searchgrid Documentation

Release 0.2.0

Joel Nothman

Feb 16, 2020

Contents

Helps building parameter grids for `scikit-learn grid search`.

Specifying a parameter grid for `sklearn.model_selection.GridSearchCV` in Scikit-Learn can be annoying, particularly when:

- you change your code to wrap some estimator in, say, a `Pipeline` and then need to prefix all the parameters in the grid using lots of __s
- you are searching over multiple grids (i.e. your `param_grid` is a list) and you want to make a change to all of those grids

searchgrid allows you to define (and change) the grid together with the esimator, reducing effort and sometimes code. It stores the parameters you want to search on each particular estimator object. This makes it much more straightforward to specify complex parameter grids, and means you don't need to update your grid when you change the structure of your composite estimator.

It provides two main functions:

- *`searchgrid.set_grid`* is used to specify the parameter values to be searched for an estimator or GP kernel.
- *`searchgrid.make_grid_search`* is used to construct the `GridSearchCV` object using the parameter space the estimator is annotated with.

Other utilities for constructing search spaces include:

- *`searchgrid.build_param_grid`*
- *`searchgrid.make_pipeline`*
- *`searchgrid.make_union`*

CHAPTER 1

Quick Start

If scikit-learn is installed, then, in a terminal:

```
pip install searchgrid
```

and use in Python:

```
from searchgrid import set_grid, make_grid_search
estimator = set_grid(MyEstimator(), param=[value1, value2, value3])
search = make_grid_search(estimator, cv=..., scoring=...)
search.fit(X, y)
```

Or search for the best among multiple distinct estimators/pipelines:

```
search = make_grid_search([estimator1, estimator2], cv=..., scoring=...)
search.fit(X, y)
```

Motivating examples

Let's look over some of the messy change cases. We'll get some imports out of the way.:

```
>>> from sklearn.pipeline import Pipeline
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.feature_selection import SelectKBest
>>> from sklearn.decomposition import PCA
>>> from searchgrid import set_grid, make_grid_search
>>> from sklearn.model_selection import GridSearchCV
```

Wrapping an estimator in a pipeline. You had code which searched over parameters for a classifier. Now you want to search for that classifier in a Pipeline. With plain old scikit-learn, you have to insert __s and change:

```
>>> gs = GridSearchCV(LogisticRegression(), {'C': [.1, 1, 10]})
```

to:

```
>>> gs = GridSearchCV(Pipeline([('reduce', SelectKBest()),
...                             ('clf', LogisticRegression())]),
...                   {'clf__C': [.1, 1, 10]})
```

With searchgrid we only have to wrap our classifier in a Pipeline, and do not have to change the parameter grid, adding the `clf__` prefix. From:

```
>>> lr = set_grid(LogisticRegression(), C=[.1, 1, 10])
>>> gs = make_grid_search(lr)
```

to:

```
>>> lr = set_grid(LogisticRegression(), C=[.1, 1, 10])
>>> gs = make_grid_search(Pipeline([('reduce', SelectKBest()),
...                                 ('clf', lr)]))
```

You want to change the estimator being searched in a pipeline. With scikit-learn, to use PCA instead of SelectKBest, you change:

```
>>> pipe = Pipeline([('reduce', SelectKBest()),
...                  ('clf', LogisticRegression())])
>>> gs = GridSearchCV(pipe,
...                   {'reduce__k': [5, 10, 20],
...                    'clf__C': [.1, 1, 10]})
```

to:

```
>>> pipe = Pipeline([('reduce', PCA()),
...                  ('clf', LogisticRegression())])
>>> gs = GridSearchCV(pipe,
...                   {'reduce__n_components': [5, 10, 20],
...                    'clf__C': [.1, 1, 10]})
```

Note that `reduce__k` became `reduce__n_components`.

With searchgrid it's easier because you change the estimator and the parameters in the same place:

```
>>> reduce = set_grid(SelectKBest(), k=[5, 10, 20])
>>> lr = set_grid(LogisticRegression(), C=[.1, 1, 10])
>>> pipe = Pipeline([('reduce', reduce),
...                  ('clf', lr)])
>>> gs = make_grid_search(pipe)
```

becomes:

```
>>> reduce = set_grid(PCA(), n_components=[5, 10, 20])
>>> lr = set_grid(LogisticRegression(), C=[.1, 1, 10])
>>> pipe = Pipeline([('reduce', reduce),
...                  ('clf', lr)])
>>> gs = make_grid_search(pipe)
```

Searching over multiple grids. You want to take the code from the previous example, but instead search over feature selection and PCA reduction in the same search.

Without searchgrid:

```
>>> pipe = Pipeline([('reduce', None),
...                  ('clf', LogisticRegression())])
>>> gs = GridSearchCV(pipe, [{'reduce': [SelectKBest()],
...                           'reduce__k': [5, 10, 20],
...                           'clf__C': [.1, 1, 10]},
...                          {'reduce': [PCA()],
...                           'reduce__n_components': [5, 10, 20],
...                           'clf__C': [.1, 1, 10]}])
```

With searchgrid:

```
>>> kbest = set_grid(SelectKBest(), k=[5, 10, 20])
>>> pca = set_grid(PCA(), n_components=[5, 10, 20])
>>> lr = set_grid(LogisticRegression(), C=[.1, 1, 10])
>>> pipe = set_grid(Pipeline([('reduce', None),
...                           ('clf', lr)]),
...                 reduce=[kbest, pca])
>>> gs = make_grid_search(pipe)
```

And since you no longer care about step names, use *searchgrid.make_pipeline* to express alternative steps even more simply:

```
>>> from searchgrid import make_pipeline
>>> kbest = set_grid(SelectKBest(), k=[5, 10, 20])
>>> pca = set_grid(PCA(), n_components=[5, 10, 20])
>>> lr = set_grid(LogisticRegression(), C=[.1, 1, 10])
>>> pipe = make_pipeline([kbest, pca], lr)
>>> gs = make_grid_search(pipe)
```

CHAPTER 3

API Reference

`searchgrid.`**`build_param_grid`**(*estimator*)

Determine the parameter grid annotated on the estimator

Parameters

estimator [scikit-learn compatible estimator] Should have been annotated using *`set_grid()`*

Notes

Most often, it is unnecessary for this to be used directly, and *`make_grid_search()`* should be used instead.

`searchgrid.`**`make_grid_search`**(*estimator*, ***kwargs*)

Construct a GridSearchCV with the given estimator and its set grid

Parameters

estimator [(list of) estimator] When a list, the estimators are searched over.

kwargs Other parameters to the `sklearn.model_selection.GridSearchCV` constructor.

`searchgrid.`**`make_pipeline`**(**steps*, ***kwargs*)

Construct a Pipeline with alternative estimators to search over

Parameters

steps Each step is specified as one of:

- an estimator instance
- None (meaning no transformation)
- a list of the above, indicating that a grid search should alternate over the estimators (or None) in the list

kwargs Keyword arguments to the constructor of `sklearn.pipeline.Pipeline`.

Notes

Each step is named according to the set of estimator types in its list:

- if a step has only one type of estimator (disregarding None), it takes that estimator's class name (lowercased)
- if a step has estimators of mixed type, the step is named 'alt'
- if there are multiple steps of the same name using the above rules, a suffix '-1', '-2', etc. is added.

Examples

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> from sklearn.feature_extraction.text import TfidfTransformer
>>> from sklearn.feature_selection import SelectKBest
>>> from sklearn.decomposition import PCA
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.model_selection import ParameterGrid
>>> from searchgrid import make_pipeline, build_param_grid
>>> pipe = make_pipeline(CountVectorizer(),
...                      [TfidfTransformer(), None],
...                      [PCA(n_components=5), SelectKBest(k=5)],
...                      [set_grid(LogisticRegression(),
...                                C=[.1, 1., 10.]),
...                       RandomForestClassifier()])
>>> pipe.steps  # doctest: +NORMALIZE_WHITESPACE +ELLIPSIS
[('countvectorizer', CountVectorizer(...)),
 ('tfidftransformer', TfidfTransformer(...)),
 ('alt-1', PCA(...)),
 ('alt-2', LogisticRegression(...))]
>>> n_combinations = len(ParameterGrid(build_param_grid(pipe)))
>>> n_combinations
... # 2 * 2 * (3 + 1)
16
```

searchgrid.**make_union**(**transformers*, ***kwargs*)

Construct a FeatureUnion with alternative estimators to search over

Parameters

steps Each step is specified as one of:

- an estimator instance
- None (meaning no features)
- a list of the above, indicating that a grid search should alternate over the estimators (or None) in the list

kwargs Keyword arguments to the constructor of `sklearn.pipeline.FeatureUnion`.

Notes

Each step is named according to the set of estimator types in its list:

- if a step has only one type of estimator (disregarding None), it takes that estimator's class name (lowercased)

- if a step has estimators of mixed type, the step is named 'alt'
- if there are multiple steps of the same name using the above rules, a suffix '-1', '-2', etc. is added.

`searchgrid.`**`set_grid`**`(`*estimator*, ***grid*`)`

Set the grid to search for the specified estimator

Overwrites any previously set grid.

Parameters

grid [dict (str -> list of values)] Keyword arguments define the values to be searched for each specified parameter.

Returns

estimator Useful for chaining

CHAPTER 4

Changelog

4.1 v0.2

- Added *searchgrid.make_pipeline* and *searchgrid.make_union*. #12

4.2 v0.1.1

- Fixed a bug where the grid of the default estimator in a Pipeline step was attributed to alternatives for that step. #10

Python Module Index

Index