***class* imbbag.MRBBag(n_estimator=*10*, estimator=*DecisionTreeClassifier()*, sampling_method = "*undersampling*")**

Multi-class Roughly Balanced Bagging

MRBBag algorithm estimates the number of instances for each class of bootstraps using a multinomial distribution defined by a specific mass function. In the original bagging, $p_1, p_2, \ldots, p_c$ are chosen in proportion to the presence of classes in the training set. However, in MRBBag, they are set to a constant value of $1/c$, where c is the number of classes.

Source: Lango, M., & Stefanowski, J. (2018). Multi-class and feature selection extensions of roughly balanced bagging for imbalanced data. Journal of Intelligent Information Systems, 50, 97-127.

Parameters: n_estimator: ***int (default=10)***

The number of nearest neighbors to search for.

estimator: ***object (default= DecisionTreeClassifier())***

An instance of a base classifier used in the ensemble.

sampling_method: ***string (default= "undersampling")***

The algorithm applies a sampling method, which can be either undersampling or oversampling, by setting the values "undersampling" and "oversampling" to this parameter, respectively.

Examples:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn. model_selection import train_test_split
from ImbBag import MRBBag

dataframe = read_csv('dataset.csv')
data = dataframe.values
X = data[:,:-1]
Y = data[:,:-1]

# split the dataset into training and test sets
X_train ,X_test ,y_train ,y_test = train_test_split (X, y,
test_size =0.2)
```

```
# instantiate the imbalance bagging classifier, training,
prediction
cls = MRBBag(n_estimator = 50, estimator =
DecisionTreeClassifier(), sampling_method="oversampling")
clf.fit(X_train , y_train)
y_pred = clf.predict(X_test)
```

Methods

fit(self, X_train, y_train)	Fit the model.
predict(self, X)	Predict the class label for sample X
predict_proba(self, X)	Estimate the probability of X belonging to each class-labels.

fit(self, X_train,y_train)

Parameters X_train : ***numpy.ndarray of shape (n_samples, n_features)***

The features to train the model.

y_train : ***numpy.ndarray of shape (n_samples,)***

An array-like with the class labels of all samples in X_train.

Returns : self

predict(self, X):

Parameters X : ***numpy.ndarray of shape (n_samples, n_features)***

All the samples we want to predict the label for.

Returns : ***numpy.ndarray***

A 1D array of shape (, n_samples), containing the predicted class labels for all instances in X.

predict_proba(self, X):

Parameters X : ***numpy.ndarray of shape (n_samples, n_features)***

All the samples we want to predict the label for.

Returns : ***numpy.ndarray***

A 2D array of shape (n_samples, n_classes). Where each i-th row contains len(self.target_value) elements, representing the probability that the i-th sample of X belongs to a certain class label.