class **imbbag.RBBag(*n_estimator=10*, estimator=*DecisionTreeClassifier()*)**

Roughly Balanced Bagging

The RBBag method is a binary-class bagging method that determines the instance counts for each class by balancing the sampling probabilities. In each bootstrap sample, the count of minority instances is fixed to the size of the minority class in the original dataset. Conversely, the count of majority instances is probabilistically determined using the negative binomial distribution.

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.

Examples:

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

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 = RBBag(n_estimator = 50, estimator =
DecisionTreeClassifier())
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.