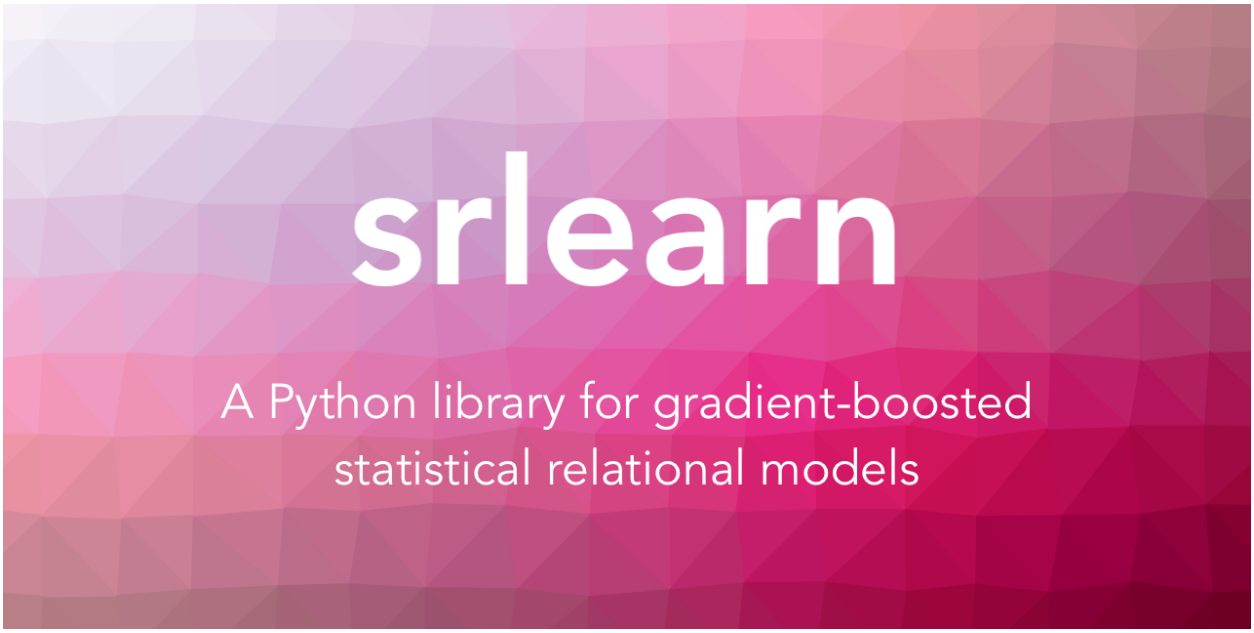
srlearn

Release 0.5.5-development

Nov 28, 2020

Getting Started


srlearn
A Python library for gradient-boosted
statistical relational models

CHAPTER 1

Getting Started

1.1 1. Prerequisites

- Java (1.8)
- Python (3.6, 3.7)
- Unix-based operating system (Linux/Mac)

If you do not have Java, you might install it with your operating system's package manager.

For example, on Ubuntu:

```
sudo apt-get install openjdk-8-jdk
```

Jenv might be a helpful way to manage Java versions as well. If you're on MacOS it's also failry easy to set up with Homebrew.

1.2 2. Installation

The package can be installed from the Python Package Index (PyPi) with `pip`.

```
pip install srlearn
```

1.3 3. Test Installation

A simple test should be whether `srlearn` can be imported:

```
>>> import srlearn
```

If you've reached this point, you should be ready for the User Guide.

CHAPTER 2

User Guide

This guide walks through how to initialize, parametrize, and invoke the core methods. It may be helpful to consult the API documentation for the following modules as you progress:

- *srlearn.Database*
- *srlearn.Background*
- *srlearn.rdn.BoostedRDN*

2.1 Parametrize the core classes

2.1.1 1. Looking at our Data

This example uses the built-in example data set: "smokes-friends-cancer". We want to model whether a person will develop cancer based on their smoking habits and their social network.

The conventional way to do machine learning might be to list the people and list their attributes. However, for social network problems it may be difficult or impossible to represent arbitrary social networks in a vector representation.

In order to get around this, we adopt Prolog clauses to represent our data:

```
>>> from srlearn import example_data
>>> for predicate in example_data.train.pos:
...     print(predicate)
...
cancer(Alice).
cancer(Bob).
cancer(Chuck).
cancer(Fred).
```

```
>>> from srlearn import example_data
>>> for predicate in example_data.train.facts:
...     print(predicate)
```

(continues on next page)

(continued from previous page)

```
...
friends(Alice, Bob).
friends(Alice, Fred).
friends(Chuck, Bob).
friends(Chuck, Fred).
friends(Dan, Bob).
friends(Earl, Bob).
friends(Bob, Alice).
friends(Fred, Alice).
friends(Bob, Chuck).
friends(Fred, Chuck).
friends(Bob, Dan).
friends(Bob, Earl).
smokes(Alice).
smokes(Chuck).
smokes(Bob).
```

Since this differs from the vector representation, this uses a *srlearn.Database* object to represent positive examples, negative examples, and facts.

2.1.2 2. Declaring our Backround Knowledge

The *srlearn.Background* object helps declare background knowledge for a domain, as well as some parameters for model learning (this last point may seem strange, but it is designed in order to remain compatible with how BoostSRL accepts background as input).

```
>>> from srlearn import Background
>>> bk = Background()
>>> print(bk)
setParam: nodeSize=2.
setParam: maxTreeDepth=3.
setParam: numberOfClauses=100.
setParam: numberOfCycles=100.
<BLANKLINE>
```

This gives us a view into some of the default parameters. However, it is missing mode declarations[1].

We can declare modes as a list of strings:

```
>>> from srlearn import Background
>>> bk = Background(
...     modes=[
...         "friends(+person,-person).",
...         "friends(-person,+person).",
...         "cancer(+person).",
...         "smokes(+person).",
...     ],
... )
```

A full description of modes and how they constrain the search space is beyond the scope of the discussion here, but further reading may be warranted[1].

[1] https://starling.utdallas.edu/software/boostsrl/wiki/basic-modes/

2.1.3 3. Initializing a Classifier

Here we will learn Relational Dependency Networks (RDNs)[2][3] as classifiers for predicting if a person in this fictional data set will develop cancer.

```
>>> from srlearn.rdn import BoostedRDN
>>> from srlearn import Background
>>> bk = Background(
...     modes=[
...         "friends(+person,-person).",
...         "friends(-person,+person).",
...         "cancer(+person).",
...         "smokes(+person).",
...     ],
... )
>>> clf = BoostedRDN()
>>> print(clf)
BoostedRDN(background=None, max_tree_depth=3, n_estimators=10, node_size=2,
           target='None')
```

This pattern should begin to look familiar if you've worked with scikit-learn before. This classifier is built on top of `sklearn.base.BaseEstimator` and `sklearn.base.ClassifierMixin`, but there are still a few things we need to declare before invoking *`srlearn.rdn.BoostedRDN.fit()`*.

Specifically, we need to include a "target" and "background" as parameters. The "background" is what we described above, and the "target" is what we aim to learn about: the **cancer** predicate.

```
>>> clf = BoostedRDN(background=bk, target="cancer")
```

2.2 Putting the pieces together

Now that we have seen each of the examples, we can put them together to learn a series of trees.

```
>>> from srlearn.rdn import BoostedRDN
>>> from srlearn import Background
>>> from srlearn import example_data
>>> bk = Background(
...     modes=[
...         "friends(+person,-person).",
...         "friends(-person,+person).",
...         "cancer(+person).",
...         "smokes(+person).",
...     ],
... )
>>> clf = BoostedRDN(background=bk, target="cancer")
>>> clf.fit(example_data.train)
BoostedRDN(background=setParam: nodeSize=2.
setParam: maxTreeDepth=3.
setParam: numberOfClauses=100.
setParam: numberOfCycles=100.
useStdLogicVariables: true.
```

(continues on next page)

[2] Sriraam Natarajan, Tushar Khot, Kristian Kersting, and Jude Shavlik, "*Boosted Statistical Relational Learners: From Benchmarks to Data-Driven Medicine*". SpringerBriefs in Computer Science, ISBN: 978-3-319-13643-1, 2015

[3] Sriraam Natarajan, Tushar Khot, Kristian Kersting, Bernd Gutmann, and Jude Shavlik, "Gradient-based boosting for statistical relational learning: The relational dependency network case". Machine Learning Journal (MLJ) 2011.

(continued from previous page)

```
mode: friends(+person,-person).
mode: friends(-person,+person).
mode: cancer(+person).
mode: smokes(+person).
'
           max_tree_depth=3, n_estimators=10, node_size=2, target='cancer')
>>> clf.predict(example_data.test)
array([ True,  True,  True, False, False])
```

2.3 Conclusion

For further reading, see the example gallery.

2.4 References

CHAPTER 3

srlearn API

3.1 Core Classes

These classes form the set of core pieces for describing the data, providing background knowledge, and learning.

Database()	Database of examples and facts.
Background([modes, ok_if_unknown, bridgers, ...])	Background Knowledge for a database.
rdn.BoostedRDN([background, target, ...])	Relational Dependency Networks Estimator
rdn.BoostedRDNRegressor([background, ...])	Relational Dependency Networks Regressor

3.1.1 srlearn.Database

class srlearn.**Database**

Database of examples and facts.

__init__()

Initialize a Database object

A database (in this respect) contains positive examples, negative examples, facts, and is augmented with background knowledge.

The implementation is done with four attributes: `pos`, `neg`, `facts`, and `modes`. Each attribute is a list that may be set by mutating, or loaded from files with *Database.from_files()*.

Examples

This initializes a Database object, then sets the `pos` attribute.

```
>>> from srlearn import Database
>>> db = Database()
>>> db.pos = ["student(alexander)."]
```

add_fact (*example: str*) → None
Append a fact to the list of facts.

add_neg (*example: str*) → None
Append a negative example to the list of negative examples.

add_pos (*example: str*) → None
Append a positive example to the list of positive examples.

static from_files (*pos='pos.pl'*, *neg='neg.pl'*, *facts='facts.pl'*, *lazy_load=True*)
Load files into a Database

Return an instance of a Database with pos, neg, and facts set to the contents of files. By default this performs a "lazy load," where the files are not loaded into Python lists, but copied at learning time.

Parameters

pos [str or pathlib.Path] Location of positive examples

neg [str or pathlib.Path] Location of negative examples

facts [str or pathlib.Path] Location of facts

lazy_load [bool (default: True)] Skip loading the files into a list

Returns

db [srlearn.Database] Instance of a Database object

write (*filename='train'*, *location=PosixPath('train')*) → None
Write the database to disk

Parameters

filename [str] Name of the file to write to: 'train' or 'test'

location [`pathlib.Path`] Path where data should be written to.

Notes

This function has polymorphic behavior. When attributes (`self.pos`, `self.neg`, `self.facts`) are lists of strings, the lists are written to files. When the attributes are (path-like) strings or pathlib Paths (`pathlib.Path`), the files are copied.

Examples using `srlearn.Database`

- *Family Relationships Domain*

3.1.2 srlearn.Background

class srlearn.**Background**(*modes=None*, *ok_if_unknown=None*, *bridgers=None*, *node_size=2*, *number_of_clauses=100*, *number_of_cycles=100*, *max_tree_depth=3*, *recursion=False*, *line_search=False*, *use_std_logic_variables=False*, *use_prolog_variables=True*, *load_all_libraries=False*, *load_all_basic_modes=False*)
Background Knowledge for a database.

Background knowledge expressed in the form of modes.

__init__(*modes=None*, *ok_if_unknown=None*, *bridgers=None*, *node_size=2*, *number_of_clauses=100*, *number_of_cycles=100*, *max_tree_depth=3*, *recursion=False*, *line_search=False*, *use_std_logic_variables=False*, *use_prolog_variables=True*, *load_all_libraries=False*, *load_all_basic_modes=False*)

Initialize a set of background knowledge

Parameters

modes [list of str (default: None)] Modes constrain the search space for hypotheses.

ok_if_unknown [list of str (default: None)] Okay if not known.

bridgers [list of str (default: None)] List of bridger predicates.

node_size [int, optional (default: 2)] Maximum number of literals in each node.

max_tree_depth [int, optional (default: 3)] Maximum number of nodes from root to leaf (height) in the tree.

number_of_clauses [int, optional (default: 100)] Maximum number of clauses in the tree (i.e. maximum number of leaves)

number_of_cycles [int, optional (default: 100)] Maximum number of times the code will loop to learn clauses, increments even if no new clauses are learned.

line_search [bool, optional (default: False)] Use lineSearch

recursion [bool, optional (default: False)] Use recursion

use_std_logic_variables [bool, optional (default: False)] Set the stdLogicVariables parameter to True

use_prolog_variables [bool, optional (default: True)] Set the usePrologVariables parameter to True

load_all_libraries [bool, optional (default: False)] Load libraries: `arithmeticInLogic`, `comparisonInLogic`, `differentInLogic`, `listsInLogic`

load_all_basic_modes [bool, optional (default: False)] Load `modes_arithmeticInLogic`, `modes_comparisonInLogic`, `modes_differentInLogic`, `modes_listsInLogic` These may require many cycles while proving.

Notes

Descriptions of these parameters are lifted almost word-for-word from the BoostSRL-Wiki "Advanced Parameters" page [1].

Some of these parameters are defined in multiple places. This is mostly to follow the sklearn-style requirement for all tune-able parameters to be part of the object while still being relatively similar to the style where BoostSRL has parameters defined in a modes file.

Examples

This demonstrates how to add parameters to the Background object. The main thing to take note of is the `modes` parameter, where background knowledge of the Toy-Cancer domain is specified.

```
>>> from srlearn import Background
>>> bk = Background(
...     modes=[
...         "cancer(+Person).",
...         "smokes(+Person).",
...         "friends(+Person,-Person).",
...         "friends(-Person,+Person).",
...     ],
...     max_tree_depth=2,
... )
>>> print(bk)
setParam: nodeSize=2.
setParam: maxTreeDepth=2.
setParam: numberOfClauses=100.
setParam: numberOfCycles=100.
useStdLogicVariables: true.
mode: cancer(+Person).
mode: smokes(+Person).
mode: friends(+Person,-Person).
mode: friends(-Person,+Person).
<BLANKLINE>
```

This Background object is used by the *srlearn.rdn.BoostedRDN* class to write the parameters to a `background.txt` file before running BoostSRL.

```
>>> from srlearn import Background
>>> from srlearn import example_data
>>> bk = Background(
...     modes=example_data.train.modes,
...     max_tree_depth=2,
...     node_size=1,
... )
>>> bk.write("training/")
```

write (*filename='train'*, *location=PosixPath('train')*) → None

Write the background to disk for learning.

Parameters

filename [str] Name of the file to write to: 'train_bk.txt' or 'test_bk.txt'

location [`pathlib.Path`] This should be handled by a manager to ensure locations do not overlap.

Examples using `srlearn.Background`

- *Estimating Feature Importance*
- *Family Relationships Domain*

3.1.3 srlearn.rdn.BoostedRDN

class srlearn.rdn.**BoostedRDN** (*background=None*, *target='None'*, *n_estimators=10*, *node_size=2*, *max_tree_depth=3*)

Relational Dependency Networks Estimator

Wrappers around BoostSRL for learning and inference with Relational Dependency Networks written with a scikit-learn-style interface derived from `sklearn.base.BaseEstimator`

Similar to `sklearn.ensemble.GradientBoostingClassifier`, this builds a model by fitting a series of regression trees.

Examples

```
>>> from srlearn.rdn import BoostedRDN
>>> from srlearn import Background
>>> from srlearn import example_data
>>> bk = Background(modes=example_data.train.modes)
>>> dn = BoostedRDN(background=bk, target="cancer")
>>> dn.fit(example_data.train)
BoostedRDN(background=setParam: nodeSize=2.
setParam: maxTreeDepth=3.
setParam: numberOfClauses=100.
setParam: numberOfCycles=100.
useStdLogicVariables: true.
mode: friends(+Person,-Person).
mode: friends(-Person,+Person).
mode: smokes(+Person).
mode: cancer(+Person).
,
           max_tree_depth=3, n_estimators=10, node_size=2, target='cancer')
>>> dn.predict(example_data.test)
array([ True,  True,  True, False, False])
```

__init__ (*background=None*, *target='None'*, *n_estimators=10*, *node_size=2*, *max_tree_depth=3*)
Initialize a BoostedRDN

Parameters

background [`srlearn.background.Background` (default: None)] Background knowledge with respect to the database

target [str (default: "None")] Target predicate to learn

n_estimators [int, optional (default: 10)] Number of trees to fit

node_size [int, optional (default: 2)] Maximum number of literals in each node.

max_tree_depth [int, optional (default: 3)] Maximum number of nodes from root to leaf (height) in the tree.

Attributes

estimators_ [array, shape (n_estimators)] Return the boosted regression trees

feature_importances_ [array, shape (n_features)] Return the feature importances (based on how often each feature appears)

feature_importances_
Return the features contained in a tree.

Parameters

tree_number: int Index of the tree to read.

fit (*database*)
Learn structure and parameters.

Fit the structure and parameters of a Relational Dependency Network using a database of positive examples, negative examples, facts, and any relevant background knowledge.

Parameters

database [srlearn.database.Database] Database containing examples and facts.

Returns

self [object] Returns self.

Notes

The underlying algorithm is based on the "Relational Functional Gradient Boosting" as described in [1].

This fit function is based on subprocess calling the BoostSRL jar files. This will require a Java runtime to also be available. See [2].

from_json(*file_name*)
Load a learned model from json.

Parameters

file_name [str (or pathlike)] Path to a saved json file.

get_params(*deep=True*)
Get parameters for this estimator.

Parameters

deep [bool, default=True] If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params [mapping of string to any] Parameter names mapped to their values.

predict(*database*)
Use the learned model to predict on new data.

Parameters

database [*srlearn.Database*] Database containing examples and facts.

Returns

results [ndarray] Positive or negative class.

predict_proba(*database*)
Return class probabilities.

Parameters

database [*srlearn.Database*] Database containing examples and facts.

Returns

results [ndarray] Probability of belonging to the positive class

score(*X*, *y*, *sample_weight=None*)
Return the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like of shape (n_samples, n_features)] Test samples.

y [array-like of shape (n_samples,) or (n_samples, n_outputs)] True labels for X.

sample_weight [array-like of shape (n_samples,), default=None] Sample weights.

Returns

score [float] Mean accuracy of self.predict(X) wrt. y.

set_params (***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters

****params** [dict] Estimator parameters.

Returns

self [object] Estimator instance.

to_json (*file_name*) → None
Serialize a learned model to json.

Parameters

file_name [str (or pathlike)] Path to a saved json file.

Examples using `srlearn.rdn.BoostedRDN`

- *Estimating Feature Importance*
- *Family Relationships Domain*

3.1.4 `srlearn.rdn`.BoostedRDNRegressor

class srlearn.rdn.**BoostedRDNRegressor**(*background=None*, *target='None'*, *n_estimators=10*, *node_size=2*, *max_tree_depth=3*)
Relational Dependency Networks Regressor

Wrappers around BoostSRL for learning and inference of RDNs for regression task.

Similar to `sklearn.ensemble.GradientBoostingRegressor`, this builds a model by fitting a series of regression trees.

Examples

```
>>> from srlearn.rdn import BoostedRDNRegressor
>>> from srlearn import Background
>>> from srlearn import Database
>>> train = Database.from_files(
        pos="./datasets/Boston/train/pos.pl",
        neg="./datasets/Boston/train/neg.pl",
        facts="./datasets/Boston/train/facts.pl",
        lazy_load=False
```

(continues on next page)

(continued from previous page)

```
    )
>>> test = Database.from_files(
        pos="./datasets/Boston/test/pos.pl",
        neg="./datasets/Boston/test/neg.pl",
        facts="./datasets/Boston/test/facts.pl",
        lazy_load=False
    )
>>> train.modes = ["crim(+id,#varsrim).",
        "zn(+id,#varzn).",
        "indus(+id,#varindus).",
        "chas(+id,#varchas).",
        "nox(+id,#varnox).",
        "rm(+id,#varrm).",
        "age(+id,#varage).",
        "dis(+id,#vardis).",
        "rad(+id,#varrad).",
        "tax(+id,#vartax).",
        "ptratio(+id,#varptrat).",
        "b(+id,#varb).",
        "lstat(+id,#varlstat).",
        "medv(+id)."]
>>> bk = Background(modes=train.modes)
>>> reg = BoostedRDNRegressor(background=bk, target="medv", n_estimators=20)
>>> reg.fit(train)
BoostedRegressionTrees(background=setParam: nodeSize=2.
setParam: maxTreeDepth=3.
setParam: numberOfClauses=100.
setParam: numberOfCycles=100.
mode: crim(+id,#varsrim).
mode: zn(+id,#varzn).
mode: indus(+id,#varindus).
mode: chas(+id,#varchas).
mode: nox(+id,#varnox).
mode: rm(+id,#varrm).
mode: age(+id,#varage).
mode: dis(+id,#vardis).
mode: rad(+id,#varrad).
mode: tax(+id,#vartax).
mode: ptratio(+id,#varptrat).
mode: b(+id,#varb).
mode: lstat(+id,#varlstat).
mode: medv(+id).
,
                       n_estimators=20, target='medv')
>>> reg.predict(test)
array([10.04313307 13.55804603 20.549378   18.14681934 23.9393469  10.01292162
     29.83298024 20.34668817 27.81642572 32.04067867  9.41342835 20.975001
     19.21966845])
```

__init__(*background=None*, *target='None'*, *n_estimators=10*, *node_size=2*, *max_tree_depth=3*)

Initialize a BoostedRDN

Parameters

background [`srlearn.background.Background` (default: None)] Background knowledge with respect to the database

target [str (default: "None")] Target predicate to learn

n_estimators [int, optional (default: 10)] Number of trees to fit

node_size [int, optional (default: 2)] Maximum number of literals in each node.

max_tree_depth [int, optional (default: 3)] Maximum number of nodes from root to leaf (height) in the tree.

Attributes

estimators_ [array, shape (n_estimators)] Return the boosted regression trees

feature_importances_ [array, shape (n_features)] Return the feature importances (based on how often each feature appears)

feature_importances_
Return the features contained in a tree.

Parameters

tree_number: int Index of the tree to read.

fit(*database*)
Learn structure and parameters.

Fit the structure and parameters of a Relational Dependency Network using a database of positive examples, negative examples, facts, and any relevant background knowledge.

Parameters

database [`srlearn.database.Database`] Database containing examples and facts.

Returns

self [object] Returns self.

Notes

The underlying algorithm is based on the "Relational Functional Gradient Boosting" as described in [1] and **[2]**_.

This fit function is based on subprocess calling the BoostSRL jar files. This will require a Java runtime to also be available. See [R81fc0061240d-3].

– [2] https://starling.utdallas.edu/software/boostsrl/wiki/regression/ .. [R81fc0061240d-3] https://starling.utdallas.edu/software/boostsrl/

from_json(*file_name*)
Load a learned model from json.

Parameters

file_name [str (or pathlike)] Path to a saved json file.

get_params(*deep=True*)
Get parameters for this estimator.

Parameters

deep [bool, default=True] If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params [mapping of string to any] Parameter names mapped to their values.

predict (*database*)
Use the learned model to predict values on new data.

Parameters

database [*srlearn.Database*] Database containing examples and facts.

Returns

pred [ndarray] regression value predicted for each example.

score (*X*, *y*, *sample_weight=None*)
Return the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like of shape (n_samples, n_features)] Test samples.

y [array-like of shape (n_samples,) or (n_samples, n_outputs)] True labels for X.

sample_weight [array-like of shape (n_samples,), default=None] Sample weights.

Returns

score [float] Mean accuracy of self.predict(X) wrt. y.

set_params (***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters

****params** [dict] Estimator parameters.

Returns

self [object] Estimator instance.

to_json (*file_name*) → None
Serialize a learned model to json.

Parameters

file_name [str (or pathlike)] Path to a saved json file.

3.2 Data Sets

example_data	A BoostSRL database with the "Toy-Cancer" dataset.

3.2.1 srlearn.example_data

A BoostSRL database with the "Toy-Cancer" dataset.

Importing this module makes `example_data.train` and `example_data.test` available in the namespace.

Examples

```
>>> from srlearn import example_data
>>> print(example_data.train)
Positive Examples:
['cancer(alice).', 'cancer(bob).', 'cancer(chuck).', 'cancer(fred).']
Negative Examples:
['cancer(dan).', 'cancer(earl).']
Facts:
['friends(alice, bob).', 'friends(alice, fred).', ..., 'smokes(bob).']
```

Ellipsis added to the facts for easier reading.

```
>>> from srlearn import example_data
>>> print(example_data.test)
Positive Examples:
['cancer(zod).', 'cancer(xena).', 'cancer(yoda).']
Negative Examples:
['cancer(voldemort).', 'cancer(watson).']
Facts:
['friends(zod, xena).', 'friends(xena, watson).', ..., 'smokes(yoda).']
```

Examples using `srlearn.example_data`

- *Smokes-Friends-Cancer*

3.3 Utilities

Some of these are for behind-the-scenes operations, but tend to be useful for further development (contributions are welcome!).

base.BaseBoostedRelationalModel([...])	Base class for deriving boosted relational models
system_manager.FileSystem()	BoostSRL File System

3.3.1 `srlearn.base`.BaseBoostedRelationalModel

class srlearn.base.**BaseBoostedRelationalModel**(*background=None*, *target='None'*, *n_estimators=10*, *node_size=2*, *max_tree_depth=3*)

Base class for deriving boosted relational models

This class extends `sklearn.base.BaseEstimator` and `sklearn.base.ClassifierMixin` while providing several utilities for instantiating a model and performing learning/inference with the BoostSRL jar files.

Note: This is not a complete treatment of *how to derive estimators*. Contributions would be appreciated.

Examples

The actual *srlearn.rdn.BoostedRDN* is derived from this class, so this example is similar to the implementation (but the actual implementation passes model parameters instead of leaving them with the defaults). This example derives a new class `BoostedRDN`, which inherits the default values of the superclass while also setting a 'special_parameter' which may be unique to this model.

All that remains is to implement the specific cases of `fit()`, `predict()`, and `predict_proba()`.

```
>>> from srlearn.base import BaseBoostedRelationalModel
>>> class BoostedRDN(BaseBoostedRelationalModel):
...     def __init__(self, special_parameter=5):
...         super().__init__(self)
...         self.special_parameter = special_parameter
...
>>> dn = BoostedRDN(special_parameter=8)
>>> print(dn)
BoostedRDN(special_parameter=8)
>>> print(dn.n_estimators)
10
```

__init__ (*background=None*, *target='None'*, *n_estimators=10*, *node_size=2*, *max_tree_depth=3*)
Initialize a BaseEstimator

feature_importances_
Return the features contained in a tree.

Parameters

tree_number: int Index of the tree to read.

from_json (*file_name*)
Load a learned model from json.

Parameters

file_name [str (or pathlike)] Path to a saved json file.

get_params (*deep=True*)
Get parameters for this estimator.

Parameters

deep [bool, default=True] If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params [mapping of string to any] Parameter names mapped to their values.

score (*X*, *y*, *sample_weight=None*)
Return the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

X [array-like of shape (n_samples, n_features)] Test samples.

y [array-like of shape (n_samples,) or (n_samples, n_outputs)] True labels for X.

sample_weight [array-like of shape (n_samples,), default=None] Sample weights.

Returns

score [float] Mean accuracy of self.predict(X) wrt. y.

set_params (***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters

****params** [dict] Estimator parameters.

Returns

self [object] Estimator instance.

to_json (*file_name*) → None
Serialize a learned model to json.

Parameters

file_name [str (or pathlike)] Path to a saved json file.

Examples using `srlearn.base.BaseBoostedRelationalModel`

- *Smokes-Friends-Cancer*

3.3.2 `srlearn.system_manager`.FileSystem

class srlearn.system_manager.**FileSystem**
BoostSRL File System

BoostSRL has an implicit assumption that it has access to a file system. At runtime it needs to both read and write with files. This object provides a view into the files needed, requests files from the operating system, and (most importantly) prepares and cleans up the file system at allocation/de-allocation time.

Notes

Ideally, each instance of a *srlearn.rdn.BoostedRDN* should have its own directory where it can operate independently. But this can be problematic and will often lead to duplicated data and other problems if multiple models are learned in parallel on the same database.

Another option (which may be more suited to parallel tree learning) would be to store data in a single location, but write the log files and models to separate locations.

Examples

This first example may not appear to do much, but behind the scenes it is creating directories for each instance of `FileSystem`, and removing them upon `exit()`.

```
>>> from srlearn.system_manager import FileSystem
>>> systems = []
>>> for _ in range(5):
...     systems.append(FileSystem())
```

Attributes

files [enum.Enum] Enum providing key,value pairs for a BoostSRL database

__init__()
Initialize a BoostSRL File System.

This will create directories that are cleaned up when the instance is de-allocated.

plotting.export_digraph(booster[, ...])	Create a digraph representation of a tree.
plotting.plot_digraph(dot_string[, format])	Plot a digraph as an image.
system_manager.reset([soft])	Reset the FileSystem

3.3.3 srlearn.plotting.export_digraph

srlearn.plotting.**export_digraph**(*booster*, *tree_index=0*, *out_file=None*)
Create a digraph representation of a tree.

Parameters

booster [BaseBoostedRelationalModel] Model to create a tree from

tree_index [int] Index of the tree to visualize

out_file [str, pathlike, or None] Handle or name of the output file. If None, returns a string

Examples

This can be used in two ways: returning a string, or directly writing the result to a file.

```
from srlearn.rdn import BoostedRDN
from srlearn import Background
from srlearn import example_data
from srlearn.plotting import export_digraph

bkg = Background(
    modes=example_data.train.modes,
)

clf = BoostedRDN(
    background=bkg,
    target="cancer",
)

clf.fit(example_data.train)

print(export_digraph(clf, tree_index=0))
```

3.3.4 srlearn.plotting.plot_digraph

srlearn.plotting.**plot_digraph**(*dot_string*, *format='png'*)
Plot a digraph as an image.

Parameters

dot_string [str] String representing a dot

format [str] Format passed to Source (default: `png`)

Returns

source [graphviz.files.Source] Graphviz `Source` object

3.3.5 srlearn.system_manager.reset

srlearn.system_manager.**reset**(*soft=False*)

Reset the FileSystem

In some circumstances, a *FileSystem* object may not properly clean up temporary files on object deallocation. This function performs a hard reset by explicitly removing the directory tree from the operating system.

Parameters

soft [bool (Default: False)] A *soft reset* reports the contents without removing.

Returns

results [list] A "soft reset" returns a list of the contents. A "hard reset" returns an empty list.

Notes

Since the FileSystem object reads and writes from a location alongside the package, these files are also removed when the package is uninstalled. But uninstalling and reinstalling is overkill when a solution like this is available.

Examples

This method has a few uses. This could be used for cleaning up your operating system in the event of failure. It could be a shorthand method for triggering behavior when the data directory is/not empty. Or the obvious case where you need to ensure temporary files are gone.

1. Use a "soft reset" to list contents without removing:

```
$ python -c "from srlearn import system_manager; print(system_manager.
↪reset(soft=True))"
['data0', 'data1']
```

2. Trigger conditional behavior. Here we report that the directory is empty.

```
>>> from srlearn import system_manager
>>> if not system_manager.reset(soft=True):
...     print("Currently Empty")
Currently Empty
```

3. Use a hard reset to remove any temporary files.

```
>>> from srlearn import system_manager
>>> system_manager.reset()
[]
```

3.4 Deprecated boostsrl objects

This is the old API style that has been deprecated. It is no longer tested or actively developed and is pending removal in 0.6.0.

srlearn.boostsrl	(Deprecated) boostsrl class for training and testing.

3.4.1 srlearn.boostsrl

(Deprecated) boostsrl class for training and testing.

Warning: This module is deprecated, pending removal in 0.6.0. See `srlearn.rdn` instead.

Functions

call_process(call)	Deprecated since version 0.5.0.
example_data(example)	Deprecated since version 0.5.0.
inspect_example_syntax(example)	Deprecated since version 0.5.0.
inspect_mode_syntax(example)	Deprecated since version 0.5.0.
write_to_file(content, path)	Deprecated since version 0.5.0.

Classes

modes(background, target[, bridgers, ...])	
test(model, test_pos, test_neg, test_facts)	Deprecated since version 0.5.0.
train(background, train_pos, train_neg, ...)	Deprecated since version 0.5.0.

srlearn.boostsrl.**call_process**(*call*)

Deprecated since version 0.5.0: Not intended as a public method.

Create a subprocess and wait for it to finish. Raise an Exception if errors occur.

srlearn.boostsrl.**example_data**(*example*)

Deprecated since version 0.5.0: Use *srlearn.example_data* instead.

For demo purposes, include some sample data.

```
from srlearn.boostsrl import example_data
train_pos = example_data('train_pos')
train_neg = example_data('train_neg')
train_facts = example_data('train_facts')
```

srlearn.boostsrl.**inspect_example_syntax**(*example*)
Deprecated since version 0.5.0: Not intended as a public method.

Uses a regular expression to check whether all of the examples in a list are in the correct form.

srlearn.boostsrl.**inspect_mode_syntax**(*example*)
Deprecated since version 0.5.0: Not intended as a public method.

Uses a regular expression to check whether all of the examples in a list are in the correct form.

class srlearn.boostsrl.**test**(*model*, *test_pos*, *test_neg*, *test_facts*, *trees=10*)
Deprecated since version 0.5.0: Use `srlearn.rdn` instead.

inference_results(*target*)
Converts BoostSRL results into a Python dictionary.

class srlearn.boostsrl.**train**(*background*, *train_pos*, *train_neg*, *train_facts*, *save=False*, *advice=False*, *softm=False*, *alpha=0.5*, *beta=-2*, *trees=10*)
Deprecated since version 0.5.0: Use `srlearn.rdn` instead.

traintime() → float
Returns a float representing seconds.

tree(*treenumber*, *target*, *image=False*)

srlearn.boostsrl.**write_to_file**(*content*, *path*)
Deprecated since version 0.5.0: Not intended as a public method.

Takes a list (content) and a path/file (path) and writes each line of the list to the file location.

CHAPTER 4

Basic Examples

These are some simple examples demonstrating `srlearn`.

4.1 Estimating Feature Importance

This demonstrates how to estimate feature importance based on how often features are used as splitting criteria.

```
from srlearn.rdn import BoostedRDN
from srlearn import Background
from srlearn.database import Database

import numpy as np
import matplotlib.pyplot as plt

webkb_train = Database.from_files(
    pos="../../datasets/webkb/train1/train1_pos.txt",
    neg="../../datasets/webkb/train1/train1_neg.txt",
    facts="../../datasets/webkb/train1/train1_facts.txt",
)

bkg = Background(
    modes=[
        "courseprof(-Course, +Person).",
        "courseprof(+Course, -Person).",
        "courseta(+Course, -Person).",
        "courseta(-Course, +Person).",
        "project(-Proj, +Person).",
        "project(+Proj, -Person).",
        "sameperson(-Person, +Person).",
        "faculty(+Person).",
        "student(+Person).",
    ],
    number_of_clauses=8,
```

(continues on next page)

(continued from previous page)

```
)

clf = BoostedRDN(
    background=bkg,
    target="faculty",
    max_tree_depth=3,
    node_size=3,
    n_estimators=10,
)

clf.fit(webkb_train)
print(clf.feature_importance())
```

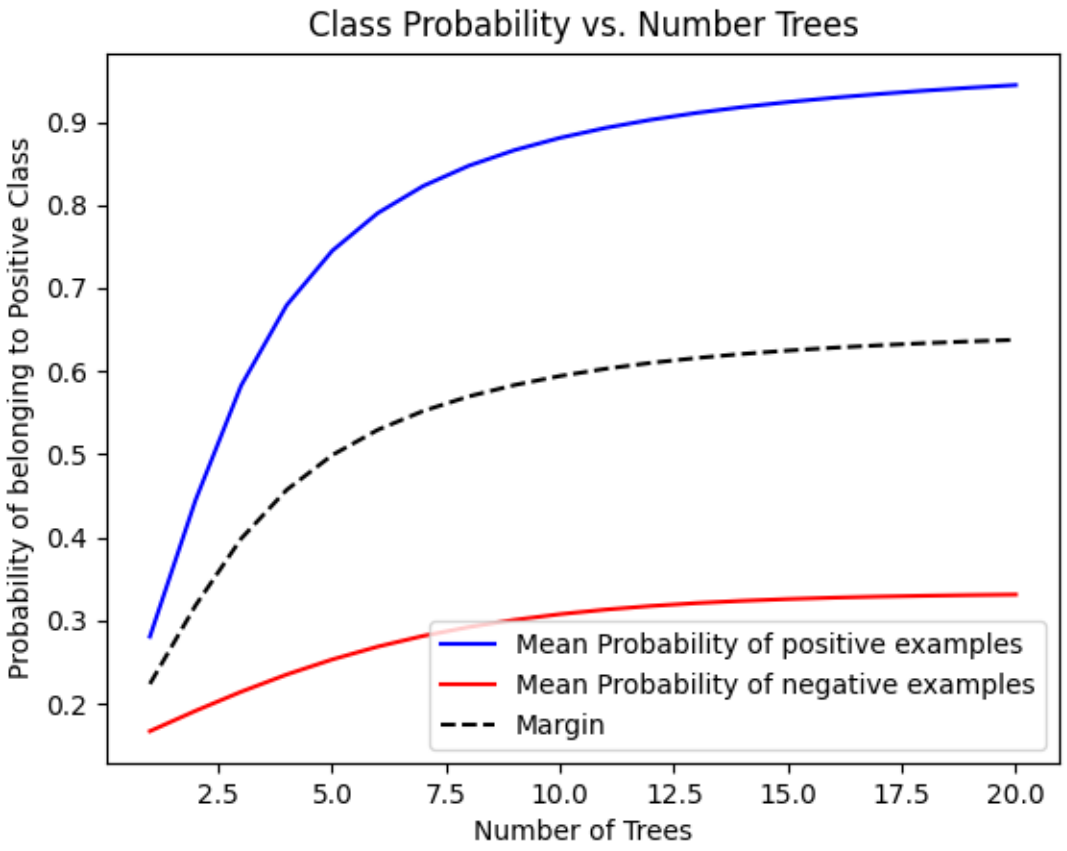
Total running time of the script: (0 minutes 0.000 seconds)

4.2 Smokes-Friends-Cancer

The smokes-friends-cancer example is a common first example in probabilistic relational models, here we use this set to learn a Relational Dependency Network (*srlearn.rdn.BoostedRDN*).

This makes use of *srlearn.example_data*, which provides two *srlearn.Database* objects named `example_data.train` and `example_data.test`.

This shows how the margin between positive and negative examples is maximized as the number of iterations of boosting increases.




```
from srlearn.rdn import BoostedRDN
from srlearn import Background
from srlearn import example_data

import numpy as np
import matplotlib.pyplot as plt

bk = Background(
    modes=example_data.train.modes,
)

clf = BoostedRDN(
    background=bk,
    target="cancer",
    max_tree_depth=2,
    node_size=2,
    n_estimators=20,
)

clf.fit(example_data.train)

x = np.arange(1, 21)
y_pos = []
y_neg = []
thresholds = []
```

(continues on next page)

(continued from previous page)

```
for n_trees in x:
    clf.set_params(n_estimators=n_trees)
    probs = clf.predict_proba(example_data.test)

    thresholds.append(clf.threshold_)
    y_pos.append(np.mean(probs[np.nonzero(clf.classes_)]))
    y_neg.append(np.mean(probs[clf.classes_ == 0]))

thresholds = np.array(thresholds)
y_pos = np.array(y_pos)
y_neg = np.array(y_neg)

plt.plot(x, y_pos, "b-", label="Mean Probability of positive examples")
plt.plot(x, y_neg, "r-", label="Mean Probability of negative examples")
plt.plot(x, thresholds, "k--", label="Margin")
plt.title("Class Probability vs. Number Trees")
plt.xlabel("Number of Trees")
plt.ylabel("Probability of belonging to Positive Class")
plt.legend(loc="best")
plt.show()
```

Total running time of the script: (0 minutes 31.251 seconds)

4.3 Family Relationships Domain

This example motivates learning about family relationships from examples of *Harry Potter* characters, then applies those rules to characters from *Pride and Prejudice*.

```
from srlearn.rdn import BoostedRDN
from srlearn import Background
from srlearn import Database

train_db = Database()
train_db.pos = [
    "father(harrypotter,jamespotter).",
    "father(dracomalfoy,luciusmalfoy).",
    "father(ginnyweasley,arthurweasley).",
    "father(ronweasley,arthurweasley).",
    "father(fredweasley,arthurweasley).",
]
train_db.neg = [
    "father(harrypotter,mrgranger).",
    "father(harrypotter,mrsgranger).",
    "father(georgeweasley,xenophiliuslovegood).",
    "father(luciusmalfoy,xenophiliuslovegood).",
    "father(harrypotter,hagrid).",
    "father(ginnyweasley,dracomalfoy).",
    "father(hagrid,dracomalfoy).",
    "father(hagrid,dumbledore).",
    "father(lunalovegood,dumbledore).",
    "father(hedwig,narcissamalfoy).",
    "father(hedwig,lunalovegood).",
    "father(ronweasley,hedwig).",
```

(continues on next page)

(continued from previous page)

```
    "father(mollyweasley,cygnusblack).",
    "father(arthurweasley,mollyweasley).",
    "father(georgeweasley,fredweasley).",
    "father(fredweasley,georgeweasley).",
    "father(ronweasley,georgeweasley).",
    "father(ronweasley,hermione).",
    "father(dracomalfoy,narcissamalfoy).",
    "father(hermione,mrsgranger).",
    "father(ginnyweasley,mollyweasley).",
]
train_db.facts = [
    "male(mrgranger).",
    "male(jamespotter).",
    "male(harrypotter).",
    "male(luciusmalfoy).",
    "male(dracomalfoy).",
    "male(arthurweasley).",
    "male(ronweasley).",
    "male(fredweasley).",
    "male(georgeweasley).",
    "male(hagrid).",
    "male(dumbledore).",
    "male(xenophiliuslovegood).",
    "male(cygnusblack).",
    "siblingof(ronweasley,fredweasley).",
    "siblingof(ronweasley,georgeweasley).",
    "siblingof(ronweasley,ginnyweasley).",
    "siblingof(fredweasley,ronweasley).",
    "siblingof(fredweasley,georgeweasley).",
    "siblingof(fredweasley,ginnyweasley).",
    "siblingof(georgeweasley,ronweasley).",
    "siblingof(georgeweasley,fredweasley).",
    "siblingof(georgeweasley,ginnyweasley).",
    "siblingof(ginnyweasley,ronweasley).",
    "siblingof(ginnyweasley,fredweasley).",
    "siblingof(ginnyweasley,georgeweasley).",
    "childof(mrgranger,hermione).",
    "childof(mrsgranger,hermione).",
    "childof(jamespotter,harrypotter).",
    "childof(lilypotter,harrypotter).",
    "childof(luciusmalfoy,dracomalfoy).",
    "childof(narcissamalfoy,dracomalfoy).",
    "childof(arthurweasley,ronweasley).",
    "childof(mollyweasley,ronweasley).",
    "childof(arthurweasley,fredweasley).",
    "childof(mollyweasley,fredweasley).",
    "childof(arthurweasley,georgeweasley).",
    "childof(mollyweasley,georgeweasley).",
    "childof(arthurweasley,ginnyweasley).",
    "childof(mollyweasley,ginnyweasley).",
    "childof(xenophiliuslovegood,lunalovegood).",
    "childof(cygnusblack,narcissamalfoy).",
]

bk = Background(
    modes=[
        "male(+name).",
```

(continues on next page)

(continued from previous page)

```
        "father(+name,+name).",
        "childof(+name,+name).",
        "siblingof(+name,+name)."
    ],
    number_of_clauses=8,
)

clf = BoostedRDN(
    background=bk,
    target="father",
    n_estimators=5,
)

clf.fit(train_db)

test_db = Database()

test_db.pos = [
    "father(elizabeth,mrbennet).",
    "father(jane,mrbennet).",
    "father(charlotte,mrlucas).",
]

test_db.neg = [
    "father(charlotte,mrsbennet).",
    "father(jane,mrlucas).",
    "father(mrsbennet,mrbennet).",
    "father(jane,elizabeth).",
]

test_db.facts = [
    "male(mrbennet).",
    "male(mrlucas).",
    "male(darcy).",
    "childof(mrbennet,elizabeth).",
    "childof(mrsbennet,elizabeth).",
    "childof(mrbennet,jane).",
    "childof(mrsbennet,jane).",
    "childof(mrlucas,charlotte).",
    "childof(mrslucas,charlotte).",
    "siblingof(jane,elizabeth).",
    "siblingof(elizabeth,jane).",
]

print(clf.predict_proba(test_db))
```

Total running time of the script: (0 minutes 0.000 seconds)

Questions? Contact Alexander L. Hayes (hayesall@iu.edu)

CHAPTER 5

Getting started

Prerequisites and installation instructions for getting started with this package.

CHAPTER 6

User guide

Guide to instantiate, parametrize, and invoke the core methods using a built-in data set.

CHAPTER 7

API documentation

Full documentation for the modules.

CHAPTER 8

Example gallery

A gallery of examples with figures and expected outputs. It complements and extends past the basic example from the User Guide.

Bibliography

[1] https://starling.utdallas.edu/software/boostsrl/wiki/advanced-parameters/

[1] Sriraam Natarajan, Tushar Khot, Kristian Kersting, and Jude Shavlik, "*Boosted Statistical Relational Learners: From Benchmarks to Data-Driven Medicine*". SpringerBriefs in Computer Science, ISBN: 978-3-319-13643-1, 2015

[2] https://starling.utdallas.edu/software/boostsrl/

[1] Sriraam Natarajan, Tushar Khot, Kristian Kersting, and Jude Shavlik, "*Boosted Statistical Relational Learners: From Benchmarks to Data-Driven Medicine*". SpringerBriefs in Computer Science, ISBN: 978-3-319-13643-1, 2015

Python Module Index

S

Index

Symbols

A

B

C

D

E

F

G

I

P

R

S

T

W