***class* imbbag.LAZYBag(n_estimator=10, estimator=DecisionTreeClassifier(),beta = 0.99)**

Lazy Bagging

The Lazy Bagging (LazyBag) algorithm operates by delaying the learning process until a test instance arrives. When a test instance needs to be classified, the algorithm first identifies the k-nearest neighbors (kNN) from the training set. These kNNs, in conjunction with the original training set D, are used to construct bootstrap bags for bagging prediction.

Source: Zhu, X. (2007, October). Lazy bagging for classifying imbalanced data. In Seventh IEEE International Conference on Data Mining (ICDM 2007) (pp. 763-768). IEEE.

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.

beta: ***real (default= 0.99)***

The algorithm employs a β-similar concept to automatically determine the value of k for each dataset. The author theoretically proved an inequality, presented as $\omega \leq \log_4 N^{1-\beta}$, and used it to determine the value of k. The value of k is determined by the equation $k = \omega N$. where N is the number of samples in the training set. Here, β is experimentally set to a value of 0.99 by the author.

.

Examples:

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

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 = LazyBag(n_estimator = 50, estimator =
                      DecisionTreeClassifier)
y_pred = clf.predict(X_train , y_train, 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.

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.