Sparsereg

Release 0.10.0+8.g9b409a0.dirty

Dec 02, 2019

Contents

CHAPTER 1

sparsereg

sparsereg is a collection of modern sparse (regularized) regression algorithms.

CHAPTER 2

Installation

```
pip install sparsereg
```

CHAPTER 3

Citation

If you use sparsereg please consider a citation:

```
@misc{markus_quade_sparsereg,
  author       = {Markus Quade},
  title        = {sparsereg - collection of modern sparse regression algorithms},
  month        = feb,
  year         = 2018,
  doi          = {10.5281/zenodo.1173754},
  url          = {https://github.com/ohjeah/sparsereg}
}
```

CHAPTER 4

Implemented algorithms

- Mcconaghy, T. (2011). FFX: Fast, Scalable, Deterministic Symbolic Regression Technology. Genetic Programming Theory and Practice IX, 235-260. DOI: 10.1007/978-1-4614-1770-5_13
- Brunton, Steven L., Joshua L. Proctor, and J. Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems." Proceedings of the National Academy of Sciences 113.15 (2016): 3932-3937. DOI: 10.1073/pnas.1517384113
- Bouchard, Kristofer E. "Bootstrapped Adaptive Threshold Selection for Statistical Model Selection and Estimation." arXiv preprint arXiv:1505.03511 (2015).
- Ignacio Arnaldo, Una-May O'Reilly, and Kalyan Veeramachaneni. "Building Predictive Models via Feature Synthesis." In Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO '15), Sara Silva (Ed.). ACM, New York, NY, USA, 983-990. DOI: 10.1145/2739480.2754693

CHAPTER 5

Contents

5.1 Sparsereg Gallery

Note: Click *here* to download the full example code

5.1.1 STRidge

Out:

```
[32.12551734 76.33080772 33.6926875   9.42759779  5.16621758 58.28693612
 29.43481665  7.18075454 10.30191944 75.31997019]
[28.61707383 69.15831382 29.89240922  8.88785683  4.38809077 53.08038878
 26.86251397  6.64254402  9.76695373 68.28360746]
```

```
import numpy as np
from sklearn.datasets import make_regression

from sparsereg.model.base import STRidge

x, y = make_regression(n_samples=1000, n_features=10, n_informative=10, n_targets=1,␣
↪random_state=42)
coef = STRidge().fit(x, y).coef_
print(STRidge().fit(x, y).coef_)
print(STRidge(unbias=False).fit(x, y).coef_)
```

Total running time of the script: (0 minutes 0.034 seconds)

Note: Click *here* to download the full example code

5.1.2 FFX

Out:

```
0.966 x_0 / ( 1.000 x_1 + 1.000 )
0.0008601017834973202
```

```python
import numpy as np

from sparsereg.model.ffx import FFX

np.random.seed(42)
x = np.random.normal(size=(33333, 2))
y = x[:, 0] / (1 + x[:, 1])
model = FFX(
    n_jobs=-1,
    l1_ratios=(0.8, 0.9, 0.95),
    exponents=[1, 2],
    target_score=1e-5,
    max_complexity=250,
    num_alphas=1000,
    eps=1e-70,
    rational=True,
)
model.fit(x, y)
print(model.print_model())
print(model.score(x, y))
```

Total running time of the script: (0 minutes 10.229 seconds)

Note: Click *here* to download the full example code

5.1.3 EFS

Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/sparsereg/envs/latest/lib/python3.6/
↪site-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default␣
↪value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to␣
↪silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
/home/docs/checkouts/readthedocs.org/user_builds/sparsereg/envs/latest/lib/python3.6/
↪site-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default␣
↪value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to␣
↪silence this warning.
```

(continues on next page)

(continued from previous page)

```
  warnings.warn(CV_WARNING, FutureWarning)
/home/docs/checkouts/readthedocs.org/user_builds/sparsereg/envs/latest/lib/python3.6/
↪site-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default␣
↪value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to␣
↪silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
1.0
```

```
import numpy as np
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.multioutput import MultiOutputRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

from sparsereg.model.efs import EFS

x, y = make_regression(n_samples=1000, n_features=10, n_informative=10, n_targets=3)
x_train, x_test, y_train, y_test = train_test_split(x, y)
steps = ("scaler", StandardScaler()), ("estimator", EFS(mu=1, q=3, max_stall_iter=5))
model = MultiOutputRegressor(Pipeline(list(steps)))
model.fit(x_train, y_train)
print(model.score(x_test, y_test))
```

Total running time of the script: (0 minutes 0.583 seconds)

Note: Click *here* to download the full example code

5.1.4 FFX2

Out:

```
Actual function: x_0^2 + sin(x_1) + x_2 cos(x_1)
Fit result: 0.005 x_0**2 + 0.021 x_0**2*x_1**2 + 0.021 x_0**2*x_2**2 + -0.008 x_
↪1**2*x_2**2 + 6.301
fit score (on training data): 0.05223809944750665
fitting took 163.28 seconds.
```

```
import datetime

import numpy as np

from sparsereg.model.ffx import FFX
```

(continues on next page)

(continued from previous page)

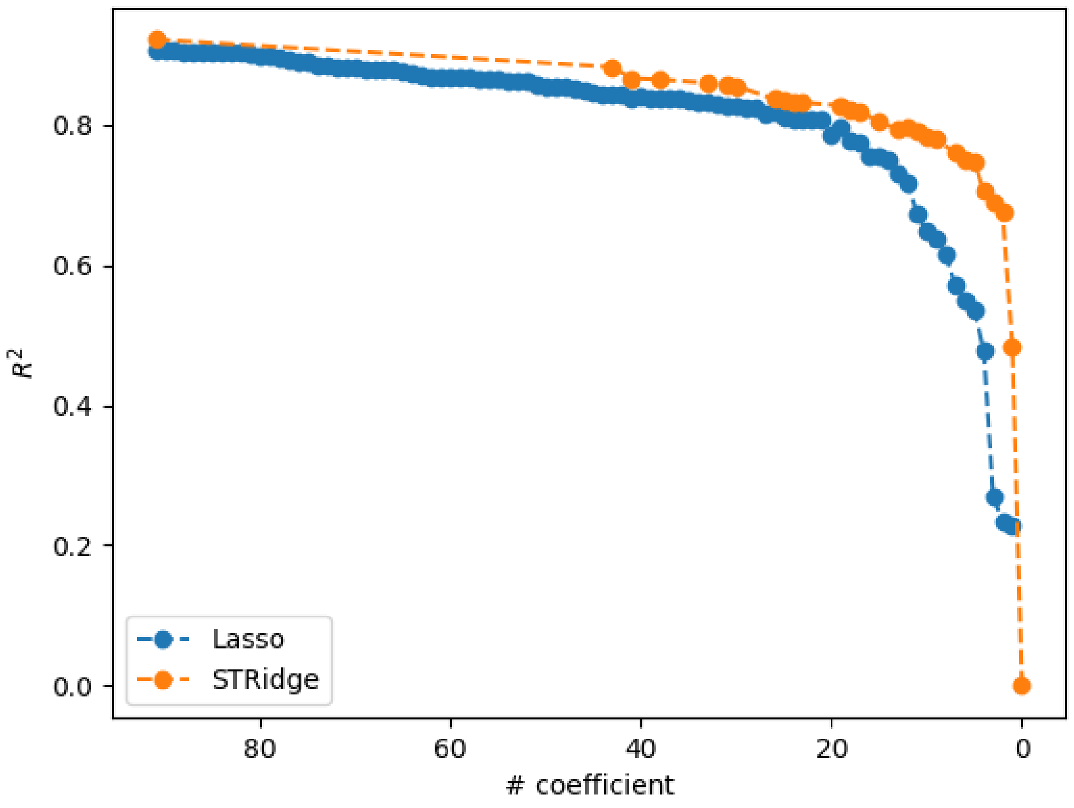
```
np.random.seed(42)
x = np.random.normal(scale=3.0, size=(33333, 3))

print("Actual function: x_0^2 + sin(x_1) + x_2 cos(x_1)")
y = x[:, 0] ** 2 + np.sin(x[:, 1]) + x[:, 2] * np.cos(x[:, 1])
model = FFX(
    n_jobs=-1,
    l1_ratios=(0.001, 0.01, 0.5, 0.9, 0.999),
    exponents=[1, 2],
    operators={"sin": np.sin, "cos": np.cos},
    target_score=1e-6,
    max_complexity=250,
    num_alphas=1000,
    eps=1e-70,
    rational=False,
)
t0 = datetime.datetime.utcnow()
model.fit(x, y)
dur = datetime.datetime.utcnow() - t0
dur = dur.total_seconds()
print(f"Fit result: {model.print_model()}")
print(f"fit score (on training data): {model.score(x, y)}")
print(f"fitting took {dur:.2f} seconds.")
```

Total running time of the script: (2 minutes 44.718 seconds)

Note: Click *here* to download the full example code

5.1.5 NET



```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_boston
from sklearn.linear_model import Lasso

import sparsereg.preprocessing.symfeat as sf
from sparsereg.model import STRidge
from sparsereg.util.net import net

data = load_boston()
x, y = data.data, data.target
exponents = [1]
operators = {}
sym = sf.SymbolicFeatures(exponents=exponents, operators=operators)
features = sym.fit_transform(x)
ests = [Lasso, STRidge]
attrs = ["alpha", "threshold"]
names = ["Lasso", "STRidge"]
for est, attr, name in zip(ests, attrs, names):
    models = net(est, features, y, attr, filter=True, max_coarsity=5, r_max=1e5)
    m = sorted(models)
    scores = np.array([models[k].score(features, y) for k in m])
    plt.plot(m, scores, "o--", label=name)
```

(continues on next page)

(continued from previous page)

```
plt.legend()
plt.xlabel("# coefficient")
plt.ylabel(r"$R^2$")
plt.gca().invert_xaxis()
plt.show()
```

Total running time of the script: (0 minutes 16.710 seconds)

Note: Click *here* to download the full example code

5.1.6 JMAP

Out:

```
JMAP
/home/docs/checkouts/readthedocs.org/user_builds/sparsereg/checkouts/latest/sparsereg/
↪model/bayes.py:49: UserWarning: Consider using sklearn.linear_model.BayesianRidge␣
↪instead of JMAP.
  f"Consider using sklearn.linear_model.BayesianRidge instead of {self.__class__.__
↪name__}."
lambda 5.846788253508231e-07
alpha 15.959280652140635
coef [0.98553912 2.46879503] [0.00086711 0.0012283 ]
SKLEARN BayesianRidge
lambda 1.898742424623188e-07
alpha 95.90650799118377
coef [1.00020773 2.50007688] [0.0059014 0.0028405]
```

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures

from sparsereg.model.bayes import JMAP
from sparsereg.model.bayes import scale_sigma

size = 10000
scale = 0.1
x = 3 * np.sort(np.random.normal(size=(size, 1)), axis=0)

y = x[:, 0] + 2.5 * x[:, 0] ** 2 + np.random.normal(scale=scale, size=size)

normalize = True
degree = 2
poly = PolynomialFeatures(degree=degree, include_bias=False)
xfeat = poly.fit_transform(x, y)
print("JMAP")
model = JMAP(normalize=normalize)
model.fit(xfeat, y)
# print("ve", model.ve_, model.ve_.shape)
```

(continues on next page)

(continued from previous page)

```
# print("vf", model.vf_, np.sqrt(model.vf_))
print("lambda", model.lambda_)
print("alpha", model.alpha_)
print("coef", model.coef_, model.std_coef_)

from sklearn.linear_model import BayesianRidge

print("SKLEARN BayesianRidge")
model = BayesianRidge(normalize=normalize)
model.fit(xfeat, y)

print("lambda", model.lambda_)
print("alpha", model.alpha_)
print("coef", model.coef_, scale_sigma(model, model.X_offset_, model.X_scale_)[1])
```

Total running time of the script: (0 minutes 8.749 seconds)

Note: Click *here* to download the full example code

5.1.7 SINDy Example

Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/sparsereg/envs/latest/lib/python3.6/
↪site-packages/sklearn/preprocessing/_function_transformer.py:97: FutureWarning: The␣
↪default validate=True will be replaced by validate=False in 0.22.
  "validate=False in 0.22.", FutureWarning)
Score on test data  0.9999975958704718
Selected hyperparameter (alpha, threshold):  0.1 0.1
dx_0 / dt =  1.000 x1
dx_1 / dt =  -0.300 x0
Complexity of the model (sum of coefficients and intercetps bigger than the␣
↪threshold):  2
```

```
import warnings

import numpy as np
from scipy.integrate import odeint
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.utils import check_random_state

from sparsereg.model import SINDy


def rhs_harmonic_oscillator(y, t):
    dy0 = y[1]
    dy1 = -0.3 * y[0]
```

(continues on next page)

(continued from previous page)

```
    return [dy0, dy1]


x0 = [0, 1]
t = np.linspace(0, 10, 1000)
x = odeint(rhs_harmonic_oscillator, x0, t)
x_train, x_test = x[:750], x[750:]
kw = dict(fit_intercept=True, normalize=False)
model = SINDy(dt=t[1] - t[0], degree=2, alpha=0.3, kw=kw)
rng = check_random_state(42)
cv = KFold(n_splits=5, random_state=rng, shuffle=False)
params = {"alpha": [0.1, 0.2, 0.3, 0.4, 0.5], "threshold": [0.1, 0.3, 0.5]}
grid = GridSearchCV(model, params, cv=cv)
with warnings.catch_warnings():  # suppress matrix illconditioned warning
    warnings.filterwarnings("ignore")
    grid.fit(x_train)
selected_model = grid.best_estimator_
print("Score on test data ", selected_model.score(x_test))
print("Selected hyperparameter (alpha, threshold): ", selected_model.alpha, selected_
↪model.threshold)
for i, eq in enumerate(selected_model.equations()):
    print("dx_{} / dt = ".format(i), eq)
print(
    "Complexity of the model (sum of coefficients and \
intercetps bigger than the threshold): ",
    selected_model.complexity,
)
```

Total running time of the script: (0 minutes 0.684 seconds)

Note: Click *here* to download the full example code

5.1.8 Group Lasso

Out:

```
defaultdict(<class 'list'>, {1: ['x_0', 'sin(x_0)', 'x_0**2*sin(x_0)'], 2: ['x_0**2'],
↪ 0: ['cos(x_0)', 'sin(x_0**2)', 'cos(x_0**2)', 'x_0*sin(x_0)', 'x_0*cos(x_0)', 'x_
↪0**2*cos(x_0)', 'x_0*sin(x_0**2)', 'x_0**2*sin(x_0**2)', 'x_0*cos(x_0**2)', 'x_
↪0**2*cos(x_0**2)'], 3: ['x_0**3']})
/home/docs/checkouts/readthedocs.org/user_builds/sparsereg/envs/latest/lib/python3.6/
↪site-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default␣
↪value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to␣
↪silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
0.9977261167415085
0.640 x_0**2 + 0.975 x_0**3 + 0.968
/home/docs/checkouts/readthedocs.org/user_builds/sparsereg/envs/latest/lib/python3.6/
↪site-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default␣
↪value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to␣
↪silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
0.9984230583925218
0.561 x_0**2 + -0.126 x_0**2*cos(x_0) + 0.975 x_0**3 + 1.046
```

```python
from collections import defaultdict

import numpy as np
from sklearn.cluster import AgglomerativeClustering
from sklearn.linear_model import Lasso
from sklearn.metrics import explained_variance_score
from sklearn.metrics import make_scorer
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.utils.validation import check_random_state

from sparsereg.model.base import print_model
from sparsereg.model.group_lasso import SparseGroupLasso
from sparsereg.preprocessing.symfeat import SymbolicFeatures

rng = check_random_state(42)
x = rng.normal(size=(10000, 1))
y = np.cos(x[:, 0]) + x[:, 0] ** 2 + x[:, 0] ** 3  # + 0.01*rng.normal(size=1000)
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=rng)
pre = SymbolicFeatures(exponents=[1, 2], operators={"sin": np.sin, "cos": np.cos}).
↪fit(x_train)
features_train = pre.transform(x_train)
features_test = pre.transform(x_test)
km = AgglomerativeClustering(n_clusters=4).fit(features_train.T)
labels = defaultdict(list)
for k, v in zip(pre.get_feature_names(), km.labels_):
    labels[v].append(k)
print(labels)
params = {"alpha": [0.001, 0.01, 0.02, 0.05], "normalize": [True]}
scorer = make_scorer(explained_variance_score)
sgl = SparseGroupLasso(groups=km.labels_, rho=0.3, alpha=0.02)
l = Lasso()
for model in [sgl, l]:
    grid = GridSearchCV(model, params, n_jobs=1, scoring=scorer, error_score=0).
↪fit(features_train, y_train)
    print(grid.score(features_test, y_test))
    print(
        print_model(
            grid.best_estimator_.coef_, pre.get_feature_names(), intercept=grid.best_
↪estimator_.intercept_
        )
    )
```

Total running time of the script: (0 minutes 2.273 seconds)

5.2 sparsereg

5.2.1 sparsereg package

5.2.1.1 Subpackages

sparsereg.model package

Submodules

sparsereg.model.base module

`sparsereg.model.base.`**`print_model`**(*coef*, *input_features*, *errors=None*, *intercept=None*, *error_intercept=None*, *precision=3*, *pm='±'*)

Parameters

- **`coef`** –
- **`input_features`** –
- **`errors`** –
- **`intercept`** –
- **`sigma_intercept`** –
- **`precision`** –
- **`pm`** –

Returns:

`sparsereg.model.base.`**`equation`**(*pipeline*, *input_features=None*, *precision=3*, *input_fmt=None*)

`class` `sparsereg.model.base.`**`RationalFunctionMixin`**

Bases: `object`

`fit`(*x*, *y*, ***kwargs*)

`predict`(*x*)

`print_model`(*input_features=None*, *precision=3*)

`class` `sparsereg.model.base.`**`PrintMixin`**

Bases: `object`

`print_model`(*input_features=None*, *precision=3*)

`class` `sparsereg.model.base.`**`STRidge`**(*threshold=0.01*, *alpha=0.1*, *max_iter=100*, *normalize=True*, *fit_intercept=True*, *threshold_intercept=False*, *copy_X=True*, *unbias=True*, *ridge_kw=None*)

Bases: `sklearn.linear_model.base.LinearModel`, `sklearn.base.RegressorMixin`

`fit`(*x_*, *y*, *sample_weight=None*)

Fit model.

`complexity`

`class` `sparsereg.model.base.`**`BoATS`**(*alpha=0.01*, *sigma=0.01*, *n=10*, *copy_X=True*, *fit_intercept=True*, *normalize=True*)

Bases: `sklearn.linear_model.base.LinearModel`, `sklearn.base.RegressorMixin`

`fit`(*x_*, *y*, *sample_weight=None*)

Fit model.

`sparsereg.model.base.`**`fit_with_noise`**(*x*, *y*, *sigma*, *alpha*, *n*, *lmc=<class 'sklearn.linear_model.base.LinearRegression'>*)

sparsereg.model.bayes module

`sparsereg.model.bayes.`**`scale_sigma`**(*est*, *X_offset*, *X_scale*)

`class` `sparsereg.model.bayes.`**`JMAP`**(*ae0=1e-06*, *be0=1e-06*, *af0=1e-06*, *bf0=1e-06*, *max_iter=300*, *tol=0.001*, *normalize=False*, *fit_intercept=True*, *copy_X=True*)

Bases: `sklearn.linear_model.base.LinearModel`, `sklearn.base.RegressorMixin`, *`sparsereg.model.base.PrintMixin`*

`fit`(*x*, *y*)

Fit model.

`predict`(*X*, *return_std=False*)

Predict using the linear model

Parameters **`X`** (*`array_like or sparse matrix, shape (n_samples, n_features)`*) – Samples.

Returns **C** – Returns predicted values.

Return type array, shape (n_samples,)

`sparsereg.model.bayes.`**`jmap`**(*g*, *H*, *ae0*, *be0*, *af0*, *bf0*, *max_iter=1000*, *tol=0.0001*, *rcond=None*, *observer=None*)

Maximum a posteriori estimator for g = H @ f + e

p(g | f) = normal(H f, ve I) p(ve) = inverse_gauss(ae0, be0) p(f | vf) = normal(0, vf I) p(vf) = inverse_gauss(af0, bf0)

JMAP: maximizes p(f,ve,vf|g) = p(g | f) p(f | vf) p(ve) p(vf) / p(g) with respect to f, ve and vf

Original Author: Ali Mohammad-Djafari, April 2015

Parameters

- **`g`** –
- **`H`** –
- **`ae0`** –
- **`be0`** –
- **`af0`** –
- **`bf0`** –
- **`max_iter`** –
- **`rcond`** –

Returns:

sparsereg.model.efs module

`sparsereg.model.efs.`**`size`**(*name*)

`sparsereg.model.efs.`**`mutate`**(*names*, *importance*, *toursize*, *operators*, *rng=<module 'random' from '/home/docs/checkouts/readthedocs.org/user_builds/sparsereg/envs/latest/lib/python3.6/randon*

`sparsereg.model.efs.`**`get_importance`**(*coefs*, *scores*)

`class` `sparsereg.model.efs.`**`LibTrafo`**(*names*, *operators*)

Bases: `sklearn.base.BaseEstimator`, `sklearn.base.TransformerMixin`

fit(*x*, *y=None*)

transform(*x*, *y=None*)

class `sparsereg.model.efs.`**EFS**(*q=1*, *mu=1*, *max_size=5*, *t=0.95*, *toursize=5*, *max_stall_iter=20*, *max_iter=2000*, *random_state=None*, *operators={'add': <ufunc 'add'>, 'cos': <ufunc 'cos'>, 'div': <ufunc 'true_divide'>, 'exp': <ufunc 'exp'>, 'log': <ufunc 'log'>, 'mul': <ufunc 'multiply'>, 'sin': <ufunc 'sin'>, 'sqrt': <ufunc 'sqrt'>, 'square': <ufunc 'square'>, 'subtract': <ufunc 'subtract'>}*, *max_coarsity=2*, *n_jobs=1*)

Bases: `sklearn.base.BaseEstimator`, `sklearn.base.RegressorMixin`, `sklearn.base.TransformerMixin`

Evolutionary feature synthesis.

fit(*x*, *y*)

predict(*x*)

transform(*x*, *y=None*)

sparsereg.model.ffx module

class `sparsereg.model.ffx.`**FFXModel**(*strategy*, ***kw*)

Bases: `sklearn.pipeline.Pipeline`

print_model(*input_features=None*)

pre_compute(*x*, *y*)

class `sparsereg.model.ffx.`**FFXElasticNet**(*alpha=1.0*, *l1_ratio=0.5*, *fit_intercept=True*, *normalize=False*, *precompute=False*, *max_iter=1000*, *copy_X=True*, *tol=0.0001*, *warm_start=False*, *positive=False*, *random_state=None*, *selection='cyclic'*)

Bases: *`sparsereg.model.base.PrintMixin`*, `sklearn.linear_model.coordinate_descent.ElasticNet`

Mixin, implements only the *`score`* method.

score(*x*, *y*)

Score using the nrmse.

class `sparsereg.model.ffx.`**FFXRationalElasticNet**(*alpha=1.0*, *l1_ratio=0.5*, *fit_intercept=True*, *normalize=False*, *precompute=False*, *max_iter=1000*, *copy_X=True*, *tol=0.0001*, *warm_start=False*, *positive=False*, *random_state=None*, *selection='cyclic'*)

Bases: *`sparsereg.model.base.RationalFunctionMixin`*, *`sparsereg.model.ffx.FFXElasticNet`*

class `sparsereg.model.ffx.`**Strategy**

Bases: `tuple`

Create new instance of Strategy(exponents, operators, consider_products, index, base)

base

Alias for field number 4

consider_products
Alias for field number 2

exponents
Alias for field number 0

index
Alias for field number 3

operators
Alias for field number 1

sparsereg.model.ffx.**build_strategies**(*exponents*, *operators*, *rational=True*)

sparsereg.model.ffx.**enet_path**(*est*, *x_train*, *x_test*, *y_train*, *y_test*, *num_alphas*, *eps*, *l1_ratio*, *target_score*, *n_tail*, *max_complexity*)

sparsereg.model.ffx.**run_strategy**(*strategy*, *x_train*, *x_test*, *y_train*, *y_test*, *num_alphas*, *eps*, *l1_ratios*, *target_score*, *n_tail*, *max_complexity*, *n_jobs*, ***kw*)

sparsereg.model.ffx.**run_ffx**(*x_train*, *x_test*, *y_train*, *y_test*, *exponents*, *operators*, *num_alphas=100*, *l1_ratios=(0.1*, *0.3*, *0.5*, *0.7*, *0.9*, *0.95)*, *eps=1e-30*, *target_score=0.01*, *max_complexity=50*, *n_tail=15*, *random_state=None*, *strategies=None*, *n_jobs=1*, *rational=True*, ***kw*)

class sparsereg.model.ffx.**WeightedEnsembleEstimator**(*estimators*, *weights*)
Bases: `sklearn.base.BaseEstimator`, `sklearn.base.TransformerMixin`

fit(*x*, *y=None*)

predict(*x*)

print_model(*input_features=None*)

class sparsereg.model.ffx.**FFX**(*l1_ratios=(0.4, 0.8, 0.95), num_alphas=30, eps=1e-05, random_state=None, strategies=None, target_score=0.01, n_tail=5, decision='min', max_complexity=50, exponents=[1, 2], operators={}, n_jobs=1, rational=True, **kw*)
Bases: `sklearn.base.BaseEstimator`, `sklearn.base.RegressorMixin`

Fast Function eXtraction model.

Parameters

- **l1_ratios** (*iterable*) – Determines ratio of l1 to l2 penalty term
- **num_alphas** (*int*) – Determines numbers of different ratios of cost function to penalty term. `0<= l1_ratio <= 1`.
- **eps** (*float*) – ratio of smallest to largest alpha considered. (`0 < eps < 1`)
- **random_state** (*int*) –
- **strategies** (*iterable*) – *Strategy* s to consider
- **target_score** (*float*) – break condition on cost function for innermost loop
- **n_tail** (*int*) – length of path (in alpha) to check into past for saturation
- **decision** (*str*) – one of `'weight'` or `'min'`
- **max_complexity** (*float*) – break condition on model complexity for innermost loop
- **exponents** (*iterable*) – can contain float and negative values

- **operators** (*dict*) – mapping operator name to callable (of one variable)
- **n_jobs** (*int*) –
- **rational** (*bool*) – Whether to consider general rational functions as well
- **kw** –

The implemented algorithm is found in `http://dx.doi.org/10.1007/978-1-4614-1770-5_13`.

A *Strategy* is determined by a set of nonlinear functions from which an extended set of features will be generated by evaluating these functions on all given features. You can either supply the strategies directly via the `strategies` parameter or let the strategies be generated. Generation of strategies is configured by the parameters *exponents*, *operators* and `rational`. When `strategies` is given, *exponents*, *operators* and `rational` have no effect.

Strategy generation takes place in the following manner:

***exponents*:** Orders of the monomials to consider for each single feature. (No products between features here). *exponents* is an iterable of numbers (floats and negative values are possible, 1 will always automatically be included.) The first step in strategy generation is calculating all monomials.

***operators*:** mapping of str to callable taking one parameter. All callables in *operators* will be evaluated on all monomials from the first step

products Not configurable. Always consider all products of each operator feature from the second step with each monomial feature from the first. And all products of monomial features with all monomial features based on a different feature (thus generating mixed products up to order `2*max(exponents)`).

rational If true, do not only consider generalized linear models from all basis functions but consider also rational functions using the rational function trick described here

For each *Strategy*, an elastic net optimizer will be run with many combinations of l1_ratio and alpha. A `l1_ratio` of 0 corresponds to ridge regression (only l2 penalty), a `l1_ratio` of 1 corresponds to LASSO regression (only l1 penalty). `alpha` determines the amount of regularization, where `alpha=0` would mean now regularization and `alpha -> infty` would mean only regularization. For details on the used elastic net algorithm see `sklearn.linear_model.ElasticNet`.

The number of alphas is loosely determined by `num_alpha` (the actual number is close and never smaller). The maximum value of the considered `alpha` is determined dynamically based on Tibshirani's "Strong Rules", see **'https://doi.org/10.1111/j.1467-9868.2011.01004.x'_** The rule gives an `alpha` for which the fitted model will (in most relevant cases) have a complexity of 0 (no nonzero terms). This maximum alpha also depends on the l1_ratio, therefore the iteration over alpha takes place in the innermost loop.

The innermost loop would iterate from the maximum alpha to `eps` times the maximum alpha. With increasing `alpha`, the complexity (number of non-zero terms) is expected to increase, whereas the cost (nrmse evaluated on the training set) is expected to decrease.

The innermost loop has three break conditions:

1. train_score If the cost is less or equal to `target_score`
2. complexity If the complexity is greater or equal to `max_complexity`
3. saturation No significant improvement in the cost during the last `n_tail` iterations. (Significant -> last 4 decimal digits)

To obtain a single model from the Pareto front of models, the Akaike information criterion (AIC) is used (see `https://en.wikipedia.org/wiki/Akaike_information_criterion`). How it is used is determined by the `decision` parameter. If `decision == 'min'`, the model with the smallest AIC is taken, if `decision == 'weighted'`, the resulting model will be a linear combination of all models the front consists of, weighted by `exp((min(AIC)-AIC)/2)`.

fit (*x*, *y=None*)

predict (*x*)

score (*x*, *y*)

Returns the coefficient of determination R^2 of the prediction.

The coefficient R^2 is defined as (1 - u/v), where u is the residual sum of squares ((y_true - y_pred) ** 2).sum() and v is the total sum of squares ((y_true - y_true.mean()) ** 2).sum(). The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

Parameters

- **X** (*array-like, shape = (n_samples, n_features)*) – Test samples. For some estimators this may be a precomputed kernel matrix instead, shape = (n_samples, n_samples_fitted], where n_samples_fitted is the number of samples used in the fitting for the estimator.
- **y** (*array-like, shape = (n_samples) or (n_samples, n_outputs)*) – True values for X.
- **sample_weight** (*array-like, shape = [n_samples], optional*) – Sample weights.

Returns **score** – R^2 of self.predict(X) wrt. y.

Return type float

Notes

The R2 score used when calling `score` on a regressor will use `multioutput='uniform_average'` from version 0.23 to keep consistent with `metrics.r2_score`. This will influence the `score` method of all the multioutput regressors (except for `multioutput.MultiOutputRegressor`). To specify the default value manually and avoid the warning, please either call `metrics.r2_score` directly or make a custom scorer with `metrics.make_scorer` (the built-in scorer `'r2'` uses `multioutput='uniform_average'`).

make_model (*x_test*, *y_test*)

print_model (*input_features=None*)

sparsereg.model.group_lasso module

class `sparsereg.model.group_lasso.`**SparseGroupLasso** (*groups*, *alpha=1.0*, *rho=0.5*, *max_iter=1000*, *tol=0.0001*, *normalize=False*, *fit_intercept=True*, *copy_X=True*)

Bases: `sklearn.linear_model.base.LinearModel`, `sklearn.base.RegressorMixin`

fit (*x*, *y*, *sample_weight=None*)

Fit model.

sparsereg.model.sindy module

class `sparsereg.model.sindy.`**`SINDy`**(*alpha=1.0, threshold=0.1, degree=3, operators=None, dt=1.0, n_jobs=1, derivative=None, feature_names=None, kw={}*)

Bases: `sklearn.base.BaseEstimator`

`fit`(*x, y=None*)

`predict`(*x*)

`equations`(*precision=3*)

`score`(*x, y=None, multioutput='uniform_average'*)

`complexity`

Module contents

sparsereg.preprocessing package

Submodules

sparsereg.preprocessing.symfeat module

class `sparsereg.preprocessing.symfeat.`**`Base`**

Bases: `sklearn.base.BaseEstimator`, `sklearn.base.TransformerMixin`

`name`

class `sparsereg.preprocessing.symfeat.`**`SimpleFeature`**(*exponent, index=0*)

Bases: `sparsereg.preprocessing.symfeat.Base`

Base to create polynomial features.

`transform`(*x*)

class `sparsereg.preprocessing.symfeat.`**`OperatorFeature`**(*feat_cls, operator, operator_name=None*)

Bases: `sparsereg.preprocessing.symfeat.Base`

`transform`(*x*)

class `sparsereg.preprocessing.symfeat.`**`ProductFeature`**(*feat_cls1, feat_cls2*)

Bases: `sparsereg.preprocessing.symfeat.Base`

`transform`(*x*)

`sparsereg.preprocessing.symfeat.`**`hashed_hash_`**()

`sparsereg.preprocessing.symfeat.`**`hashed_hash`**(*x*)

class `sparsereg.preprocessing.symfeat.`**`SymbolicFeatures`**(*exponents=[1], operators={}, consider_products=True*)

Bases: `sparsereg.preprocessing.symfeat.Base`

`fit`(*x, y=None*)

`transform`(*x*)

get_feature_names(*input_features=None*)
Get all the feature names. Only Available after fitting.

Module contents

sparsereg.util package

Submodules

sparsereg.util.net module

`sparsereg.util.net.`**complexity**(*estimator*)

`sparsereg.util.net.`**net**(*estimator, x, y, attr='alpha', max_coarsity=2, filter=True, r_max=1000.0, **kw*)

sparsereg.util.pipeline module

class `sparsereg.util.pipeline.`**ColumnSelector**(*index=slice(None, None, None)*)
Bases: `sklearn.base.TransformerMixin`, `sklearn.base.BaseEstimator`

fit(*x, y=None*)

transform(*x, y=None*)

get_feature_names(*input_features=None*)

Module contents

`sparsereg.util.`**dominates**(*a, b*)

`sparsereg.util.`**pareto_front**(*models, *attrs, all=False*)
Simple cull. Can recursively determine all fronts.

`sparsereg.util.`**crowding_distance**(*models, *attrs*)
Assumes models in lexicographical sorted.

`sparsereg.util.`**sort_non_dominated**(*models, *attrs, index=False*)
NSGA2 based sorting

`sparsereg.util.`**normalize**(*x, order=2*)

`sparsereg.util.`**cardinality**(*x, null=1e-09*)

`sparsereg.util.`**rmse**(*x*)

`sparsereg.util.`**nrmse**(*x, y*)

class `sparsereg.util.`**ReducedLinearModel**(*mask, lm*)
Bases: `sklearn.linear_model.base.LinearModel`

fit(*x, y*)
Fit model.

predict(*x*)
Predict using the linear model

Parameters **X** (*array_like or sparse matrix, shape (n_samples, n_features)*) – Samples.

Returns **C** – Returns predicted values.

Return type array, shape (n_samples,)

scores (*x, y*)

sparsereg.util.**aic** (*residuals, k, correct=False*)
Akaike information criterion

sparsereg.vendor package

Subpackages

sparsereg.vendor.group_lasso package

Submodules

sparsereg.vendor.group_lasso.group_lasso module

sparsereg.vendor.group_lasso.group_lasso.**soft_threshold** (*a, b*)

sparsereg.vendor.group_lasso.group_lasso.**sparse_group_lasso** (*X, y, alpha, rho, groups, max_iter=1000, rtol=1e-06, verbose=False*)

Linear least-squares with l2/l1 + l1 regularization solver.

Solves problem of the form:

(1 / (2 n_samples)) * ||Xb - y||^2_2 + [(alpha * (1 - rho) * sum(sqrt(#j) * ||b_j||_2) + alpha * rho ||b_j||_1)]

where b_j is the coefficients of b in the j-th group. Also known as the `sparse group lasso`.

Parameters

- **X** (*array of shape (n_samples, n_features)*) – Design Matrix.
- **y** (*array of shape (n_samples,)*) –
- **alpha** (*float or array*) – Amount of penalization to use.
- **groups** (*array of shape (n_features,)*) – Group label. For each column, it indicates its group apertenance.
- **rtol** (*float*) – Relative tolerance. ensures ||(x - **x_**) / x_|| < rtol, where **x_** is the approximate solution and x is the true solution. TODO duality gap

Returns **x** – vector of coefficients

Return type array

References

“A sparse-group lasso”, Noah Simon et al.

`sparsereg.vendor.group_lasso.group_lasso.`**`group_lasso_check_kkt`**(*A*, *b*, *x*, *alpha*, *groups*)

Auxiliary function. Check KKT conditions for the group lasso

Returns True if conditions are satisfied, False otherwise

Module contents

Module contents

5.2.1.2 Module contents

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index