# How to Contribute

---

I would be very happy about any kind of contributions that help to improve and extend the functionality of mlxtend.

## Quick Contributor Checklist

This is a quick checklist about the different steps of a typical contribution to mlxtend (and

other open source projects). Consider copying this list to a local text file (or the issue tracker)

and checking off items as you go.

1. [ ] Open a new "issue" on GitHub to discuss the new feature / bug fix

2. [ ] Fork the mlxtend repository from GitHub (if not already done earlier)

3. [ ] Create and check out a new topic branch (please don't make modifications in the master branch)

4. [ ] Implement the new feature or apply the bug-fix

5. [ ] Add appropriate unit test functions in `mlxtend/\*/tests`

6. [ ] Run `PYTHONPATH='.' pytest ./mlxtend -sv` and make sure that all unit tests pass

7. [ ] Check for style issues by running `flake8 ./mlxtend` (you may want to run `pytest` again after you made modifications to the code)

8. [ ] Add a note about the modification/contribution to the `./docs/sources/changelog.md` file

9. [ ] Modify documentation in the appropriate location under `mlxtend/docs/sources/`

10. [ ] Push the topic branch to the server and create a pull request

11. [ ] Check the Travis-CI build passed at [https://travis-ci.org/rasbt/mlxtend](https://travis-ci.org/rasbt/mlxtend)

12. [ ] Check/improve the unit test coverage at [https://coveralls.io/github/rasbt/mlxtend](https://coveralls.io/github/rasbt/mlxtend)

13. [ ] Check/improve the code health at [https://landscape.io/github/rasbt/mlxtend](https://landscape.io/github/rasbt/mlxtend)

<hr>

# Tips for Contributors

## Getting Started - Creating a New Issue and Forking the Repository

- If you don't have a [GitHub](https://github.com) account, yet, please create one to contribute to this project.

- Please submit a ticket for your issue to discuss the fix or new feature before too much time and effort is spent for the implementation.

![](./img/contributing/new\_issue.png)

- Fork the `mlxtend` repository from the GitHub web interface.

![](./img/contributing/fork.png)

- Clone the `mlxtend` repository to your local machine by executing

```git clone https://github.com/<your\_username>/mlxtend.git```

## Syncing an Existing Fork

If you already forked mlxtend earlier, you can bring you "Fork" up to date

with the master branch as follows:

#### 1. Configuring a remote that points to the upstream repository on GitHub

List the current configured remote repository of your fork by executing

```bash

$ git remote -v

```

If you see something like

```bash

origin https://github.com/<your username>/mlxtend.git (fetch)

origin https://github.com/<your username>/mlxtend.git (push)

```

you need to specify a new remote \*upstream\* repository via

```bash

$ git remote add upstream https://github.com/rasbt/mlxtend.git

```

Now, verify the new upstream repository you've specified for your fork by executing

```bash

$ git remote -v

```

You should see following output if everything is configured correctly:

```bash

origin https://github.com/<your username>/mlxtend.git (fetch)

origin https://github.com/<your username>/mlxtend.git (push)

upstream https://github.com/rasbt/mlxtend.git (fetch)

upstream https://github.com/rasbt/mlxtend.git (push)

```

#### 2. Syncing your Fork

First, fetch the updates of the original project's master branch by executing:

```bash

$ git fetch upstream

```

You should see the following output

```bash

remote: Counting objects: xx, done.

remote: Compressing objects: 100% (xx/xx), done.

remote: Total xx (delta xx), reused xx (delta x)

Unpacking objects: 100% (xx/xx), done.

From https://github.com/rasbt/mlxtend

\* [new branch] master -> upstream/master

```

This means that the commits to the `rasbt/mlxtend` master branch are now

stored in the local branch `upstream/master`.

If you are not already on your local project's master branch, execute

```bash

$ git checkout master

```

Finally, merge the changes in upstream/master to your local master branch by

executing

```bash

$ git merge upstream/master

```

which will give you an output that looks similar to

```bash

Updating xxx...xxx

Fast-forward

SOME FILE1 | 12 +++++++

SOME FILE2 | 10 +++++++

2 files changed, 22 insertions(+),

```

## \*The Main Workflow - Making Changes in a New Topic Branch

Listed below are the 9 typical steps of a contribution.

#### 1. Discussing the Feature or Modification

Before you start coding, please discuss the new feature, bugfix, or other modification to the project

on the project's [issue tracker](https://github.com/rasbt/mlxtend/issues). Before you open a "new issue," please

do a quick search to see if a similar issue has been submitted already.

#### 2. Creating a new feature branch

Please avoid working directly on the master branch but create a new feature branch:

```bash

$ git branch <new\_feature>

```

Switch to the new feature branch by executing

```bash

$ git checkout <new\_feature>

```

#### 3. Developing the new feature / bug fix

Now it's time to modify existing code or to contribute new code to the project.

#### 4. Testing your code

Add the respective unit tests and check if they pass:

```bash

$ PYTHONPATH='.' pytest ./mlxtend ---with-coverage

```

#### 5. Documenting changes

Please add an entry to the `mlxtend/docs/sources/changelog.md` file.

If it is a new feature, it would also be nice if you could update the documentation in appropriate location in `mlxtend/sources`.

#### 6. Committing changes

When you are ready to commit the changes, please provide a meaningful `commit` message:

```bash

$ git add <modifies\_files> # or `git add .`

$ git commit -m '<meaningful commit message>'

```

#### 7. Optional: squashing commits

If you made multiple smaller commits, it would be nice if you could group them into a larger, summarizing commit. First, list your recent commit via

\*\*Note\*\*

\*\*Due to the improved GitHub UI, this is no longer necessary/encouraged.\*\*

```bash

$ git log

```

which will list the commits from newest to oldest in the following format by default:

```bash

commit 046e3af8a9127df8eac879454f029937c8a31c41

Author: rasbt <mail@sebastianraschka.com>

Date: Tue Nov 24 03:46:37 2015 -0500

fixed setup.py

commit c3c00f6ba0e8f48bbe1c9081b8ae3817e57ecc5c

Author: rasbt <mail@sebastianraschka.com>

Date: Tue Nov 24 03:04:39 2015 -0500

documented feature x

commit d87934fe8726c46f0b166d6290a3bf38915d6e75

Author: rasbt <mail@sebastianraschka.com>

Date: Tue Nov 24 02:44:45 2015 -0500

added support for feature x

```

Assuming that it would make sense to group these 3 commits into one, we can execute

```bash

$ git rebase -i HEAD~3

```

which will bring our default git editor with the following contents:

```bash

pick d87934f added support for feature x

pick c3c00f6 documented feature x

pick 046e3af fixed setup.py

```

Since `c3c00f6` and `046e3af` are related to the original commit of `feature x`, let's keep the `d87934f` and squash the 2 following commits into this initial one by changes the lines to

```

pick d87934f added support for feature x

squash c3c00f6 documented feature x

squash 046e3af fixed setup.py

```

Now, save the changes in your editor. Now, quitting the editor will apply the `rebase` changes, and the editor will open a second time, prompting you to enter a new commit message. In this case, we could enter `support for feature x` to summarize the contributions.

#### 8. Uploading changes

Push your changes to a topic branch to the git server by executing:

```bash

$ git push origin <feature\_branch>

```

#### 9. Submitting a `pull request`

Go to your GitHub repository online, select the new feature branch, and submit a new pull request:

![](./img/contributing/pull\_request.png)

<hr>

# Notes for Developers

## Building the documentation

The documentation is built via [MkDocs](http://www.mkdocs.org); to ensure that the documentation is rendered correctly, you can view the documentation locally by executing `mkdocs serve` from the `mlxtend/docs` directory.

For example,

```bash

~/github/mlxtend/docs$ mkdocs serve

```

### 1. Building the API documentation

To build the API documentation, navigate to `mlxtend/docs` and execute the `make\_api.py` file from this directory via

```python

~/github/mlxtend/docs$ python make\_api.py

```

This should place the API documentation into the correct directories into the two directories:

- `mlxtend/docs/sources/api\_modules`

- `mlxtend/docs/sources/api\_subpackes`

### 2. Editing the User Guide

The documents containing code examples for the "User Guide" are generated from IPython Notebook files. In order to convert a IPython notebook file to markdown after editing, please follow the following steps:

1. Modify or edit the existing notebook.

2. Execute all cells in the current notebook and make sure that no errors occur.

3. Convert the notebook to markdown using the `ipynb2markdown.py` converter

```python

~/github/mlxtend/docs$ python ipynb2markdown.py --ipynb\_path ./sources/user\_guide/subpackage/notebookname.ipynb

```

\*\*Note\*\*

If you are adding a new document, please also include it in the pages section in the `mlxtend/docs/mkdocs.yml` file.

### 3. Building static HTML files of the documentation

First, please check the documenation via localhost (http://127.0.0.1:8000/):

```bash

~/github/mlxtend/docs$ mkdocs serve

```

Next, build the static HTML files of the mlxtend documentation via

```bash

~/github/mlxtend/docs$ mkdocs build --clean

```

To deploy the documentation, execute

```bash

~/github/mlxtend/docs$ mkdocs gh-deploy --clean

```

### 4. Generate a PDF of the documentation

To generate a PDF version of the documentation, simply `cd` into the `mlxtend/docs` directory and execute:

```bash

python md2pdf.py

```

## Uploading a new version to PyPI

### 1. Creating a new testing environment

Assuming we are using `conda`, create a new python environment via

```bash

$ conda create -n 'mlxtend-testing' python=3 numpy scipy pandas

```

Next, activate the environment by executing

```bash

$ source activate mlxtend-testing

```

### 2. Installing the package from local files

Test the installation by executing

```bash

$ python setup.py install --record files.txt

```

the `--record files.txt` flag will create a `files.txt` file listing the locations where these files will be installed.

Try to import the package to see if it works, for example, by executing

```bash

$ python -c 'import mlxtend; print(mlxtend.\_\_file\_\_)'

```

If everything seems to be fine, remove the installation via

```bash

$ cat files.txt | xargs rm -rf ; rm files.txt

```

Next, test if `pip` is able to install the packages. First, navigate to a different directory, and from there, install the package:

```bash

$ pip install mlxtend

```

and uninstall it again

```bash

$ pip uninstall mlxtend

```

### 3. Deploying the package

Consider deploying the package to the PyPI test server first. The setup instructions can be found [here](https://wiki.python.org/moin/TestPyPI).

```bash

$ python setup.py sdist bdist\_wheel upload -r https://testpypi.python.org/pypi

```

Test if it can be installed from there by executing

```bash

$ pip install -i https://testpypi.python.org/pypi mlxtend

```

and uninstall it

```bash

$ pip uninstall mlxtend

```

After this dry-run succeeded, repeat this process using the "real" PyPI:

```bash

$ python setup.py sdist bdist\_wheel upload

```

### 4. Removing the virtual environment

Finally, to cleanup our local drive, remove the virtual testing environment via

```bash

$ conda remove --name 'mlxtend-testing' --all

```

### 5. Updating the conda-forge recipe

Once a new version of mlxtend has been uploaded to PyPI, update the conda-forge build recipe at https://github.com/conda-forge/mlxtend-feedstock by changing the version number in the `recipe/meta.yaml` file appropriately.