LossFunctions.jl Documentation

Release 0.1

Christof Stocker, Tom Breloff, Alex Williams

November 05, 2016

This package represents a community effort to centralize the definition and implementation of **loss functions** in Julia. As such, it is a part of the JuliaML ecosystem.

For details on a specific aspect, see the documentation outlined below.

CHAPTER 1

Installation

To install LossFunctions.jl, start up Julia and type the following code-snipped into the REPL. It makes use of the native Julia package manger.

```
Pkg.add("LossFunctions")
```

Additionally, for example if you encounter any sudden issues, or in the case you would like to contribute to the package, you can manually choose to be on the latest (untagged) version.

```
Pkg.checkout("LossFunctions")
```

CHAPTER 2

Getting Started

This section outlines the basic steps needed to start utilizing the LossFunctions.jl package. To that end we will provide a condensed overview of the package.

Note: In order to keep this overview consise, we will not discuss any background information or theory on the losses here in detail.

2.1 Using LossFunctions.jl

Once installed the LossFunctions package can be imported just as any other Julia package.

```
using LossFunctions
```

To get help on specific functionality you can either look up the information here, or if you prefer you can make use of Julia's native doc-system. The following example shows how to get additional information on *L1HingeLoss* within Julia's REPL:

```
?L1HingeLoss
```

2.2 Hello World

The following code snippets show a simple scenario of how a *Loss* can be used to compute the element-wise values.

```
using LossFunctions

true_targets = [  1,  0, -2]
pred_outputs = [0.5,  1, -1]

value(L2DistLoss(), true_targets, pred_outputs)
```

```
3-element Array{Float64,1}:
 0.25
 1.0
 1.0
```

The same function signatures also apply to the derivatives.

```
deriv(L2DistLoss(), true_targets, pred_outputs)
```

```
3-element Array{Float64,1}:
 -1.0
  2.0
  2.0
```

Additionally, we provide mutating versions of most functions.

```
buffer = zeros(3)
deriv!(buffer, L2DistLoss(), true_targets, pred_outputs)
```

If need be, one can also compute the *meanvalue()* or *sumvalue()* efficiently, without allocating a temporary array.

```
# or meanvalue
sumvalue(L2DistLoss(), true_targets, pred_outputs)
```

```
0.75
```

2.3 Overview

All the concrete loss "functions" that this package provides are defined as types and are subtypes of the abstract `Loss`.

Typically the losses we work with in Machine Learning are bivariate functions of the true `target` and the predicted `output` of some prediction model. All losses that can be expressed this way are subtypes for *SupervisedLoss*. To compute the value of some *SupervisedLoss* we use the function *value()*.

```
value(L2DistLoss(), true_target, pred_output)
```

We can further divide the supervised losses into two useful sub-categories: *DistanceLoss* and *MarginLoss*.

2.3.1 Losses for Regression

Supervised losses that can be expressed as a univariate function of `output - target` are referred to as distance-based losses.

```
value(L2DistLoss(), difference)
```

Distance-based losses are typically utilized for regression problems. That said, there are also other losses that are useful for regression problems that don't fall into this category, such as the *PeriodicLoss*.

Note: In the literature that this package is partially based on, the convention for the distance-based losses is `target - output` (see *[STEINWART2008]* p. 38). We chose to diverge from this definition because that would cause the the sign of the derivative to flip.

2.3.2 Losses for Classification

Margin-base losses are supervised losses where the values of the targets are restricted to be in {-1, 1}, and which can be expressed as a univariate function `output * target`.

```
value(L1HingeLoss(), agreement)
```

Note: Throughout the codebase we refer to the result of `output * target` as `agreement`. The discussion that lead to this convention can be found issue #9

Margin-based losses are usually used for binary classification. In contrast to other formalism, they do not natively provide probabilities as output.

Note: Even though distance-based losses and margin-based losses can be expressed in univariate form, we still provide the bivariate form of `value` for both.

CHAPTER 3

Shared Interface

All provided loss functions share a common interface with other losses, but to a varying degree.

This section will provide an overview of the shared functions are available to the different sub-class of an abstract `Loss`

3.1 Supervised Losses

class SupervisedLoss

Abstract subtype of `Loss`. A loss is considered **supervised**, if all the information needed to compute `value(loss, features, targets, outputs)` are contained in `targets` and `outputs`, and thus allows for the simplification `value(loss, targets, outputs)`.

3.1.1 Computing the values

value(*loss*, *targets*, *outputs*)

Computes the value of the loss function for each observation-pair in `targets` and `outputs` individual and returns the result as an array of the same size as the parameters.

Parameters

- **loss** (*SupervisedLoss*) – An instance of the loss we are interested in.
- **targets** (`AbstractArray`) – The true targets that we would like your model to predict.
- **outputs** (`AbstractArray`) – The predicted outputs that your model produced.

Returns The values of the loss function for the elements in `targets` and `outputs`.

Return type `AbstractArray`

sumvalue(*loss*, *targets*, *outputs*)

Same as *value()*, but returns the **sum** of all values as a `Number` instead of all the individual values as `Array`.

Return type `Number`

meanvalue(*loss*, *targets*, *outputs*)

Same as *value()*, but returns the unweighted **mean** of all values as a single `Number` instead of all the individual values as `Array`.

Return type `Number`

value!(buffer, loss, targets, outputs)
Computes the values of the loss function for each observation-pair in `targets` and `outputs` individually and stores them in the preallocated `buffer`, which has to be the same size as the parameters.

Parameters

- **buffer** (`AbstractArray`) – Array to store the computed values in. Old values will be overwritten and lost.
- **loss** (*SupervisedLoss*) – An instance of the loss we are interested in.
- **targets** (`AbstractArray`) – The true targets that we would like your model to predict.
- **outputs** (`AbstractArray`) – The predicted outputs that your model produced.

Returns `buffer`

3.1.2 Computing the derivatives

deriv(*loss*, *targets*, *outputs*)
Computes the derivative of the loss function for each observation-pair in `targets` and `outputs` individually and returns the result as an array of the same size as the parameters.

Parameters

- **loss** (*SupervisedLoss*) – An instance of the loss we are interested in.
- **targets** (`AbstractArray`) – The true targets that we would like your model to predict.
- **outputs** (`AbstractArray`) – The predicted outputs that your model produced.

Returns The derivatives of the loss function for the elements in `targets` and `outputs`.

Return type `AbstractArray`

sumderiv(*loss*, *targets*, *outputs*)
Same as *deriv()*, but returns the **sum** of all derivatives as a single `Number`, instead of all the individual derivatives as `Array`.

Return type `Number`

meanderiv(*loss*, *targets*, *outputs*)
Same as *deriv()*, but returns the unweighted **mean** of all derivatives as a single `Number`, instead of all the individual derivatives as `Array`.

Return type `Number`

deriv!(buffer, loss, targets, outputs)
Computes the derivative of the loss function for each observation-pair in `targets` and `outputs` individually and stores them in the preallocated `buffer`, which has to be the same size as the parameters.

Parameters

- **buffer** (`AbstractArray`) – Array to store the computed derivatives in. Old values will be overwritten and lost.
- **loss** (*SupervisedLoss*) – An instance of the loss we are interested in.
- **targets** (`AbstractArray`) – The true targets that we would like your model to predict.
- **outputs** (`AbstractArray`) – The predicted outputs that your model produced.

Returns `buffer`

value_deriv(*loss*, *targets*, *outputs*)

Returns the results of `value()` and `deriv()` as a tuple. In some cases this function can yield better performance, because the losses can make use of shared variable when computing the values.

3.1.3 Closures

In some circumstances it may be convenient to have the loss function or its derivative as a proper Julia function. Closures provide just that as the following examples demonstrate.

```
f = value_fun(L2DistLoss())
f(targets, outputs) # computes the value of L2DistLoss

d = deriv_fun(L2DistLoss())
d(targets, outputs) # computes the derivative of L2DistLoss
```

value_fun(*loss*)

deriv_fun(*loss*)

deriv2_fun(*loss*)

value_deriv_fun(*loss*)

3.1.4 Querying loss properties

The losses implemented in this package provide a range of properties that can be queried by functions defined in *LearnBase.jl*.

isminimizable(*loss*)

isconvex(*loss*)

isstrictlyconvex(*loss*)

isstronglyconvex(*loss*)

isdifferentiable(*loss*[, *at*])

istwicedifferentiable(*loss*[, *at*])

isnemitski(*loss*)

islipschitzcont(*loss*)

islocallylipschitzcont(*loss*)

isclipable(*loss*)

ismarginbased(*loss*)

isclasscalibrated(*loss*)

isdistancebased(*loss*)

issymmetric(*loss*)

3.2 Distance-based Losses

class `DistanceLoss`
Abstract subtype of *`SupervisedLoss`*. A supervised loss that can be simplified to `L(targets, outputs) = L(targets - outputs)` is considered distance-based.

`value`(*loss*, *difference*)
Computes the value of the loss function for each observation in `difference` individually and returns the result as an array of the same size as the parameter.

Parameters

- **`loss`** (*`DistanceLoss`*) – An instance of the loss we are interested in.
- **`difference`** (`AbstractArray`) – The result of subtracting the true targets from the predicted outputs.

Returns The value of the loss function for the elements in `difference`.

Return type `AbstractArray`

`deriv`(*loss*, *difference*)
Computes the derivative of the loss function for each observation in ‘‘difference‘ individually and returns the result as an array of the same size as the parameter.

Parameters

- **`loss`** (*`DistanceLoss`*) – An instance of the loss we are interested in.
- **`difference`** (`AbstractArray`) – The result of subtracting the true targets from the predicted outputs.

Returns The derivatives of the loss function for the elements in `difference`.

Return type `AbstractArray`

`value_deriv`(*loss*, *difference*)
Returns the results of *`value()`* and *`deriv()`* as a tuple. In some cases this function can yield better performance, because the losses can make use of shared variable when computing the values.

3.3 Margin-based Losses

class `MarginLoss`
Abstract subtype of *`SupervisedLoss`*. A supervised loss, where the targets are in {-1, 1}, and which can be simplified to `L(targets, outputs) = L(targets * outputs)` is considered margin-based.

`value`(*loss*, *agreement*)
Computes the value of the loss function for each observation in ‘‘agreement‘ individually and returns the result as an array of the same size as the parameter.

Parameters

- **`loss`** (*`MarginLoss`*) – An instance of the loss we are interested in.
- **`agreement`** (`AbstractArray`) – The result of multiplying the true targets with the predicted outputs.

Returns The value of the loss function for the elements in `agreement`.

Return type `AbstractArray`

deriv(*loss*, *agreement*)

Computes the derivative of the loss function for each observation in `agreement` individually and returns the result as an array of the same size as the parameter.

Parameters

- **loss** (*MarginLoss*) – An instance of the loss we are interested in.
- **agreement** (`AbstractArray`) – The result of multiplying the true targets with the predicted outputs.

Returns The derivatives of the loss function for the elements in `agreement`.

Return type `AbstractArray`

value_deriv(*loss*, *agreement*)

Returns the results of *value()* and *deriv()* as a tuple. In some cases this function can yield better performance, because the losses can make use of shared variable when computing the values.

CHAPTER 4

Distance-based Losses

This section lists all the subtypes of *DistanceLoss* that are implemented in this package.

Fig. 4.1: Distance-based Losses (Regression)

4.1 LPDistLoss

class LPDistLoss

The p-th power absolute distance loss. It is Lipschitz continuous iff $p = 1$, convex if and only if $p \geq 1$, and strictly convex iff $p > 1$.

$$L(r) = |r|^p$$

4.2 L1DistLoss

class L1DistLoss

The absolute distance loss. Special case of the *LPDistLoss* with `P=1`. It is Lipshitz continuous and convex, but not strictly convex.

$$L(r) = |r|$$

4.3 L2DistLoss

class L2DistLoss

The least squares loss. Special case of the *LPDistLoss* with `P=2`. It is strictly convex.

$$L(r) = |r|^2$$

4.4 LogitDistLoss

class `LogitDistLoss`

The distance-based logistic loss for regression. It is strictly convex and Lipshitz continuous.

$$L(r) = -\ln \frac{4e^r}{(1+e^r)^2}$$

4.5 HuberLoss

class `HuberLoss`

α

Loss function commonly used for robustness to outliers. For large values of α it becomes close to the *`L1DistLoss`*, while for small values of α it resembles the *`L2DistLoss`*. It is Lipshitz continuous and convex, but not strictly convex.

$$L(r) = \begin{cases} \frac{r^2}{2} & \text{if } |r| \leq \alpha \\ \alpha|r| - \frac{\alpha^2}{2} & \text{otherwise} \end{cases}$$

4.6 L1EpsilonInsLoss

class `L1EpsilonInsLoss`

The ϵ-insensitive loss. Typically used in linear support vector regression. It ignores deviances smaller than ϵ , but penalizes larger deviances linarily. It is Lipshitz continuous and convex, but not strictly convex.

$$L(r) = \max\{0, |r| - \epsilon\}$$

4.7 L2EpsilonInsLoss

class `L2EpsilonInsLoss`

The ϵ-insensitive loss. Typically used in linear support vector regression. It ignores deviances smaller than ϵ , but penalizes larger deviances quadratically. It is convex, but not strictly convex.

$$L(r) = \max\{0, |r| - \epsilon\}^2$$

4.8 PeriodicLoss

class `PeriodicLoss`

`c`

Measures distance on a circle of specified circumference c.

$$L(r) = 1 - \cos\left(\frac{2r\pi}{c}\right)$$

4.9 QuantileLoss

class `QuantileLoss`

τ

The quantile loss, aka pinball loss. Typically used to estimate the conditional τ-quantiles. It is convex, but not strictly convex. Furthermore it is Lipschitz continuous.

$$L(r) = \begin{cases} -(1-\tau)\,r & \text{if } r < 0 \\ \tau r & \text{if } r \geq 0 \end{cases}$$

CHAPTER 5

Margin-based Losses

This section lists all the subtypes of `MarginLoss` that are implemented in this package.

Fig. 5.1: Margin-based Losses (Classification)

5.1 ZeroOneLoss

class `ZeroOneLoss`

The classical classification loss. It penalizes every missclassified observation with a loss of *1* while every correctly classified observation has a loss of *0*. It is not convex nor continuous and thus seldomly used directly. Instead one usually works with some classification-calibrated surrogate loss, such as one of those listed below.

$$L(a) = \begin{cases} 1 & \text{if } a < 0 \\ 0 & \text{if } a >= 0 \end{cases}$$

5.2 PerceptronLoss

class `PerceptronLoss`

The perceptron loss linearly penalizes every prediction where the resulting `agreement` $a \leq 0$. It is Lipshitz continuous and convex, but not strictly convex.

$$L(a) = \max\{0, -a\}$$

5.3 L1HingeLoss

class `L1HingeLoss`

The hinge loss linearly penalizes every predicition where the resulting `agreement` $a \leq 1$. It is Lipshitz continuous and convex, but not strictly convex.

$$L(a) = \max\{0, 1 - a\}$$

5.4 L2HingeLoss

class `L2HingeLoss`

The truncated least squares loss quadratically penalizes every predicition where the resulting `agreement` $a \leq 1$. It is locally Lipshitz continuous and convex, but not strictly convex.

$$L(a) = \max\{0, 1 - a\}^2$$

5.5 LogitMarginLoss

class `LogitMarginLoss`

The margin version of the logistic loss. It is infinitely many times differentiable, strictly convex, and lipschitz continuous.

$$L(a) = \ln(1 + e^{-a})$$

5.6 SmoothedL1HingeLoss

class `SmoothedL1HingeLoss`

γ

As the name suggests a smoothed version of the L1 hinge loss. It is Lipshitz continuous and convex, but not strictly convex.

$$L(a) = \begin{cases} \frac{0.5}{\gamma} \cdot \max\{0, 1 - a\}^2 & \text{if } a \geq 1 - \gamma \\ 1 - \frac{\gamma}{2} - a & \text{otherwise} \end{cases}$$

5.7 ModifiedHuberLoss

class `ModifiedHuberLoss`

A special (scaled) case of the *`SmoothedL1HingeLoss`* with $\gamma = 4$. It is Lipshitz continuous and convex, but not strictly convex.

$$L(a) = \begin{cases} \max\{0, 1 - a\}^2 & \text{if } a \geq -1 \\ -4a & \text{otherwise} \end{cases}$$

5.8 L2MarginLoss

class `L2MarginLoss`

The margin-based least-squares loss for classification, which quadratically penalizes every prediction where $a \neq 1$. It is locally Lipschitz continuous and strongly convex.

$$L(a) = (1 - a)^2$$

CHAPTER 6

Acknowledgements

The basic design of this package is heavily modelled after the loss-related definitions in *[STEINWART2008]*.

We would also like to mention that some early inspiration was drawn from EmpiricalRisks.jl

6.1 References

CHAPTER 7

LICENSE

The LossFunctions.jl package is licensed under the **MIT "Expat" License**

see LICENSE.md in the Github repository.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Bibliography

[STEINWART2008] Steinwart, Ingo, and Andreas Christmann. "Support vector machines". Springer Science & Business Media, 2008.