GPU-imLearn: an open-source software for imbalanced data classification based on GPU

(User Manual)

CONTENTS

1. Overview

In recent years, multi-imbalance learning is becoming more and more popular, however, with the scale of data increasing fast and classification algorithms getting more complex, there is an urgent need for more efficient ways to solve the low efficiency of multi-class imbalanced data classification. we develop the "GPU-imLearn" software package and share it with the community, to boost research in this field.

The developed GPU-ImLearn software contains 6 different algorithms for multi-class imbalance learning based on GPU acceleration, 5 of them were proposed in recent years, and one(FocalBoost) was recently proposed by us. We will introduce the framework and functionalities of this software in the next sections.

2. Installation

Before installation of GPU-imLearn, please ensure that CUDA 9 or above was installed correctly in your environment.

Then, you can install like this:

```
>>> git clone https://github.com/inbliz/gpuimlearn.git
>>> cd gpuimlearn/codes/gpuimlearn
>>> python setup.py install
```

Here is a sample for installation, see figure 1.

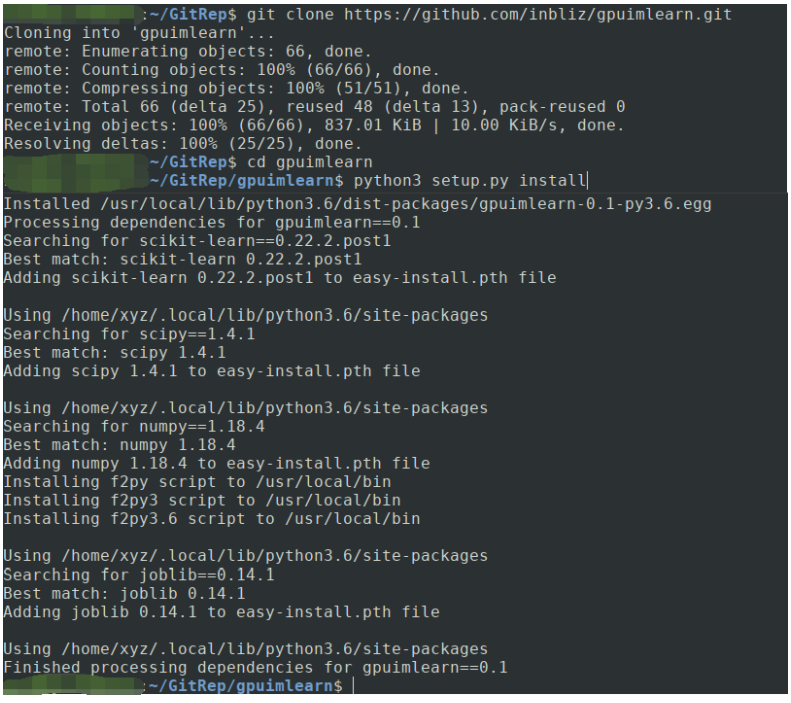

Figure 1 installation of GPU-imLearn

3. API Reference

Each algorithm has 3 functions mainly: initialization, train and predict.

3.1 DECOC

model initialization:

clf = DECOC.DECOC()

model train:

clf.fit(X_train, y_train)

Table 3.1 DECOC.fit

Parameters	**X_train** The data matrix in the training dataset.
	y_train The corresponding labels of each instance in the training dataset.

model predict:

pre = clf.predict(X_test)

Table 3.2 : DECOC.predict

Parameters	**X_test** The data matrix waiting to be predicted.
Returns	**pre** The prediction results for X_test

3.2 FocalBoost

model initialization:

clf = FocalBoost. FocalBoost ()

model train:

clf.fit(X_train, y_train, imcls)

Table 3.3 FocalBoost.fit

Parameters	**X_train** The data matrix in the training dataset.
	y_train The corresponding labels of each instance in the training dataset.
	Imcls an array specified by user, It should contain the minority classes in the dataset.

model predict:

pre = clf.predict(X_test)

Table 3.4 FocalBoost.predict

Parameters	**X_test** The data matrix waiting to be predicted.
Returns	**pre** The prediction results for X_test

3.3 DOVO

model initialization:

clf = DOVO. DOVO ()

model train:

clf.fit(X_train, y_train)

Table 3.5 DOVO.fit

Parameters	**X_train** The data matrix in the training dataset.
	y_train The corresponding labels of each instance in the training dataset.

model predict:

pre = clf.predict(X_test)

Table 3.6 : DOVO.predict

Parameters	**X_test** The data matrix waiting to be predicted.
Returns	**pre** The prediction results for X_test

3.4 AdaBoost.M1

model initialization:

clf = AdaBoostM1. AdaBoostM1 ()

model train:

clf.fit(X_train, y_train)

Table 3.7 AdaBoost.M1.fit

Parameters	**X_train** The data matrix in the training dataset.
	y_train The corresponding labels of each instance in the training dataset.

model predict:

pre = clf.predict(X_test)

Table 3.8 AdaBoost.M1.predict

Parameters	**X_test** The data matrix waiting to be predicted.
Returns	**pre** The prediction results for X_test

3.5 SAMME

model initialization:

clf = SAMME.SAMME ()

model train:

clf.fit(X_train, y_train)

Table 3.9 SAMME.fit

Parameters	**X_train** The data matrix in the training dataset.
	y_train The corresponding labels of each instance in the training dataset.

model predict:

pre = clf.predict(X_test)

Table 3.10 SAMME.predict

Parameters	**X_test** The data matrix waiting to be predicted.
Returns	**pre** The prediction results for X_test

3.6 imECOC

model initialization:

clf = imECOC. imECOC ()

model train:

clf.fit(X_train, y_train)

Table 3.11 imECOC.fit

Parameters	**X_train** The data matrix in the training dataset.
	y_train The corresponding labels of each instance in the training dataset.

model predict:

pre = clf.predict(X_test)

Table 3.12 imECOC.predict

Parameters	**X_test** The data matrix waiting to be predicted.
Returns	**pre** The prediction results for X_test

4. Usage Examples

4.1 DECOC

Usage example:

DECOC take as input two arrays: an array X of shape (n_samples, n_features) holding the training samples, and an array y of class labels (strings or integers), of shape (n_samples):

```
>>> from gpuimlearn import DECOC
>>> X_train = [[1, 0.5], [2, 1], [1, 1], [2, 2], [1, 2.5], [2, 4.5]]
>>> y_train = [1, 1, 2, 2, 3, 3]
>>> clf = DECOC.DECOC()
>>> clf.fit(X_train, y_train)
DECOC()
```

After being fitted, the model can then be used to predict new values:

```
>>> X_test = [[1.5, 0.5], [1.5, 1.5], [1.5, 4.5]]
>>> clf.predict(X_test)
array([1, 2, 3])
```

4.2 FocalBoost

Usage example:

FocalBoost take as input two arrays: an array X of shape (n_samples, n_features) holding the training samples, and an array y of class labels (strings or integers), of shape (n_samples):

```
>>> from gpuimlearn import FocalBoost
>>> X_train = [[1, 0.5], [2, 1], [1, 1], [2, 2], [1, 2.5], [2, 4.5]]
>>> y_train = [1, 1, 2, 2, 3, 3]
>>> clf = FocalBoost. FocalBoost ()
>>> clf.fit(X_train, y_train)
FocalBoost()
```

After being fitted, the model can then be used to predict new values:

```
>>> X_test = [[1.5, 0.5], [1.5, 1.5], [1.5, 4.5]]
>>> clf.predict(X_test)
array([1, 2, 3])
```

4.3 DOVO

Usage example:

DOVO take as input two arrays: an array X of shape (n_samples, n_features) holding the training samples, and an array y of class labels (strings or integers), of shape (n_samples):

```
>>> from gpuimlearn import DOVO
>>> X_train = [[1, 0.5], [2, 1], [1, 1], [2, 2], [1, 2.5], [2, 4.5]]
>>> y_train = [1, 1, 2, 2, 3, 3]
>>> clf = DOVO.DOVO ()
>>> clf.fit(X_train, y_train)
DOVO ()
```

After being fitted, the model can then be used to predict new values:

```
>>> X_test = [[1.5, 0.5], [1.5, 1.5], [1.5, 4.5]]
>>> clf.predict(X_test)
array([1, 2, 3])
```

4.4 AdaBoost.M1

Usage example:

AdaBoost.M1 take as input two arrays: an array X of shape (n_samples, n_features) holding the training samples, and an array y of class labels (strings or integers), of shape (n_samples):

```
>>> from gpuimlearn import AdaBoostM1
>>> X_train = [[1, 0.5], [2, 1], [1, 1], [2, 2], [1, 2.5], [2, 4.5]]
>>> y_train = [1, 1, 2, 2, 3, 3]
>>> clf = AdaBoostM1. AdaBoostM1 ()
>>> clf.fit(X_train, y_train)
AdaBoostM1 ()
```

After being fitted, the model can then be used to predict new values:

```
>>> X_test = [[1.5, 0.5], [1.5, 1.5], [1.5, 4.5]]
>>> clf.predict(X_test)
array([1, 2, 3])
```

4.5 SAMME

Usage example:

SAMME take as input two arrays: an array X of shape (n_samples, n_features) holding the training samples, and an array y of class labels (strings or integers), of shape (n_samples):

```
>>> from gpuimlearn import SAMME
>>> X_train = [[1, 0.5], [2, 1], [1, 1], [2, 2], [1, 2.5], [2, 4.5]]
>>> y_train = [1, 1, 2, 2, 3, 3]
>>> clf = SAMME. SAMME ()
>>> clf.fit(X_train, y_train)
SAMME ()
```

After being fitted, the model can then be used to predict new values:

```
>>> X_test = [[1.5, 0.5], [1.5, 1.5], [1.5, 4.5]]
>>> clf.predict(X_test)
array([1, 2, 3])
```

4.6 imECOC

Usage example:

imECOC take as input two arrays: an array X of shape (n_samples, n_features) holding the training samples, and an array y of class labels (strings or integers), of shape (n_samples):

```
>>> from gpuimlearn import imECOC
>>> X_train = [[1, 0.5], [2, 1], [1, 1], [2, 2], [1, 2.5], [2, 4.5]]
>>> y_train = [1, 1, 2, 2, 3, 3]
>>> clf = imECOC. imECOC ()
>>> clf.fit(X_train, y_train)
imECOC ()
```

After being fitted, the model can then be used to predict new values:

```
>>> X_test = [[1.5, 0.5], [1.5, 1.5], [1.5, 4.5]]
>>> clf.predict(X_test)
array([1, 2, 3])
```