LazyGrid

Dec 13, 2019

User Guide

LazyGrid is a python package providing an automatic, efficient and flexible implementation of complex machine learning pipeline generation and cross-validation.

Before fitting a model or a pipeline step, LazyGrid checks inside an internal SQLite database if the model has already been fitted. If the model is found, it won't be fitted again.

CHAPTER 1

Quick start

You can install LazyGrid along with all its dependencies from PyPI:

```
$ pip install -r requirements.txt lazygrid
```

CHAPTER 2

Source

The source code and minimal working examples can be found on GitHub.

2.1 Installation

You can install LazyGrid along with all its dependencies from PyPI:

```
$ pip install -r requirements.txt lazygrid
```

or from source code:

```
$ git clone https://github.com/glubbdubdrib/lazygrid.git
$ cd ./lazygrid
$ pip install -r requirements.txt .
```

LazyGrid is compatible with Python 3.5 and above.

2.2 Tutorial

LazyGrid has three main features:

- it can generate all possible pipelines given a set of steps (*Pipeline generation*) or all possible models given a grid of parameters (*Grid search*)
- it can compare the performance of a list of models using cross-validation and statistical tests (*Model comparison*), and
- it follows the memoization paradigm, avoiding fitting a model or a pipeline step twice.

2.2.1 Environment setup

Input data

In order to make each LazyPipeline transformer unique for different cross-validation splits, you must provide input data as DataFrame objects. The easiest way to transform numpy arrays into `DataFrame` data structures is the following:

```
import pandas as pd
...
X, y = ...
X = pd.DataFrame(X)
```

Organizing data sets and databases

If you are using more than one data set in your project, it is highly recommended to generate a hierarchy of database directories so that models fitted on different data sets can be easily identified:

```
import os
...
database_root_dir = "database"
data_set_name = "foo"
database_dir = os.path.join(database_root_dir, data_set_name)
if not os.path.isdir(database_dir):
    os.makedirs(database_dir)
```

This code will generate a directory structure as the following:

```
database
+-- foo
|   +-- database.sqlite
+-- baz
|   +-- database.sqlite
+-- ...
```

2.2.2 Model generation

Pipeline generation

In order to generate all possible pipelines given a set of steps, you should define a list of elements, which in turn are lists of pipeline steps, i.e. preprocessors, feature selectors, classifiers, etc. Each step could be either a `sklearn` object or a `keras` model.

Once you have defined the pipeline elements, the `generate_grid` method will return a list of models of type `lazygrid.lazy_estimator.LazyPipeline`.

The `LazyPipeline` class extends the `sklearn.pipeline.Pipeline` class by providing an interface to SQLite databases.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.preprocessing import RobustScaler, StandardScaler
import lazygrid as lg

```

(continues on next page)

(continued from previous page)

```
preprocessors = [StandardScaler(), RobustScaler()]
feature_selectors = [SelectKBest(score_func=f_classif, k=1), SelectKBest(score_func=f_
↪classif, k=2)]
classifiers = [RandomForestClassifier(random_state=42), SVC(random_state=42)]

elements = [preprocessors, feature_selectors, classifiers]

list_of_models = lg.grid.generate_grid(elements)
```

Grid search

LazyGrid implements a useful functionality to emulate the grid search algorithm by generating all possible models given the model structure and its parameters.

In this case, you should define a dictionary of arguments for the model constructor and a dictionary of arguments for the fit method. The `generate_grid_search` method will return the list of all possible models.

The following example illustrates how to use this functionality to compare keras models with different optimizers and fit parameters.

```
import keras
from keras import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.utils import to_categorical
from sklearn.metrics import f1_score
from sklearn.datasets import load_digits
from sklearn.model_selection import StratifiedKFold
import lazygrid as lg
import numpy as np
import pandas as pd
from keras.wrappers.scikit_learn import KerasClassifier


# define keras model generator
def create_keras_model(optimizer):

    kmodel = Sequential()
    kmodel.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1),
                      activation='relu',
                      input_shape=x_train.shape[1:]))
    kmodel.add(MaxPooling2D(pool_size=(2, 2)))
    kmodel.add(Flatten())
    kmodel.add(Dense(1000, activation='relu'))
    kmodel.add(Dense(n_classes, activation='softmax'))

    kmodel.compile(loss=keras.losses.categorical_crossentropy,
                   optimizer=optimizer,
                   metrics=['accuracy'])
    return kmodel


# load data set
X, y = load_digits(return_X_y=True)
X = pd.DataFrame(X)

skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

(continues on next page)

(continued from previous page)

```
list_of_splits = [split for split in skf.split(x, y)]
train_index, val_index = list_of_splits[0]
x_train, x_val = x[train_index], x[val_index]
y_train, y_val = y[train_index], y[val_index]
x_train = np.reshape(x_train, (x_train.shape[0], 8, 8, 1))
x_val = np.reshape(x_val, (x_val.shape[0], 8, 8, 1))
n_classes = len(np.unique(y_train))
if n_classes > 2:
    y_train = to_categorical(y_train)
    y_val = to_categorical(y_val)


# cast keras model into sklearn model
kmodel = KerasClassifier(create_keras_model, verbose=1, epochs=0)

# define all possible model parameters of the grid
model_params = {"optimizer": ['SGD', 'RMSprop']}
fit_params = {"epochs": [5, 10, 20], "batch_size": [10, 20]}

# generate all possible models given the parameters' grid
models, fit_parameters = lg.grid.generate_grid_search(kmodel, model_params, fit_
↪params)
```

You will find the conclusion of this example in the *plot section*.

2.2.3 Model comparison

Optimized cross-validation

`LazyPipeline` objects can be extremely useful when a large number of machine learning pipelines need to be compared through cross-validation techniques.

In fact, once a pipeline step has been fitted, LazyGrid saves the fitted step into a SQLite database. Therefore, should the step be required by another pipeline, LazyGrid fetches the model that has already been fitted from the database.

This approach may boost the speed of time-consuming steps as recursive feature elimination techniques, voting classifiers or deep neural networks.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.feature_selection import SelectKBest, f_classif, RFE
from sklearn.preprocessing import RobustScaler, StandardScaler
from sklearn.datasets import make_classification
import lazygrid as lg
import pandas as pd

X, y = make_classification(random_state=42)
X = pd.DataFrame(X)

preprocessors = [StandardScaler(), RobustScaler()]
feature_selectors = [RFE(RandomForestClassifier, n_features_to_select=10),
                     SelectKBest(score_func=f_classif, k=10)]
classifiers = [RandomForestClassifier(random_state=42), SVC(random_state=42)]

elements = [preprocessors, feature_selectors, classifiers]
```

(continues on next page)

(continued from previous page)

```
models = lg.grid.generate_grid(elements)

for model in models:
    scores = cross_validate(model, X, y, cv=10)
```

Statistical hypothesis tests

Once you have generated a list of models (or pipelines), LazyGrid provides friendly APIs to compare models' performances by using a cross-validation procedure and by analyzing the outcomes applying statistical hypothesis tests.

You can collect the cross-validation scores into a single list and call the `find_best_solution` method provided by LazyGrid. Such method applies the following algorithm: it looks for the model having the highest mean value over its cross-validation scores ("the best model"); it compares the distribution of the scores of each model against the distribution of the scores of the best model applying a statistical hypothesis test.

You can customize the comparison by modifying the statistical hypothesis test (it should be compatible with `scipy.stats`) or the significance level for the test.

```
...
scores = []
for model in models:
    score = cross_validate(model, X, y, cv=10)
    scores.append(score["test_score"])

best_idx, best_solutions_idx, pvalues = lg.statistics.find_best_solution(scores,
                                                                         ␣
↪test=mannwhitneyu,
                                                                          alpha=0.05)
```

2.2.4 Data set APIs

LazyGrid includes a set of easy-to-use APIs to fetch OpenML data sets (NB: OpenML has a database of more than 20000 data sets).

The `fetch_datasets` method allows you to smartly handle such data sets: it looks for OpenML data sets compliant with the requirements specified; for such data sets, it fetches the characteristics of their latest version; it saves in a local cache file the properties of such data sets, so that experiments can be easily reproduced using the same data sets and versions. You will find the list of downloaded data sets inside `./data/<datetime>-datalist.csv`.

The `load_openml_dataset` method can then be used to download the required data set version.

```
import lazygrid as lg

datasets = lg.datasets.fetch_datasets(task="classification", min_classes=2,
                                      max_samples=1000, max_features=10)

# get the latest (or cached) version of the iris data set
data_id = datasets.loc["iris"].did

x, y, n_classes = lg.datasets.load_openml_dataset(data_id)
```

2.3 Contributing to LazyGrid

First off, thanks for taking the time to contribute! :+1:

2.3.1 How Can I Contribute?

- Obviously source code: patches, as well as completely new files
- Bug report
- Code review

2.3.2 Coding Style

Notez Bien: All these rules are meant to be broken, **BUT** you need a very good reason **AND** you must explain it in a comment.

- Names (TL;DR): *module_name*, *package_name*, *ClassName*, *method_name*, *ExceptionName*, *function_name*, *GLOBAL_CONSTANT_NAME*, *global_var_name*, *instance_var_name*, *function_parameter_name*, *local_var_name*.
- Start names internal to a module or protected or private within a class with a single underscore (_); don't dunder (__).
- Use nouns for variables and properties names (*y = foo.baz*). Use full sentences for functions and methods names (*x = foo.calculate_next_bar(previous_bar)*); functions returning a boolean value (a.k.a., predicates) should start with the *is_* prefix (*if is_gargled(quz)*).
- Do not implement getters and setters, use properties instead. Whether a function does not need parameters consider using a property (*foo.first_bar* instead of *foo.calculate_first_bar()*). However, do not hide complexity: if a task is computationally intensive, use an explicit method (e.g., *big_number.get_prime_factors()*).
- Do not override *__repr__*.
- Use *assert* to check the internal consistency and verify the correct usage of methods, not to check for the occurrence of unexpected events. That is: The optimized bytecode should not waste time verifying the correct invocation of methods or running sanity checks.
- Explain the purpose of all classes and functions in docstrings; be verbose when needed, otherwise use single-line descriptions (note: each verbose description also includes a concise one as its first line). Be terse describing methods, but verbose in the class docstring, possibly including usage examples. Comment public attributes and properties in the *Attributes* section of the class docstring (even though PyCharm is not supporting it, yet); don't explain basic customizations (e.g., *__str__*). Comment *__init__* only when its parameters are not obvious. Use the formats suggested in the Google's style guide).
- Annotate all functions (refer to PEP-483) and PEP-484) for details).
- Use English for names, in docstrings and in comments (favor formal language over slang, wit over humor, and American English over British).
- Format source code using Yapf)'s style *"{based_on_style: google, column_limit=120, blank_line_before_module_docstring=true}"*
- Follow PEP-440) for version identification.
- Follow the Google's style guide) whenever in doubt.

2.4 Running tests

You can run all unittests from command line after having downloaded the source code from GitHub:

```
$ git clone https://github.com/glubbdubdrib/lazygrid.git
$ cd ./lazygrid
```

You can use either python:

```
$ python -m unittest discover
```

or coverage:

```
$ coverage run -m unittest discover
```

2.5 Database

lazygrid.database

lazygrid.database.**drop_db**(*db_name: str*) → None

Drop database table if it exists.

Parameters **db_name** – Database name

Returns

Return type None

Examples

```
>>> import lazygrid as lg
>>>
>>> lg.database.drop_db(db_name="my-database.sqlite")
```

lazygrid.database.**load_all_from_db**(*db_name: str*, *table_name: str = 'MODEL'*) → Optional[Any]

Load all database items.

Parameters

- **db_name** – Database name
- **table_name** – Database table to load

Returns Query result

Return type Optional[Any]

2.6 Datasets

lazygrid.datasets

lazygrid.datasets.**fetch_datasets**(*output_dir: str = './data', update_data: bool = False, min_classes: int = 0, task: str = 'classification', max_samples: int = inf, max_features: int = inf*) → pandas.core.frame.DataFrame

Load OpenML data sets compatible with the requirements.

Parameters

- **output_dir** – Directory where the .csv file will be stored
- **update_data** – If True it deletes cached data sets and downloads their latest version; otherwise it loads data sets as specified inside the cache
- **min_classes** – Minimum number of classes required for each data set
- **task** – Classification or regression
- **max_samples** – Maximum number of samples required for each data set
- **max_features** – Maximum number of features required for each data set

Returns Information required to load the latest version of each data set

Return type Dataframe

Examples

```
>>> import lazygrid as lg
>>>
>>> datasets = lg.datasets.fetch_datasets(task="classification", min_classes=2,␣
↪max_samples=1000, max_features=10)
>>> datasets.loc["iris"]
version          45
did           42098
n_samples       150
n_features        4
n_classes         3
Name: iris, dtype: int64
```

lazygrid.datasets.**load_npy_dataset**(*path_x: str, path_y: str) -> (<class 'numpy.ndarray'>, <class 'numpy.ndarray'>, <class 'int'>*)

Load npy data set.

Parameters

- **path_x** – Path to data matrix
- **path_y** – Path to data labels

Returns Data matrix, data labels, and number of classes

Return type Tuple

Examples

```
>>> import os
>>> from sklearn.datasets import make_classification
>>> import numpy as np
>>> import lazygrid as lg
>>>
```

(continues on next page)

(continued from previous page)

```
>>> x, y = make_classification(random_state=42)
>>>
>>> path_x, path_y = "x.npy", "y.npy"
>>> np.save(path_x, x)
>>> np.save(path_y, y)
>>>
>>> x, y, n_classes = lg.datasets.load_npy_dataset(path_x, path_y)
```

lazygrid.datasets.**load_openml_dataset**(*data_id: int = None*, *dataset_name: str = None*) -> (<*class 'numpy.ndarray'*>, <*class 'numpy.ndarray'*>, <*class 'int'*>)

Load OpenML data set.

Parameters

- **data_id** – Data set identifier
- **dataset_name** – Data set name

Returns Data matrix, data labels, and number of classes

Return type Tuple

Examples

```
>>> import lazygrid as lg
>>>
>>> x, y, n_classes = lg.datasets.load_openml_dataset(dataset_name="iris")
>>> n_classes
3
```

2.7 Grid

lazygrid.grid

lazygrid.grid.**generate_grid**(*elements: list*, *lazy: bool = True*, ***kwargs*) → list

Generate all possible combinations of sklearn Pipelines given the input steps.

Parameters

- **elements** – List of elements used to generate the pipelines
- **lazy** – If True it generates LazyPipelines objects; if False it generates standard sklearn Pipeline objects
- **kwargs** – Keyword arguments to generate Pipeline objects

Returns List of pipelines

Return type list

Example

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.svm import SVC
>>> from sklearn.feature_selection import SelectKBest, f_classif
>>> from sklearn.preprocessing import RobustScaler, StandardScaler
>>> import lazygrid as lg
>>>
>>> preprocessors = [StandardScaler(), RobustScaler()]
>>> feature_selectors = [SelectKBest(score_func=f_classif, k=1),␣
↪SelectKBest(score_func=f_classif, k=2)]
>>> classifiers = [RandomForestClassifier(random_state=42), SVC(random_state=42)]
>>>
>>> elements = [preprocessors, feature_selectors, classifiers]
>>>
>>> pipelines = lg.grid.generate_grid(elements)
```

lazygrid.grid.**generate_grid_search**(*model: keras.wrappers.scikit_learn.KerasClassifier*, *model_params: dict*, *fit_params: dict*) → Tuple[List[keras.engine.training.Model], List[dict]]

Generate all possible combinations of models.

Parameters

- **model** – Model architecture
- **model_params** – Model parameters. For each key the dictionary should contain a list of possible values
- **fit_params** – Fit parameters. For each key the dictionary should contain a list of possible values

Returns Models and their corresponding fit parameters

Return type Tuple

Example

```
>>> import keras
>>> from keras import Sequential
>>> from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
>>> import lazygrid as lg
>>> from keras.wrappers.scikit_learn import KerasClassifier
>>>
>>> # define keras model generator
>>> def create_keras_model(input_shape, optimizer, n_classes):
...     kmodel = Sequential()
...     kmodel.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1), activation='relu
↪', input_shape=input_shape))
...     kmodel.add(MaxPooling2D(pool_size=(2, 2)))
...     kmodel.add(Flatten())
...     kmodel.add(Dense(1000, activation='relu'))
...     kmodel.add(Dense(n_classes, activation='softmax'))
...
...     kmodel.compile(loss=keras.losses.categorical_crossentropy,
...                    optimizer=optimizer, metrics=['accuracy'])
...     return kmodel
>>>
>>> # cast keras model into sklearn model
>>> kmodel = KerasClassifier(create_keras_model)
```

(continues on next page)

(continued from previous page)

```
>>>
>>> # define all possible model parameters of the grid
>>> model_params = {"optimizer": ['SGD', 'RMSprop'], "input_shape": [(28, 28, 3)],
↪ "n_classes": [10]}
>>> fit_params = {"epochs": [5, 10, 20], "batch_size": [10, 20]}
>>>
>>> # generate all possible models given the parameters' grid
>>> models, fit_parameters = lg.grid.generate_grid_search(kmodel, model_params,␣
↪fit_params)
```

2.8 Plotter

lazygrid.plotter

lazygrid.plotter.**plot_boxplots**(*scores: List, labels: List[str], file_name: str, title: str, output_dir: str = './figures'*) → dict

Generate and save boxplots.

Parameters

- **scores** – List of scores to compare
- **labels** – Name / identifier of each score list
- **file_name** – Output file name
- **title** – Figure title
- **output_dir** – Output directory

Returns boxplot object

Return type Boxplot

lazygrid.plotter.**plot_learning_curve**(*estimator*, *title*, *X*, *y*, *axes=None*, *ylim=None*, *cv=None*, *n_jobs=None*, *train_sizes=array([0.1, 0.325, 0.55, 0.775, 1.])*)

Generate 3 plots: the test and training learning curve, the training samples vs fit times curve, the fit times vs score curve.

Parameters

- **estimator** (*object type that implements the "fit" and "predict" methods*) – An object of that type which is cloned for each validation.
- **title** (*string*) – Title for the chart.
- **X** (*array-like, shape (n_samples, n_features)*) – Training vector, where n_samples is the number of samples and n_features is the number of features.
- **y** (*array-like, shape (n_samples) or (n_samples, n_features), optional*) – Target relative to X for classification or regression; None for unsupervised learning.
- **axes** (*array of 3 axes, optional (default=None)*) – Axes to use for plotting the curves.
- **ylim** (*tuple, shape (ymin, ymax), optional*) – Defines minimum and maximum yvalues plotted.

- **cv** (*int, cross-validation generator or an iterable, optional*) – Determines the cross-validation splitting strategy. Possible inputs for cv are:
  - None, to use the default 5-fold cross-validation,
  - integer, to specify the number of folds.
  - CV splitter,
  - An iterable yielding (train, test) splits as arrays of indices.

  For integer/None inputs, if `y` is binary or multiclass, `StratifiedKFold` used. If the estimator is not a classifier or if `y` is neither binary nor multiclass, `KFold` is used.

  Refer User Guide for the various cross-validators that can be used here.
- **n_jobs** (*int or None, optional (default=None)*) – Number of jobs to run in parallel. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details.
- **train_sizes** (*array-like, shape (n_ticks,), dtype float or int*) – Relative or absolute numbers of training examples that will be used to generate the learning curve. If the dtype is float, it is regarded as a fraction of the maximum size of the training set (that is determined by the selected validation method), i.e. it has to be within (0, 1]. Otherwise it is interpreted as absolute sizes of the training sets. Note that for classification the number of samples usually have to be big enough to contain at least one sample from each class. (default: np.linspace(0.1, 1.0, 5))

2.9 Statistics

lazygrid.statistics

lazygrid.statistics.**confidence_interval_mean_t**(*x: numpy.ndarray*, *cl: float = 0.05*) → List

Compute the confidence interval of the mean from sample data.

Parameters

- **x** – Sample
- **cl** – Confidence level

Returns confidence interval

Return type List

Examples

```
>>> import numpy as np
>>> import lazygrid as lg
>>>
>>> np.random.seed(42)
>>> x = np.random.normal(loc=0, scale=2, size=10)
>>> confidence_level = 0.05
>>>
>>> lg.statistics.confidence_interval_mean_t(x, confidence_level)
[-0.13829578539063092, 1]
```

Notes

You should use the t distribution rather than the normal distribution when the variance is not known and has to be estimated from sample data.

When the sample size is large, say 100 or above, the t distribution is very similar to the standard normal distribution. However, with smaller sample sizes, the t distribution is leptokurtic, which means it has relatively more scores in its tails than does the normal distribution. As a result, you have to extend farther from the mean to contain a given proportion of the area.

lazygrid.statistics.**find_best_solution**(*solutions: list*, *test: Callable = <function mannwhitneyu>*, *alpha: float = 0.05*, ***kwargs) -> (<class 'int'>, <class 'list'>, <class 'list'>*)

Find the best solution in a list of candidates, according to a statistical test and a significance level (alpha).

The best solution is defined as the one having the highest mean value.

Parameters

- **solutions** – List of candidate solutions
- **test** – Statistical test
- **alpha** – Significance level
- **kwargs** – Keyword arguments required by the statistical test

Returns

- the position of the best solution inside the candidate input list;
- the positions of the solutions which are not separable from the best one;
- the list of p-values returned by the statistical test while comparing the best solution to the other candidates

Return type Tuple

Examples

```
>>> from sklearn.linear_model import LogisticRegression, RidgeClassifier
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.datasets import make_classification
>>> from sklearn.model_selection import cross_val_score
>>> import lazygrid as lg
>>>
>>> x, y = make_classification(random_state=42)
>>>
>>> model1 = LogisticRegression(random_state=42)
>>> model2 = RandomForestClassifier(random_state=42)
>>> model3 = RidgeClassifier(random_state=42)
>>> model_names = ["LogisticRegression", "RandomForestClassifier",
↪"RidgeClassifier"]
>>>
>>> score1 = cross_val_score(estimator=model1, X=x, y=y, cv=10)
>>> score2 = cross_val_score(estimator=model2, X=x, y=y, cv=10)
>>> score3 = cross_val_score(estimator=model3, X=x, y=y, cv=10)
>>>
>>> scores = [score1, score2, score3]
>>> best_idx, best_solutions_idx, pvalues = lg.statistics.find_best_
↪solution(scores)
```

(continues on next page)

(continued from previous page)

```
>>> model_names[best_idx]
'LogisticRegression'
>>> best_solutions_idx
[0, 2]
>>> pvalues #doctest: +ELLIPSIS
[0.4782..., 0.0360..., 0.1610...]
```

2.10 Lazy Estimator

lazygrid.lazy_estimator

class lazygrid.lazy_estimator.**LazyPipeline**(*steps*, *database: str = './database/'*, *verbose: bool = False*)

A LazyPipeline estimator.

A lazy pipeline is a sklearn-like pipeline that follows the memoization paradigm. Once the pipeline has been fitted, its steps are pickled and stored in a local database. Therefore, when the program starts again, the pipeline will fetch its fitted steps from the database and will skip the fit operation.

Parameters

- **steps** – List of (name, transform) tuples (implementing fit/transform) that are chained, in the order in which they are chained, with the last object an estimator.
- **database** – Used to cache the fitted transformers of the pipeline. It is the path to the database directory. Caching the transformers is advantageous when fitting is time consuming.
- **verbose** (*bool, default=False*) – If True, the time elapsed while fitting each step will be printed as it is completed.

named_steps

Read-only attribute to access any step parameter by user given name. Keys are step names and values are steps parameters.

Type bunch object, a dictionary with attribute access

See also:

sklearn.pipeline.Pipeline

Examples

```
>>> from sklearn import svm
>>> from sklearn.datasets import make_classification
>>> from sklearn.feature_selection import SelectKBest
>>> from sklearn.feature_selection import f_regression
>>> from lazygrid.lazy_estimator import LazyPipeline
>>> import pandas as pd
>>> # generate some data to play with
>>> X, y = make_classification(
...     n_informative=5, n_redundant=0, random_state=42)
>>> X = pd.DataFrame(X)
>>> # ANOVA SVM-C
>>> anova_filter = SelectKBest(f_regression, k=5)
>>> clf = svm.SVC(kernel='linear')
```

(continues on next page)

(continued from previous page)

```
>>> anova_svm = LazyPipeline([('anova', anova_filter), ('svc', clf)])
>>> # You can set the parameters using the names issued
>>> # For instance, fit using a k of 10 in the SelectKBest
>>> # and a parameter 'C' of the svm
>>> anova_svm.set_params(anova__k=10, svc__C=.1).fit(X, y)
Pipeline(steps=[('anova', SelectKBest(...)), ('svc', SVC(...))])
>>> prediction = anova_svm.predict(X)
>>> anova_svm.score(X, y)
0.83
>>> # getting the selected features chosen by anova_filter
>>> anova_svm['anova'].get_support()
array([False, False,  True,  True, False, False,  True,  True, False,
       True, False,  True,  True, False,  True, False,  True,  True,
       False, False])
>>> # Another way to get selected features chosen by anova_filter
>>> anova_svm.named_steps.anova.get_support()
array([False, False,  True,  True, False, False,  True,  True, False,
       True, False,  True,  True, False,  True, False,  True,  True,
       False, False])
>>> # Indexing can also be used to extract a sub-pipeline.
>>> sub_pipeline = anova_svm[:1]
>>> sub_pipeline
Pipeline(steps=[('anova', SelectKBest(...))])
>>> coef = anova_svm[-1].coef_
>>> anova_svm['svc'] is anova_svm[-1]
True
>>> coef.shape
(1, 10)
>>> sub_pipeline.inverse_transform(coef).shape
(1, 20)
```

fit (*X: pandas.core.frame.DataFrame*, *y: Iterable = None*, ***fit_params*)
Fit the model

Fit all the transforms one after the other and transform the data, then fit the transformed data using the final estimator.

Parameters

- **X** – Training data. Must fulfill input requirements of first step of the pipeline.
- **y** – Training targets. Must fulfill label requirements for all steps of the pipeline.
- ****fit_params** (*dict of string -> object*) – Parameters passed to the `fit` method of each step, where each parameter name is prefixed such that parameter `p` for step `s` has key `s__p`.

Returns This estimator

Return type self

2.11 Authors

- Pietro Barbiero - Mathematical engineer - GitHub
- Giovanni Squillero - Professor of computer science at Politecnico di Torino - GitHub

2.12 Apache License

Version 2.0

Date January 2004

URL http://www.apache.org/licenses/

2.12.1 TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means *(i)* the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or *(ii)* ownership of fifty percent (50%) or more of the outstanding shares, or *(iii)* beneficial ownership of such entity.

"You" (or **"Your"**) shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

- You must give any other recipients of the Work or Derivative Works a copy of this License; and
- You must cause any modified files to carry prominent notices stating that You changed the files; and
- You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
- If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions.

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks.

This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty.

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an **"AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND**, either express or implied, including, without limitation, any warranties or conditions of **TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE**. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability.

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability.

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

```
Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

2.13 Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index