***class* imbbag.EBBag(estimator=*DecisionTreeClassifier()*)**

Exactly Balanced Bagging classifier

Exactly Balanced Bagging (EBBag), also called Binary Ensemble Variation (BEV), is a binary class classification algorithm that divides the samples into $N = \lfloor N_{maj}/N_{min} \rfloor$ disjoint sets. Here, N_{maj} and N_{min} represent the number of data points in the majority and minority classes, respectively. Each of these disjoint sets is then combined with the data from the minority class. This results in a series of combined sets, each containing a balanced mix of majority and minority class data. The final step involves training a base classifier for each of these combined sets. It's important to note that in this bagging algorithm, the number of base classifiers is a fixed value, equal to *N*.

Source: Li, C. (2007, March). Classifying imbalanced data using a bagging ensemble variation (BEV). In Proceedings of the 45th annual southeast regional conference (pp. 203-208).

Liu, X. Y., Wu, J., & Zhou, Z. H. (2008). Exploratory undersampling for class-imbalance learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 39(2), 539-550.

Parameters : 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 EBBag

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 = EBBag(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.