DiFF-RF Documentation

Contents

Module DiFF_RF

Created on Tue Mar 24 12:19:32 2020

@author: Pierre-François Marteau (https://people.irisa.fr/Pierre-Francois.Marteau/)

Functions

Function EE

```
def EE(
    hist
)
```

given a list of positive values as a histogram drawn from any information source, returns the empirical entropy of its discrete probability function.

Parameters

hist : array histogram

Returns

float empirical entropy estimated from the histogram

Function `create_tree`

```
def create_tree(
    X,
    featureDistrib,
    sample_size,
    max_height
)
```

Creates an DiFF tree using a sample of size sample_size of the original data.

Parameters

`X` : nD array. nD array with the observations. Dimensions should be (n_obs, n_features).
`sample_size` : int Size of the sample from which a DiFF tree is built.
`max_height` : int Maximum height of the tree.

Returns

a DiFF tree

Function `getSplit`

```
def getSplit(
    X
)
```

Randomly selects a split value from set of scalar data 'X'. Returns the split value.

Parameters

`X` : array Array of scalar values

Returns

float split value

Function `similarityScore`

```
def similarityScore(
    S,
    node,
    alpha
)
```

Given a set of instances S falling into node and a value alpha >=0, returns for all element x in S the weighted similarity score between x and the centroid M of S (node.M)

Parameters

`S` : array of instances Array of instances that fall into a node
`node` : a DiFF tree node S is the set of instances "falling" into the node
`alpha` : float alpha is the distance scaling hyper-parameter

Returns

array the array of similarity values between the instances in S and the mean of training instances falling in node

Function `walk_tree`

```
def walk_tree(
    forest,
    node,
    treeIdx,
    obsIdx,
    X,
    featureDistrib,
```

```
    depth=0,
    alpha=0.01
)
```

Recursive function that walks a tree from an already fitted forest to compute the path length of the new observations.

Parameters

forest : DiFF_RF A fitted forest of DiFF trees
node : DiFF Tree node the current node
treeIdx : int index of the tree that is being walked.
obsIdx : array 1D array of length n_obs. 1/0 if the obs has reached / has not reached the node.
X : nD array. array of observations/instances.
depth : int current depth.

Returns

None

Function `weightFeature`

```
def weightFeature(
    s,
    nbins
)
```

Given a list of values corresponding to a feature dimension, returns a weight (in [0,1]) that is one minus the normalized empirical entropy, a way to characterize the importance of the feature dimension.

Parameters

s : array list of scalar values corresponding to a feature dimension
nbins : int the number of bins used to discretize the feature dimension using an histogram.

Returns

float the importance weight for feature s.

Classes

Class `DiFF_Tree`

```
class DiFF_Tree(
    height_limit
)
```

Construct a tree via randomized splits with maximum height height_limit.

Parameters

height_limit : int Maximum height of the tree.

Returns

None

Methods

Method `fit`

```
def fit(
    self,
    X: numpy.ndarray,
    featureDistrib: <built-in function array>
)
```

Given a 2D matrix of observations, create an DiFF tree. Set field self.root to the root of that tree and return it.

Parameters

`X` : nD array. nD array with the observations. Dimensions should be (n_obs, n_features).
`featureDistrib` : 1D array The distribution weight affected to each dimension

Returns

A DIFF tree root.

Class `DiFF_TreeEnsemble`

```
class DiFF_TreeEnsemble(
    sample_size: int,
    n_trees: int = 10
)
```

DiFF Forest. Even though all the methods are thought to be public the main functionality of the class is given by: - **init** - **fit** - **predict**

Creates the DiFF-RF object.

Parameters

`sample_size` : int. size of the sample randomly drawn from the train instances to build each DiFF tree.
`n_trees` : int The number of trees in the forest

Returns

`None`

Methods

Method `anomaly_score`

```
def anomaly_score(
    self,
    X: numpy.ndarray,
    alpha=1
) -> numpy.ndarray
```

Given a nD matrix of observations, X, compute the anomaly scores for instances in X, returning 3 1D arrays of anomaly scores

Parameters

`X` : nD array. nD array with the tested observations to be predicted. Dimensions should be (n_obs, n_features).
`alpha` : float scaling distance hyper-parameter.

Returns

`scD`, `scF`, `scFF` : 1d arrays respectively the distance scores (point-wise anomaly score), the frequency of visit socres and the collective anomaly scores

Method `fit`

```
def fit(
    self,
    X: numpy.ndarray,
    n_jobs: int = 4
)
```

Fits the algorithm into a model. Given a 2D matrix of observations, create an ensemble of IsolationTree objects and store them in a list: self.trees. Convert DataFrames to ndarray objects. Uses parallel computing.

Parameters

`X` : nD array. nD array with the train instances. Dimensions should be (n_obs, n_features).
`n_jobs` : int number of parallel jobs that will be launched

Returns

`the object itself.`

Method `predict`

```
def predict(
    self,
    X: numpy.ndarray,
    threshold: float
) -> numpy.ndarray
```

A shorthand for calling anomaly_score() and predict_from_anomaly_scores().

Parameters

`X` : nD array. nD array with the tested observations to be predicted. Dimensions should be (n_obs, n_features).
`threshold` : float Threshold for considering a observation an anomaly, the higher the less anomalies.

Returns

1D array The prediction array corresponding to 1/0 if anomaly/not anomaly respectively.

Method `predict_from_anomaly_scores`

```
def predict_from_anomaly_scores(
    self,
    scores: numpy.ndarray,
    threshold: float
) -> numpy.ndarray
```

Given an array of scores and a score threshold, return an array of the predictions: 1 for any score >= the threshold and 0 otherwise.

Parameters

`scores` : 1D array. 1D array of scores. Dimensions should be (n_obs, n_features).
`threshold` : float Threshold for considering a observation an anomaly, the higher the less anomalies.

Returns

1D array The prediction array corresponding to 1/0 if anomaly/not anomaly respectively.

:param scores: 1D array. Scores produced by the random forest. :param threshold: Threshold for considering a observation an anomaly, the higher the less anomalies. :return: Return predictions

Method `walk`

```
def walk(
    self,
    X: numpy.ndarray
) -> numpy.ndarray
```

Given a nD matrix of observations, X, compute the average path length, the distance, frequency and collective anomaly scores for instances in X. Compute the path length for x_i using every tree in self.trees then compute the average for each x_i. Return an ndarray of shape (len(X),1).

Parameters

`X` : nD array. nD array with the instances to be tested. Dimensions should be (n_obs, n_features).

Returns

None

Class `InNode`

```
class InNode(
    X,
    height_limit,
    featureDistrib,
    sample_size,
    current_height
)
```

Node of the tree that is not a leaf node. The functionality of the class is: - Do the best split from a sample of randomly chosen dimensions and split points. - Partition the space of observations according to the split and send the along to two different nodes The method usually has a higher complexity than doing it for every point. But because it's using NumPy it's more efficient time-wise.

Parameters

`X` : nD array. nD array with the training instances that have reached the node.
`height_limit` : int Maximum height of the tree.
`Xf` : nD array. distribution used to randomly select a dimension (feature) used at parent level.
`sample_size` : int Size of the sample used to build the tree.
`current_height` : int Current height of the tree.

Returns

None

Class `LeafNode`

```
class LeafNode(
    X,
    height,
    Xp,
    sample_size
)
```

Leaf node The base funcitonality is storing the Mean and standard deviation of the observations in that node. We also evaluate the frequency of visit for training data.

Parameters

`X` : nD array. nD array with the training instances falling into the leaf node.
`height` : int Current height of the tree.
`Xf` : nD array. nD array with the training instances falling into the parent node.
`sample_size` : int Size of the sample used to build the tree.

Returns

None
