of rows in the Matrix
cols	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

mat	the Vector (embedded in a Matrix)

Returns

double Value of the largest element in the Vector

Exceptions

Exception	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

mat	the Vector (embedded in a Matrix)

Returns

int Index of the largest element in the Vector

Exceptions

Exception	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

mat	the Vector (embedded in a Matrix)
k	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

X	Input features
y	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

model	Model to be used
X_new	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

model	Existing model
X	New input features
y	New input labels

2.5.2.4 KNN_free()

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

Free memory allocated by KNN Model.

Parameters

model	

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

model	
x_new	

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

k	Number of neighbours used by the model
X	Input features
y	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

k	Number of neighbours used by the model
X	Input features
y	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

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

Returns

LinearRegressionModel with default computation mode set to BATCH

Exceptions

Exception	when X->rows != y->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

X	Input features
y	Input labels
params	Current parameters
hyper_params	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

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

X	Input features
y	Input labels
hyper_params	Contains lambda in data[0][0]

Returns

Matrix Global optimum value of params

Exceptions

Exception	

2.5.2.13 LinearRegression()

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

Construct a Linear Regression model with no regularisation.

Parameters

X	input data in a Matrix
y	output labels in a Vector (embedded as a Matrix)

Returns

LinearRegressionModel with default computation mode set to ALGEBRAIC

Exceptions

Exception	when X->rows != y->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

X	Input features
y	Input labels
params	Current parameters
hyper_params	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

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

X	Input features
y	Input labels
hyper_params	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

model	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

model	Linear Regression Model to predict with
x_new	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

model	model to be changed
loss_function	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

model	The model to be changed
mode	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

model	Model to be trained

2.5.2.23 LogisticRegression()

```
LogisticRegressionModel LogisticRegression (
             const Matrix * X,
             const Matrix * y )
```

Construct a Logistic Regression Model.

Parameters

X	Input features
y	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

model	Model to be used
X_new	Data to be predicted

Returns

Vector (embedded in a Matrix) containing the predicted labels

Exceptions

Exception	requires model to be trained

2.5.2.25 LogisticRegression_train()

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

Train a Logistic Regression model.

Parameters

model	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

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

Returns

LinearRegressionModel with default computation mode set to ALGEBRAIC

Exceptions

Exception	when X->rows != y->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

X	Input features
y	Input labels
params	Current parameters
hyper_params	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

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

X	Input features
y	Input labels
hyper_params	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

point	Array of values representing point in euclidean space
centroids	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

filename	

Returns

LabelledData parsed data

Exceptions

Exception	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

k	Number of clusters
X	Data

Returns

KMeansModel

2.6.2.2 KMeans_free()

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

Free memory allocated to KMeans Model.

Parameters

model	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

model	
X_new	

Returns

Matrix Predicted clusters

2.6.2.4 KMeans_train()

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

Train the model using lloyd's algorithm.

Parameters

model	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

filename	

Returns

Matrix read data

Index