Logistic Regression

NAME

Logistic Regression - A classification algorithm to predict the binary output with logistic loss.

SYNOPSIS

class frovedis.mllib.linear_model.LogisticRegression (penalty='l2', dual=False,
tol=1e-4, C=0.01, fit_intercept=True, intercept_scaling=1,
class_weight=None, random_state=None, solver='sag',
max_iter=1000, multi_class='ovr', verbose=0, warm_start=False,
n_jobs=1)

Public Member Functions

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

DESCRIPTION

Classification aims to divide the items into categories. The most common classification type is binary classification, where there are two categories, usually named positive and negative. Frovedis supports binary classification algorithm only.

Logistic regression is widely used to predict a binary response. It is a linear method with the loss function given by the **logistic loss**:

```
L(w;x,y) := log(1 + exp(-ywTx))
```

Where the vectors x are the training data examples and y are their corresponding labels (Frovedis considers negative response as -1 and positive response as 1, but when calling from scikit-learn interface, user should pass 0 for negative response and 1 for positive response according to the scikit-learn requirement) which we want to predict. w is the linear model (also called as weight) which uses a single weighted sum of features to make a prediction. Frovedis Logistic Regression supports ZERO, L1 and L2 regularization to address the overfit problem.

The gradient of the logistic loss is: -y(1 - 1 / (1 + exp(-ywTx))).x
The gradient of the L1 regularizer is: sign(w)
And The gradient of the L2 regularizer is: w

For binary classification problems, the algorithm outputs a binary logistic regression model. Given a new data point, denoted by x, the model makes predictions by applying the logistic function:

```
f(z) := 1 / 1 + exp(-z)
```

Where z = wTx. By default (threshold=0.5), if f(wTx) > 0.5, the response is positive (1), else the response is negative (0).

Frovedis provides implementation of logistic 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(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 nxn 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 rapider 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 Logistic 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/logistic_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 Logistic 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

LogisticRegression()

Parameters *penalty*: A string object containing the regularizer type to use. (Default: 'l2')
dual: A boolean parameter (unused)
tol: A double parameter specifying the convergence tolerance value, (Default: 1e-4)
C: 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)
intercept_scaling: An integer parameter. (unused)
*class*weight_: A python dictionary or a string object. (unused)
random_state: An integer, None or RandomState instance. (unused)
solver: A string object specifying the solver to use. (Default: 'sag')
max_iter: An integer parameter specifying maximum iteration count. (Default: 1000)

multi_class: A string object specifying type of classification. (Default: 'ovr')
verbose: An integer object specifying the log level to use. (Default: 0)
warm_start: A boolean parameter. (unused)
n_jobs: An integer parameter. (unused)

Purpose
It initialized a Lasso object with the given parameters.

The parameters: "dual", "intercept_scaling", "class_weight", "warm_start", "random_state" and "n_jobs" 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.

"penalty" can be either 'l1' or 'l2' (Default: 'l2').

"solver" can be either 'sag' for frovedis side stochastic gradient descent or 'lbfgs' for frovedis side LBFGS optimizer when optimizing the linear regression model.

"multi_class" can only be 'ovr' as frovedis suppots binary classification algorithms only at this moment.

"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 specifed 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([1,0,1,1,1,0,1,1])

# fitting input matrix and label on logistic regression object
lr = LogisticRegression(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.

predict_proba(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. But unlike predict(), it returns the probability values against each input sample to be positive.

Return Value
It returns a numpy array of double (float64) type containing the prediction probability values.

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_svm, dvector, crs_matrix