

EasyML Library

Generated by Doxygen 1.9.7

1 Documentation	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Class Documentation	9
5.1 AutoRegExtractor Class Reference	9
5.1.1 Detailed Description	9
5.1.2 Constructor & Destructor Documentation	9
5.1.2.1 AutoRegExtractor()	9
5.1.3 Member Function Documentation	10
5.1.3.1 extract_X()	10
5.1.3.2 extract_y()	10
5.1.3.3 get_name()	10
5.1.4 Member Data Documentation	10
5.1.4.1 p_	10
5.1.4.2 name_	11
5.2 AutoRegModel< SolverType > Class Template Reference	11
5.2.1 Detailed Description	12
5.2.2 Constructor & Destructor Documentation	12
5.2.2.1 AutoRegModel()	12
5.2.3 Member Function Documentation	13
5.2.3.1 get_weights()	13
5.2.3.2 get_sigma()	13
5.2.3.3 get_order()	13
5.2.3.4 fit()	13
5.2.3.5 predict()	14
5.2.4 Member Data Documentation	14
5.2.4.1 weights_	14
5.2.4.2 sigma_	14
5.2.4.3 p_	14
5.2.4.4 solver_	15
5.3 BaseModel Class Reference	15
5.3.1 Detailed Description	16
5.3.2 Constructor & Destructor Documentation	16
5.3.2.1 BaseModel()	16
5.3.2.2 ~BaseModel()	16
5.3.3 Member Function Documentation	16

5.3.3.1 fit()	16
5.3.3.2 predict()	16
5.3.3.3 is_fitted()	17
5.3.3.4 get_name()	17
5.3.3.5 mark_as_fitted_()	17
5.3.4 Member Data Documentation	17
5.3.4.1 name_	17
5.3.4.2 fitted_	18
5.3.4.3 y_mean_	18
5.4 BaseSolver Class Reference	18
5.4.1 Detailed Description	19
5.4.2 Constructor & Destructor Documentation	19
5.4.2.1 BaseSolver()	19
5.4.2.2 ~BaseSolver()	19
5.4.3 Member Function Documentation	19
5.4.3.1 optimize()	19
5.4.3.2 get_name()	19
5.4.4 Member Data Documentation	20
5.4.4.1 name_	20
5.4.4.2 verbose_	20
5.5 BaseTransformer Class Reference	20
5.5.1 Detailed Description	21
5.5.2 Constructor & Destructor Documentation	21
5.5.2.1 BaseTransformer()	21
5.5.2.2 ~BaseTransformer()	21
5.5.3 Member Function Documentation	21
5.5.3.1 fit()	21
5.5.3.2 transform()	21
5.5.3.3 fit_transform()	22
5.5.3.4 is_fitted()	22
5.5.3.5 get_name()	22
5.5.3.6 mark_as_fitted_()	23
5.5.4 Member Data Documentation	23
5.5.4.1 name_	23
5.5.4.2 fitted_	23
5.6 DerivativeSolver Class Reference	23
5.6.1 Detailed Description	24
5.6.2 Constructor & Destructor Documentation	24
5.6.2.1 DerivativeSolver()	24
5.6.3 Member Function Documentation	25
5.6.3.1 optimize()	25
5.6.3.2 compute_derivative()	25

5.6.4 Member Data Documentation	26
5.6.4.1 diff_loss_func_	26
5.6.4.2 learning_rate_	26
5.6.4.3 max_iter_	26
5.6.4.4 min_derivative_size_	26
5.7 LinRegModel< SolverType > Class Template Reference	26
5.7.1 Detailed Description	27
5.7.2 Constructor & Destructor Documentation	28
5.7.2.1 LinRegModel()	28
5.7.3 Member Function Documentation	28
5.7.3.1 get_weights()	28
5.7.3.2 fit()	28
5.7.3.3 predict()	29
5.7.4 Member Data Documentation	29
5.7.4.1 weights_	29
5.7.4.2 solver_	29
5.8 LogRegModel< SolverType > Class Template Reference	29
5.8.1 Detailed Description	30
5.8.2 Constructor & Destructor Documentation	31
5.8.2.1 LogRegModel()	31
5.8.3 Member Function Documentation	31
5.8.3.1 get_weights()	31
5.8.3.2 fit()	31
5.8.3.3 predict()	32
5.8.3.4 predict_proba()	32
5.8.4 Member Data Documentation	32
5.8.4.1 weights_	32
5.8.4.2 solver_	33
5.9 NewtonShapeException Class Reference	33
5.9.1 Detailed Description	33
5.9.2 Constructor & Destructor Documentation	33
5.9.2.1 NewtonShapeException()	33
5.9.3 Member Function Documentation	34
5.9.3.1 what()	34
5.10 NotFittedException Class Reference	34
5.10.1 Detailed Description	34
5.10.2 Constructor & Destructor Documentation	34
5.10.2.1 NotFittedException()	34
5.10.3 Member Function Documentation	35
5.10.3.1 what()	35
5.10.4 Member Data Documentation	35
5.10.4.1 obj_name_	35

5.11 OLSSolver Class Reference	35
5.11.1 Detailed Description	36
5.11.2 Constructor & Destructor Documentation	36
5.11.2.1 OLSSolver()	36
5.11.3 Member Function Documentation	36
5.11.3.1 optimize()	36
5.12 QRSolver Class Reference	37
5.12.1 Detailed Description	38
5.12.2 Constructor & Destructor Documentation	38
5.12.2.1 QRSolver()	38
5.12.3 Member Function Documentation	38
5.12.3.1 optimize()	38
5.13 StandardScaler Class Reference	39
5.13.1 Detailed Description	40
5.13.2 Constructor & Destructor Documentation	40
5.13.2.1 StandardScaler()	40
5.13.3 Member Function Documentation	40
5.13.3.1 fit()	40
5.13.3.2 transform()	41
5.13.3.3 fit_transform()	41
5.13.3.4 get_means()	41
5.13.3.5 get_stddevs()	42
5.13.4 Member Data Documentation	42
5.13.4.1 means_	42
5.13.4.2 stddevs_	42
6 File Documentation	43
6.1 autoreg_extractor.hpp File Reference	43
6.1.1 Detailed Description	43
6.2 autoreg_model.hpp File Reference	44
6.2.1 Detailed Description	44
6.3 base_model.hpp File Reference	44
6.3.1 Detailed Description	45
6.4 base_solver.hpp File Reference	45
6.4.1 Detailed Description	45
6.5 base_transformer.hpp File Reference	46
6.5.1 Detailed Description	46
6.6 derivative_solver.hpp File Reference	46
6.6.1 Detailed Description	47
6.7 diff_loss_functions.hpp File Reference	47
6.7.1 Detailed Description	48
6.7.2 Function Documentation	48

6.7.2.1 mean_squared_error_loss_grad()	48
6.7.2.2 mean_squared_error_loss_lapl()	49
6.7.2.3 mean_squared_error_loss_newton()	49
6.7.2.4 log_likelihood_loss_grad()	50
6.7.2.5 log_likelihood_loss_lapl()	50
6.7.2.6 log_likelihood_loss_newton()	51
6.7.3 Variable Documentation	51
6.7.3.1 MEAN_SQUARED_ERROR_LOSS_GRAD	51
6.7.3.2 MEAN_SQUARED_ERROR_LOSS_NEWTON	51
6.7.3.3 LOG_LIKELIHOOD_LOSS_GRAD	51
6.7.3.4 LOG_LIKELIHOOD_LOSS_NEWTON	51
6.8 exceptions.hpp File Reference	52
6.8.1 Detailed Description	52
6.9 linreg_model.hpp File Reference	52
6.9.1 Detailed Description	53
6.10 logreg_model.hpp File Reference	53
6.10.1 Detailed Description	53
6.11 metrics.hpp File Reference	54
6.11.1 Detailed Description	55
6.11.2 Function Documentation	55
6.11.2.1 mse()	55
6.11.2.2 sse()	56
6.11.2.3 sst()	56
6.11.2.4 r2()	56
6.11.2.5 accuracy()	57
6.11.2.6 tp_count()	57
6.11.2.7 fp_count()	58
6.11.2.8 tn_count()	58
6.11.2.9 fn_count()	58
6.11.2.10 confusion_matrix()	59
6.11.2.11 precision()	59
6.11.2.12 recall()	60
6.11.2.13 fpr()	60
6.11.2.14 f1_score()	60
6.11.2.15 pr_curve()	61
6.11.2.16 roc_curve()	61
6.11.2.17 tp_curve()	62
6.11.2.18 fp_curve()	62
6.11.2.19 tn_curve()	62
6.11.2.20 fn_curve()	63
6.11.2.21 auc()	63
6.12 ols_solver.hpp File Reference	64

6.12.1 Detailed Description	64
6.13 predict_functions.hpp File Reference	64
6.13.1 Detailed Description	65
6.13.2 Function Documentation	65
6.13.2.1 linreg()	65
6.13.2.2 logistic_function()	66
6.13.2.3 logreg_proba()	66
6.13.2.4 logreg_class()	67
6.13.2.5 ols()	67
6.13.2.6 qr()	67
6.14 qr_solver.hpp File Reference	68
6.14.1 Detailed Description	68
6.15 standard_scaler.hpp File Reference	69
6.15.1 Detailed Description	69
6.16 types.hpp File Reference	69
6.16.1 Detailed Description	70
6.16.2 Typedef Documentation	70
6.16.2.1 Features	70
6.16.2.2 Target	71
6.16.2.3 Weights	71
6.16.2.4 Derivative	71
6.16.2.5 ConfusionMatrix	71
6.16.2.6 Precisions	71
6.16.2.7 Recalls	71
6.16.2.8 Fallouts	71
6.16.2.9 PRCurve	71
6.16.2.10 ROCCurve	72
6.16.2.11 TPs	72
6.16.2.12 FPs	72
6.16.2.13 TNs	72
6.16.2.14 FNs	72
6.16.3 Function Documentation	72
6.16.3.1 get_name()	72
Index	75

Chapter 1

Documentation

Welcome to the EasyML library documentation! EasyML is a set of classic machine learning algorithms written in C++. It extensively uses the [Armadillo](#) library which in turn uses the LAPACK library to work with vectors and matrices. [Boost](#) is also used, for example, to obtain object's human-readable type during runtime and to work with probability distributions.

Supported models and solvers:

- Linear Regression
 - Ordinary Least Squares
 - QR-decomposition
 - Derivative-based: Gradient Descent, Newton (single feature only)
- Logistic Regression
 - Derivative-based: Gradient Descent, Newton (single feature only)
- Autoregressive AR(p)
 - Ordinary Least Squares
 - QR-decomposition
 - Derivative-based: Gradient Descent, Newton (single lag only)

Supported transformers and extractors:

- Standard scaler (z -score transformation)
- Time series (extract features and target from process)

More models and possibly transformers to be implemented in future versions.

Installation instructions and examples of usage can be found in the GitHub [repo](#) of the author.

A good way to start is to browse the Class Hierarchy section.

Author: Andrei Batyrov (arbatyrov@edu.hse.ru)

Copyright (c) 2024

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

AutoRegExtractor	9
BaseModel	15
AutoRegModel< SolverType >	11
LinRegModel< SolverType >	26
LogRegModel< SolverType >	29
BaseSolver	18
DerivativeSolver	23
OLSSolver	35
QRSolver	37
BaseTransformer	20
StandardScaler	39
std::exception	
NewtonShapeException	33
NotFittedException	34

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

AutoRegExtractor	Features and target extractor class for Autoregressive AR(p) model	9
AutoRegModel< SolverType >	Autoregressive AR(p) model class template. Inherits from BaseModel class	11
BaseModel	Base Model class. All concrete model classes must inherit from this class	15
BaseSolver	Base Solver class. All concrete solver classes must inherit from this class	18
BaseTransformer	Base Transformer class. All concrete transformer classes must inherit from this class	20
DerivativeSolver	Derivative Solver class. Inherits from BaseSolver class	23
LinRegModel< SolverType >	Linear Regression model class template. Inherits from BaseModel class	26
LogRegModel< SolverType >	Logistic Regression model class template. Inherits from BaseModel class	29
NewtonShapeException	NewtonShapeException class. Inherits from std::exception class	33
NotFittedException	NotFittedException class. Inherits from std::exception class	34
OLSSolver	Ordinary Least Squares solver class. Inherits from BaseSolver class	35
QRSolver	QR-decomposition solver class. Inherits from BaseSolver class	37
StandardScaler	Standard Scaler (z -score transformation) class. Inherits from BaseTransformer class	39

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

autoreg_extractor.hpp	
AutoRegExtractor class declarations	43
autoreg_model.hpp	
AutoRegModel class declarations	44
base_model.hpp	
BaseModel class declarations	44
base_solver.hpp	
BaseSolver class declarations	45
base_transformer.hpp	
BaseTransformer class declarations	46
derivative_solver.hpp	
DerivativeSolver class declarations	46
diff_loss_functions.hpp	
Derivatives of loss functions declarations and implementation	47
exceptions.hpp	
Custom exceptions declarations and implementation	52
linreg_model.hpp	
LinRegModel class declarations	52
logreg_model.hpp	
LogRegModel class declarations	53
metrics.hpp	
Model metrics declarations and implementation	54
ols_solver.hpp	
OLSSolver class declarations	64
predict_functions.hpp	
Prediction functions declarations and implementation	64
qr_solver.hpp	
QRSolver class declarations	68
standard_scaler.hpp	
StandardScaler class declarations	69
types.hpp	
Custom types used in the library	69

Chapter 5

Class Documentation

5.1 AutoRegExtractor Class Reference

Features and target extractor class for Autoregressive AR(p) model.

```
#include <autoreg_extractor.hpp>
```

Public Member Functions

- [AutoRegExtractor](#) (const size_t p)
Construct a new [AutoRegExtractor](#) object of order p.
- const [Features extract_X](#) (TimeSeries &process)
Extract feature variables: $\{X_{t-i}\}, i = 1 \dots p$.
- const [Target extract_y](#) (TimeSeries &process)
Extract target variable: y_t .
- const std::string [get_name](#) () const
Get extractor's name.

Private Attributes

- size_t [p_](#)
- std::string [name_](#)
Extractor's name (string).

5.1.1 Detailed Description

Features and target extractor class for Autoregressive AR(p) model.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 AutoRegExtractor()

```
AutoRegExtractor::AutoRegExtractor (  
    const size_t p )
```

Construct a new [AutoRegExtractor](#) object of order p.

5.1.3 Member Function Documentation

5.1.3.1 extract_X()

```
const Features AutoRegExtractor::extract_X (
    TimeSeries & process )
```

Extract feature variables: $\{X_{t-i}\}, i = 1 \dots p$.

Parameters

<i>process</i>	Time series vector
----------------	--------------------

Returns

const [Types::Features](#)

5.1.3.2 extract_y()

```
const Target AutoRegExtractor::extract_y (
    TimeSeries & process )
```

Extract target variable: y_t .

Parameters

<i>process</i>	Time series vector
----------------	--------------------

Returns

const [Types::Target](#)

5.1.3.3 get_name()

```
const std::string AutoRegExtractor::get_name ( ) const
```

Get extractor's name.

Returns

std::string

5.1.4 Member Data Documentation

5.1.4.1 p_

```
size_t AutoRegExtractor::p_ [private]
```

Order of lag.

5.1.4.2 name_

```
std::string AutoRegExtractor::name_ [private]
```

Extractor's name (string).

The documentation for this class was generated from the following file:

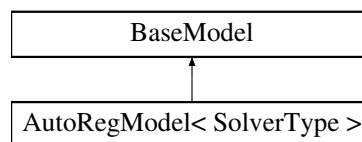
- [autoreg_extractor.hpp](#)

5.2 AutoRegModel< SolverType > Class Template Reference

Autoregressive AR(p) model class template. Inherits from [BaseModel](#) class.

```
#include <autoreg_model.hpp>
```

Inheritance diagram for AutoRegModel< SolverType >:



Public Member Functions

- [AutoRegModel](#) (const SolverType &solver)
Construct a new [AutoRegModel](#) object.
- const [Weights](#) [get_weights](#) () const
Get model's weights.
- const double [get_sigma](#) () const
Get model's sigma (standard deviation).
- const size_t [get_order](#) () const
Get model's order (lag).
- const [AutoRegModel](#) [fit](#) ([Features](#) &X, const [Target](#) &y)
Fit AR(p) model.
- const TimeSeries [predict](#) (const [Features](#) &X, const size_t num_periods) const
Predict future values of time series with fitted model.

Public Member Functions inherited from [BaseModel](#)

- [BaseModel](#) ()
Construct a new [BaseModel](#) object.
- virtual [~BaseModel](#) ()
Destroy the [BaseModel](#) object.
- const [BaseModel](#) [fit](#) (const [Features](#) &X, const [Target](#) &y)
Fit model.
- const [Target](#) [predict](#) (const [Features](#) &X) const
Predict target variable with fitted model.
- const bool [is_fitted](#) () const
Check if model is fitted.
- const std::string [get_name](#) () const
Get model's name.

Private Attributes

- [Weights](#) `weights_`
Row vector of model's weights.
- double `sigma_`
Model's sigma (standard deviation).
- size_t `p_`
Model's order (lag).
- SolverType `solver_`
Solver to fit model with.

Additional Inherited Members

Protected Member Functions inherited from [BaseModel](#)

- void `mark_as_fitted_()`
Mark model as fitted.

Protected Attributes inherited from [BaseModel](#)

- std::string `name_`
Model's name (string).

5.2.1 Detailed Description

```
template<typename SolverType>
class AutoRegModel< SolverType >
```

Autoregressive AR(p) model class template. Inherits from [BaseModel](#) class.

Template Parameters

<i>SolverType</i>	class of solver: BaseSolver , OLSSolver , QRSolver , DerivativeSolver
-------------------	---

5.2.2 Constructor & Destructor Documentation

5.2.2.1 AutoRegModel()

```
template<typename SolverType >
AutoRegModel< SolverType >::AutoRegModel (
    const SolverType & solver )
```

Construct a new [AutoRegModel](#) object.

Parameters

<i>solver</i>	Solver type
---------------	-------------

5.2.3 Member Function Documentation

5.2.3.1 get_weights()

```
template<typename SolverType >
const Weights AutoRegModel< SolverType >::get_weights ( ) const
```

Get model's weights.

Returns

const [Types::Weights](#)

5.2.3.2 get_sigma()

```
template<typename SolverType >
const double AutoRegModel< SolverType >::get_sigma ( ) const
```

Get model's sigma (standard deviation).

Returns

const double

5.2.3.3 get_order()

```
template<typename SolverType >
const size_t AutoRegModel< SolverType >::get_order ( ) const
```

Get model's order (lag).

Returns

const size_t

5.2.3.4 fit()

```
template<typename SolverType >
const AutoRegModel AutoRegModel< SolverType >::fit (
    Features & X,
    const Target & y )
```

Fit AR(p) model.

Parameters

<i>X</i>	Matrix of feature variables extracted with AutoRegExtractor
<i>y</i>	Column vector of target variable extracted with AutoRegExtractor

Returns

[AutoRegModel](#)

5.2.3.5 predict()

```
template<typename SolverType >
const TimeSeries AutoRegModel< SolverType >::predict (
    const Features & X,
    const size_t num_periods ) const
```

Predict future values of time series with fitted model.

Parameters

<i>X</i>	Matrix of feature variables extracted with AutoRegExtractor
<i>num_periods</i>	Number of forecast periods

Returns

const TimeSeries

5.2.4 Member Data Documentation**5.2.4.1 weights_**

```
template<typename SolverType >
Weights AutoRegModel< SolverType >::weights_ [private]
```

Row vector of model's weights.

5.2.4.2 sigma_

```
template<typename SolverType >
double AutoRegModel< SolverType >::sigma_ [private]
```

Model's sigma (standard deviation).

5.2.4.3 p_

```
template<typename SolverType >
size_t AutoRegModel< SolverType >::p_ [private]
```

Model's order (lag).

5.2.4.4 solver_

```
template<typename SolverType >
SolverType AutoRegModel< SolverType >::solver_ [private]
```

Solver to fit model with.

The documentation for this class was generated from the following file:

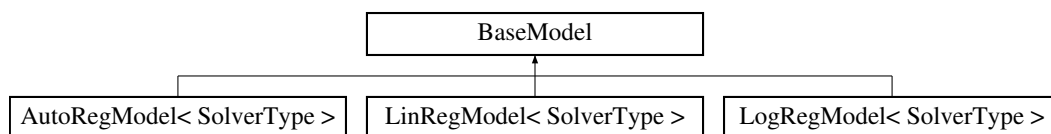
- [autoreg_model.hpp](#)

5.3 BaseModel Class Reference

Base Model class. All concrete model classes must inherit from this class.

```
#include <base_model.hpp>
```

Inheritance diagram for BaseModel:



Public Member Functions

- [BaseModel](#) ()
Construct a new [BaseModel](#) object.
- virtual [~BaseModel](#) ()
Destroy the [BaseModel](#) object.
- const [BaseModel](#) fit (const [Features](#) &X, const [Target](#) &y)
Fit model.
- const [Target](#) predict (const [Features](#) &X) const
Predict target variable with fitted model.
- const bool [is_fitted](#) () const
Check if model is fitted.
- const std::string [get_name](#) () const
Get model's name.

Protected Member Functions

- void [mark_as_fitted](#) ()
Mark model as fitted.

Protected Attributes

- std::string [name](#)_
Model's name (string).

Private Attributes

- bool `fitted_`
- Target `y_mean_`

Base predictions – simply the mean value of target variable.

5.3.1 Detailed Description

Base Model class. All concrete model classes must inherit from this class.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 BaseModel()

```
BaseModel::BaseModel ( )
```

Construct a new [BaseModel](#) object.

5.3.2.2 ~BaseModel()

```
virtual BaseModel::~~BaseModel ( ) [virtual]
```

Destroy the [BaseModel](#) object.

5.3.3 Member Function Documentation

5.3.3.1 fit()

```
const BaseModel BaseModel::fit (
    const Features & X,
    const Target & y )
```

Fit model.

Returns

[BaseModel](#)

5.3.3.2 predict()

```
const Target BaseModel::predict (
    const Features & X ) const
```

Predict target variable with fitted model.

Parameters

<i>X</i>	Matrix of feature variables
----------	-----------------------------

Returns

const [Types::Target](#)

5.3.3.3 is_fitted()

```
const bool BaseModel::is_fitted ( ) const
```

Check if model is fitted.

Returns

true

false

5.3.3.4 get_name()

```
const std::string BaseModel::get_name ( ) const
```

Get model's name.

Returns

std::string

5.3.3.5 mark_as_fitted_()

```
void BaseModel::mark_as_fitted_ ( ) [protected]
```

Mark model as fitted.

5.3.4 Member Data Documentation**5.3.4.1 name_**

```
std::string BaseModel::name_ [protected]
```

Model's name (string).

5.3.4.2 fitted_

```
bool BaseModel::fitted_ [private]
```

Model is fitted flag.

5.3.4.3 y_mean_

```
Target BaseModel::y_mean_ [private]
```

Base predictions – simply the mean value of target variable.

The documentation for this class was generated from the following file:

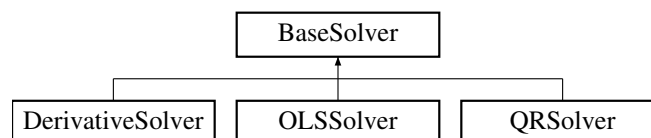
- [base_model.hpp](#)

5.4 BaseSolver Class Reference

Base Solver class. All concrete solver classes must inherit from this class.

```
#include <base_solver.hpp>
```

Inheritance diagram for BaseSolver:



Public Member Functions

- [BaseSolver](#) (const bool verbose=false)
Construct a new Base Solver object.
- virtual [~BaseSolver](#) ()
Destroy the Base Solver object.
- const [Weights](#) [optimize](#) ([Weights](#) &w, const [Features](#) &X, const [Target](#) &y)
Return optimized weights.
- const std::string [get_name](#) () const
Get solver's name.

Protected Attributes

- std::string [name_](#)
Solver's name (string).
- bool [verbose_](#)
Show solver's steps flag.

5.4.1 Detailed Description

Base Solver class. All concrete solver classes must inherit from this class.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 BaseSolver()

```
BaseSolver::BaseSolver (
    const bool verbose = false )
```

Construct a new Base Solver object.

Parameters

<i>verbose</i>	Show solver's steps flag
----------------	--------------------------

5.4.2.2 ~BaseSolver()

```
virtual BaseSolver::~BaseSolver ( ) [virtual]
```

Destroy the Base Solver object.

5.4.3 Member Function Documentation

5.4.3.1 optimize()

```
const Weights BaseSolver::optimize (
    Weights & w,
    const Features & X,
    const Target & y )
```

Return optimized weights.

Parameters

<i>w</i>	Column vector of weights
<i>X</i>	Matrix of feature variables
<i>y</i>	Column vector of target variable

Returns

[Types::Weights](#)

5.4.3.2 get_name()

```
const std::string BaseSolver::get_name ( ) const
```

Get solver's name.

Returns

std::string

5.4.4 Member Data Documentation

5.4.4.1 name_

```
std::string BaseSolver::name_ [protected]
```

Solver's name (string).

5.4.4.2 verbose_

```
bool BaseSolver::verbose_ [protected]
```

Show solver's steps flag.

The documentation for this class was generated from the following file:

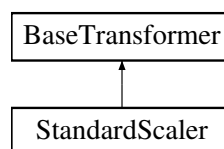
- [base_solver.hpp](#)

5.5 BaseTransformer Class Reference

Base Transformer class. All concrete transformer classes must inherit from this class.

```
#include <base_transformer.hpp>
```

Inheritance diagram for BaseTransformer:



Public Member Functions

- [BaseTransformer](#) ()
Construct a new Base Transformer object.
- virtual [~BaseTransformer](#) ()
Destroy the Base Transformer object.
- [BaseTransformer fit](#) ([Features](#) &X)
Fit transformer.
- const [Features transform](#) ([Features](#) &X)
Transform feature variables with fitted model.
- const [Features fit_transform](#) ([Features](#) &X)
Fit transformer, then transform feature variables.
- const bool [is_fitted](#) () const
Check if transformer is fitted.
- const std::string [get_name](#) () const
Get transformer's name.

Protected Member Functions

- void `mark_as_fitted_()`
Mark transformer as fitted.

Protected Attributes

- std::string `name_`
Transformer's name (string).

Private Attributes

- bool `fitted_`

5.5.1 Detailed Description

Base Transformer class. All concrete transformer classes must inherit from this class.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 BaseTransformer()

```
BaseTransformer::BaseTransformer ( )
```

Construct a new Base Transformer object.

5.5.2.2 ~BaseTransformer()

```
virtual BaseTransformer::~~BaseTransformer ( ) [virtual]
```

Destroy the Base Transformer object.

5.5.3 Member Function Documentation

5.5.3.1 fit()

```
BaseTransformer BaseTransformer::fit (
    Features & X )
```

Fit transformer.

Returns

`BaseTransformer`

5.5.3.2 transform()

```
const Features BaseTransformer::transform (
    Features & X )
```

Transform feature variables with fitted model.

Parameters

<i>X</i>	Matrix of feature variables
----------	-----------------------------

Returns

const [Types::Target](#)

5.5.3.3 fit_transform()

```
const Features BaseTransformer::fit_transform (
    Features & X )
```

Fit transformer, then transform feature variables.

Parameters

<i>X</i>	Matrix of feature variables
----------	-----------------------------

Returns

const [Types::Features](#)

5.5.3.4 is_fitted()

```
const bool BaseTransformer::is_fitted ( ) const
```

Check if transformer is fitted.

Returns

true

false

5.5.3.5 get_name()

```
const std::string BaseTransformer::get_name ( ) const
```

Get transformer's name.

Returns

std::string

5.5.3.6 mark_as_fitted_()

```
void BaseTransformer::mark_as_fitted_ ( ) [protected]
```

Mark transformer as fitted.

5.5.4 Member Data Documentation

5.5.4.1 name_

```
std::string BaseTransformer::name_ [protected]
```

Transformer's name (string).

5.5.4.2 fitted_

```
bool BaseTransformer::fitted_ [private]
```

Transformer is fitted flag.

The documentation for this class was generated from the following file:

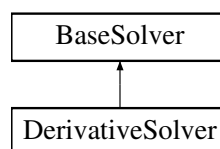
- [base_transformer.hpp](#)

5.6 DerivativeSolver Class Reference

Derivative Solver class. Inherits from [BaseSolver](#) class.

```
#include <derivative_solver.hpp>
```

Inheritance diagram for DerivativeSolver:



Public Member Functions

- [DerivativeSolver](#) (const std::function< [Derivative](#)(const [Weights](#) &, const [Features](#) &, const [Target](#) &)> &diff←_loss_func, const double learning_rate, const size_t max_iter, const double min_grad_size, const bool verbose)
- Construct a new Derivative Solver object.
- const [Weights](#) [optimize](#) ([Weights](#) &w, const [Features](#) &X, const [Target](#) &y)
- Return learned weights by using gradient descent for MSE loss function.
- const [Derivative](#) [compute_derivative](#) (const [Weights](#) &w, const [Features](#) &X, const [Target](#) &y)
- Compute derivative of loss function.

Public Member Functions inherited from [BaseSolver](#)

- [BaseSolver](#) (const bool verbose=false)
Construct a new Base Solver object.
- virtual [~BaseSolver](#) ()
Destroy the Base Solver object.
- const [Weights optimize](#) ([Weights](#) &w, const [Features](#) &X, const [Target](#) &y)
Return optimized weights.
- const std::string [get_name](#) () const
Get solver's name.

Private Attributes

- const std::function< [Derivative](#)(const [Weights](#) &, const [Features](#) &, const [Target](#) &)> & [diff_loss_func_](#)
Derivative of loss function used for computing optimization step.
- const double [learning_rate_](#)
Learning rate.
- const size_t [max_iter_](#)
Max number of optimization iterations.
- const double [min_derivative_size_](#)
Min size of the vector of derivative.

Additional Inherited Members

Protected Attributes inherited from [BaseSolver](#)

- std::string [name_](#)
Solver's name (string).
- bool [verbose_](#)
Show solver's steps flag.

5.6.1 Detailed Description

Derivative Solver class. Inherits from [BaseSolver](#) class.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 [DerivativeSolver\(\)](#)

```
DerivativeSolver::DerivativeSolver (
    const std::function< Derivative(const Weights &, const Features &, const Target
&)> & diff\_loss\_func,
    const double learning\_rate,
    const size_t max\_iter,
    const double min\_grad\_size,
    const bool verbose )
```

Construct a new Derivative Solver object.

Parameters

<i>diff_loss_func</i>	Derivative of loss function
<i>learning_rate</i>	Learning rate
<i>max_iter</i>	Max number of optimization iterations
<i>min_grad_size</i>	Min size of the vector of derivative
<i>verbose</i>	

5.6.3 Member Function Documentation

5.6.3.1 optimize()

```
const Weights DerivativeSolver::optimize (
    Weights & w,
    const Features & X,
    const Target & y )
```

Return learned weights by using gradient descent for MSE loss function.

Parameters

<i>w</i>	Row vector of weights
<i>X</i>	Matrix of feature variables
<i>y</i>	Column vector of target variable

Returns

[Types::Weights](#)

5.6.3.2 compute_derivative()

```
const Derivative DerivativeSolver::compute_derivative (
    const Weights & w,
    const Features & X,
    const Target & y )
```

Compute derivative of loss function.

Parameters

<i>w</i>	Row vector of weights
<i>X</i>	Matrix of feature variables
<i>y</i>	Column vector of target variable

Returns

[Types::Derivative](#)

5.6.4 Member Data Documentation

5.6.4.1 `diff_loss_func_`

```
const std::function<Derivative(const Weights&, const Features&, const Target&)>& Derivative←  
Solver::diff_loss_func_ [private]
```

Derivative of loss function used for computing optimization step.

5.6.4.2 `learning_rate_`

```
const double DerivativeSolver::learning_rate_ [private]
```

Learning rate.

5.6.4.3 `max_iter_`

```
const size_t DerivativeSolver::max_iter_ [private]
```

Max number of optimization iterations.

5.6.4.4 `min_derivative_size_`

```
const double DerivativeSolver::min_derivative_size_ [private]
```

Min size of the vector of derivative.

The documentation for this class was generated from the following file:

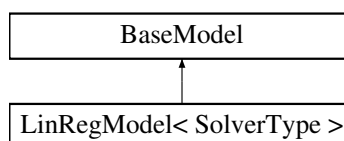
- [derivative_solver.hpp](#)

5.7 `LinRegModel< SolverType >` Class Template Reference

Linear Regression model class template. Inherits from [BaseModel](#) class.

```
#include <linreg_model.hpp>
```

Inheritance diagram for `LinRegModel< SolverType >`:



Public Member Functions

- [LinRegModel](#) (const SolverType & solver)
Construct a new [LinRegModel](#) object.
- const [Weights](#) [get_weights](#) () const
Get model's weights.
- const [LinRegModel](#) [fit](#) ([Features](#) &X, const [Target](#) &y)
Fit model.
- const [Target](#) [predict](#) (const [Features](#) &X) const
Predict target variable with fitted model.

Public Member Functions inherited from [BaseModel](#)

- [BaseModel](#) ()
Construct a new [BaseModel](#) object.
- virtual [~BaseModel](#) ()
Destroy the [BaseModel](#) object.
- const [BaseModel](#) [fit](#) (const [Features](#) &X, const [Target](#) &y)
Fit model.
- const [Target](#) [predict](#) (const [Features](#) &X) const
Predict target variable with fitted model.
- const bool [is_fitted](#) () const
Check if model is fitted.
- const std::string [get_name](#) () const
Get model's name.

Private Attributes

- [Weights](#) [weights_](#)
Row vector of model's weights.
- SolverType [solver_](#)
Solver to fit model with.

Additional Inherited Members**Protected Member Functions inherited from [BaseModel](#)**

- void [mark_as_fitted](#) ()
Mark model as fitted.

Protected Attributes inherited from [BaseModel](#)

- std::string [name_](#)
Model's name (string).

5.7.1 Detailed Description

```
template<typename SolverType>
class LinRegModel< SolverType >
```

Linear Regression model class template. Inherits from [BaseModel](#) class.

Template Parameters

<i>SolverType</i>	class of solver: BaseSolver , OLSSolver , QRSolver , DerivativeSolver
-------------------	---

5.7.2 Constructor & Destructor Documentation

5.7.2.1 LinRegModel()

```
template<typename SolverType >
LinRegModel< SolverType >::LinRegModel (
    const SolverType & solver )
```

Construct a new [LinRegModel](#) object.

5.7.3 Member Function Documentation

5.7.3.1 get_weights()

```
template<typename SolverType >
const Weights LinRegModel< SolverType >::get_weights ( ) const
```

Get model's weights.

Returns

const [Types::Weights](#)

5.7.3.2 fit()

```
template<typename SolverType >
const LinRegModel LinRegModel< SolverType >::fit (
    Features & X,
    const Target & y )
```

Fit model.

Parameters

<i>X</i>	Matrix of feature variables
<i>y</i>	Column vector of target variable

Returns

[LinRegModel](#)

5.7.3.3 predict()

```
template<typename SolverType >
const Target LinRegModel< SolverType >::predict (
    const Features & X ) const
```

Predict target variable with fitted model.

Parameters

X	Matrix of feature variables
----------	-----------------------------

Returns

const [Types::Target](#)

5.7.4 Member Data Documentation

5.7.4.1 weights_

```
template<typename SolverType >
Weights LinRegModel< SolverType >::weights_ [private]
```

Row vector of model's weights.

5.7.4.2 solver_

```
template<typename SolverType >
SolverType LinRegModel< SolverType >::solver_ [private]
```

Solver to fit model with.

The documentation for this class was generated from the following file:

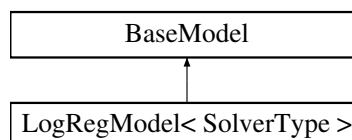
- [linreg_model.hpp](#)

5.8 LogRegModel< SolverType > Class Template Reference

Logistic Regression model class template. Inherits from [BaseModel](#) class.

```
#include <logreg_model.hpp>
```

Inheritance diagram for LogRegModel< SolverType >:



Public Member Functions

- [LogRegModel](#) (const SolverType &solver)
Construct a new [LogRegModel](#) object.
- const [Weights](#) [get_weights](#) () const
Get model's weights.
- const [LogRegModel](#) [fit](#) ([Features](#) &X, const [Target](#) &y)
Fit model.
- const [Target](#) [predict](#) (const [Features](#) &X, const double &threshold=0.5) const
Predict (classify) target variable's class with fitted model at given threshold.
- const [Target](#) [predict_proba](#) (const [Features](#) &X) const
Predict probability of positive class of target variable with fitted model.

Public Member Functions inherited from [BaseModel](#)

- [BaseModel](#) ()
Construct a new [BaseModel](#) object.
- virtual [~BaseModel](#) ()
Destroy the [BaseModel](#) object.
- const [BaseModel](#) [fit](#) (const [Features](#) &X, const [Target](#) &y)
Fit model.
- const [Target](#) [predict](#) (const [Features](#) &X) const
Predict target variable with fitted model.
- const bool [is_fitted](#) () const
Check if model is fitted.
- const std::string [get_name](#) () const
Get model's name.

Private Attributes

- [Weights](#) [weights_](#)
Row vector of model's weights.
- SolverType [solver_](#)
Solver to fit model with.

Additional Inherited Members

Protected Member Functions inherited from [BaseModel](#)

- void [mark_as_fitted](#) ()
Mark model as fitted.

Protected Attributes inherited from [BaseModel](#)

- std::string [name_](#)
Model's name (string).

5.8.1 Detailed Description

```
template<typename SolverType>
class LogRegModel< SolverType >
```

Logistic Regression model class template. Inherits from [BaseModel](#) class.

Template Parameters

<i>SolverType</i>	class of solver: BaseSolver , DerivativeSolver
-------------------	--

5.8.2 Constructor & Destructor Documentation

5.8.2.1 LogRegModel()

```
template<typename SolverType >
LogRegModel< SolverType >::LogRegModel (
    const SolverType & solver )
```

Construct a new [LogRegModel](#) object.

5.8.3 Member Function Documentation

5.8.3.1 get_weights()

```
template<typename SolverType >
const Weights LogRegModel< SolverType >::get_weights ( ) const
```

Get model's weights.

Returns

const [Types::Weights](#)

5.8.3.2 fit()

```
template<typename SolverType >
const LogRegModel LogRegModel< SolverType >::fit (
    Features & X,
    const Target & y )
```

Fit model.

Parameters

<i>X</i>	Matrix of feature variables
<i>y</i>	Column vector of target variable

Returns

[LogRegModel](#)

5.8.3.3 predict()

```
template<typename SolverType >
const Target LogRegModel< SolverType >::predict (
    const Features & X,
    const double & threshold = 0.5 ) const
```

Predict (classify) target variable's class with fitted model at given threshold.

Positive class is 1 and negative class is 0

Parameters

<i>X</i>	Matrix of feature variables
<i>threshold</i>	Threshold [0, 1] (double)

Returns

const [Types::Target](#)

5.8.3.4 predict_proba()

```
template<typename SolverType >
const Target LogRegModel< SolverType >::predict_proba (
    const Features & X ) const
```

Predict probability of positive class of target variable with fitted model.

Parameters

<i>X</i>	Matrix of feature variables
----------	-----------------------------

Returns

const [Types::Target](#)

5.8.4 Member Data Documentation

5.8.4.1 weights_

```
template<typename SolverType >
Weights LogRegModel< SolverType >::weights_ [private]
```

Row vector of model's weights.

5.8.4.2 solver_

```
template<typename SolverType >
SolverType LogRegModel< SolverType >::solver_ [private]
```

Solver to fit model with.

The documentation for this class was generated from the following file:

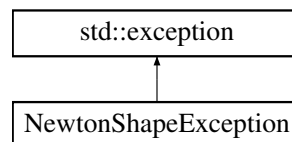
- [logreg_model.hpp](#)

5.9 NewtonShapeException Class Reference

[NewtonShapeException](#) class. Inherits from `std::exception` class.

```
#include <exceptions.hpp>
```

Inheritance diagram for `NewtonShapeException`:



Public Member Functions

- [NewtonShapeException](#) ()
Construct a new [NewtonShapeException](#) object.
- `const std::string` [what](#) ()
Return detailed description of exception.

5.9.1 Detailed Description

[NewtonShapeException](#) class. Inherits from `std::exception` class.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 NewtonShapeException()

```
NewtonShapeException::NewtonShapeException ( ) [inline]
```

Construct a new [NewtonShapeException](#) object.

5.9.3 Member Function Documentation

5.9.3.1 what()

```
const std::string NewtonShapeException::what ( ) [inline]
```

Return detailed description of exception.

Returns

const std::string

The documentation for this class was generated from the following file:

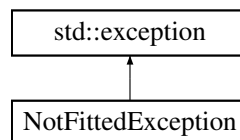
- [exceptions.hpp](#)

5.10 NotFittedException Class Reference

[NotFittedException](#) class. Inherits from `std::exception` class.

```
#include <exceptions.hpp>
```

Inheritance diagram for NotFittedException:



Public Member Functions

- [NotFittedException](#) (const std::string &obj_name)
Construct a new [NotFittedException](#) object.
- const std::string [what](#) ()
Return detailed description of exception.

Private Attributes

- std::string [obj_name_](#)
Object's name.

5.10.1 Detailed Description

[NotFittedException](#) class. Inherits from `std::exception` class.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 NotFittedException()

```
NotFittedException::NotFittedException (
    const std::string & obj_name ) [inline]
```

Construct a new [NotFittedException](#) object.

Parameters

<code>obj_name</code>	Object's name
-----------------------	---------------

5.10.3 Member Function Documentation

5.10.3.1 what()

```
const std::string NotFittedException::what ( ) [inline]
```

Return detailed description of exception.

Returns

const std::string

5.10.4 Member Data Documentation

5.10.4.1 obj_name_

```
std::string NotFittedException::obj_name_ [private]
```

Object's name.

The documentation for this class was generated from the following file:

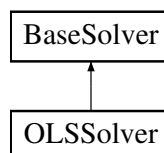
- [exceptions.hpp](#)

5.11 OLSSolver Class Reference

Ordinary Least Squares solver class. Inherits from [BaseSolver](#) class.

```
#include <ols_solver.hpp>
```

Inheritance diagram for OLSSolver:



Public Member Functions

- [OLSSolver](#) ()
Construct a new Ordinary Least Squares Solver object.
- const [Weights optimize](#) ([Weights](#) &w, const [Features](#) &X, const [Target](#) &y)
Return optimized weights using ordinary least squares closed-form formula.

Public Member Functions inherited from [BaseSolver](#)

- [BaseSolver](#) (const bool verbose=false)
Construct a new Base Solver object.
- virtual [~BaseSolver](#) ()
Destroy the Base Solver object.
- const [Weights](#) [optimize](#) ([Weights](#) &w, const [Features](#) &X, const [Target](#) &y)
Return optimized weights.
- const std::string [get_name](#) () const
Get solver's name.

Additional Inherited Members

Protected Attributes inherited from [BaseSolver](#)

- std::string [name_](#)
Solver's name (string).
- bool [verbose_](#)
Show solver's steps flag.

5.11.1 Detailed Description

Ordinary Least Squares solver class. Inherits from [BaseSolver](#) class.

Template Parameters

<i>SolverType</i>	Type (class) of solver
-------------------	------------------------

5.11.2 Constructor & Destructor Documentation

5.11.2.1 [OLSSolver](#)()

```
OLSSolver::OLSSolver ( )
```

Construct a new Ordinary Least Squares Solver object.

5.11.3 Member Function Documentation

5.11.3.1 [optimize](#)()

```
const Weights OLSSolver::optimize (
    Weights & w,
    const Features & X,
    const Target & y )
```

Return optimized weights using ordinary least squares closed-form formula.

Parameters

w	Row vector of weights – not used
X	Matrix of feature variables
y	Column vector of target variable

Returns

[Types::Weights](#)

The documentation for this class was generated from the following file:

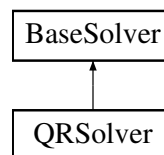
- [ols_solver.hpp](#)

5.12 QRSolver Class Reference

QR-decomposition solver class. Inherits from [BaseSolver](#) class.

```
#include <qr_solver.hpp>
```

Inheritance diagram for QRSolver:



Public Member Functions

- [QRSolver](#) ()
Construct a new QR Solver object.
- const [Weights](#) optimize ([Weights](#) &w, const [Features](#) &X, const [Target](#) &y)
Return optimized weights using QR-decomposition closed-form formula.

Public Member Functions inherited from [BaseSolver](#)

- [BaseSolver](#) (const bool verbose=false)
Construct a new Base Solver object.
- virtual [~BaseSolver](#) ()
Destroy the Base Solver object.
- const [Weights](#) optimize ([Weights](#) &w, const [Features](#) &X, const [Target](#) &y)
Return optimized weights.
- const std::string [get_name](#) () const
Get solver's name.

Additional Inherited Members

Protected Attributes inherited from [BaseSolver](#)

- `std::string` [name_](#)
Solver's name (string).
- `bool` [verbose_](#)
Show solver's steps flag.

5.12.1 Detailed Description

QR-decomposition solver class. Inherits from [BaseSolver](#) class.

Template Parameters

SolverType	Type (class) of solver
----------------------------	------------------------

5.12.2 Constructor & Destructor Documentation

5.12.2.1 [QRSolver\(\)](#)

```
QRSolver::QRSolver ( )
```

Construct a new QR Solver object.

https://en.wikipedia.org/wiki/QR_decomposition

5.12.3 Member Function Documentation

5.12.3.1 [optimize\(\)](#)

```
const Weights QRSolver::optimize (
    Weights & w,
    const Features & X,
    const Target & y )
```

Return optimized weights using QR-decomposition closed-form formula.

Parameters

<i>w</i>	Row vector of weights – not used
<i>X</i>	Matrix of feature variables
<i>y</i>	Column vector of target variable

Returns

[Types::Weights](#)

The documentation for this class was generated from the following file:

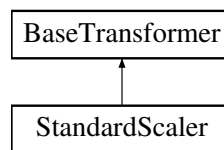
- [qr_solver.hpp](#)

5.13 StandardScaler Class Reference

Standard Scaler (z -score transformation) class. Inherits from [BaseTransformer](#) class.

```
#include <standard_scaler.hpp>
```

Inheritance diagram for StandardScaler:



Public Member Functions

- [StandardScaler](#) ()
Construct a new Standard Scaler object.
- [StandardScaler fit](#) ([Features](#) &X)
Fit scaler.
- const [Features transform](#) ([Features](#) &X)
Transform feature variables with fitted scaler.
- const [Features fit_transform](#) ([Features](#) &X)
Fit scaler, then transform feature variables.
- const [Features get_means](#) () const
Return matrix of learned means with shape of features.
- const [Features get_stddevs](#) () const
Return matrix of learned standard deviations with shape of features.

Public Member Functions inherited from [BaseTransformer](#)

- [BaseTransformer](#) ()
Construct a new Base Transformer object.
- virtual [~BaseTransformer](#) ()
Destroy the Base Transformer object.
- [BaseTransformer fit](#) ([Features](#) &X)
Fit transformer.
- const [Features transform](#) ([Features](#) &X)
Transform feature variables with fitted model.
- const [Features fit_transform](#) ([Features](#) &X)
Fit transformer, then transform feature variables.
- const bool [is_fitted](#) () const
Check if transformer is fitted.
- const std::string [get_name](#) () const
Get transformer's name.

Private Attributes

- [Features means_](#)
Learned means of features.
- [Features stddevs_](#)
Learned standard deviations of features.

Additional Inherited Members**Protected Member Functions inherited from [BaseTransformer](#)**

- void [mark_as_fitted_\(\)](#)
Mark transformer as fitted.

Protected Attributes inherited from [BaseTransformer](#)

- std::string [name_](#)
Transformer's name (string).

5.13.1 Detailed Description

Standard Scaler (z -score transformation) class. Inherits from [BaseTransformer](#) class.

Template Parameters

<i>SolverType</i>	Type (class) of solver
-------------------	------------------------

5.13.2 Constructor & Destructor Documentation**5.13.2.1 StandardScaler()**

```
StandardScaler::StandardScaler ( )
```

Construct a new Standard Scaler object.

5.13.3 Member Function Documentation**5.13.3.1 fit()**

```
StandardScaler StandardScaler::fit (
    Features & X )
```

Fit scaler.

Parameters

X	Matrix of feature variables
-----	-----------------------------

Returns

[StandardScaler](#)

5.13.3.2 transform()

```
const Features StandardScaler::transform (
    Features &  $X$  )
```

Transform feature variables with fitted scaler.

Parameters

X	Matrix of feature variables
-----	-----------------------------

Returns

const [Types::Features](#)

5.13.3.3 fit_transform()

```
const Features StandardScaler::fit_transform (
    Features &  $X$  )
```

Fit scaler, then transform feature variables.

Parameters

X	Matrix of feature variables
-----	-----------------------------

Returns

const [Types::Features](#)

5.13.3.4 get_means()

```
const Features StandardScaler::get_means ( ) const
```

Return matrix of learned means with shape of features.

Returns

const [Types::Features](#)

5.13.3.5 `get_stddevs()`

```
const Features StandardScaler::get_stddevs ( ) const
```

Return matrix of learned standard deviations with shape of features.

Returns

```
const Types::Features
```

5.13.4 Member Data Documentation

5.13.4.1 `means_`

```
Features StandardScaler::means_ [private]
```

Learned means of features.

5.13.4.2 `stddevs_`

```
Features StandardScaler::stddevs_ [private]
```

Learned standard deviations of features.

The documentation for this class was generated from the following file:

- [standard_scaler.hpp](#)

Chapter 6

File Documentation

6.1 autoreg_extractor.hpp File Reference

[AutoRegExtractor](#) class declarations.

```
#include "types.hpp"
```

Classes

- class [AutoRegExtractor](#)
Features and target extractor class for Autoregressive AR(p) model.

6.1.1 Detailed Description

[AutoRegExtractor](#) class declarations.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2024-04-25

Copyright

Copyright (c) 2024

6.2 autoreg_model.hpp File Reference

[AutoRegModel](#) class declarations.

```
#include "types.hpp"
#include "base_model.hpp"
```

Classes

- class [AutoRegModel](#)< [SolverType](#) >
Autoregressive AR(p) model class template. Inherits from [BaseModel](#) class.

6.2.1 Detailed Description

[AutoRegModel](#) class declarations.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2024-04-25

Copyright

Copyright (c) 2024

6.3 base_model.hpp File Reference

[BaseModel](#) class declarations.

```
#include "types.hpp"
```

Classes

- class [BaseModel](#)
Base Model class. All concrete model classes must inherit from this class.

6.3.1 Detailed Description

[BaseModel](#) class declarations.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2024-03-20

Copyright

Copyright (c) 2024

6.4 base_solver.hpp File Reference

[BaseSolver](#) class declarations.

```
#include <functional>
#include "types.hpp"
```

Classes

- class [BaseSolver](#)

Base Solver class. All concrete solver classes must inherit from this class.

6.4.1 Detailed Description

[BaseSolver](#) class declarations.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2024-03-20

Copyright

Copyright (c) 2024

6.5 base_transformer.hpp File Reference

[BaseTransformer](#) class declarations.

```
#include "types.hpp"
```

Classes

- class [BaseTransformer](#)

Base Transformer class. All concrete transformer classes must inherit from this class.

6.5.1 Detailed Description

[BaseTransformer](#) class declarations.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2024-03-20

Copyright

Copyright (c) 2024

6.6 derivative_solver.hpp File Reference

[DerivativeSolver](#) class declarations.

```
#include <functional>
#include "types.hpp"
#include "base_solver.hpp"
```

Classes

- class [DerivativeSolver](#)

Derivative Solver class. Inherits from [BaseSolver](#) class.

6.6.1 Detailed Description

[DerivativeSolver](#) class declarations.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2024-03-28

Copyright

Copyright (c) 2024

6.7 diff_loss_functions.hpp File Reference

Derivatives of loss functions declarations and implementation.

```
#include <functional>
#include <armadillo>
#include "types.hpp"
#include "predict_functions.hpp"
#include "exceptions.hpp"
```

Functions

- static const [Derivative](#) [DiffLoss::mean_squared_error_loss_grad](#) (const [Weights](#) &w, const [Features](#) &X, const [Target](#) &y_true)
Gradient (first derivative) of Mean Squared Error loss.
- static const [Derivative](#) [DiffLoss::mean_squared_error_loss_lapl](#) (const [Features](#) &X)
Laplacian (second derivative) of Mean Squared Error loss.
- static const [Derivative](#) [DiffLoss::mean_squared_error_loss_newton](#) (const [Weights](#) &w, const [Features](#) &X, const [Target](#) &y_true)
Gradient to Laplacian ratio (Newton) of Mean Squared Error loss.
- static const [Derivative](#) [DiffLoss::log_likelihood_loss_grad](#) (const [Weights](#) &w, const [Features](#) &X, const [Target](#) &y_true)
Gradient (first derivative) of Log Likelihood loss.
- static const [Derivative](#) [DiffLoss::log_likelihood_loss_lapl](#) (const [Weights](#) &w, const [Features](#) &X)
Laplacian (second derivative) of Log Likelihood loss.
- static const [Derivative](#) [DiffLoss::log_likelihood_loss_newton](#) (const [Weights](#) &w, const [Features](#) &X, const [Target](#) &y_true)
Gradient to Laplacian ratio (Newton) of Log Likelihood loss.

Variables

- static const std::function< [Derivative](#)(const [Weights](#) &, const [Features](#) &, const [Target](#) &)> [DiffLoss::MEAN_SQUARED_ERROR](#)
= [mean_squared_error_loss_grad](#)
Alias for [DiffLoss::mean_squared_error_loss_grad](#) function.
- static const std::function< [Derivative](#)(const [Weights](#) &, const [Features](#) &, const [Target](#) &)> [DiffLoss::MEAN_SQUARED_ERROR](#)
= [mean_squared_error_loss_newton](#)
Alias for [mean_squared_error_loss_newton](#) function.
- static const std::function< [Derivative](#)(const [Weights](#) &, const [Features](#) &, const [Target](#) &)> [DiffLoss::LOG_LIKELIHOOD_LOSS](#)
= [log_likelihood_loss_grad](#)
Alias for [log_likelihood_loss_grad](#) function.
- static const std::function< [Derivative](#)(const [Weights](#) &, const [Features](#) &, const [Target](#) &)> [DiffLoss::LOG_LIKELIHOOD_LOSS](#)
= [log_likelihood_loss_newton](#)
Alias for [log_likelihood_loss_newton](#) function.

6.7.1 Detailed Description

Derivatives of loss functions declarations and implementation.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2024-03-20

Copyright

Copyright (c) 2024

6.7.2 Function Documentation

6.7.2.1 [mean_squared_error_loss_grad\(\)](#)

```
static const Derivative DiffLoss::mean_squared_error_loss_grad (
    const Weights & w,
    const Features & X,
    const Target & y_true ) [static]
```

Gradient (first derivative) of Mean Squared Error loss.

$\nabla L_{MSE} = -\frac{2}{n} \sum_{i=1}^n X^T (y_i - \hat{y}_i)$, where X is the features matrix, y is the target vector, \hat{y} is the model's predictions vector, n is the number of predictions.

Parameters

w	Row vector of weights
X	Matrix of feature variables
y_true	Column vector of target variable

Returns

const [Types::Derivative](#)

6.7.2.2 mean_squared_error_loss_lapl()

```
static const Derivative DiffLoss::mean_squared_error_loss_lapl (
    const Features & X ) [static]
```

Laplacian (second derivative) of Mean Squared Error loss.

$\nabla(\nabla L_{MSE}) = \frac{2}{n} \sum X^T X$, where X is the features matrix, n is the number of observations.

Parameters

X	Matrix of feature variables
-----	-----------------------------

Returns

const [Types::Derivative](#)

6.7.2.3 mean_squared_error_loss_newton()

```
static const Derivative DiffLoss::mean_squared_error_loss_newton (
    const Weights & w,
    const Features & X,
    const Target & y_true ) [static]
```

Gradient to Laplacian ratio (Newton) of Mean Squared Error loss.

$$\text{Step} = \frac{\nabla L_{MSE}}{\nabla(\nabla L_{MSE})}$$

Parameters

w	Row vector of weights
X	Matrix of feature variables
y_true	Column vector of target variable

Returns

const [Types::Derivative](#)

6.7.2.4 log_likelihood_loss_grad()

```
static const Derivative DiffLoss::log_likelihood_loss_grad (
    const Weights & w,
    const Features & X,
    const Target & y_true ) [static]
```

Gradient (first derivative) of Log Likelihood loss.

$\nabla L_{LOG} = -\frac{1}{n} \sum_{i=1}^n X^T (y_i - \hat{y}_i)$, where X is the features matrix, y is the target vector, \hat{y} is the model's predictions vector, n is the number of predictions.

Parameters

w	Row vector of weights
X	Matrix of feature variables
y_true	Column vector of target variable

Returns

const [Types::Derivative](#)

6.7.2.5 log_likelihood_loss_lapl()

```
static const Derivative DiffLoss::log_likelihood_loss_lapl (
    const Weights & w,
    const Features & X ) [static]
```

Laplacian (second derivative) of Log Likelihood loss.

$\nabla(\nabla L_{LOG}) = \frac{1}{n} \sum_{i=1}^n X^T \hat{y}_i (1 - \hat{y}_i)^T$, where X is the features matrix, \hat{y} is the model's predictions vector, n is the number of predictions.

Parameters

w	Row vector of weights
X	Matrix of feature variables

Returns

const [Types::Derivative](#)

6.7.2.6 log_likelihood_loss_newton()

```
static const Derivative DiffLoss::log_likelihood_loss_newton (
    const Weights & w,
    const Features & X,
    const Target & y_true ) [static]
```

Gradient to Laplacian ratio (Newton) of Log Likelihood loss.

$$\text{Step} = \frac{\nabla L_{LOG}}{\nabla(\nabla L_{LOG})}$$

Parameters

<i>w</i>	Row vector of weights
<i>X</i>	Matrix of feature variables
<i>y_true</i>	Column vector of target variable

Returns

const [Types::Derivative](#)

6.7.3 Variable Documentation

6.7.3.1 MEAN_SQUARED_ERROR_LOSS_GRAD

```
const std::function<Derivative(const Weights&, const Features&, const Target&)> DiffLoss::↔
MEAN_SQUARED_ERROR_LOSS_GRAD = mean_squared_error_loss_grad [static]
```

Alias for [DiffLoss::mean_squared_error_loss_grad](#) function.

6.7.3.2 MEAN_SQUARED_ERROR_LOSS_NEWTON

```
const std::function<Derivative(const Weights&, const Features&, const Target&)> DiffLoss::↔
MEAN_SQUARED_ERROR_LOSS_NEWTON = mean_squared_error_loss_newton [static]
```

Alias for [mean_squared_error_loss_newton](#) function.

6.7.3.3 LOG_LIKELIHOOD_LOSS_GRAD

```
const std::function<Derivative(const Weights&, const Features&, const Target&)> DiffLoss::↔
LOG_LIKELIHOOD_LOSS_GRAD = log_likelihood_loss_grad [static]
```

Alias for [log_likelihood_loss_grad](#) function.

6.7.3.4 LOG_LIKELIHOOD_LOSS_NEWTON

```
const std::function<Derivative(const Weights&, const Features&, const Target&)> DiffLoss::↔
LOG_LIKELIHOOD_LOSS_NEWTON = log_likelihood_loss_newton [static]
```

Alias for [log_likelihood_loss_newton](#) function.

6.8 exceptions.hpp File Reference

Custom exceptions declarations and implementation.

```
#include <string>
```

Classes

- class [NotFittedException](#)
[NotFittedException](#) class. Inherits from `std::exception` class.
- class [NewtonShapeException](#)
[NewtonShapeException](#) class. Inherits from `std::exception` class.

6.8.1 Detailed Description

Custom exceptions declarations and implementation.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2024-03-20

Copyright

Copyright (c) 2024

6.9 linreg_model.hpp File Reference

[LinRegModel](#) class declarations.

```
#include "types.hpp"  
#include "base_model.hpp"
```

Classes

- class [LinRegModel< SolverType >](#)
Linear Regression model class template. Inherits from [BaseModel](#) class.

6.9.1 Detailed Description

[LinRegModel](#) class declarations.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2024-03-20

Copyright

Copyright (c) 2024

6.10 logreg_model.hpp File Reference

[LogRegModel](#) class declarations.

```
#include "types.hpp"
#include "base_model.hpp"
#include "base_solver.hpp"
#include "derivative_solver.hpp"
```

Classes

- class [LogRegModel](#)< [SolverType](#) >
Logistic Regression model class template. Inherits from [BaseModel](#) class.

6.10.1 Detailed Description

[LogRegModel](#) class declarations.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2024-03-27

Copyright

Copyright (c) 2024

6.11 metrics.hpp File Reference

Model metrics declarations and implementation.

```
#include <armadillo>
#include "types.hpp"
#include "predict_functions.hpp"
```

Functions

- static const double [Metrics::mse](#) (const [Target](#) &y_true, const [Target](#) &y_pred)
Compute Mean Squared Error of predictions.
- static const double [Metrics::sse](#) (const [Target](#) &y_true, const [Target](#) &y_pred)
Compute Sum Squared Error of predictions. In fact, $nMSE(y, \hat{y})$.
- static const double [Metrics::sst](#) (const [Target](#) &y_true)
Compute Sum Squared Total variance of target. In fact, $nVar(y)$.
- static const double [Metrics::r2](#) (const [Target](#) &y_true, const [Target](#) &y_pred)
Coefficient of determination of predictions.
- static const double [Metrics::accuracy](#) (const [Target](#) &y_true, const [Target](#) &y_pred)
Normalized accuracy score of predictions for binary classifier.
- static const size_t [Metrics::tp_count](#) (const [Target](#) &y_true, const [Target](#) &y_pred)
True Positives count.
- static const size_t [Metrics::fp_count](#) (const [Target](#) &y_true, const [Target](#) &y_pred)
False Positives count.
- static const size_t [Metrics::tn_count](#) (const [Target](#) &y_true, const [Target](#) &y_pred)
True Negatives count.
- static const size_t [Metrics::fn_count](#) (const [Target](#) &y_true, const [Target](#) &y_pred)
False Negatives count.
- static const [ConfusionMatrix](#) [Metrics::confusion_matrix](#) (const [Target](#) &y_true, const [Target](#) &y_pred)
Return confusion matrix.
- static const double [Metrics::precision](#) (const [Target](#) &y_true, const [Target](#) &y_pred)
Precision (Positive class).
- static const double [Metrics::recall](#) (const [Target](#) &y_true, const [Target](#) &y_pred)
Recall (Positive class). Same as True Positive Rate. Same as Sensitivity.
- static const double [Metrics::fpr](#) (const [Target](#) &y_true, const [Target](#) &y_pred)
False Positive Rate. Same as Fall-out. Same as probability of Type I error (falsely rejecting correct null hypothesis).
- static const double [Metrics::f1_score](#) (const [Target](#) &y_true, const [Target](#) &y_pred)
F1-score (Positive class).
- static const [PRCurve](#) [Metrics::pr_curve](#) (const [Target](#) &y_true, const [Target](#) &y_pred_proba, const size_t num=101)
Compute Precision-Recall curve: [Metrics::precision](#) vs [Metrics::recall](#) for different classification thresholds.
- static const [ROCCurve](#) [Metrics::roc_curve](#) (const [Target](#) &y_true, const [Target](#) &y_pred_proba, const size_t num=101)
Compute Receiver Operating Characteristic (ROC) curve: [Metrics::recall](#) vs [Metrics::fpr](#) for different classification thresholds.
- static const [TPs](#) [Metrics::tp_curve](#) (const [Target](#) &y_true, const [Target](#) &y_pred_proba, const size_t num=101)
Compute True Positives curve: [Metrics::tp_count](#) vs classification threshold.
- static const [FPs](#) [Metrics::fp_curve](#) (const [Target](#) &y_true, const [Target](#) &y_pred_proba, const size_t num=101)
Compute False Positives curve: [Metrics::fp_count](#) vs classification threshold.

- static const [TNs](#) [Metrics::tn_curve](#) (const [Target](#) &y_true, const [Target](#) &y_pred_proba, const size_t num=101)
Compute True Negatives curve: [Metrics::tn_count](#) vs classification threshold.
- static const [FNs](#) [Metrics::fn_curve](#) (const [Target](#) &y_true, const [Target](#) &y_pred_proba, const size_t num=101)
Compute False Negatives curve: [Metrics::fn_count](#) vs classification threshold.
- static const double [Metrics::auc](#) (const std::pair< const arma::drowvec, const arma::drowvec > &pair)
Area Under Curve for [Metrics::pr_curve](#) or [Metrics::roc_curve](#).

6.11.1 Detailed Description

Model metrics declarations and implementation.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2024-03-20

Copyright

Copyright (c) 2024

6.11.2 Function Documentation

6.11.2.1 mse()

```
static const double Metrics::mse (
    const Target & y_true,
    const Target & y_pred ) [static]
```

Compute Mean Squared Error of predictions.

$MSE(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$, where y is the target vector, \hat{y} is model's predictions vector, n is the number of predictions.

Parameters

y_true	Column vector of ground truth target
y_pred	Column vector of predicted target

Returns

const double

6.11.2.2 sse()

```
static const double Metrics::sse (
    const Target & y_true,
    const Target & y_pred ) [static]
```

Compute Sum Squared Error of predictions. In fact, $nMSE(y, \hat{y})$.

$SSE(y, \hat{y}) = \sum_{i=1}^n (y_i - \hat{y}_i)^2$, where y is the target vector, \hat{y} is model's predictions vector, n is the number of predictions.

Parameters

<code>y_true</code>	Column vector of ground truth target
<code>y_pred</code>	Column vector of predicted target

Returns

const double

6.11.2.3 sst()

```
static const double Metrics::sst (
    const Target & y_true ) [static]
```

Compute Sum Squared Total variance of target. In fact, $nVar(y)$.

$SST(y) = \sum_{i=1}^n (y_i - \bar{y})^2$, where y is the target vector, \bar{y} is target vector's mean, n is the number of predictions.

Parameters

<code>y_true</code>	Column vector of ground truth target
---------------------	--------------------------------------

Returns

const double

6.11.2.4 r2()

```
static const double Metrics::r2 (
    const Target & y_true,
    const Target & y_pred ) [static]
```


Coefficient of determination of predictions.

$R^2 = 1 - \frac{SSE(y, \hat{y})}{SST(y)}$. Can also be expressed as $R^2 = 1 - \frac{nMSE(y, \hat{y})}{nVar(y)} = 1 - \frac{MSE(y, \hat{y})}{Var(y)}$, where SSE is [Metrics::sse](#), SST is [Metrics::sst](#), MSE is [Metrics::mse](#).

Parameters

<code>y_true</code>	Column vector of ground truth target
<code>y_pred</code>	Column vector of predicted target

Returns

const double

6.11.2.5 accuracy()

```
static const double Metrics::accuracy (
    const Target & y_true,
    const Target & y_pred ) [static]
```

Normalized accuracy score of predictions for binary classifier.

$Acc = \frac{1}{n} \sum_{i=1}^n [\hat{y} = y]$, where y is the target vector, \hat{y} is the model's predictions vector, n is the number of predictions.

Parameters

<code>y_true</code>	Column vector of ground truth target
<code>y_pred</code>	Column vector of predicted target

Returns

const double

6.11.2.6 tp_count()

```
static const size_t Metrics::tp_count (
    const Target & y_true,
    const Target & y_pred ) [static]
```

True Positives count.

$TP = |\{y|y = 1\} \cap \{\hat{y}|\hat{y} = 1\}|$, where y is the target vector, \hat{y} is the model's predictions vector.

Parameters

<code>y_true</code>	Column vector of ground truth target
<code>y_pred</code>	Column vector of predicted target

Returns

const size_t

6.11.2.7 fp_count()

```
static const size_t Metrics::fp_count (
    const Target & y_true,
    const Target & y_pred ) [static]
```

False Positives count.

$FP = |\{y|y = 0\} \cap \{\hat{y}|\hat{y} = 1\}|$, where y is the target vector, \hat{y} is the model's predictions vector.

Parameters

<i>y_true</i>	Column vector of ground truth target
<i>y_pred</i>	Column vector of predicted target

Returns

const size_t

6.11.2.8 tn_count()

```
static const size_t Metrics::tn_count (
    const Target & y_true,
    const Target & y_pred ) [static]
```

True Negatives count.

$TN = |\{y|y = 0\} \cap \{\hat{y}|\hat{y} = 0\}|$, where y is the target vector, \hat{y} is the model's predictions vector.

Parameters

<i>y_true</i>	Column vector of ground truth target
<i>y_pred</i>	Column vector of predicted target

Returns

const size_t

6.11.2.9 fn_count()

```
static const size_t Metrics::fn_count (
    const Target & y_true,
    const Target & y_pred ) [static]
```

False Negatives count.

$FN = |\{y|y = 1\} \cap \{\hat{y}|\hat{y} = 0\}|$, where y is the target vector, \hat{y} is the model's predictions vector.

Parameters

<i>y_true</i>	Column vector of ground truth target
<i>y_pred</i>	Column vector of predicted target

Returns

const size_t

6.11.2.10 confusion_matrix()

```
static const ConfusionMatrix Metrics::confusion_matrix (
    const Target & y_true,
    const Target & y_pred ) [static]
```

Return confusion matrix.

	$y = 1$	$y = 0$
$\hat{y} = 1$	Metrics::tp_count	Metrics::fp_count
$\hat{y} = 0$	Metrics::fn_count	Metrics::tn_count

Parameters

<i>y_true</i>	Column vector of ground truth target
<i>y_pred</i>	Column vector of predicted target

Returns

const Types::ConfusionMatrix

6.11.2.11 precision()

```
static const double Metrics::precision (
    const Target & y_true,
    const Target & y_pred ) [static]
```

Precision (Positive class).

$Precision = P(y = 1 | \hat{y} = 1) = \frac{P(y = 1 \cap \hat{y} = 1)}{P(\hat{y} = 1)} = \frac{TP}{TP + FP}$, where y is the target vector, \hat{y} is the model's predictions vector.

Parameters

<i>y_true</i>	Column vector of ground truth target
<i>y_pred</i>	Column vector of predicted target

Returns

const double

6.11.2.12 recall()

```
static const double Metrics::recall (
    const Target & y_true,
    const Target & y_pred ) [static]
```

Recall (Positive class). Same as True Positive Rate. Same as Sensitivity.

$Recall = P(y = 1 | \hat{y} = 1) = \frac{P(y = 1 \cap \hat{y} = 1)}{P(\hat{y} = 1)} = \frac{TP}{TP + FN}$, where y is the target vector, \hat{y} is the model's predictions vector.

Parameters

<i>y_true</i>	Column vector of ground truth target
<i>y_pred</i>	Column vector of predicted target

Returns

const double

6.11.2.13 fpr()

```
static const double Metrics::fpr (
    const Target & y_true,
    const Target & y_pred ) [static]
```

False Positive Rate. Same as Fall-out. Same as probability of Type I error (falsely rejecting correct null hypothesis).

$Fallout = P(y = 0 | \hat{y} = 1) = \frac{P(y = 0 \cap \hat{y} = 1)}{P(\hat{y} = 1)} = \frac{FP}{FP + TN}$, where y is the target vector, \hat{y} is the model's predictions vector.

Parameters

<i>y_true</i>	Column vector of ground truth target
<i>y_pred</i>	Column vector of predicted target

Returns

const double

6.11.2.14 f1_score()

```
static const double Metrics::f1_score (
    const Target & y_true,
    const Target & y_pred ) [static]
```

F1-score (Positive class).

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Parameters

<i>y_true</i>	Column vector of ground truth target
<i>y_pred</i>	Column vector of predicted target

Returns

const double

6.11.2.15 pr_curve()

```
static const PRCurve Metrics::pr_curve (
    const Target & y_true,
    const Target & y_pred_proba,
    const size_t num = 101 ) [static]
```

Compute Precision-Recall curve: [Metrics::precision](#) vs [Metrics::recall](#) for different classification thresholds.

Parameters

<i>y_true</i>	Column vector of ground truth target
<i>y_pred_proba</i>	Column vector of predicted probabilities of positive class
<i>num</i>	Number of thresholds

Returns

const Types::PrecisionsRecalls

6.11.2.16 roc_curve()

```
static const ROCCurve Metrics::roc_curve (
    const Target & y_true,
    const Target & y_pred_proba,
    const size_t num = 101 ) [static]
```

Compute Receiver Operating Characteristic (ROC) curve: [Metrics::recall](#) vs [Metrics::fpr](#) for different classification thresholds.

Parameters

<i>y_true</i>	Column vector of ground truth target
<i>y_pred_proba</i>	Column vector of predicted probabilities of positive class
<i>num</i>	Number of thresholds

Returns

const [Types::PrecisionsRecalls](#)

6.11.2.17 tp_curve()

```
static const TPs Metrics::tp_curve (
    const Target & y_true,
    const Target & y_pred_proba,
    const size_t num = 101 ) [static]
```

Compute True Positives curve: [Metrics::tp_count](#) vs classification threshold.

Parameters

<i>y_true</i>	Column vector of ground truth target
<i>y_pred_proba</i>	Column vector of predicted probabilities of positive class
<i>num</i>	Number of thresholds

Returns

const [Types::TPs](#)

6.11.2.18 fp_curve()

```
static const FPs Metrics::fp_curve (
    const Target & y_true,
    const Target & y_pred_proba,
    const size_t num = 101 ) [static]
```

Compute False Positives curve: [Metrics::fp_count](#) vs classification threshold.

Parameters

<i>y_true</i>	Column vector of ground truth target
<i>y_pred_proba</i>	Column vector of predicted probabilities of positive class
<i>num</i>	Number of thresholds

Returns

const [Types::FPs](#)

6.11.2.19 tn_curve()

```
static const TNs Metrics::tn_curve (
    const Target & y_true,
    const Target & y_pred_proba,
    const size_t num = 101 ) [static]
```

Compute True Negatives curve: [Metrics::tn_count](#) vs classification threshold.

Parameters

<i>y_true</i>	Column vector of ground truth target
<i>y_pred_proba</i>	Column vector of predicted probabilities of positive class
<i>num</i>	Number of thresholds

Returns

const [Types::TNs](#)

6.11.2.20 fn_curve()

```
static const FNs Metrics::fn_curve (
    const Target & y_true,
    const Target & y_pred_proba,
    const size_t num = 101 ) [static]
```

Compute False Negatives curve: [Metrics::fn_count](#) vs classification threshold.

Parameters

<i>y_true</i>	Column vector of ground truth target
<i>y_pred_proba</i>	Column vector of predicted probabilities of positive class
<i>num</i>	Number of thresholds

Returns

const [Types::FNs](#)

6.11.2.21 auc()

```
static const double Metrics::auc (
    const std::pair< const arma::drowvec, const arma::drowvec > & pair ) [static]
```

Area Under Curve for [Metrics::pr_curve](#) or [Metrics::roc_curve](#).

$$AUC_{PR} = \int_0^1 Precision(Recall) dRecall$$

$$AUC_{ROC} = \int_0^1 Recall(Fallout) dFallout$$

Parameters

<i>pair</i>	Metrics::PRCurve or Metrics::ROCCurve (arma::drowvec)
-------------	---

Returns

const double

6.12 `ols_solver.hpp` File Reference

[OLSSolver](#) class declarations.

```
#include "types.hpp"
#include "base_solver.hpp"
```

Classes

- class [OLSSolver](#)

Ordinary Least Squares solver class. Inherits from [BaseSolver](#) class.

6.12.1 Detailed Description

[OLSSolver](#) class declarations.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2024-03-20

Copyright

Copyright (c) 2024

6.13 `predict_functions.hpp` File Reference

Prediction functions declarations and implementation.

```
#include <armadillo>
#include "types.hpp"
```


Functions

- static const [Target Predict::linreg](#) (const [Features](#) &X, const [Weights](#) &w)
Predict function for linear regression.
- static const [Target Predict::logistic_function](#) (const [Target](#) &z)
Logistic function.
- static const [Target Predict::logreg_proba](#) (const [Features](#) &X, const [Weights](#) &w)
Predict probability of positive class for logistic regression.
- static const [Target Predict::logreg_class](#) (const [Target](#) &y_pred_proba, const double threshold)
Predict class for logistic regression. Positive class = 1. Negative class = 0.
- static const [Weights Predict::ols](#) (const [Features](#) &X, const [Target](#) &y_true)
Compute weights for Ordinary Least Squares.
- static const [Weights Predict::qr](#) (const [Features](#) &X, const [Target](#) &y_true)
Compute weights for QR-decomposition.

6.13.1 Detailed Description

Prediction functions declarations and implementation.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2024-03-20

Copyright

Copyright (c) 2024

6.13.2 Function Documentation

6.13.2.1 linreg()

```
static const Target Predict::linreg (
    const Features & X,
    const Weights & w ) [static]
```

Predict function for linear regression.

$$\hat{y} = w_0x_0 + w_1x_1 + \dots + w_nx_n = wX,$$

where X is the features matrix, w is the model's weights vector.

Parameters

X	Matrix of feature variables
w	Row vector of weights

Returns

const [Types::Target](#)

6.13.2.2 logistic_function()

```
static const Target Predict::logistic_function (
    const Target & z ) [static]
```

Logistic function.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Parameters

z	Column vector of target variable
-----	----------------------------------

Returns

const [Types::Target](#)

6.13.2.3 logreg_proba()

```
static const Target Predict::logreg_proba (
    const Features & X,
    const Weights & w ) [static]
```

Predict probability of positive class for logistic regression.

$$\hat{y}_{proba} = \sigma(-wX),$$

where $\sigma(x)$ is the logistic function X is the features matrix, w is the model's weights vector.

Parameters

X	Matrix of feature variables
w	Row vector of weights

Returns

const [Types::Target](#)

6.13.2.4 logreg_class()

```
static const Target Predict::logreg_class (
    const Target & y_pred_proba,
    const double threshold ) [static]
```

Predict class for logistic regression. Positive class = 1. Negative class = 0.

$$\hat{y}(\hat{y}_{proba}, t) = \begin{cases} 1, & \text{if } \hat{y}_{proba} \geq t \\ 0, & \text{if } \hat{y}_{proba} < t \end{cases}$$

where \hat{y}_{proba} is the predicted probability of positive class, t is the decision threshold.

Parameters

<i>y_pred_proba</i>	Predicted probability of positive class
<i>threshold</i>	Decision threshold

Returns

const [Types::Target](#)

6.13.2.5 ols()

```
static const Weights Predict::ols (
    const Features & X,
    const Target & y_true ) [static]
```

Compute weights for Ordinary Least Squares.

$$w = (X^T X)^{-1} X y,$$

where X is the features matrix, y is the target vector.

Note: X must be full rank, i.e. features must be linearly independent Otherwise, its inverse is undefined, and thus solution is also undefined

Parameters

<i>X</i>	Matrix of feature variables
<i>y_true</i>	Column vector of target variable

Returns

const [Types::Weights](#)

6.13.2.6 qr()

```
static const Weights Predict::qr (
    const Features & X,
    const Target & y_true ) [static]
```

Compute weights for QR-decomposition.

$$X = QR,$$

$$w = R^{-1}(Q^T y),$$

where X is the features matrix, y is the target vector.

Parameters

X	Matrix of feature variables
y_true	Column vector of target variable

Returns

const [Types::Weights](#)

6.14 qr_solver.hpp File Reference

[QRSolver](#) class declarations.

```
#include "types.hpp"
#include "base_solver.hpp"
```

Classes

- class [QRSolver](#)
QR-decomposition solver class. Inherits from [BaseSolver](#) class.

6.14.1 Detailed Description

[QRSolver](#) class declarations.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2024-03-20

Copyright

Copyright (c) 2024

6.15 standard_scaler.hpp File Reference

[StandardScaler](#) class declarations.

```
#include "types.hpp"
#include "base_transformer.hpp"
```

Classes

- class [StandardScaler](#)
Standard Scaler (z -score transformation) class. Inherits from [BaseTransformer](#) class.

6.15.1 Detailed Description

[StandardScaler](#) class declarations.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2024-03-20

Copyright

Copyright (c) 2024

6.16 types.hpp File Reference

Custom types used in the library.

```
#include <boost/type_index.hpp>
#include <armadillo>
```

Typedefs

- using `Types::Features` = `arma::dmat`
- using `Types::Target` = `arma::dvec`
- using `Types::Weights` = `arma::drowvec`
- using `Types::Derivative` = `arma::drowvec`
- using `Types::ConfusionMatrix` = `arma::umat`
- using `Types::Precisions` = `arma::drowvec`
- using `Types::Recalls` = `arma::drowvec`
- using `Types::Fallouts` = `arma::drowvec`
- using `Types::PRCurve` = `std::pair< const Precisions, const Recalls >`
Pair of row vectors of precisions and recalls – PR curve.
- using `Types::ROCCurve` = `std::pair< const Recalls, const Fallouts >`
Pair of row vectors of recalls and fall-outs – ROC curve.
- using `Types::TPs` = `arma::urowvec`
- using `Types::FPs` = `arma::urowvec`
- using `Types::TNs` = `arma::urowvec`
- using `Types::FNs` = `arma::urowvec`

Functions

- `template<typename T>`
`std::string Types::get_name (T)`
Get the object's name as string representation of its type.

6.16.1 Detailed Description

Custom types used in the library.

Author

Andrei Batyrov (arbatyrov@edu.hse.ru)

Version

0.1

Date

2024-03-26

Copyright

Copyright (c) 2024

6.16.2 Typedef Documentation

6.16.2.1 Features

using `Types::Features` = `typedef arma::dmat`

Matrix of feature variables (doubles).

6.16.2.2 Target

```
using Types::Target = typedef arma::dvec
```

Column vector of target variable (doubles).

6.16.2.3 Weights

```
using Types::Weights = typedef arma::drowvec
```

Row vector of model's weights (doubles).

6.16.2.4 Derivative

```
using Types::Derivative = typedef arma::drowvec
```

Row vector of n-th order derivative of loss function (doubles).

6.16.2.5 ConfusionMatrix

```
using Types::ConfusionMatrix = typedef arma::umat
```

Confusion Matrix (doubles).

6.16.2.6 Precisions

```
using Types::Precisions = typedef arma::drowvec
```

Row vector of precisions computed for different thresholds (doubles).

6.16.2.7 Recalls

```
using Types::Recalls = typedef arma::drowvec
```

Row vector of recalls computed for different thresholds (doubles).

6.16.2.8 Fallouts

```
using Types::Fallouts = typedef arma::drowvec
```

Row vector of fall-outs computed for different thresholds (doubles).

6.16.2.9 PRCurve

```
using Types::PRCurve = typedef std::pair<const Precisions, const Recalls>
```

Pair of row vectors of precisions and recalls – PR curve.

6.16.2.10 ROCCurve

```
using Types::ROCCurve = typedef std::pair<const Recalls, const Fallouts>
```

Pair of row vectors of recalls and fall-outs – ROC curve.

6.16.2.11 TPs

```
using Types::TPs = typedef arma::urowvec
```

Row vector of true positives computed for different thresholds (unsigned int).

6.16.2.12 FPs

```
using Types::FPs = typedef arma::urowvec
```

Row vector of false positives computed for different thresholds (unsigned int).

6.16.2.13 TNs

```
using Types::TNs = typedef arma::urowvec
```

Row vector of true negatives computed for different thresholds (unsigned int).

6.16.2.14 FNs

```
using Types::FNs = typedef arma::urowvec
```

Row vector of false negatives computed for different thresholds (unsigned int).

6.16.3 Function Documentation

6.16.3.1 get_name()

```
template<typename T >
std::string Types::get_name (
    T )
```

Get the object's name as string representation of its type.

Template Parameters

<i>T</i>	
----------	--

Returns

`std::string`

Index

- ~BaseModel
 - BaseModel, [16](#)
- ~BaseSolver
 - BaseSolver, [19](#)
- ~BaseTransformer
 - BaseTransformer, [21](#)
- accuracy
 - metrics.hpp, [57](#)
- auc
 - metrics.hpp, [63](#)
- autoreg_extractor.hpp, [43](#)
- autoreg_model.hpp, [44](#)
- AutoRegExtractor, [9](#)
 - AutoRegExtractor, [9](#)
 - extract_X, [10](#)
 - extract_y, [10](#)
 - get_name, [10](#)
 - name_, [10](#)
 - p_, [10](#)
- AutoRegModel
 - AutoRegModel< SolverType >, [12](#)
- AutoRegModel< SolverType >, [11](#)
 - AutoRegModel, [12](#)
 - fit, [13](#)
 - get_order, [13](#)
 - get_sigma, [13](#)
 - get_weights, [13](#)
 - p_, [14](#)
 - predict, [14](#)
 - sigma_, [14](#)
 - solver_, [14](#)
 - weights_, [14](#)
- base_model.hpp, [44](#)
- base_solver.hpp, [45](#)
- base_transformer.hpp, [46](#)
- BaseModel, [15](#)
 - ~BaseModel, [16](#)
 - BaseModel, [16](#)
 - fit, [16](#)
 - fitted_, [17](#)
 - get_name, [17](#)
 - is_fitted, [17](#)
 - mark_as_fitted_, [17](#)
 - name_, [17](#)
 - predict, [16](#)
 - y_mean_, [18](#)
- BaseSolver, [18](#)
 - ~BaseSolver, [19](#)
- BaseSolver, [19](#)
 - get_name, [19](#)
 - name_, [20](#)
 - optimize, [19](#)
 - verbose_, [20](#)
- BaseTransformer, [20](#)
 - ~BaseTransformer, [21](#)
 - BaseTransformer, [21](#)
 - fit, [21](#)
 - fit_transform, [22](#)
 - fitted_, [23](#)
 - get_name, [22](#)
 - is_fitted, [22](#)
 - mark_as_fitted_, [22](#)
 - name_, [23](#)
 - transform, [21](#)
- compute_derivative
 - DerivativeSolver, [25](#)
- confusion_matrix
 - metrics.hpp, [59](#)
- ConfusionMatrix
 - types.hpp, [71](#)
- Derivative
 - types.hpp, [71](#)
- derivative_solver.hpp, [46](#)
- DerivativeSolver, [23](#)
 - compute_derivative, [25](#)
 - DerivativeSolver, [24](#)
 - diff_loss_func_, [26](#)
 - learning_rate_, [26](#)
 - max_iter_, [26](#)
 - min_derivative_size_, [26](#)
 - optimize, [25](#)
- diff_loss_func_
 - DerivativeSolver, [26](#)
- diff_loss_functions.hpp, [47](#)
 - LOG_LIKELIHOOD_LOSS_GRAD, [51](#)
 - log_likelihood_loss_grad, [50](#)
 - log_likelihood_loss_lapl, [50](#)
 - LOG_LIKELIHOOD_LOSS_NEWTON, [51](#)
 - log_likelihood_loss_newton, [50](#)
 - MEAN_SQUARED_ERROR_LOSS_GRAD, [51](#)
 - mean_squared_error_loss_grad, [48](#)
 - mean_squared_error_loss_lapl, [49](#)
 - MEAN_SQUARED_ERROR_LOSS_NEWTON, [51](#)
 - mean_squared_error_loss_newton, [49](#)
- Documentation, [1](#)

- exceptions.hpp, 52
- extract_X
 - AutoRegExtractor, 10
- extract_y
 - AutoRegExtractor, 10
- f1_score
 - metrics.hpp, 60
- Fallouts
 - types.hpp, 71
- Features
 - types.hpp, 70
- fit
 - AutoRegModel< SolverType >, 13
 - BaseModel, 16
 - BaseTransformer, 21
 - LinRegModel< SolverType >, 28
 - LogRegModel< SolverType >, 31
 - StandardScaler, 40
- fit_transform
 - BaseTransformer, 22
 - StandardScaler, 41
- fitted_
 - BaseModel, 17
 - BaseTransformer, 23
- fn_count
 - metrics.hpp, 58
- fn_curve
 - metrics.hpp, 63
- FNs
 - types.hpp, 72
- fp_count
 - metrics.hpp, 58
- fp_curve
 - metrics.hpp, 62
- fpr
 - metrics.hpp, 60
- FPs
 - types.hpp, 72
- get_means
 - StandardScaler, 41
- get_name
 - AutoRegExtractor, 10
 - BaseModel, 17
 - BaseSolver, 19
 - BaseTransformer, 22
 - types.hpp, 72
- get_order
 - AutoRegModel< SolverType >, 13
- get_sigma
 - AutoRegModel< SolverType >, 13
- get_stddevs
 - StandardScaler, 41
- get_weights
 - AutoRegModel< SolverType >, 13
 - LinRegModel< SolverType >, 28
 - LogRegModel< SolverType >, 31
- is_fitted
 - BaseModel, 17
 - BaseTransformer, 22
- learning_rate_
 - DerivativeSolver, 26
- linreg
 - predict_functions.hpp, 65
- linreg_model.hpp, 52
- LinRegModel
 - LinRegModel< SolverType >, 28
- LinRegModel< SolverType >, 26
 - fit, 28
 - get_weights, 28
 - LinRegModel, 28
 - predict, 28
 - solver_, 29
 - weights_, 29
- LOG_LIKELIHOOD_LOSS_GRAD
 - diff_loss_functions.hpp, 51
- log_likelihood_loss_grad
 - diff_loss_functions.hpp, 50
- log_likelihood_loss_lapl
 - diff_loss_functions.hpp, 50
- LOG_LIKELIHOOD_LOSS_NEWTON
 - diff_loss_functions.hpp, 51
- log_likelihood_loss_newton
 - diff_loss_functions.hpp, 50
- logistic_function
 - predict_functions.hpp, 66
- logreg_class
 - predict_functions.hpp, 66
- logreg_model.hpp, 53
- logreg_proba
 - predict_functions.hpp, 66
- LogRegModel
 - LogRegModel< SolverType >, 31
- LogRegModel< SolverType >, 29
 - fit, 31
 - get_weights, 31
 - LogRegModel, 31
 - predict, 31
 - predict_proba, 32
 - solver_, 32
 - weights_, 32
- mark_as_fitted_
 - BaseModel, 17
 - BaseTransformer, 22
- max_iter_
 - DerivativeSolver, 26
- MEAN_SQUARED_ERROR_LOSS_GRAD
 - diff_loss_functions.hpp, 51
- mean_squared_error_loss_grad
 - diff_loss_functions.hpp, 48
- mean_squared_error_loss_lapl
 - diff_loss_functions.hpp, 49
- MEAN_SQUARED_ERROR_LOSS_NEWTON
 - diff_loss_functions.hpp, 51

- mean_squared_error_loss_newton
 - diff_loss_functions.hpp, 49
- means_
 - StandardScaler, 42
- metrics.hpp, 54
 - accuracy, 57
 - auc, 63
 - confusion_matrix, 59
 - f1_score, 60
 - fn_count, 58
 - fn_curve, 63
 - fp_count, 58
 - fp_curve, 62
 - fpr, 60
 - mse, 55
 - pr_curve, 61
 - precision, 59
 - r2, 56
 - recall, 60
 - roc_curve, 61
 - sse, 56
 - sst, 56
 - tn_count, 58
 - tn_curve, 62
 - tp_count, 57
 - tp_curve, 62
- min_derivative_size_
 - DerivativeSolver, 26
- mse
 - metrics.hpp, 55
- name_
 - AutoRegExtractor, 10
 - BaseModel, 17
 - BaseSolver, 20
 - BaseTransformer, 23
- NewtonShapeException, 33
 - NewtonShapeException, 33
 - what, 34
- NotFittedException, 34
 - NotFittedException, 34
 - obj_name_, 35
 - what, 35
- obj_name_
 - NotFittedException, 35
- ols
 - predict_functions.hpp, 67
- ols_solver.hpp, 64
- OLSSolver, 35
 - OLSSolver, 36
 - optimize, 36
- optimize
 - BaseSolver, 19
 - DerivativeSolver, 25
 - OLSSolver, 36
 - QRSolver, 38
- p_
 - AutoRegExtractor, 10
 - AutoRegModel< SolverType >, 14
- pr_curve
 - metrics.hpp, 61
- PRCurve
 - types.hpp, 71
- precision
 - metrics.hpp, 59
- Precisions
 - types.hpp, 71
- predict
 - AutoRegModel< SolverType >, 14
 - BaseModel, 16
 - LinRegModel< SolverType >, 28
 - LogRegModel< SolverType >, 31
- predict_functions.hpp, 64
 - linreg, 65
 - logistic_function, 66
 - logreg_class, 66
 - logreg_proba, 66
 - ols, 67
 - qr, 67
- predict_proba
 - LogRegModel< SolverType >, 32
- qr
 - predict_functions.hpp, 67
- qr_solver.hpp, 68
- QRSolver, 37
 - optimize, 38
 - QRSolver, 38
- r2
 - metrics.hpp, 56
- recall
 - metrics.hpp, 60
- Recalls
 - types.hpp, 71
- roc_curve
 - metrics.hpp, 61
- ROCCurve
 - types.hpp, 71
- sigma_
 - AutoRegModel< SolverType >, 14
- solver_
 - AutoRegModel< SolverType >, 14
 - LinRegModel< SolverType >, 29
 - LogRegModel< SolverType >, 32
- sse
 - metrics.hpp, 56
- sst
 - metrics.hpp, 56
- standard_scaler.hpp, 69
- StandardScaler, 39
 - fit, 40
 - fit_transform, 41
 - get_means, 41
 - get_stddevs, 41

- means_, [42](#)
- StandardScaler, [40](#)
- stddevs_, [42](#)
- transform, [41](#)
- stddevs_
 - StandardScaler, [42](#)
- Target
 - types.hpp, [70](#)
- tn_count
 - metrics.hpp, [58](#)
- tn_curve
 - metrics.hpp, [62](#)
- TNs
 - types.hpp, [72](#)
- tp_count
 - metrics.hpp, [57](#)
- tp_curve
 - metrics.hpp, [62](#)
- TPs
 - types.hpp, [72](#)
- transform
 - BaseTransformer, [21](#)
 - StandardScaler, [41](#)
- types.hpp, [69](#)
 - ConfusionMatrix, [71](#)
 - Derivative, [71](#)
 - Fallouts, [71](#)
 - Features, [70](#)
 - FNs, [72](#)
 - FPs, [72](#)
 - get_name, [72](#)
 - PRCurve, [71](#)
 - Precisions, [71](#)
 - Recalls, [71](#)
 - ROCCurve, [71](#)
 - Target, [70](#)
 - TNs, [72](#)
 - TPs, [72](#)
 - Weights, [71](#)
- verbose_
 - BaseSolver, [20](#)
- Weights
 - types.hpp, [71](#)
- weights_
 - AutoRegModel< SolverType >, [14](#)
 - LinRegModel< SolverType >, [29](#)
 - LogRegModel< SolverType >, [32](#)
- what
 - NewtonShapeException, [34](#)
 - NotFittedException, [35](#)
- y_mean_
 - BaseModel, [18](#)