scikit-nni Documentation

Release 0.2.1

Kapil Sachdeva

Oct 22, 2019

Contents:

CHAPTER 1

scikit-nni

Hyper parameters search for scikit-learn components using Microsoft NNI

- Free software: Apache Software License 2.0
- Documentation: https://scikit-nni.readthedocs.io.

1.1 Introduction

Microsoft NNI (Neural Network Intelligence) is a toolkit to help users run automated machine learning (AutoML) experiments. The tool dispatches and runs trial jobs generated by tuning algorithms to search the best neural architecture and/or hyper-parameters in different environments like local machine, remote servers and cloud.

Read and explore more about Microsoft NNI here - https://github.com/microsoft/nni

scikit-nni is a helper tool (and a package) that :

- generates the configuration (config.yaml & search-space.json) required for NNI
- automatically builds the scikit-learn pipelines based on your specification and becomes an experiment/trial code for Microsoft NNI to run.

1.1.1 What value does this tool add to Microsoft NNI ?

First note that this tool is specifically written to only help with scikit-learn pipelines and to tune classification algorithms. In near future, I would add the support for regression algorithms as well.

Now when you use Microsoft NNI you need to specify (at minimum) 3 files :

- A search space (json) file that contains the parameters that you want to search/tune.

- **Your code/experiment. In your experiment code, you perform these tasks in sequence :**
  - Request for the parameters from NNI server
  - Create your model using these parameters
  - Fit your model
  - Score your model
  - Report the score to NNI server.
- a configuration file where you specify the tuner, which mode to use to run, path to your code file and search space file.

scikit-nni eliminiates the second step i.e. it builds the scikit pipelines, request NNI server for parameters and also report back the score of your model. It also simplifies (in IMHO) the input specification by only requiring one file instead of 3.

Sounds interesting ? Then read the documentation below, install scikit-nni, and more importantly provide feedback if it does not work for you and/or you think it can be improved.

1.2 Features

- Hyperparameters search for scikit-learn pipelines using Microsoft NNI
- No code required to define the pipelines
- Built-in datasource reader for reading npz files for classification
- Support for using custom datasource reader
- Single configuration file to define NNI configuration and search space

I plan to add more datasource readers (e.g. CSV, libSVM format files etc). Contributions are always welcome !

1.3 Usage

1.3.1 Step 1 - Write a specification file

The specification file is essentially a YAML file but with extension *.nni.yml*

There are 4 parts (sections) in the configuration file.

Datasource Section

This is where you will specify the (python) callable that *sknni* would be invoking to get the training and test dataset.

The callable **must** return two values where each value is a *tuple* of two items. The first tuple consists of training data *(X_train, y_train)* and the second tuple consists of test data *(X_test, y_test)*.

An example callable would look like this:

```
import numpy as np

from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

class ACustomDataSource(object):
    def __init__(self):
        pass

    def __call__(self, test_size:float=0.25):
        digits = load_digits()
        X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.
↪target, random_state=99, test_size=test_size)

        return (X_train, y_train), (X_test, y_test)
```

In the above example, the callable generates the train and test dataset. The callable can even have paramaters for e.g. in this example you could optionally pass the fraction of data to be used for testing.

Now let's see how you would add it in the specification file.

```
# Datasource is how you specify which callable
# sknni will invoke to get the data
dataSource:
    reader: yourmodule.ACustomDataSource
    params:
        test_size: 0.30
```

Make sure that during the exeuction of the experiment your datasource (i.e. in this case *yourmodule.ACustomDataSource*) is available in the PYTHONPATH.

Here is an additional example showing the usage of a built-in datasource reader

```
dataSource:
    reader: sknni.datasource.NpzClassificationSource
    params:
        dir_path: /Users/ksachdeva/Desktop/Dev/myoss/scikit-nni/examples/data/
↪multiclass-classification
```

NpzClassificationSource expects that at *dir_path* you have two folders - train and test. In each folder are the files named as 0.npz, 1.npz etc. Every file contains that features for that corresponding class.

The repository contains two such datasources to do binary and multiclass classifications.

Pipline definition Section

Below is an example of this type of section. You simply specify the list of steps of your scikit-learn Pipeline.

Note - The sequence of steps is very important.

What you **MUST** ensure is that the full qualified name of your scikit-learn preprocessors, transformers and estimators is correctly specified & spelled. *sknni* uses reflection and introspection to create the instances of these components so if you have a typo in the names and/or they are not available in your PYTHONPATH you will get an error at experiment execution time.

```
sklearnPipeline:
    name: normalizer_svc
```

(continues on next page)

(continued from previous page)

```
    steps:
        normalizer:
            type: sklearn.preprocessing.Normalizer
            classArgs:
                norm: l2
        svc:
            type: sklearn.svm.SVC
```

In above example, there are 2 steps. The first step is to normalize the data and the second step is train a classifier using Support Vector Machine.

Search Space Section

This section corresponds to the search space for your hyperparameters. When you use `nnictrl` this is typically specified in search-space.json file.

See https://nni.readthedocs.io/en/latest/Tutorial/SearchSpaceSpec.html to learn more about the search space syntax.

Here are the important things to note about this section -

- The syntax is the same (except we are using YAML here instead of JSON) for specifiying parameter types and ranges.
- You **MUST** specifiy the parameters corresponding to the step in your scikit pipeline.
- You **MUST** use the names of the parameters that are **same as** the ones accepted by the constructors of scikit-learn components (i.e. preprocessors, estimators etc).

Below is an example of this type of section.

```
nniConfigSearchSpace:
    - normalizer:
        norm:
            _type: choice
            _value: [l2, l1]
    - svc:
        C:
            _type: uniform
            _value: [0.1,0.0]
        kernel:
            _type: choice
            _value: [linear,rbf,poly,sigmoid]
        degree:
            _type: choice
            _value: [1,2,3,4]
        gamma:
            _type: uniform
            _value: [0.01,0.1]
        coef0:
            _type: uniform
            _value: [0.01,0.1]
```

Note that *sklearn.svm.SVC* takes C, kernel, degree, gamman and coef0 is the paramaters and hence we have used here the same names (keys) in the search space specification. You can add as many or as little parameters to search for.

NNI Config Section

This is the simplest of all sections as there is nothing new here from sknni perspective. You just copy-paste here your NNI's config.yaml here. You do not have to specify *codedir* and *command* field in the *trial* subsection as this is added by the sknni in the generated configuration files.

See https://nni.readthedocs.io/en/latest/Tutorial/ExperimentConfig.html

Here is an example of this type of section.

```
# This is exactly same as the one that of NNI
# except that you do not have to specify the command
# and code fields. They are automatically added by the sknni generator
nniConfig:
    authorName: default
    experimentName: example_sklearn-classification
    trialConcurrency: 1
    maxExecDuration: 1h
    maxTrialNum: 100
    trainingServicePlatform: local
    useAnnotation: false
    tuner:
        builtinTunerName: TPE
        classArgs:
            optimize_mode: maximize
    trial:
        gpuNum: 0
```

You can look at the various examples in the repository to learn how to define your own specification file.

1.3.2 Step 2 - Generate your experiment

```
sknni generate-experiment --spec example/basic_svc.nni.yml --output-dir experiments
```

Above command will create a directory experiments/svc-classification with the following files

- The original specification file i.e. basic_svc.nni.yml (used during experiment run as well)
- Generated Microsoft NNI's config.yml
- Generated Microsoft NNI's search-space.json

Note - there is no python file as typically shown in the examples of Microsoft NNI as the command in ends up invoking *sknni* entry point when the experiment is run.

1.3.3 Step 3 - Run your experiment

This is same as running *nnitctl*

```
nnictl create --config experiments/svc-classification/config.yml
```

1.4 Troubleshooting

1.4.1 My trials are failing what is wrong ?

Your trial could fail for many reasons -

- Bug in your DataSource code resulting the exception/error
- Wrong inputs to your (or built-in) DataSources resulting in exception/error
- Your DataSource (python callable) could not be found

Here is what I would recommend -

- Test your DataSource code
- The webui does not always display all the errors/logs so look at the log of your trials and more specifically stderr file

```
cat $HOME/nni/experiments/<YOUR_EXPERIMENT_ID>/trials/<TRIAL_ID>/stderr

cat $HOME/nni/experiments/<YOUR_EXPERIMENT_ID>/trials/<TRIAL_ID>/trial.log
```

1.5 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install scikit-nni, run this command in your terminal:

```
$ pip install scikit-nni
```

This is the preferred method to install scikit-nni, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for scikit-nni can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/ksachdeva/scikit-nni
```

Or download the tarball:

```
$ curl -OJL https://github.com/ksachdeva/scikit-nni/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

3.1 Step 1 - Write specification file

The specification file is essentially a YAML file but with extension *.nni.yml*

There are 4 parts (sections) in the configuration file.

3.1.1 Datasource Section

This is where you will specify the (python) callable that *sknni* would invoking to the training and test dataset.

The callable should return 2 values where each value is a *tuple* of two items. The first tuple consists of training data *(X_train, y_train)* and the second tuple consists of test data *(X_test, y_test)*.

An example callable would look like this:

```
import numpy as np

from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

class ACustomDataSource(object):
    def __init__(self):
        pass

    def __call__(self, test_size:float=0.25):
        digits = load_digits()
        X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.
↪target, random_state=99, test_size=test_size)

        return (X_train, y_train), (X_test, y_test)
```

In the above example, the callable generates the train and test dataset. The callable can even have paramaters for e.g. in this example you could optionally pass the fraction of data to be used for testing purposes.

Now let's see how you would specify in the specification file.

```
# Datasource is how you specify which callable
# sknni will invoke to get the data
dataSource:
    reader: yourmodule.ACustomDataSource
    params:
        test_size: 0.30
```

Make sure that during the exeuction of the experiment your datasource (i.e. in this case *yourmodule.ACustomDataSource*) is available in the PYTHONPATH.

Here is an additional example showing the usage of an built-in datasource reader

```
dataSource:
    reader: sknni.datasource.NpzClassificationSource
    params:
        dir_path: /Users/ksachdeva/Desktop/Dev/myoss/scikit-nni/examples/data/
↪multiclass-classification
```

3.1.2 Pipline definition Section

Below is the example of the section. You simply specify the list of steps of your typical scikit-learn Pipeline.

Note - The sequence of steps is very important.

What you **MUST** ensure is that the full qualified name of your scikit-learn preprocessors, transformers and estimators is correctly specified. *sknni* uses reflection and introspection to create the instances so if you have a typo in the names and/or they are not available in your PYTHONPATH you will get an error at experiment execution time.

```
sklearnPipeline:
    name: normalizer_svc
    steps:
        normalizer:
            type: sklearn.preprocessing.Normalizer
        svc:
            type: sklearn.svm.SVC
```

In above example, there are 2 steps. The first step is to normalize the data and the second step is train a classifier using Support Vector Machine.

3.1.3 Search Space Section

This section corresponds to the search space for your hyperparameters. When you `nnictrl` this is typically specified in search-space.json file.

Here are the important things to note about this section -

- The syntax is the same (except we are using YAML here instead of JSON) for specifiying parameter types and ranges.
- You **MUST** specifiy the parameters corresponding to the step in your scikit pipeline.
- You **MUST** use the names of the parameters that are same as the ones accepted by scikit-learn components (i.e. preprocessors, estimators etc).

Below is an example of this section.

```
nniConfigSearchSpace:
    - normalizer:
        norm:
            _type: choice
            _value: [l2, l1]
    - svc:
        C:
            _type: uniform
            _value: [0.1,0.0]
        kernel:
            _type: choice
            _value: [linear,rbf,poly,sigmoid]
        degree:
            _type: choice
            _value: [1,2,3,4]
        gamma:
            _type: uniform
            _value: [0.01,0.1]
        coef0:
            _type: uniform
            _value: [0.01,0.1]
```

Note that *sklearn.svm.SVC* takes C, kernel, degree, gamman and coef0 is the paramaters and hence we have used here the same names (keys) in the search space specification. You can add as many or as little parameters to search for.

3.1.4 NNI Config Section

This is the simplest of all sections as there is nothing new here from sknni perspective. You just copy-paste here your NNI's config.yaml here. You do not have to specify *codedir* and *command* field in the *trial* subsection as this is added by the sknni in the generated configuration files.

Here is an example.

```
# This is exactly same as the one that of NNI
# except that you do not have to specify the command
# and code fields. They are automatically added by the sknni generator
nniConfig:
    authorName: default
    experimentName: example_sklearn-classification
    trialConcurrency: 1
    maxExecDuration: 1h
    maxTrialNum: 100
    trainingServicePlatform: local
    useAnnotation: false
    tuner:
        builtinTunerName: TPE
        classArgs:
            optimize_mode: maximize
    trial:
        gpuNum: 0
```

You can look at the various examples in the repository to learn how to define your own specification file.

3.2 Step 2 - Generate your experiment

```
sknni generate-experiment --spec example/basic_svc.nni.yml --output-dir experiments
```

Above command will create a directory experiments/svc-classification will following files

- The original specification file i.e. basic_svc.nni.yml (used during experiment run as well)
- Generated Microsoft NNI's config.yml
- Generated Microsoft NNI's search-space.json

3.3 Step 3 - Run your experiment

This is same as running *nnitctl*

```
nnictl create --config experiments/svc-classification/config.yml
```

CHAPTER 4

sknni

4.1 sknni package

4.1.1 Subpackages

sknni.datasource package

Module contents

sknni.internals package

Module contents

4.1.2 Submodules

4.1.3 sknni.cli module

4.1.4 Module contents

Top-level package for scikit-nni.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/ksachdeva/scikit-nni/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

scikit-nni could always use more documentation, whether as part of the official scikit-nni docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/ksachdeva/scikit-nni/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *sknni* for local development.

1. Fork the *scikit-nni* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/scikit-nni.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv sknni
$ cd sknni/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 sknni tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6 and 3.7, and for PyPy. Check https://travis-ci.org/ksachdeva/scikit-nni/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_sknni
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

History

6.1 0.2.1 (2019-10-21)

- Support for classArgs to steps in pipeline section

6.2 0.1.1 (2019-10-20)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

S