srlearn

Release 0.5.5

Apr 15, 2022

Getting Started

1 Introduction **3**
- 1.1 Getting Started . . . 3
- 1.2 User Guide . . . 4
- 1.3 srlearn API . . . 7
- 1.4 Basic Examples . . . 22

2 Getting started **31**

3 User guide **33**

4 API documentation **35**

5 Example gallery **37**

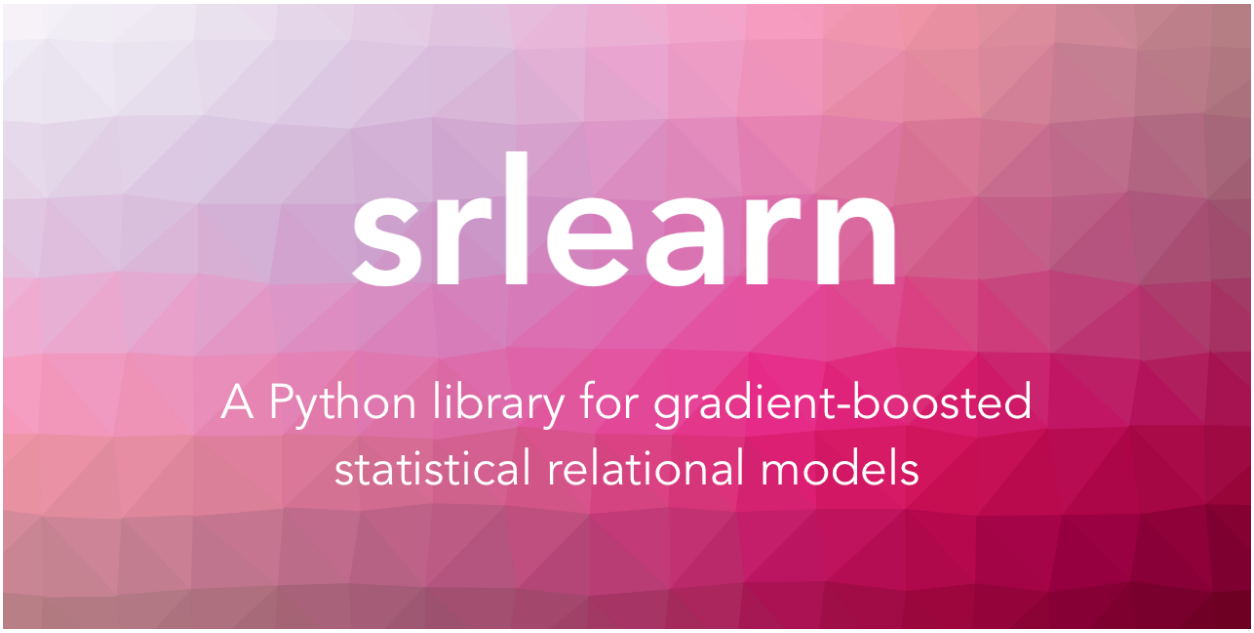
6 Citing **39**

7 Contributing **41**

Bibliography **43**

Python Module Index **45**

Index **47**


srlearn
A Python library for gradient-boosted
statistical relational models

CHAPTER 1

Introduction

srlearn is a project and set of packages for *statistical relational artificial intelligence.*

Standard machine learning tends to focus on learning and inference inside of a *feature-vector* (fit a model such that $\boldsymbol{X}$ predicts y). *Statistical Relational Learning* attempts to generalize this to arbitrary graph and hypergraph data: where the prediction problem may include a set of objects with attributes and relations on those objects.

```
from srlearn.rdn import BoostedRDNClassifier
from srlearn import Background
from srlearn.datasets import load_toy_cancer
train, test = load_toy_cancer()
bk = Background(modes=train.modes)
clf = BoostedRDNClassifier(
    background=bk,
    target='cancer',
)
clf.fit(train)
clf.predict_proba(test)
# array([0.88079619, 0.88079619, 0.88079619, 0.3075821 , 0.3075821 ])
print(clf.classes_)
# array([1., 1., 1., 0., 0.])
```

1.1 Getting Started

1.1.1 1. Prerequisites

- Java (1.8)
- Python (3.6, 3.7)

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
```

macOS:

```
brew install openjdk
```

Windows (with Chocolately):

```
choco install openjdk
```

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.

1.1.2 2. Installation

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

```
pip install srlearn
```

1.1.3 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.

1.2 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.BoostedRDNClassifier*

1.2.1 Parametrize the core classes

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.datasets import load_toy_cancer
>>> train, _ = load_toy_cancer()
>>> for predicate in train.pos:
...     print(predicate)
...
cancer(alice).
cancer(bob).
cancer(chuck).
cancer(fred).
```

```
>>> from srlearn.datasets import load_toy_cancer
>>> train, _ = load_toy_cancer()
>>> for predicate in train.facts:
...    print(predicate)
...
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.

1. 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: numOfClauses=100.
setParam: numOfCycles=100.
usePrologVariables: true.
setParam: nodeSize=2.
setParam: maxTreeDepth=3.
<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:

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

```
>>> 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].

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 BoostedRDNClassifier
>>> from srlearn import Background
>>> bk = Background(
...     modes=[
...         "friends(+person,-person).",
...         "friends(-person,+person).",
...         "cancer(+person).",
...         "smokes(+person).",
...     ],
... )
>>> clf = BoostedRDNClassifier()
>>> print(clf)
BoostedRDNClassifier(background=None, n_estimators=10, neg_pos_ratio=2, solver=
↪'BoostSRL', 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.BoostedRDNClassifier.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 = BoostedRDNClassifier(background=bk, target="cancer")
```

1.2.2 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 BoostedRDNClassifier
>>> from srlearn import Background
>>> from srlearn.datasets import load_toy_cancer
>>> train, test = load_toy_cancer()
```

(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)

```
>>> bk = Background(
...     modes=[
...         "friends(+person,-person).",
...         "friends(-person,+person).",
...         "cancer(+person).",
...         "smokes(+person).",
...     ],
... )
>>> clf = BoostedRDNClassifier(background=bk, target="cancer")
>>> clf.fit(train)
BoostedRDNClassifier(background=setParam: numOfClauses=100.
setParam: numOfCycles=100.
usePrologVariables: true.
setParam: nodeSize=2.
setParam: maxTreeDepth=3.
mode: friends(+person,-person).
mode: friends(-person,+person).
mode: cancer(+person).
mode: smokes(+person).
, n_estimators=10, neg_pos_ratio=2, solver='BoostSRL', target='cancer')
>>> clf.predict(test)
array([ True,  True,  True, False, False])
```

1.2.3 Conclusion

For further reading, see the example gallery.

1.2.4 References

1.3 srlearn API

1.3.1 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, ...])	Background Knowledge for a database.
rdn.BoostedRDNClassifier([background, ...])	Relational Dependency Networks Estimator
rdn.BoostedRDNRegressor([background, ...])	Relational Dependency Networks Regressor

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)."]
```

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`

- *Smokes-Friends-Cancer*
- *Family Relationships Domain*

srlearn.Background

class srlearn.**Background**(*, *modes=None*, *ok_if_unknown=None*, *bridgers=None*, *ranges=None*, *number_of_clauses=100*, *number_of_cycles=100*, *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*, *ranges=None*, *number_of_clauses=100*, *number_of_cycles=100*, *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.

ranges [dict of str (default: None)] Dict mapping object types to discrete categories

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).",
...     ],
... )
>>> print(bk)
setParam: numOfClauses=100.
setParam: numOfCycles=100.
usePrologVariables: true.
setParam: nodeSize=2.
setParam: maxTreeDepth=3.
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.datasets import load_toy_cancer
>>> train, _ = load_toy_cancer()
>>> bk = Background(modes=train.modes)
>>> bk.write("training/")    # doctest: +SKIP
```

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*
- *Smokes-Friends-Cancer*
- *Family Relationships Domain*

srlearn.rdn.BoostedRDNClassifier

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

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 BoostedRDNClassifier
>>> from srlearn import Background
>>> from srlearn.datasets import load_toy_cancer
>>> train, test = load_toy_cancer()
>>> bk = Background(modes=train.modes)
>>> dn = BoostedRDNClassifier(background=bk, target="cancer")
>>> dn.fit(train)
BoostedRDNClassifier(background=setParam: numOfClauses=100.
setParam: numOfCycles=100.
usePrologVariables: true.
setParam: nodeSize=2.
setParam: maxTreeDepth=3.
mode: friends(+Person,-Person).
mode: friends(-Person,+Person).
mode: smokes(+Person).
mode: cancer(+Person).
, n_estimators=10, neg_pos_ratio=2, solver='BoostSRL', target='cancer')
>>> dn.predict(test)
array([ True,  True,  True, False, False])
```

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

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.

neg_pos_ratio [int or float, optional (default: 2)] Ratio of negative to positive examples used during learning.

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.

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

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.BoostedRDNClassifier`

- *Estimating Feature Importance*
- *Smokes-Friends-Cancer*
- *Family Relationships Domain*

`srlearn.rdn`.BoostedRDNRegressor

class srlearn.rdn.**BoostedRDNRegressor**(*background=None*, *target='None'*, *n_estimators=10*, *node_size=2*, *max_tree_depth=3*, *neg_pos_ratio=2*, *solver='BoostSRL'*)

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,
... )
>>> 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=5)
>>> reg.fit(train)
BoostedRDNRegressor(background=setParam: numOfClauses=100.
setParam: numOfCycles=100.
```

(continues on next page)

(continued from previous page)

```
useProlog Variables: true.
setParam: nodeSize=2.
setParam: maxTreeDepth=3.
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=5, neg_pos_ratio=2, solver='BoostSRL', target='medv')
>>> reg.predict(test)   # doctest: +SKIP
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*, *neg_pos_ratio=2*, *solver='BoostSRL'*)

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.

neg_pos_ratio [int or float, optional (default: 2)] Ratio of negative to positive examples used during learning.

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.

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.

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

Parameters

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

1.3.2 Data Sets

There are some toy datasets built into the srlearn package. For more datasets, see the relational-datasets package.

datasets.load_toy_cancer()	Load and return the Toy Cancer dataset.
datasets.load_toy_father()	Load and return the Toy Father dataset.

srlearn.datasets.load_toy_cancer

srlearn.datasets.**load_toy_cancer**()
Load and return the Toy Cancer dataset.

Returns

toy_cancer [Bunch] Bunch contains *train* and *test* Database objects.

Examples

```
>>> from srlearn.datasets import load_toy_cancer
>>> train, test = load_toy_cancer()
>>> 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).']
```

Examples using `srlearn.datasets.load_toy_cancer`

- *Smokes-Friends-Cancer*

`srlearn.datasets`.load_toy_father

`srlearn.datasets.`**`load_toy_father`**`()`

Load and return the Toy Father dataset.

Returns

toy_father [Bunch] Bunch contains *train* and *test* Database objects.

Examples

```
>>> from srlearn.datasets import load_toy_father
>>> train, test = load_toy_father()
>>> train
Positive Examples:
['father(harrypotter,jamespotter).', ..., 'father(fredweasley,arthurweasley).']
Negative Examples:
['father(harrypotter,mrgranger).', ..., 'father(ginnyweasley,mollyweasley).']
Facts:
['male(mrgranger).', ..., 'childof(cygnusblack,narcissamalfoy).']
```

Examples using `srlearn.datasets.load_toy_father`

- *Family Relationships Domain*

1.3.3 Plotting and Visualization

These may be helpful for visualizing trees.

plotting.export_digraph(booster[, ...])	Create a digraph representation of a tree.
plotting.plot_digraph(dot_string[, format])	Plot a digraph as an image.

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 BoostedRDNClassifier
from srlearn import Background
from srlearn.datasets import load_toy_cancer
from srlearn.plotting import export_digraph

train, _ = load_toy_cancer()

bkg = Background(
    modes=train.modes,
)

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

clf.fit(train)

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

Examples using srlearn.plotting.export_digraph

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

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

Examples using `srlearn.plotting.plot_digraph`

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

1.3.4 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

srlearn.base.BaseBoostedRelationalModel

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

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.BoostedRDNClassifier* 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 `BoostedRDNClassifier`, 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()`.

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

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.

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`

- *Estimating Feature Importance*
- *Smokes-Friends-Cancer*
- *Family Relationships Domain*

`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.

system_manager.reset([soft])	Reset the FileSystem

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):    # doctest: +SKIP
...     print("Currently Empty")
Currently Empty
```

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

```
>>> from srlearn import system_manager
>>> system_manager.reset()    # doctest: +SKIP
[]
```

1.3.5 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.

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.

1.4 Basic Examples

These are some simple examples demonstrating `srlearn`.

1.4.1 Estimating Feature Importance

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

`webkb` is available in the relational-datasets package. A brief webkb overview is available with the relational-datasets documentation.

Calling `load` will return training and test folds:

```
from relational_datasets import load

train, test = load("webkb", fold=1)
```

We'll set up the learning problem and fit the classifier:

```
from srlearn.rdn import BoostedRDNClassifier
from srlearn import Background

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

clf.fit(train)
```

Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/srlearn/checkouts/stable/srlearn/
↪base.py:70: FutureWarning: solver='BoostSRL' will default to solver='SRLBoost' in 0.
↪6.0, pass one or the other as an argument to suppress this warning.
  ", pass one or the other as an argument to suppress this warning.", FutureWarning)

BoostedRDNClassifier(background=setParam: numOfClauses=8.
setParam: numOfCycles=100.
usePrologVariables: true.
setParam: nodeSize=3.
setParam: maxTreeDepth=3.
mode: courseprof(-course,+person).
mode: courseprof(+course,-person).
mode: courseta(+course,-person).
mode: courseta(-course,+person).
mode: project(-proj,+person).
mode: project(+proj,-person).
mode: sameperson(-person,+person).
mode: faculty(+person).
mode: student(+person).
, n_estimators=10, neg_pos_ratio=2, solver='BoostSRL', target='faculty')
```

The built-in `feature_importances_` attribute of a fit classifier is a Counter of how many times a features appears across the trees:

```
clf.feature_importances_
```

Out:

```
Counter({'student': 10, 'sameperson': 10})
```

These should generally be looked at while looking at the trees, so we'll plot the first tree here as well.

It appears that the only features needed to determine if someone is a faculty member can roughly be stated as: "*Is this person a student?*" and "*Do these two names refer to the same person?*"

This might be surprising, but shows that we can induce concepts like "*a faculty member is NOT a student.*"

```
from srlearn.plotting import export_digraph, plot_digraph

plot_digraph(export_digraph(clf, 0), format='html')
```

Out:

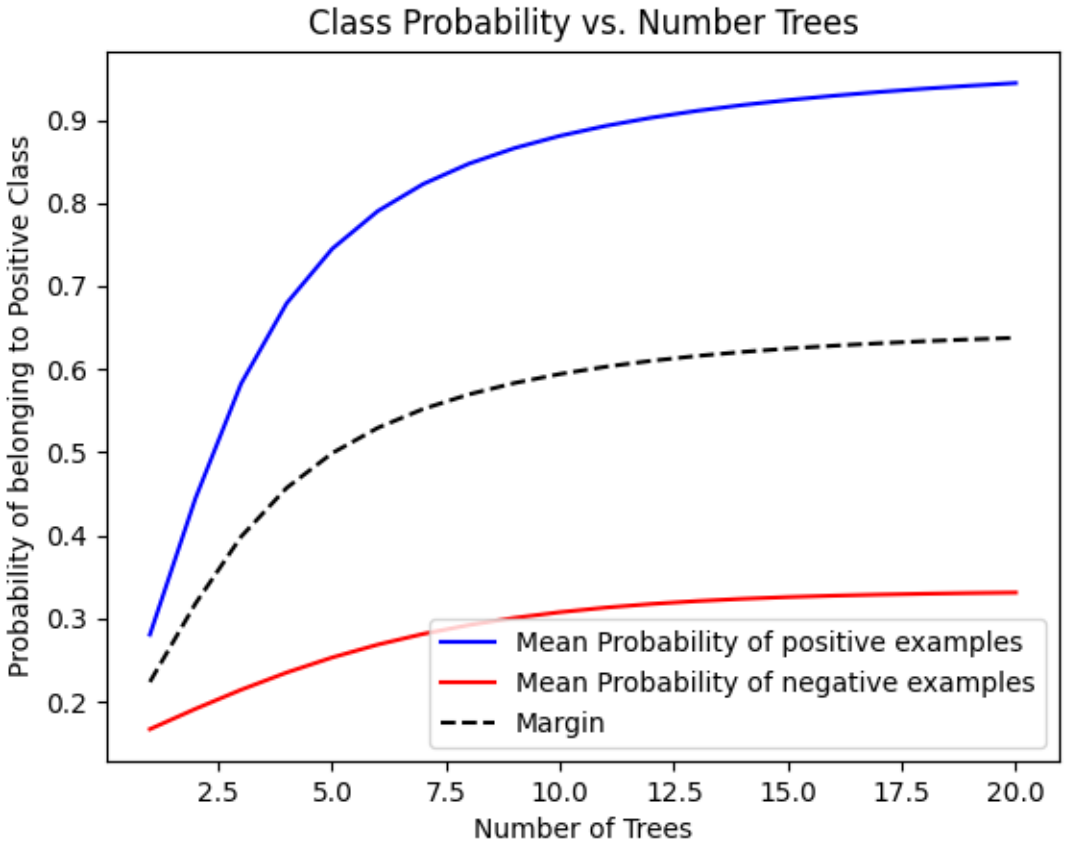
```
<srlearn.plotting._GVPlotter object at 0x7f040d9053d0>
```

Total running time of the script: (0 minutes 12.877 seconds)

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



Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/srlearn/checkouts/stable/srlearn/
↪base.py:70: FutureWarning: solver='BoostSRL' will default to solver='SRLBoost' in 0.
↪6.0, pass one or the other as an argument to suppress this warning.
  ", pass one or the other as an argument to suppress this warning.", FutureWarning)

<matplotlib.legend.Legend object at 0x7f040d75e750>
```

```
from srlearn.rdn import BoostedRDNClassifier
from srlearn import Background
from srlearn.datasets import load_toy_cancer
```

(continues on next page)

(continued from previous page)

```
import numpy as np
import matplotlib.pyplot as plt

train, test = load_toy_cancer()

bk = Background(modes=train.modes)

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

clf.fit(train)

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

for n_trees in x:
    clf.n_estimators = n_trees
    probs = clf.predict_proba(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")
```

Total running time of the script: (0 minutes 13.920 seconds)

1.4.3 Family Relationships Domain

Overview: 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.datasets import load_toy_father

train, test = load_toy_father()
```

The training examples in the "*Toy Father*" dataset describes relationships and facts about *Harry Potter* characters.

The first *positive* example: `father(harrypotter,jamespotter).` means "*James Potter is the father of Harry*

Potter."

The first *negative* example: `father(harrypotter,mrgranger).` can be interpreted as "*Mr. Granger is not the father of Harry Potter.*"

```
print(train.pos[0], "→    James Potter is the father of Harry Potter.")
print(train.neg[0], "  → Mr. Granger is not the father of Harry Potter.")
```

Out:

```
father(harrypotter,jamespotter). →    James Potter is the father of Harry Potter.
father(harrypotter,mrgranger).   → Mr. Granger is not the father of Harry Potter.
```

The *facts* contain three additional predicates: describing `children`, `male`, and who is a `siblingof`.

```
train.facts
```

Out:

```
['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).']
```

Our aim is to learn about what a "*father*" is in terms of the facts we have available. This process is usually called *induction,* and is often portrayed as "learning a definition of an object."

```
from srlearn.rdn import BoostedRDNClassifier
from srlearn import Background

bk = Background(
    modes=[
        "male(+name).",
        "father(+name,+name).",
        "childof(+name,+name).",
        "siblingof(+name,+name)."
    ],
    number_of_clauses=8,
)

clf = BoostedRDNClassifier(
    background=bk,
    target="father",
    node_size=1,
```

(continues on next page)

(continued from previous page)

```
    n_estimators=5,
)

clf.fit(train)
```

Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/srlearn/checkouts/stable/srlearn/
↪base.py:70: FutureWarning: solver='BoostSRL' will default to solver='SRLBoost' in 0.
↪6.0, pass one or the other as an argument to suppress this warning.
  ", pass one or the other as an argument to suppress this warning.", FutureWarning)

BoostedRDNClassifier(background=setParam: numOfClauses=8.
setParam: numOfCycles=100.
usePrologVariables: true.
setParam: nodeSize=1.
setParam: maxTreeDepth=3.
mode: male(+name).
mode: father(+name,+name).
mode: childof(+name,+name).
mode: siblingof(+name,+name).
, n_estimators=5, neg_pos_ratio=2, solver='BoostSRL', target='father')
```

It's important to check whether we actually learn something useful. We'll visually inspect the relational regression trees to see what they learned.

```
from srlearn.plotting import plot_digraph
from srlearn.plotting import export_digraph

plot_digraph(export_digraph(clf, 0), format="html")
```

Out:

```
<srlearn.plotting._GVPlotter object at 0x7f040d25e190>
```

There is some variance between runs, but in the concept that the trees pick up on is roughly that "*A father has a child and is male.*"

```
plot_digraph(export_digraph(clf, 1), format="html")
```

Out:

```
<srlearn.plotting._GVPlotter object at 0x7f040d25e290>
```

Here the data is fairly complete, and the concept that "*A father has a child and is male*" seems sufficient for the purposes of this data. Let's apply our learned model to the test data, which includes facts about characters from Jane Austen's *Pride and Prejudice.*

```
predictions = clf.predict_proba(test)

print("{:<35} {}".format("Predicate", "Probability of being True"), "\n", "-" * 60)
for predicate, prob in zip(test.pos + test.neg, predictions):
    print("{:<35} {:.2f}".format(predicate, prob))
```

Out:

```
Predicate                              Probability of being True
 ------------------------------------------------------------
father(elizabeth,mrbennet).            0.66
father(jane,mrbennet).                 0.66
father(charlotte,mrlucas).             0.66
father(charlotte,mrsbennet).           0.08
father(jane,mrlucas).                  0.09
father(mrsbennet,mrbennet).            0.09
father(jane,elizabeth).                0.08
```

The confidence might be a little low, which is a good excuse to mention one of the hyperparameters. "Node Size," or `node_size` corresponds to the maximum number of predicates that can be used as a split in the dependency network. We set `node_size=1` above for demonstration, but the concept that seems to be learned: `father(A, B) = [childof(B, A), male(B)]` is of size 2.

We might be able to learn a better model by taking this new information into account:

```
bk = Background(
    modes=[
        "male(+name).",
        "father(+name,+name).",
        "childof(+name,+name).",
        "siblingof(+name,+name)."
    ],
    number_of_clauses=8,
)

clf = BoostedRDNClassifier(
    background=bk,
    target="father",
    node_size=2,                # <--- Changed from 1 to 2
    n_estimators=5,
)

clf.fit(train)

plot_digraph(export_digraph(clf, 0), format="html")
```

Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/srlearn/checkouts/stable/srlearn/
↪base.py:70: FutureWarning: solver='BoostSRL' will default to solver='SRLBoost' in 0.
↪6.0, pass one or the other as an argument to suppress this warning.
  ", pass one or the other as an argument to suppress this warning.", FutureWarning)

<srlearn.plotting._GVPlotter object at 0x7f040d23f450>
```

This seems to be much more stable, which should also be reflected in the probabilities assigned on test examples.

```
predictions = clf.predict_proba(test)

print("{:<35} {}".format("Predicate", "Probability of being True"), "\n", "-" * 60)
for predicate, prob in zip(test.pos + test.neg, predictions):
    print("{:<35} {:.2f}".format(predicate, prob))
```

Out:

```
Predicate                          Probability of being True
 ------------------------------------------------------------
father(elizabeth,mrbennet).        0.74
father(jane,mrbennet).             0.74
father(charlotte,mrlucas).         0.74
father(charlotte,mrsbennet).       0.07
father(jane,mrlucas).              0.08
father(mrsbennet,mrbennet).        0.08
father(jane,elizabeth).            0.07
```

Total running time of the script: (0 minutes 3.647 seconds)

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

CHAPTER 2

Getting started

Prerequisites and installation instructions for getting started with this package.

CHAPTER 3

User guide

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

CHAPTER 4

API documentation

Full documentation for the modules.

CHAPTER 5

Example gallery

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

CHAPTER 6

Citing

If you find this helpful in your work, please consider citing:

```
@misc{hayes2019srlearn,
  title={srlearn: A Python Library for Gradient-Boosted Statistical Relational Models}
↪,
  author={Alexander L. Hayes},
  year={2019},
  eprint={1912.08198},
  archivePrefix={arXiv},
  primaryClass={cs.LG}
}
```

CHAPTER 7

Contributing

Many thanks to those who have already made contributions:

- Alexander L. Hayes, *Indiana University, Bloomington*
- Harsha Kokel, *The University of Texas at Dallas*
- Siwen Yan, *The University of Texas at Dallas*

Many thanks to the known and unknown contributors to WILL/BoostSRL/SRLBoost, including: Navdeep Kaur, Nandini Ramanan, Srijita Das, Mayukh Das, Kaushik Roy, Devendra Singh Dhami, Shuo Yang, Phillip Odom, Tushar Khot, Gautam Kunapuli, Sriraam Natarajan, Trevor Walker, and Jude W. Shavlik.

We have adopted the Contributor Covenant Code of Conduct version 1.4. Please read, follow, and report any incidents which violate this.

Questions, Issues, and Pull Requests are welcome. Please refer to CONTRIBUTING.md for information on submitting issues and pull requests.

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

srlearn.boostsrl, 21

Index

Symbols

__init__() (*srlearn.Background method*), 9
__init__() (*srlearn.Database method*), 7
__init__() (*srlearn.base.BaseBoostedRelationalModel method*), 18
__init__() (*srlearn.rdn.BoostedRDNClassifier method*), 11
__init__() (*srlearn.rdn.BoostedRDNRegressor method*), 14
__init__() (*srlearn.system_manager.FileSystem method*), 19

B

Background (*class in srlearn*), 9
BaseBoostedRelationalModel (*class in srlearn.base*), 18
BoostedRDNClassifier (*class in srlearn.rdn*), 11
BoostedRDNRegressor (*class in srlearn.rdn*), 13

C

call_process() (*in module srlearn.boostsrl*), 21

D

Database (*class in srlearn*), 7

E

example_data() (*in module srlearn.boostsrl*), 21
export_digraph() (*in module srlearn.plotting*), 17

F

feature_importances_ (*srlearn.base.BaseBoostedRelationalModel attribute*), 18
feature_importances_ (*srlearn.rdn.BoostedRDNClassifier attribute*), 12
feature_importances_ (*srlearn.rdn.BoostedRDNRegressor attribute*), 14
FileSystem (*class in srlearn.system_manager*), 19
fit() (*srlearn.rdn.BoostedRDNClassifier method*), 12
fit() (*srlearn.rdn.BoostedRDNRegressor method*), 14
from_files() (*srlearn.Database static method*), 8
from_json() (*srlearn.base.BaseBoostedRelationalModel method*), 18
from_json() (*srlearn.rdn.BoostedRDNClassifier method*), 12
from_json() (*srlearn.rdn.BoostedRDNRegressor method*), 15

I

inference_results() (*srlearn.boostsrl.test method*), 22
inspect_example_syntax() (*in module srlearn.boostsrl*), 21
inspect_mode_syntax() (*in module srlearn.boostsrl*), 21

L

load_toy_cancer() (*in module srlearn.datasets*), 15
load_toy_father() (*in module srlearn.datasets*), 16

P

plot_digraph() (*in module srlearn.plotting*), 17
predict() (*srlearn.rdn.BoostedRDNClassifier method*), 12
predict() (*srlearn.rdn.BoostedRDNRegressor method*), 15
predict_proba() (*srlearn.rdn.BoostedRDNClassifier method*), 12

R

reset() (*in module srlearn.system_manager*), 20

S

srlearn.boostsrl (*module*), 21

T

test (*class in srlearn.boostsrl*), 21
to_json() (*srlearn.base.BaseBoostedRelationalModel method*), 18
to_json() (*srlearn.rdn.BoostedRDNClassifier method*), 12
to_json() (*srlearn.rdn.BoostedRDNRegressor method*), 15
train (*class in srlearn.boostsrl*), 22
traintime() (*srlearn.boostsrl.train method*), 22
tree() (*srlearn.boostsrl.train method*), 22

W

write() (*srlearn.Background method*), 10
write() (*srlearn.Database method*), 8
write_to_file() (*in module srlearn.boostsrl*), 22