

Необходимо создать открытый репозиторий (github, gitlab, etc.), в нём создать пакет на Python, который будет являться валидным с точки зрения пакетирования и решать какую-либо задачу машинного обучения (в качестве примера можно взять https://github.com/girafe-ai/ml-course/blob/2020_spring/week0_12_CNN/week12_cnn_semi_nar.ipynb).

Выбирайте задачу, которая у вас будет обучаться не более 5 минут! Иначе и вы, и мы будем долго ждать.

Под "решать" понимается две вещи:

1. есть файл `train.py`, в котором данные загружаются, модель тренируется, пишутся метрики и сохраняется на диск
2. есть файл `infer.py`, который считывает с диска модель из предыдущего пункта, загружает тестовый датасет, предсказывает моделью ответы для этих данных, записывает ответы на диск в `.csv` файл.

Альтернативно вместо `train.py` и `infer.py` можно иметь `commands.py` с командами `train` и `infer`

Какие инструменты необходимо использовать:

1. Poetry

Зависимости должны быть представлены в `pyproject.toml`, также не забудьте добавить `poetry.lock` файл.

2. Pre-commit

Необходимо использовать хуки `black`, `isort`, `flake8` (остальные тоже стоит применить, но это опционально). Также необходимо, чтобы запуск `pre-commit run -a` выдавал зелёный результат.

2. DVC

Для `dvc` в качестве бекенда проще и доступнее всего использовать гугл диск (проверьте, что папка доступна по ссылке на чтение всем, иначе мы не сможем проверить), можно использовать и любой другой бекенд, но тут возникает такой же вопрос с доступностью.

Скачивание данных с помощью `dvc` необходимо встроить в имеющиеся команды `train` и `infer`, для этого у `dvc` есть `python api` (на крайний случай можно вызывать CLI).

3. Hydra

Переведите основные гиперпараметры препроцессинга, обучения и постпроцессинга в `yaml` конфиги `hydra`. Сами конфиги лучше всего расположить в папке `configs` в корне репозитория.

4. Logging

Необходимо добавить логирование ваших основных метрик и функций потерь (всего не менее 3 графиков). Также в эксперимент записывать использованные гиперпараметры и версию кода (`git commit id`). Считайте, что сервер `mlflow` уже поднят на порту `128.0.1.1:8080`, его адрес можно добавить в поле конфига.

Как мы будем проверять ДЗ:

1. клонируем репозиторий
2. создаём новый чистый virtualenv
3. `poetry install` - нужна успешная установка (верная конфигурация)
4. `pre-commit install` - нужна успешная установка (верная конфигурация)
5. `pre-commit run -a` - не должно быть проваленных хуков (необходимые хуки это black, isort, flake8)
6. `python train.py` - ожидаем в результате сохранённую модель
7. `python infer.py` - ожидаем в результате файл с ответами

Технические подсказки:

1. Не забывайте про `if __name__ == '__main__':`
2. Пишите в вашу дефолтную ветку (master or main), мы не будем переключать, проверка скриптованная.

[Форма для сдачи](#)

Вы можете заполнить форму со ссылкой на репозиторий сейчас, а комитить в него вплоть до дедлайна.

Список типичных ошибок

Что нужно исправить:

- Заполнить файл README, в нём объяснить, какую задачу вы решаете (у каждого что-то своё, так что нам нужно это знать)
- Нельзя объявлять переменные (кроме констант) на верхнем уровне файла (не внутри функции или класса). Это можно делать только внутри классов, функций или под `if __name__ == '__main__':`.
- Под вызовом `if __name__ == '__main__':` вызывать ровно одну функцию (можно её назвать main или как-то ещё), а не писать всю логику непосредственно под if-ом
- ~~warnings.filterwarnings("ignore")~~ - это исчадие сатаны. Никогда не делайте этого в ~~продакшен~~ любых проектах. Это огромный задел на отстреливание себе ноги. Люди пишут предостережения для вас, но вы же умнее каких-то там авторов библиотек!
- Используйте реальные данные. Нельзя использовать сгенерированные данные.
- Импорты из вашего проекта делайте либо локальными (через точку, например [как тут](#)), либо глобальными (когда начинаете с названия вашего пакета, вам это пока скорее всего не нужно, проще первый вариант)
- Нельзя сохранять данные в гит!!!!!!!!!!!!!! То есть файлы .json, .csv, .h5 и проч. То же касается файлов натренированных моделей (.cbm, .pth, .xyz, etc). Я об этом говорил на лекциях, более того, сейчас вы даже знаете, как это делать правильно (через dvc). Нужно удалить все данные из гита.
- Некоторым с первого раза не понятно: ДВА ФАЙЛА - train.py и infer.py, не всё в одном
- Назвать питон пакет (aka папка с вашим кодом) по правилам питона (snake_case), а не как попало (e.g. MYopsTools)
- Используйте дефолтный .gitignore для Питона (не пустой), его дополняйте необходимыми вам путями. Дефолтный конфиг гуглится и даже предлагается вам гитхабом при создании репозитория.
- Запустить таки pre-commit run -a и убедиться, что он не красный

- Файлы с кодом называются в snake_case, не в CamelCase (e.g. Dataset.py)
- Репозиторий должен быть виден. Скрытые (приватные) репозитории оцениваются в 0 баллов.

Что делать желательно:

- Использовать fire вместо argparse (или полностью перейти на hydra)
- Гит репозитории как правило называются в log-case (то есть слова разделяются дефисами), а не в CamelCase
- Сделать одну входную точку commands.py, где вызывать соответствующие функции из файлов
- Пересесть с иглы процедурного программирования (когда вы объявляете только функции) на ООП (ака классы).
- Для выполнения предыдущего пункта сначала нужно осмелиться создать больше двух файлов с кодом в папке (например вынести туда функции и импортировать их из этого пошеренного файла)
- Не делайте однобуквенные переменные, как будто в школе сидим программируем, ей богу...
- Конфиги лучше хранить не в json, а в yaml (это всё равно понадобится для Гидры)
- Используйте pathlib вместо os.path - важа жизнь заиграет совершенно другими красками
- Папка кода проекта как правило называется так же как проект, можете посмотреть популярные open source репозитории (numpy, scipy, pytorch, etc). На сегодняшний день пакеты в Python не требуют называть папку с кодом src.
- Не стоит все ваши функции и классы класть исключительно в utils - получается какая-то свалка

P.S. если что-то непонятно или вы хотите сделать как-то иначе - пишите в чат, обсудим; данные правила не являются догмами, а скорее набором хороших техник, которые впрочем нужно адаптировать под задачу и ситуацию в целом