Python, Git, Intent classification

...

практический минимум

План лекции (3 в 1)

1. Python
2. Машинное обучение на примере классификации интентов
3. git

Часть 1. Python / jupyter

- Совсем вкратце про python
- PEP 8 / typing
- virtualenv / conda environments
- jupyter
- nbextensions / ipykernel
- Советы

Совсем вкратце про python

Назначения языка -- позволить быстро писать программы под широкий круг задач.

Это достигается за счет:

- Разнообразных библиотек
- Динамической типизации
- “все на интерфейсах”

Обучающих ресурсов много, упомяну один

https://checkio.org/ -- сайт с задачками по “олимпиадному” программированию

Часто используемые библиотеки

- numpy -- работа с многомерными массивами
  - https://pythonworld.ru/numpy/100-exercises.html
- scipy -- библиотека математических алгоритмов
- pandas -- работа с таблицами
  - книга “python и анализ данных” -- от создателя pandas
- scikit-learn -- много разнообразных алгоритмов машинного обучения, предобработки данных и тд.
- matplotlib, seaborn, plotly -- визуализация

PEP (Python Enhancement Proposals) / typing

- PEP 20 “Zen of python” -- общая философия языка
- PEP 8 -- style-guide
  - (местами от него отходят, например, длину строки сейчас часто ограничивают 120 символами, а не 80, как указано в PEP 8)
- PEP 257 -- соглашения по docstring

весь список: https://www.python.org/dev/peps/

Что еще можно улучшить?

Аннотации типов https://habr.com/ru/company/lamoda/blog/432656/

TDD (Test Driven Development)

Общая идея: сперва пишем проверки на поведение функции/класса, потом пишем код, проходящий эти проверки, потом делаем рефакторинг.

Статья с кучей подробностей: https://habr.com/ru/post/459620/

pytest

- Как писать в нем тесты https://habr.com/ru/post/448782/
- Минигайд по использованию в pycharm https://www.jetbrains.com/help/pycharm/pytest.html#create-pytest-test

Virtual environments

Что это?

Изолированные среды для разных проектов/задач.

Для чего это?

избежать конфликта разных версий библиотек, облегчить перенос проектов между разными машинами

Как использовать?

virtualenv, anaconda (рекомендую второй вариант)

Ссылки:

в anaconda https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html

virtualenv https://python-scripts.com/virtualenv

Jupyter -- популярный и удобный инструмент

jupyter notebook Last Checkpoint: a minute ago (autosaved)

File Edit View Insert Cell Kernel Help Python 3

Markdown Cell Toolbar: None

Simple spectral analysis

An illustration of the Discrete Fourier Transform using windowing, to reveal the frequency content of a sound signal.

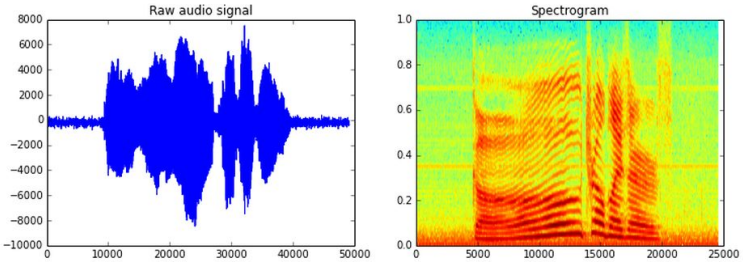
$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

We begin by loading a datafile using SciPy's audio file support:

In [1]:
```
from scipy.io import wavfile
rate, x = wavfile.read('test_mono.wav')
```

And we can easily view its spectral structure using matplotlib's builtin `specgram` routine:

In [2]:
```
%matplotlib inline
from matplotlib import pyplot as plt
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(x); ax1.set_title('Raw audio signal')
ax2.specgram(x); ax2.set_title('Spectrogram');
```



Jupyter -- полезные детали

- ipykernel -- для интеграции с виртуальными средами

В виртуальную среду устанавливается пакет ipykernel, затем из консоли (в этой виртуальной среде):

python -m ipykernel install --user --name=venv_name --display_name="Name of this kernel"

Полное руководство: https://ipython.readthedocs.io/en/stable/install/kernel_install.html

- nbextensions -- набор расширений.

Рекомендую оттуда Table of contents (2)

Статья с прочими полезностями вроде интерактивных графиков и многим другим:

https://habr.com/ru/company/wunderfund/blog/316826/

Советы

- Периодически перечитывайте PEPы
- Разработка через тестирование экономит ваше время.
- Используйте дистрибутив anaconda (и по возможности установщик пакетов conda)
- Старайтесь использовать tqdm везде, где ваш код работает дольше 10 секунд
- Начинать разработку удобно в jupyter, как только сможете вынести код в отдельные модули -- делайте это
- Используйте отдельные виртуальные среды под каждый проект
- Старайтесь делать ноутбуки короткими, следите за тем, чтобы они выполнялись от начала и до конца

Вопросы?

Часть 2. Машинное обучение

1. Задача классификации
2. Классификация интентов с помощью sklearn
3. Эмбеддинги
4. Классификация интентов с помощью эмбеддингов

Материалы

Желающим изучать машинное обучение могу порекомендовать эти материалы:

- https://mlcourse.ai/ -- курс от сообщества Open Data Science (кстати, вступайте: ods.ai). Последние запуски курса были только на английском языке, более старые -- на русском.
- Лекции Воронцова
- Червоненкис “Компьютерный анализ данных” -- для желающих лучше понять математику

Курсы по нейронным сетям:

- https://dlcourse.ai/ -- курс по основам нейронных сетей от Семена Козлова
- https://www.fast.ai/ -- сборник курсов от Jeremy Howard. К некоторым прочтениям курсов есть русские субтитры

Задача классификации

Дано:

Выборка примеров X.

Каждый пример $\boldsymbol{x} \in X$ описывается числовым вектором.

Существует некая зависимость $f: X \to \{0, \dots, n\}$, мы знаем $f(\boldsymbol{x})$ для всех $\boldsymbol{x} \in X$

Требуется:

Построить аппроксимацию зависимости f

Примеры:
Детекция аномалий в работе системы, детекция мошеннических операций или других злонамеренных действий. Фильтр спама. Кредитный скоринг, прогнозирование оттока клиентов. Медицинская диагностика.

Анализ тональности (эмоционального окраса) текста.
Выявление намерения пользователя

Классификация интентов (выявление намерений)

Демонстрация в jupyter-ноутбуке

Часть 3: git

1. Что это и зачем
2. Основные команды
3. commit message
4. Командная работа
5. Структура проекта

git: что это и зачем

Это популярная система контроля версий.

В случае, когда над проектом работает 1 человек:

- Можно гарантировать, что рабочая версия проекта всегда доступна
- Облегчает перенос проекта между разными компьютерами
- Легко переключаться между разными версиями проекта (например, для экспериментов)

В случае, когда над проектом работает команда:

- Меньше шансов, что один человек внесет правки, которые сломают версию проекта, над которой в данный момент работает другой человек
- Проще объединять результаты работы разных людей в единую версию

git: основные команды

- branch new_branch -- создает новую ветку new_branch
- checkout new_branch -- переключается на ветку new_branch
- pull -- стягивает изменения с origin (github или gitlab репозитория)
- add -- добавляет изменения к будущему коммиту
- commit -- делает коммит, то есть фиксирует произведенные (и добавленные) изменения
- stash -- откатывает сделанные изменения к последнему коммиту (позже их можно будет восстановить)
- push -- отправляет изменения в origin

Материалы:

https://try.github.io/ -- полезные ресурсы, в том числе и интерактивные

https://guides.github.com/ -- сборник мини-статей

git: commit message

commit message -- это текстовое пояснение к коммиту.

Вредные советы:

- писать бессмыслицу “lvnsdlfvw”
- делать коммиты как можно реже, копить много изменений
- писать о том, что вы сделали (“улучшил такой-то класс”)

Полезные советы:

https://chris.beams.io/posts/git-commit/

“Нормально, если на commit message потрачено больше времени, чем на код” -- совет от Senior Developer из Google

git: командная работа

Как это сделано в R&D-отделе нашей компании:

- В master-ветке -- гарантированно рабочая версия. Обновления не должны ничего ухудшить.
- Отдельная ветка development -- для слияния работы разных людей. Версия этой ветки -- кандидат для вливания в master.
- В одной ветке работает только один человек.
- На каждую задачу -- отдельная ветка.
- По завершению задачи заводится merge request в ветку development
- code review (просмотр и комментирование чужого кода) делают все желающие
- merge request принимается только после закрытия всех вопросов, поднятых в code review

Структура проекта

- data/ (каталог с данными, часто заносится в .gitignore)
- src/ (python-модули, основной каталог с кодом)
- notebooks/ (каталог с рабочими ноутбуками, заносится в .gitignore)
- reports/ (каталог с отчетами -- минималистичными ноутбуками)
- .gitignore (файл, указывающий git, какие каталоги и файлы следует игнорировать)
- README.md (общая документация по репозиторию)
- requirements.txt (файл, в котором перечислены зависимости, опционально)
- setup.py (файл, нужные для установки проекта, опционально)

Вопросы?

Спасибо за внимание!