

Lasso Regression

NAME

Lasso Regression - A regression algorithm to predict the continuous output with L1 regularization.

SYNOPSIS

```
class frovedis.mllib.linear_model.Lasso (alpha=0.01, fit_intercept=True, normalize=False,
    precompute=False, copy_X=True, max_iter=1000,
    tol=1e-4, warm_start=False, positive=False,
    random_state=None, selection='cyclic',
    verbose=0, solver='sag')
```

Public Member Functions

```
fit(X, y, sample_weight=None)
predict(X)
save(filename)
load(filename)
debug_print()
release()
```

DESCRIPTION

Linear least squares is the most common formulation for regression problems. It is a linear method with the loss function given by the **squared loss**:

$$L(\mathbf{w}; \mathbf{x}, y) := 1/2(\mathbf{w}^T \mathbf{x} - y)^2$$

Where the vectors \mathbf{x} are the training data examples and y are their corresponding labels which we want to predict. \mathbf{w} is the linear model (also known as weight) which uses a single weighted sum of features to make a prediction. The method is called linear since it can be expressed as a function of $\mathbf{w}^T \mathbf{x}$ and y . Lasso regression uses L1 regularization to address the overfit problem.

The gradient of the squared loss is: $(\mathbf{w}^T \mathbf{x} - y) \cdot \mathbf{x}$

The gradient of the regularizer is: $\text{sign}(\mathbf{w})$

Frovedis provides implementation of lasso regression with two different optimizers: (1) stochastic gradient descent with minibatch and (2) LBFGS optimizer.

The simplest method to solve optimization problems of the form **min** $f(\mathbf{w})$ is gradient descent. Such first-order optimization methods well-suited for large-scale and distributed computation. Whereas, L-BFGS

is an optimization algorithm in the family of quasi-Newton methods to solve the optimization problems of the similar form.

Like the original BFGS, L-BFGS (Limited Memory BFGS) uses an estimation to the inverse Hessian matrix to steer its search through feature space, but where BFGS stores a dense $n \times n$ approximation to the inverse Hessian (n being the number of features in the problem), L-BFGS stores only a few vectors that represent the approximation implicitly. L-BFGS often achieves rapid convergence compared with other first-order optimization.

This module provides a client-server implementation, where the client application is a normal python scikit-learn program. Scikit-learn has its own `linear_model` providing the Lasso Regression support. But that algorithm is non-distributed in nature. Hence it is slower when comparing with the equivalent Frovedis algorithm (see frovedis manual for `ml/lasso_regression`) with big dataset. Thus in this implementation, a scikit-learn client can interact with a frovedis server sending the required python data for training at frovedis side. Python data is converted into frovedis compatible data internally and the scikit-learn ML call is linked with the respective frovedis ML call to get the job done at frovedis server.

Scikit-learn side call for Lasso Regression quickly returns, right after submitting the training request to the frovedis server with a unique model ID for the submitted training request.

When operations like prediction will be required on the trained model, scikit-learn client sends the same request to frovedis server on the same model (containing the unique ID) and the request is served at frovedis server and output is sent back to the scikit-learn client.

Detailed Description

Lasso()

Parameters

alpha: A double parameter containing the learning rate. (Default: 0.01)

fit_intercept: A boolean parameter specifying whether a constant (intercept) should be added to the decision function. (Default: True)

normalize: A boolean parameter (unused)

precompute: A boolean parameter (unused)

copy_X: A boolean parameter (unused)

max_iter: An integer parameter specifying maximum iteration count. (Default: 1000)

tol: A double parameter specifying the convergence tolerance value, (Default: 1e-4)

warm_start: A boolean parameter (unused)

positive: A boolean parameter (unused)

random_state: An integer, None or RandomState instance. (unused)

selection: A string object. (unused)

verbose: An integer object specifying the log level to use. (Default: 0)

solver: A string object specifying the solver to use. (Default: 'sag')

Purpose

It initialized a Lasso object with the given parameters.

The parameters: “normalize”, “precompute”, “copy_X”, “warm_start”, “positive”, “random_state” and “selection” are not yet supported at frovedis side. Thus they don’t have any significance in this call. They are simply provided for the compatibility with scikit-learn application.

“solver” can be either ‘sag’ for frovedis side stochastic gradient descent or ‘lbfgs’ for frovedis side LBFGS optimizer when optimizing the linear regression model.

“verbose” value is set at 0 by default. But it can be set to 1 (for DEBUG mode) or 2 (for TRACE mode) for getting training time logs from frovedis server.

Return Value

It simply returns “self” reference.

fit(X, y, sample_weight=None)

Parameters

X: A scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix.

y: Any python array-like object or an instance of FrovedisDvector.

sample_weight: Python array-like optional parameter. (unused)

Purpose

It accepts the training feature matrix (X) and corresponding output labels (y) as inputs from the user and trains a linear regression model with L1 regularization with those data at frovedis server.

It doesn't support any initial weight to be passed as input at this moment. Thus the "sample_weight" parameter will simply be ignored. It starts with an initial guess of zeros for the model vector and keeps updating the model to minimize the cost function until convergence is achieved or maximum iteration count is reached.

For example,

```
# loading sample CRS data file
mat = FrovedisCRSMatrix().load("./sample")
lbl = FrovedisDvector([1.1,0.2,1.3,1.4,1.5,0.6,1.7,1.8])

# fitting input matrix and label on lasso object
lr = Lasso(solver='sgd', verbose=2).fit(mat, lbl)
```

Return Value

It simply returns "self" reference.

Note that the call will return quickly, right after submitting the fit request at frovedis server side with a unique model ID for the fit request. It may be possible that the training is not completed at the frovedis server side even though the client scikit-learn side fit() returns.

predict(X)

Parameters

X: A scipy sparse matrix or any python array-like object or an instance of FrovedisCRSMatrix.

Purpose

It accepts the test feature matrix (X) in order to make prediction on the trained model at frovedis server.

Return Value

It returns a numpy array of double (float64) type containing the predicted outputs.

save(filename)

Parameters

filename: A string object containing the name of the file on which the target model is to be saved.

Purpose

On success, it writes the model information (weight values etc.) in the specified file as little-endian binary data. Otherwise, it throws an exception.

Return Value

It returns nothing.

load(filename)

Parameters

filename: A string object containing the name of the file having model information to be loaded.

Purpose

It loads the model from the specified file (having little-endian binary data).

Return Value

It simply returns “self” instance.

debug__print()

Purpose

It shows the target model information (weight values etc.) on the server side user terminal. It is mainly used for debugging purpose.

Return Value

It returns nothing.

release()

Purpose

It can be used to release the in-memory model at frovedis server.

Return Value

It returns nothing.

SEE ALSO

linear_regression, ridge_regression, dvector, crs_matrix