STree

Release 1.2.2

Ricardo Montañana Gómez
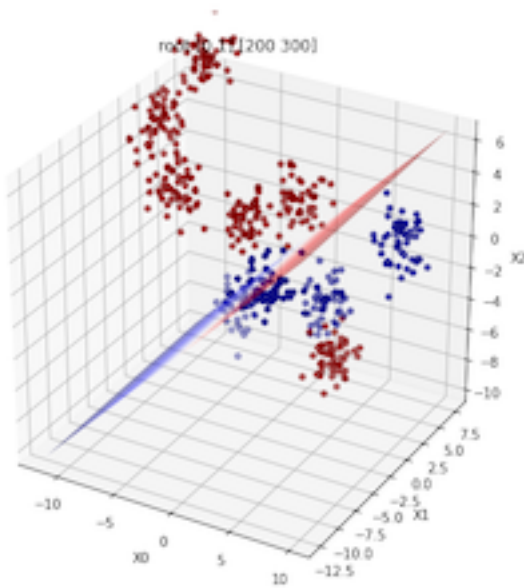
Nov 02, 2021

CONTENTS:

STREE

Oblique Tree classifier based on SVM nodes. The nodes are built and splitted with sklearn SVC models. Stree is a sklearn estimator and can be integrated in pipelines, grid searches, etc.

1.1 License

STree is MIT licensed

INSTALL

The main stable release

```
pip install stree
```

or the last development branch

```
pip install git+https://github.com/doctorado-ml/stree
```

2.1 Tests

```
python -m unittest -v stree.tests
```

CHAPTER

THREE

HYPERPARAMETERS

	Hyper-parameter	Type/Values	Default	Meaning
*	C	<float>	1.0	Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive.
*	kernel	{"liblinear", "linear", "poly", "rbf", "sigmoid"}	linear	Specifies the kernel type to be used in the algorithm. It must be one of 'liblinear', 'linear', 'poly' or 'rbf'. liblinear uses liblinear library and the rest uses libsvm library through scikit-learn library
*	max_iter	<int>	1e5	Hard limit on iterations within solver, or -1 for no limit.
*	random_state	<int>	None	Controls the pseudo random number generation for shuffling the data for probability estimates. Ignored when probability is False.Pass an int for reproducible output across multiple function calls
	max_depth	<int>	None	Specifies the maximum depth of the tree
*	tol	<float>	1e-4	Tolerance for stopping criterion.
*	degree	<int>	3	Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.
*	gamma	{"scale", "auto"} or <float>	scale	Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.if gamma='scale' (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,if 'auto', uses 1 / n_features.
	split_criteria	{"impurity", "max_samples"}	impurity	Decides (just in case of a multi class classification) which column (class) use to split the dataset in a node**. max_samples is incompatible with 'ovo' multiclass_strategy
	criterion	{"gini", "entropy"}	entropy	The function to measure the quality of a split (only used if max_features != num_features). Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.
	min_samples_split	<int>	0	The minimum number of samples required to split an internal node. 0 (default) for any
	max_features	<int> <float> or {"auto", "sqrt", "log2"}	None	The number of features to consider when looking for the split:If int, then consider max_features features at each split.If float, then max_features is a fraction and int(max_features * n_features) features are considered at each split.If "auto", then max_features=sqrt(n_features).If "sqrt", then max_features=sqrt(n_features).If "log2", then max_features=log2(n_features).If None, then max_features=n_features.
	splitter	{"best", "random",	"random"	The strategy used to choose the feature set at each node (only used if max_features < num_features). Supported strategies are: **"best"**: sklearn SelectKBest algorithm is used in every node to choose the max_features best features. **"random"**: The algo-

* Hyperparameter used by the support vector classifier of every node

** **Splitting in a STree node**

The decision function is applied to the dataset and distances from samples to hyperplanes are computed in a matrix. This matrix has as many columns as classes the samples belongs to (if more than two, i.e. multiclass classification) or 1 column if it's a binary class dataset. In binary classification only one hyperplane is computed and therefore only one column is needed to store the distances of the samples to it. If three or more classes are present in the dataset we need as many hyperplanes as classes are there, and therefore one column per hyperplane is needed.

In case of multiclass classification we have to decide which column take into account to make the split, that depends on hyperparameter *split_criteria*, if "impurity" is chosen then STree computes information gain of every split candidate using each column and chooses the one that maximize the information gain, otherwise STree choses the column with more samples with a predicted class (the column with more positive numbers in it).

Once we have the column to take into account for the split, the algorithm splits samples with positive distances to hyperplane from the rest.

EXAMPLES

4.1 Notebooks

- Benchmark
- Some features
- Gridsearch
- Ensembles

4.2 Sample Code

```python
import time
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from stree import Stree

random_state = 1
X, y = load_iris(return_X_y=True)
Xtrain, Xtest, ytrain, ytest = train_test_split(
    X, y, test_size=0.2, random_state=random_state
)
now = time.time()
print("Predicting with max_features=sqrt(n_features)")
clf = Stree(random_state=random_state, max_features="auto")
clf.fit(Xtrain, ytrain)
print(f"Took {time.time() - now:.2f} seconds to train")
print(clf)
print(f"Classifier's accuracy (train): {clf.score(Xtrain, ytrain):.4f}")
print(f"Classifier's accuracy (test) : {clf.score(Xtest, ytest):.4f}")
print("=" * 40)
print("Predicting with max_features=n_features")
clf = Stree(random_state=random_state)
clf.fit(Xtrain, ytrain)
print(f"Took {time.time() - now:.2f} seconds to train")
print(clf)
print(f"Classifier's accuracy (train): {clf.score(Xtrain, ytrain):.4f}")
print(f"Classifier's accuracy (test) : {clf.score(Xtest, ytest):.4f}")
```

API INDEX

5.1 Stree

class stree.**Stree**(*C: float = 1.0*, *kernel: str = 'linear'*, *max_iter: int = 100000.0*, *random_state: Optional[int] = None*, *max_depth: Optional[int] = None*, *tol: float = 0.0001*, *degree: int = 3*, *gamma='scale'*, *split_criteria: str = 'impurity'*, *criterion: str = 'entropy'*, *min_samples_split: int = 0*, *max_features=None*, *splitter: str = 'random'*, *multiclass_strategy: str = 'ovo'*, *normalize: bool = False*)

Bases: sklearn.base.BaseEstimator, sklearn.base.ClassifierMixin

Estimator that is based on binary trees of svm nodes can deal with sample_weights in predict, used in boosting sklearn methods inheriting from BaseEstimator implements get_params and set_params methods inheriting from ClassifierMixin implement the attribute _estimator_type with "classifier" as value

C [float, optional] Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive., by default 1.0

kernel [str, optional] Specifies the kernel type to be used in the algorithm. It must be one of 'liblinear', 'linear', 'poly' or 'rbf'. liblinear uses [liblinear](https://www.csie.ntu.edu.tw/~cjlin/liblinear/) library and the rest uses [libsvm](https://www.csie.ntu.edu.tw/~cjlin/libsvm/) library through scikit-learn library, by default "linear"

max_iter [int, optional] Hard limit on iterations within solver, or -1 for no limit., by default 1e5

random_state [int, optional] Controls the pseudo random number generation for shuffling the data for probability estimates. Ignored when probability is False.Pass an int for reproducible output across multiple function calls, by default None

max_depth [int, optional] Specifies the maximum depth of the tree, by default None

tol [float, optional] Tolerance for stopping, by default 1e-4

degree [int, optional] Degree of the polynomial kernel function ('poly'). Ignored by all other kernels., by default 3

gamma [str, optional] Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.if gamma='scale' (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,if 'auto', uses 1 / n_features., by default "scale"

split_criteria [str, optional] Decides (just in case of a multi class classification) which column (class) use to split the dataset in a node. max_samples is incompatible with 'ovo' multiclass_strategy, by default "impurity"

criterion [str, optional] The function to measure the quality of a split (only used if max_features != num_features). Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain., by default "entropy"

min_samples_split [int, optional] The minimum number of samples required to split an internal node. 0 (default) for any, by default 0

max_features [optional] The number of features to consider when looking for the split: If int, then consider max_features features at each split. If float, then max_features is a fraction and int(max_features * n_features) features are considered at each split. If "auto", then max_features= sqrt(n_features). If "sqrt", then max_features=sqrt(n_features). If "log2", then max_features=log2(n_features). If None, then max_features= n_features., by default None

splitter [str, optional] The strategy used to choose the feature set at each node (only used if max_features < num_features). Supported strategies are: "best": sklearn SelectKBest algorithm is used in every node to choose the max_features best features. "random": The algorithm generates 5 candidates and choose the best (max. info. gain) of them. "trandom": The algorithm generates only one random combination. "mutual": Chooses the best features w.r.t. their mutual info with the label. "cfs": Apply Correlation-based Feature Selection. "fcbf": Apply Fast Correlation- Based , by default "random"

multiclass_strategy [str, optional] Strategy to use with multiclass datasets, "ovo": one versus one. "ovr": one versus rest, by default "ovo"

normalize [bool, optional] If standardization of features should be applied on each node with the samples that reach it , by default False

classes_ [ndarray of shape (n_classes,)] The classes labels.

n_classes_ [int] The number of classes

n_iter_ [int] Max number of iterations in classifier

depth_ [int] Max depht of the tree

n_features_ [int] The number of features when `fit` is performed.

n_features_in_ [int] Number of features seen during fit.

max_features_ [int] Number of features to use in hyperplane computation

tree_ [Node] root of the tree

X_ [ndarray] points to the input dataset

y_ [ndarray] points to the input labels

R. Montañana, J. A. Gámez, J. M. Puerta, "STree: a single multi-class oblique decision tree based on support vector machines.", 2021 LNAI 12882

_build_clf()
Build the right classifier for the node

_initialize_max_features() → int

_more_tags() → dict
Required by sklearn to supply features of the classifier make mandatory the labels array

> **Returns** the tag required
>
> **Return type** dict

static _reorder_results(*y: numpy.array*, *indices: numpy.array*) → numpy.array
Reorder an array based on the array of indices passed

y [np.array] data untidy

indices [np.array] indices used to set order

np.array array y ordered

_train(*X: numpy.ndarray*, *y: numpy.ndarray*, *sample_weight: numpy.ndarray*, *depth: int*, *title: str*) →
Optional[stree.Splitter.Snode]
Recursive function to split the original dataset into predictor nodes (leaves)

X [np.ndarray] samples dataset

y [np.ndarray] samples labels

sample_weight [np.ndarray] weight of samples. Rescale C per sample.

depth [int] actual depth in the tree

title [str] description of the node

Optional[Snode] binary tree

fit(*X: numpy.ndarray*, *y: numpy.ndarray*, *sample_weight: Optional[numpy.array] = None*) →
stree.Strees.Stree
Build the tree based on the dataset of samples and its labels

Stree itself to be able to chain actions: fit().predict() …

ValueError if C < 0

ValueError if max_depth < 1

ValueError if all samples have 0 or negative weights

nodes_leaves() → tuple
Compute the number of nodes and leaves in the built tree

[tuple] tuple with the number of nodes and the number of leaves

predict(*X: numpy.array*) → numpy.array
Predict labels for each sample in dataset passed

X [np.array] dataset of samples

np.array array of labels

ValueError if dataset with inconsistent number of features

NotFittedError if model is not fitted

5.2 Siterator

Oblique decision tree classifier based on SVM nodes Splitter class

class Splitter.**Siterator**(*tree:* Splitter.Snode)
Bases: `object`

Stree preorder iterator

_push(*node:* Splitter.Snode)

5.3 Snode

Oblique decision tree classifier based on SVM nodes Splitter class

class Splitter.**Snode**(*clf: sklearn.svm._classes.SVC, X: numpy.ndarray, y: numpy.ndarray, features: numpy.array, impurity: float, title: str, weight: Optional[numpy.ndarray] = None, scaler: Optional[sklearn.preprocessing._data.StandardScaler] = None*)

Bases: `object`

Nodes of the tree that keeps the svm classifier and if testing the dataset assigned to it

clf [SVC] Classifier used

X [np.ndarray] input dataset in train time (only in testing)

y [np.ndarray] input labes in train time

features [np.array] features used to compute hyperplane

impurity [float] impurity of the node

title [str] label describing the route to the node

weight [np.ndarray, optional] weights applied to input dataset in train time, by default None

scaler [StandardScaler, optional] scaler used if any, by default None

classmethod copy(*node:* Splitter.Snode) → *Splitter.Snode*

get_classifier() → sklearn.svm._classes.SVC

get_down() → *Splitter.Snode*

get_features() → numpy.array

get_impurity() → float

get_partition_column() → int

get_title() → str

get_up() → *Splitter.Snode*

is_leaf() → bool

make_predictor()
    Compute the class of the predictor and its belief based on the subdataset of the node only if it is a leaf

set_classifier(*clf*)

set_down(*son*)

set_features(*features*)

set_impurity(*impurity*)

set_partition_column(*col: int*)

set_title(*title*)

set_up(*son*)

5.4 Splitter

Oblique decision tree classifier based on SVM nodes Splitter class

class Splitter.**Splitter**(*clf: Optional[sklearn.svm._classes.SVC] = None, criterion: Optional[str] = None,*
feature_select: Optional[str] = None, criteria: Optional[str] = None,
min_samples_split: Optional[int] = None, random_state=None, normalize=False)

Bases: object

Splits a dataset in two based on different criteria

clf [SVC, optional] classifier, by default None

criterion [str, optional] The function to measure the quality of a split (only used if max_features !=
num_features). Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.,
by default "entropy", by default None

feature_select [str, optional] The strategy used to choose the feature set at each node (only used if max_features
< num_features). Supported strategies are: "best": sklearn SelectKBest algorithm is used in every node
to choose the max_features best features. "random": The algorithm generates 5 candidates and choose
the best (max. info. gain) of them. "trandom": The algorithm generates only one random combination.
"mutual": Chooses the best features w.r.t. their mutual info with the label. "cfs": Apply Correlation-based
Feature Selection. "fcbf": Apply Fast Correlation- Based, by default None

criteria [str, optional] ecides (just in case of a multi class classification) which column (class) use to split the
dataset in a node. max_samples is incompatible with 'ovo' multiclass_strategy, by default None

min_samples_split [int, optional] The minimum number of samples required to split an internal node. 0 (de-
fault) for any, by default None

random_state [optional] Controls the pseudo random number generation for shuffling the data for probability
estimates. Ignored when probability is False.Pass an int for reproducible output across multiple function
calls, by default None

normalize [bool, optional] If standardization of features should be applied on each node with the samples that
reach it , by default False

ValueError clf has to be a sklearn estimator

ValueError criterion must be gini or entropy

ValueError criteria has to be max_samples or impurity

ValueError splitter must be in {random, best, mutual, cfs, fcbf}

_distances(*node:* Splitter.Snode, *data: numpy.ndarray*) → numpy.array
Compute distances of the samples to the hyperplane of the node

node [Snode] node containing the svm classifier

data [np.ndarray] samples to compute distance to hyperplane

np.array array of shape (m, nc) with the distances of every sample to the hyperplane of every class. nc =
# of classes

static _entropy(*y: numpy.array*) → float
Compute entropy of a labels set

y [np.array] set of labels

float entropy

static **_fs_best**(*dataset: numpy.array*, *labels: numpy.array*, *max_features: int*) → tuple
>   Return the variabes with higher f-score

>   **dataset** [np.array] array of samples

>   **labels** [np.array] labels of the dataset

>   **max_features** [int] number of features of the subspace (< number of features in dataset)

>   **tuple** indices of the features selected

static **_fs_cfs**(*dataset: numpy.array*, *labels: numpy.array*, *max_features: int*) → tuple
>   Correlattion-based feature selection with max_features limit

>   **dataset** [np.array] array of samples

>   **labels** [np.array] labels of the dataset

>   **max_features** [int] number of features of the subspace (< number of features in dataset)

>   **tuple** indices of the features selected

static **_fs_fcbf**(*dataset: numpy.array*, *labels: numpy.array*, *max_features: int*) → tuple
>   Fast Correlation-based Filter algorithm with max_features limit

>   **dataset** [np.array] array of samples

>   **labels** [np.array] labels of the dataset

>   **max_features** [int] number of features of the subspace (< number of features in dataset)

>   **tuple** indices of the features selected

static **_fs_iwss**(*dataset: numpy.array*, *labels: numpy.array*, *max_features: int*) → tuple
>   Correlattion-based feature selection based on iwss with max_features limit

>   **dataset** [np.array] array of samples

>   **labels** [np.array] labels of the dataset

>   **max_features** [int] number of features of the subspace (< number of features in dataset)

>   **tuple** indices of the features selected

static **_fs_mutual**(*dataset: numpy.array*, *labels: numpy.array*, *max_features: int*) → tuple
>   Return the best features with mutual information with labels

>   **dataset** [np.array] array of samples

>   **labels** [np.array] labels of the dataset

>   **max_features** [int] number of features of the subspace (< number of features in dataset)

>   **tuple** indices of the features selected

_fs_random(*dataset: numpy.array*, *labels: numpy.array*, *max_features: int*) → tuple
>   Return the best of five random feature set combinations

>   **dataset** [np.array] array of samples

>   **labels** [np.array] labels of the dataset

>   **max_features** [int] number of features of the subspace (< number of features in dataset)

tuple indices of the features selected

static _fs_trandom(*dataset: numpy.array*, *labels: numpy.array*, *max_features: int*) → tuple
  Return the a random feature set combination

  dataset [np.array] array of samples

  labels [np.array] labels of the dataset

  max_features [int] number of features of the subspace (< number of features in dataset)

  tuple indices of the features selected

static _generate_spaces(*features: int*, *max_features: int*) → list
  Generate at most 5 feature random combinations

  features [int] number of features in each combination

  max_features [int] number of features in dataset

  list list with up to 5 combination of features randomly selected

_get_subspaces_set(*dataset: numpy.array*, *labels: numpy.array*, *max_features: int*) → tuple
  Compute the indices of the features selected by splitter depending on the self._feature_select hyper parameter

  dataset [np.array] array of samples

  labels [np.array] labels of the dataset

  max_features [int] number of features of the subspace (<= number of features in dataset)

  tuple indices of the features selected

static _gini(*y: numpy.array*) → float

_impurity(*data: numpy.array*, *y: numpy.array*) → numpy.array
  return column of dataset to be taken into account to split dataset

  data [np.array] distances to hyper plane of every class

  y [np.array] vector of labels (classes)

  np.array column of dataset to be taken into account to split dataset

static _max_samples(*data: numpy.array*, *y: numpy.array*) → numpy.array
  return column of dataset to be taken into account to split dataset

  data [np.array] distances to hyper plane of every class

  y [np.array] column of dataset to be taken into account to split dataset

  np.array column of dataset to be taken into account to split dataset

_select_best_set(*dataset: numpy.array*, *labels: numpy.array*, *features_sets: list*) → list
  Return the best set of features among feature_sets, the criterion is the information gain

  dataset [np.array] array of samples (# samples, # features)

  labels [np.array] array of labels

  features_sets [list] list of features sets to check

> **list** best feature set

get_subspace(*dataset: numpy.array*, *labels: numpy.array*, *max_features: int*) → tuple
> Re3turn a subspace of the selected dataset of max_features length. Depending on hyperparameter
>
> **dataset** [np.array] array of samples (# samples, # features)
>
> **labels** [np.array] labels of the dataset
>
> **max_features** [int] number of features to form the subspace
>
> **tuple** tuple with the dataset with only the features selected and the indices of the features selected

information_gain(*labels: numpy.array*, *labels_up: numpy.array*, *labels_dn: numpy.array*) → float
> Compute information gain of a split candidate
>
> **labels** [np.array] labels of the dataset
>
> **labels_up** [np.array] labels of one side
>
> **labels_dn** [np.array] labels on the other side
>
> **float** information gain

part(*origin: numpy.array*) → list
> Split an array in two based on indices (self._up) and its complement partition has to be called first to establish up indices
>
> **origin** [np.array] dataset to split
>
> **list** list with two splits of the array

partition(*samples: numpy.array*, *node:* Splitter.Snode, *train: bool*)
> Set the criteria to split arrays. Compute the indices of the samples that should go to one side of the tree (up)
>
> **samples** [np.array] array of samples (# samples, # features)
>
> **node** [Snode] Node of the tree where partition is going to be made
>
> **train** [bool] Train time - True / Test time - False

partition_impurity(*y: numpy.array*) → numpy.array

• genindex

PYTHON MODULE INDEX

S

Symbols

C

F

G

I

M

N

P

S