

Udacity: Machine Learning Engineer

Contents

1	Software Engineering Fundamentals	1
1.1	Software Engineering Practices	1
2	Machine Learning in Production	2
2.1	Introduction to Deployment	2
2.2	Machine Learning in Production	2

1 Software Engineering Fundamentals

1.1 Software Engineering Practices

Modular Code: putting functions into separate files to be imported into workspace.

Refactoring: restructuring your code to improve its internal structure, without changing its external functionality. This means cleaning and modularizing your program after it is working.

Optimization: we want to write efficient code, so this can be either fast execution or taking up less space in memory. We also want to *vectorize* our code for speed and amount of coding used.

```
1  for book in recent_books:
2      if book in coding_books:
3          recent_coding_books.append(book) # 16.63 sec
4
5  recent_coding_books = np.intersect1d(recent_books, coding_books) # 0.035 sec
6  recent_coding_books = set(recent_books).intersection(coding_books) # 0.0097 sec
7
8  for cost in gift_costs:
9      if cost < 25:
10         total_price += cost * 1.08 # 5.55 sec
11
12  total_price = np.sum(gift_costs[gift_costs < 25] * 1.08) # 0.084 sec
13
```

Git Branches: to switch to a branch in a repository you use *git checkout (branchname)*.

To create and switch to a new branch you use *git checkout -b (newbranch)*.

When in main branch, merge another branch by using *git merge -no-ff (branchname)*.

Previous Code: to see previous commits use *git log*.

Using the commit message number, open the code using a new branch *git checkout (commit#)*.

Unit Testing: *pytest* is a tool we can use to make sure our function is outputting correctly. We can create a test file starting with *test_* and we get a . if we pass and an F if we fail.

Test Driven Deployment: writing tests before you write the code that's being tested. Your test would fail at first, and you'll know you've finished implementing a task when this test passes.

Virtual Environment: when creating packages, you want to run a program in a virtual environment so the install does not mess up with original Python installation. Doing this means all packages installed in the virtual environment only exist within it, and will be removed after closing it.

To create the virtual environment we run *python -m venv (env_name)*

To move to the virtual environment we run *source (env_name)/bin/activate*

Creating Packages: in the main directory, we have a *setup.py* file and a package folder containing all necessary files. The *setup.py* contains information about the package using *setup* from *setuptools*. In the package folder, we have our Python modules and the *__init__.py* file. When in the main directory, use *pip install .* to install package into environment (on a personal machine use virtual environment above). To update a package after changes are made, run *pip install --upgrade*

Uploading Packages: within the package directory including all Python modules you also need a *README.md*, *license.txt*, and *setup.cfg* files. [Click here](#) to see how to set these up.

To create the distribution package, run *python setup.py sdist*

To upload to test.pypi run *twine upload --repository-url https://test.pypi.org/legacy/ dist/**

To install from test.pypi run *pip install --index-url https://test.pypi.org/simple/ (packagename)*

To upload to pypi run *twine upload dist/**

To install package from pypi you now run *pip install (packagename)*

2 Machine Learning in Production

2.1 Introduction to Deployment

Workflow: explore/process data, modeling, and deployment. [Click here](#) for AWS workflow.

Production Environment: the application that customers use to receive predictions from the deployed model.

Endpoint: the interface to the model which allows the application to send user data to the model and receives predictions back from the model about that data. Think of this as the python program being the application and a function calling the model being the endpoint.

Application: a web or software application that enables the application users to use the model to retrieve predictions.

Container: a standardized collection/bundle of software that is to be used for the specific purpose of running an application. This is used to create the computational environments for the model and application. A common container is *Docker*.

- **Layers** (bottom to top): infrastructure (data center), operating system, container engine (Docker), libraries/binaries, and application.
- **Script:** the instructions used to create a container ([dockerfiles](#)).

Deployment: versioning (indicate a models current version), monitoring (model continues to meet performance metrics), update (when model fails to meet performance use new data to update), routing (test model performance compared to other variants), and predictions (*on-demand* used for customers vs *batch* used in business).

2.2 Machine Learning in Production

NOTE: refer to “Boston Housing - XGBoost (Batch Transform) - High Level” for sample code.

Session ([doc](#)): a special object that allows you to do things like manage data in S3 and create and train any models. We use this throughout the notebook workspace in SageMaker.

Role: the IAM role created when you create a notebook, this defines how data that your notebook uses/creates will be stored. We will use this later for training.

S3 Bucket: When a training job is constructed using SageMaker, a container is executed which performs the training operation and accesses data in S3. This means that we need to upload the data we want to use for training to S3. When we perform a batch transform job, SageMaker expects the input data to be stored on S3. We upload data to S3 using the session object.

Estimators ([doc](#)): an object that specifies some details about how a model will be trained. It gives you the ability to create and deploy a model. It requires a container which can be obtained from the session object. We can also set hyperparameters on this estimator object.

Input: we specify where in S3 and the type of the data that we will feed into our estimator object.

Transformer ([doc](#)): used to create a transform job and evaluate a trained model. We specify the location of the test data and the format it is in. We can then use the *wait()* method to see the progress on testing and when we can resume coding.

Predictions: predictions are stored on S3, but we can download them into a specific directory.

```
1 !aws s3 cp --recursive $xgb_transformer.output_path $data_dir
2
```