***class* imbbag.RBSBag (estimator = DecisionTreeClassifier(), n_bins = 10, threshold = 0.01, min_support = 0.05, min_estimator = 2)**

Random Balanced Sampling with Bagging (RBSBag)

RBSBag is a multi-class bagging algorithm for imbalanced data sets that distinguishes itself from its predecessors through its unique application of attribute selection, akin to the Random Forest (RF) algorithm, but with several distinct features..

Source: Huang, C., Huang, X., Fang, Y., Xu, J., Qu, Y., Zhai, P., ... & Li, J. (2020). Sample imbalance disease classification model based on association rule feature selection. Pattern Recognition Letters, 133, 280-286.

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

An instance of a base classifier used in the ensemble

n_bins: ***int (default=10)***

Number of quantiles used to discretize featues into equal-sized buckets.

threshold : ***real (default= 0.01)***

The value of the threshold controls whether the difference between the current macro-F1 value and the current optimal macro-F1 value exceeds the convergence threshold. If it does, the addition of a base classifier to the ensemble is stopped.

min_support : ***real (default= 0.05)***

A float between 0 and 1 for minimum support of the itemsets returned.

min_estimator : ***int (default= 2)***

The minimum number of base classifier in the ensemble.

Examples:

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

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 = RBSBag(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.