***class* imbbag.UnderBagKNN (n_estimator =10, subsample_rate=1, k_neighbors = 5*)**

Under-Bagging kNN classifier (UnderBagKNN)

UnderBagKNN is similar to the UnderBag algorithm, except using kNN as a base classifier and instead of sampling with eqaul probability, each sample sampled based on the accept probability.

Source: Hang, H., Cai, Y., Yang, H., & Lin, Z. (2022). Under-bagging nearest neighbors for imbalanced classification. Journal of Machine Learning Research, 23(118), 1-63

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

The number of nearest neighbors to search for.

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

The number of nearest neighbors used to generate synthetic samples.

Subsample_rate : ***real (default= 1)***

Percentage of resampling from data classes.

Examples:

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

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 = UnderBagKNN(n_estimator = 10, subsample_rate =
1,k_neighbors=5)
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.