Intro & Setup

slides by 陳麗光

Table of Contents

Course Overview

Syllabus

Week	Date	Topics
1	9/15	Course Introduction
2	9/22	Cross corpora analysis
3	9/29	Word representations
4	10/6	Website Parser
5	10/13	Collocations / Grammar Patterns
6	10/20	Streamlit, Final Project Announcement
7	10/27	Neural Network introduction
8	11/3	Sentence Proficiency Level Classification
9	11/10	Final Topic Proposal

Week	Date	Topics
10	11/17	Grammar Pattern Tagging
11	11/24	Grammatical Error Correction
12	12/1	Final Project Progress Check
13	12/8	Definition Embedding
14	12/15	Final Project Progress Check
15	12/22	TBD
16	12/29	Final Presentation 1
17	1/5	Final Presentation 2
18	1/12	Finals Week (No class)

Grading

- Weekly Assignments: 60 %
  - Turn in before or at TA hours, **10~12 a.m., Thursday** the following week

- Final Project: 40 %
  - Group of 2
  - Project specification announced at Oct/20
  - Nov/10 Proposal
  - Dec/1, Dec/15 Progress check
  - Presentation Dec/29, Jan/5

TAs

- 段凱文 kevintuan@nlplab.cc
- 李書卉 shlee@nlplab.cc
- 陳麗光 lkchen@nlplab.cc
- AC acmloria@gmail.com

Course platform: elearn

We will be using **elearn** as our course platform.

We will distribute **announcements** and **slides** on eeclass.

You will hand in this week's assignment (and some of the assignments from future weeks) via eeclass.

Setting up your OS

If you’re using a Windows computer…

- You might want to consider installing WSL (Windows subsystem for Linux) or Cygwin, and running your programs under it
  - WSL [English tutorial] , [中文教學] (recommended)
  - Cygwin [English tutorial] [中文教學] (you may try this, but TAs may not be able to help you if you encounter problems)
- Why?
  - Many convenient Unix / Linux commands and programs (e.g. tmux, nohup) require complicated equivalents in Windows
  - If you have good alternatives for a Windows system, please let us know!

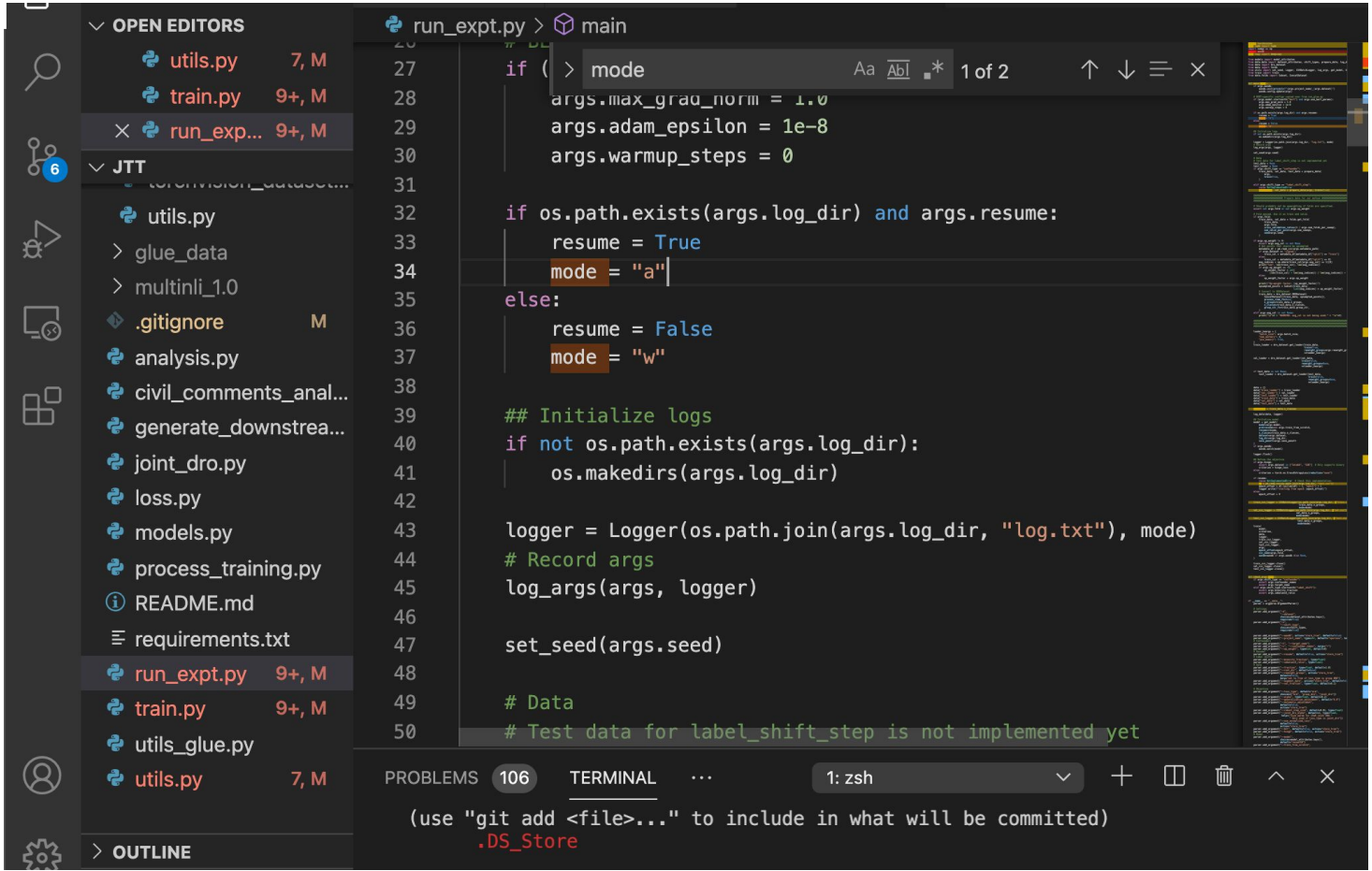
(If you’re using MacOS, your built-in terminal already accepts Unix commands (since MacOS is a Unix system))

Jupyter

Jupyter Notebook / Jupyter lab: an introduction

- An *interactive* graphic user interface
- Good for tutorials and demos

Your usual IDE



Jupyter notebook

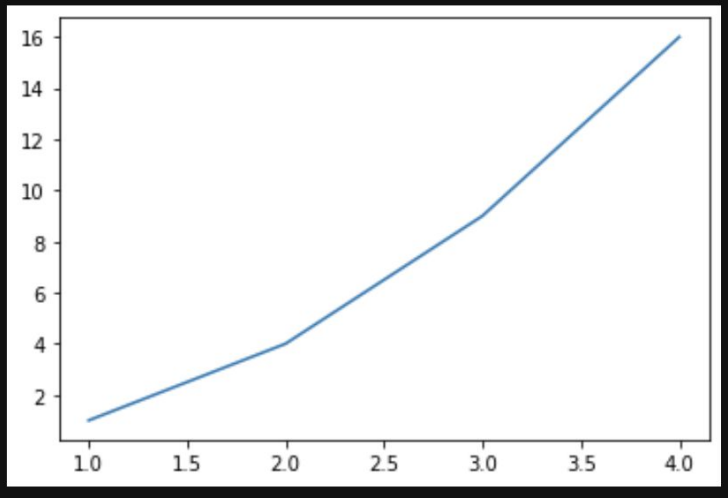
Here we import matplotlib

matplotlib is a Python library that allows you to make plots like Matlab -- and more!

[2]:
```
from matplotlib import pyplot as plt
```

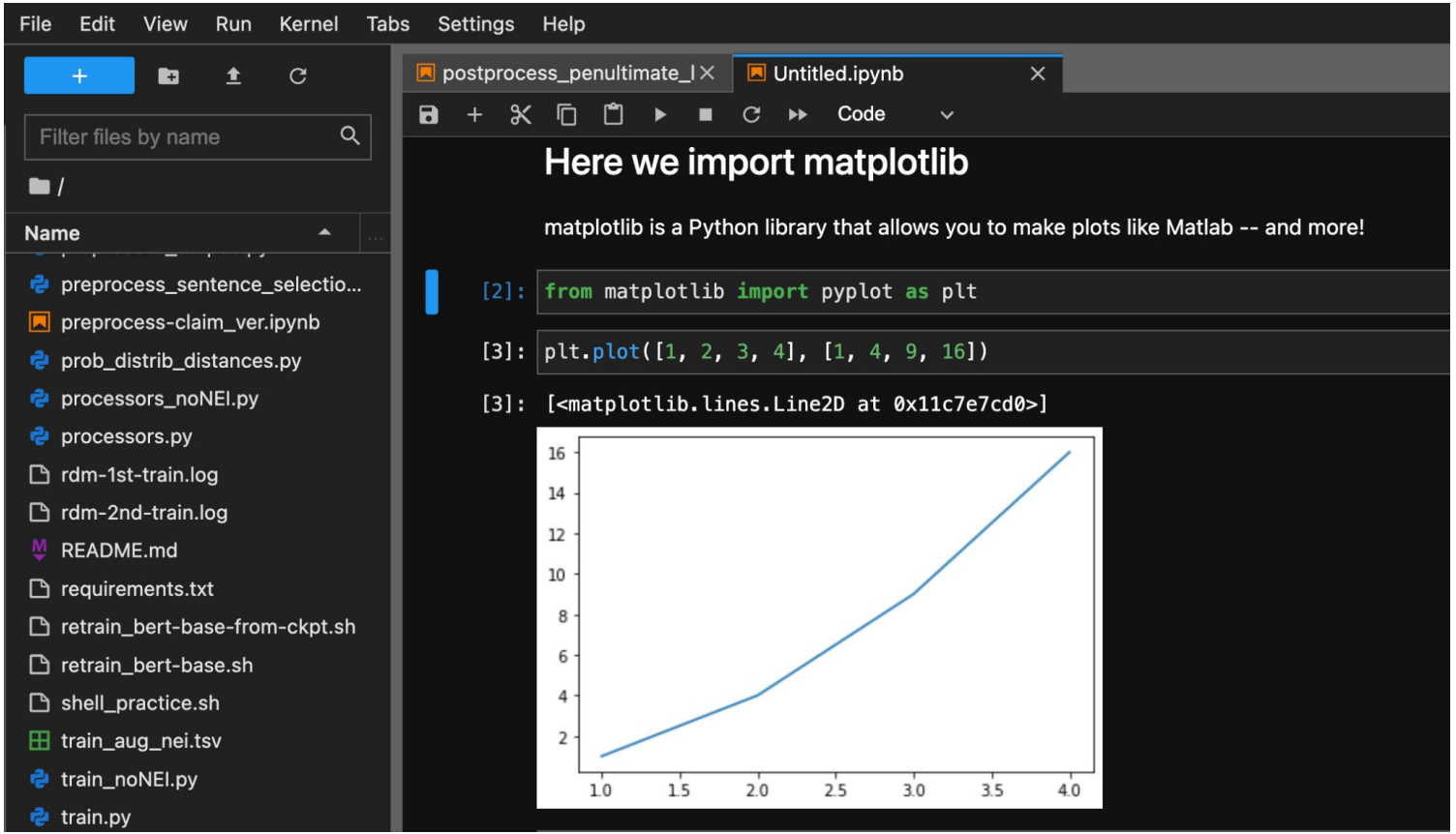
[3]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
```

[3]:
```
[<matplotlib.lines.Line2D at 0x11c7e7cd0>]
```



Jupyter Lab

Have your files and tabs all in one place



Before you install Jupyter…

Install an environment for it!

🤦 Why use an environment?

- There are different versions of Python
  - Python 2, Python 3.5, Python 3.6, Python 3.9 …
- Packages have different versions
  - Matplotlib 0.x, 1.1, 1.5, 2.2, 2.7, 3.5, …
- Packages use other packages
  - Matplotlib: Python >=3.6, NumPy >=1.11, cycler >=0.10.0…

Dependencies

Matplotlib requires the following dependencies:

- Python (>= 3.6)
- FreeType (>= 2.3)
- libpng (>= 1.2)
- NumPy (>= 1.11)
- setuptools
- cycler (>= 0.10.0)
- dateutil (>= 2.1)
- kiwisolver (>= 1.0.0)
- pyparsing

Setting up an environment

Different kinds of environment managers for Python

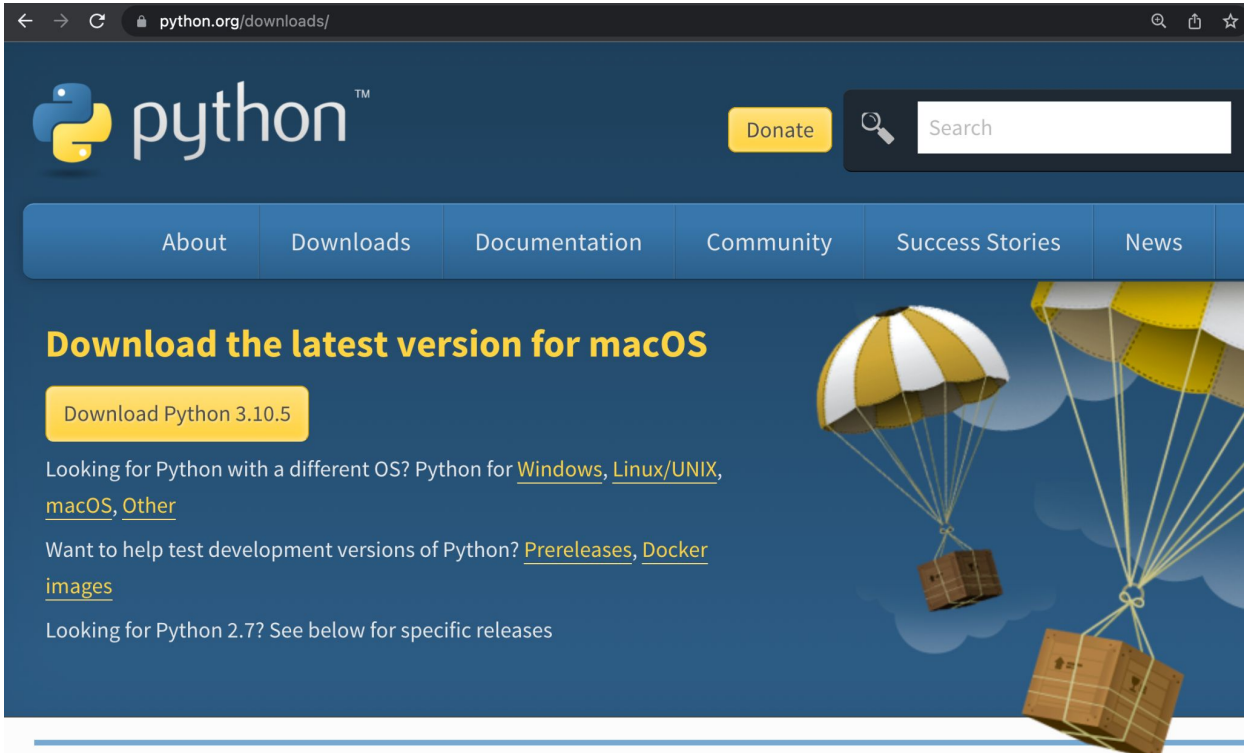
- Miniconda ⇒ we recommend this
- venv ⇒ we also recommend this
- virtualenv
- Poetry
- …

Download & Install Python if you haven't already

https://python.org/downloads/

For WSL: you may install it via apt

```
sudo apt update && sudo apt upgrade
sudo apt upgrade python3
```



Check your Python version

- Open your terminal
  - MacOS: command+space ⇒ search and open **Terminal.app**
  - Windows: search “**cmd**” or “**command prompt**” ⇒ open “**Command Prompt**”
  - Linux subsystem for Windows also has its own terminal entry point

- Check your Python version
  - Key in `python --version`, press enter/return
  - It should show Python 3.x.y , (e.g. 3.7.4, 3.8.2, 3.9.0, …)
  - We recommend having **Python >=3.7**

Create and manage environments: **Miniconda**

- **Miniconda**: https://docs.conda.io/en/latest/miniconda.html (recommended)

  Note: There's another distribution called "**Anaconda**" developed by the same company:

  - Automatically downloads loads of other packages for you
  - Requires > 3 GB disk space whereas Miniconda requires only ~400 MB
  - For more information: https://docs.conda.io/projects/conda/en/latest/user-guide/install/download.html#anaconda-or-miniconda

Download **Miniconda**: Windows

Install the appropriate version according to your system and Python version

Windows installers

Windows

Python version	Name	Size	SHA256 hash
Python 3.9	Miniconda3 Windows 64-bit	71.2 MiB	1acbc2e8277ddd54a5f724896c7edee112d068529588d944702966c867e7e9cc
Python 3.8	Miniconda3 Windows 64-bit	70.6 MiB	94f24e52e316fa935ccf94b0c504ceca8e6abc6190c68378e18550c95bb7cee1
Python 3.7	Miniconda3 Windows 64-bit	69.0 MiB	b221ccdb2bbc5a8209a292f858ae05fd87f882f79be75b37d26faa881523c057
Python 3.9	Miniconda3 Windows 32-bit	67.8 MiB	4fb64e6c9c28b88beab16994bfba4829110ea3145baa60bda5344174ab65d462
Python 3.8	Miniconda3 Windows 32-bit	66.8 MiB	60cc5874b3cce9d80a38fb2b28df96d880e8e95d1b5848b15c20f1181e2807db
Python 3.7	Miniconda3 Windows 32-bit	65.5 MiB	a6af674b984a333b53aaf99043f6af4f50b0bb2ab78e0b732aa60c47bbfb0704

Download **Miniconda**: MacOS

Install the appropriate version according to your processor and Python version

macOS installers

macOS

Python version	Name	Size	SHA256 hash
Python 3.9	Miniconda3 macOS Intel x86 64-bit bash	56.0 MiB	007bae6f18dc7b6f2ca6209b5a0c9bd2f283154152f82becf787aac709a
	Miniconda3 macOS Intel x86 64-bit pkg	62.7 MiB	cb56184637711685b08f6eba9532cef6985ed7007b38e789613d5dd3f94
	Miniconda3 macOS Apple M1 ARM 64-bit bash	52.2 MiB	4bd112168cc33f8a4a60d3ef7e72b52a85972d588cd065be803eb21d73b
	Miniconda3 macOS Apple M1 ARM 64-bit pkg	63.5 MiB	0cb5165ca751e827d91a4ae6823bfda24d22c398a0b3b01213e57377a2c
Python 3.8	Miniconda3 macOS Intel x86 64-bit bash	56.4 MiB	f930f5b1c85e509ebbf9f28e13c697a082581f21472dc5360c41905d108
	Miniconda3 macOS Intel x86 64-bit pkg	63.1 MiB	62eda1322b971d43409e5dde8dc0fd7bfe799d18a49fb2d8d6ad1f68334
	Miniconda3 macOS Apple M1 ARM 64-bit bash	52.5 MiB	13b992328ef088a49a685ae84461f132f8719bf0cabc43792fc9009b042
	Miniconda3 macOS Apple M1 ARM 64-bit pkg	63.8 MiB	e92fd40710f7123d9e1b2d44f71e7b2101e3397049b87807ccf612c964b
Python 3.7	Miniconda3 macOS Intel x86 64-bit bash	66.0 MiB	323179e4873e291f07db041f3d968da2ffc102dcf709915b48a253914d9
	Miniconda3 macOS Intel x86 64-bit pkg	72.7 MiB	9278875a235ef625d581c63b46129b27373c3cf5516d36250a1a3640978

Download **Miniconda**: WSL / Cygwin

Linux installers

Linux

Python version	Name	Size	SHA256 hash
Python 3.9	Miniconda3 Linux 64-bit	73.1 MiB	78f39f9bae971ec1ae7969f0516017f2413f17796670f7040725dd83fcff5689
	Miniconda3 Linux-aarch64 64-bit	75.3 MiB	5f4f865812101fdc747cea5b820806f678bb50fe0a61f19dc8aa369c52c4e513
	Miniconda3 Linux-ppc64le 64-bit	74.3 MiB	1fe3305d0ccc9e55b336b051ae12d82f33af408af4b560625674fa7ad915102b
	Miniconda3 Linux-s390x 64-bit	69.2 MiB	ff6fdad3068ab5b15939c6f422ac329fa005d56ee0876c985e22e622d930e424
Python 3.8	Miniconda3 Linux 64-bit	72.6 MiB	3190da6626f86eee8abf1b2fd7a5af492994eb2667357ee4243975cdbb175d7a
	Miniconda3 Linux-aarch64 64-bit	64.4 MiB	0c20f121dc4c8010032d64f8e9b27d79e52d28355eb8d7972eafc90652387777
	Miniconda3 Linux-ppc64le 64-bit	65.9 MiB	4be4086710845d10a8911856e9aea706c1464051a24c19aabf7f6e1a1aedf454
	Miniconda3 Linux-s390x 64-bit	68.7 MiB	3125961430c77eae81556fa59fe25dca9e5808f76c05f87092d6f2d57f85e933
Python 3.7	Miniconda3 Linux 64-bit	100.1 MiB	4dc4214839c60b2f5eb3efbdee1ef5d9b45e74f2c09fcae6c8934a13f36ffc3e
	Miniconda3 Linux-aarch64 64-bit	101.7 MiB	47affd9577889f80197aadbdf1198b04a41528421aaf0ec1f28b04a50b9f3ab8
	Miniconda3 Linux-ppc64le 64-bit	101.4 MiB	c99b66a726a5116f7c825f9535de45fcac9e4e8ae825428abfb190f7748a5fd0

Create a Python environment: **Miniconda**

1. Create a folder for this course / assignment
    - Try to use only English characters and numbers without white space for the folder name
2. Go to the folder you just created in your terminal
3. **Create** the environment: in the folder, type and enter in your terminal

```
conda create --name <name for your environment> <python=3.X>
```

Optional: if you want to be really specific about your python version

4. **Activate** your environment: in your terminal, type and enter

```
conda activate <name for your environment>
```

Manage environments: **Miniconda**

After you've created and activated your environment…

- To **install** a package in the environment

  `pip install <package name>`

  To be more specific about versions :

  `pip install <package name==[version]>`

Note:

On the cheatsheet, it tells you to to use `conda install …`

While that also works, the versions you need might not always be available.

Instead, use `pip` for a wider range of versions to choose from.

- **Check** that you've installed the package in the environment:

  In the environment, type and enter `conda list`. It should show the packages you just installed

- To **deactivate** the environment (so you can enter another environment / install some package elsewhere / etc.):
  `conda deactivate`

Manage environments: **Miniconda**

For more conda commands, see the cheatsheet:

https://docs.conda.io/projects/conda/en/latest/_downloads/843d9e0198f2a193a3484886fa28163c/conda-cheatsheet.pdf

Alternative environment management tool: **venv**

1. Create a folder for you project
   - Try to use only English characters and numbers without white space for the folder name
2. Go to the folder you just created in your terminal
3. Type and enter in your terminal

   ```
   python -m venv <name for your environment>
   ```

4. To **activate** the environment: type in your terminal
   - MacOS: `source <name of your environment>/bin/activate`
   - Windows: see this tutorial
   - To check you've successfully created and activated an environment:

     ```
     pip --version
     ```

     It should show that pip you're now using is under the environment you've created:
     `<path to your env>/lib/python3.8/site-packages/pip`

Managing a **venv** environment

- To install a package:

  ```
  pip install <package name(==version)>
  ```

- To check you've installed the package in the environment:

  ```
  pip list
  ```

  It should only list the packages you installed.
  If other packages show up, please contact TAs for help.

- To deactivate the environment (so you can install packages in another environment etc.)

  ```
  deactivate
  ```

Jupyter notebook / lab

Now that you've created an environment…

we can finally install Jupyter!

- In your activated environment in your terminal, type

  `pip install notebook` for Jupyter Notebook

  `pip install jupyterlab` for Jupyter Lab

- To launch the notebook, type

  `jupyter notebook`

  `jupyter-lab`

Sometimes our terminal doesn't recognize the command: `command not found jupyter-...`

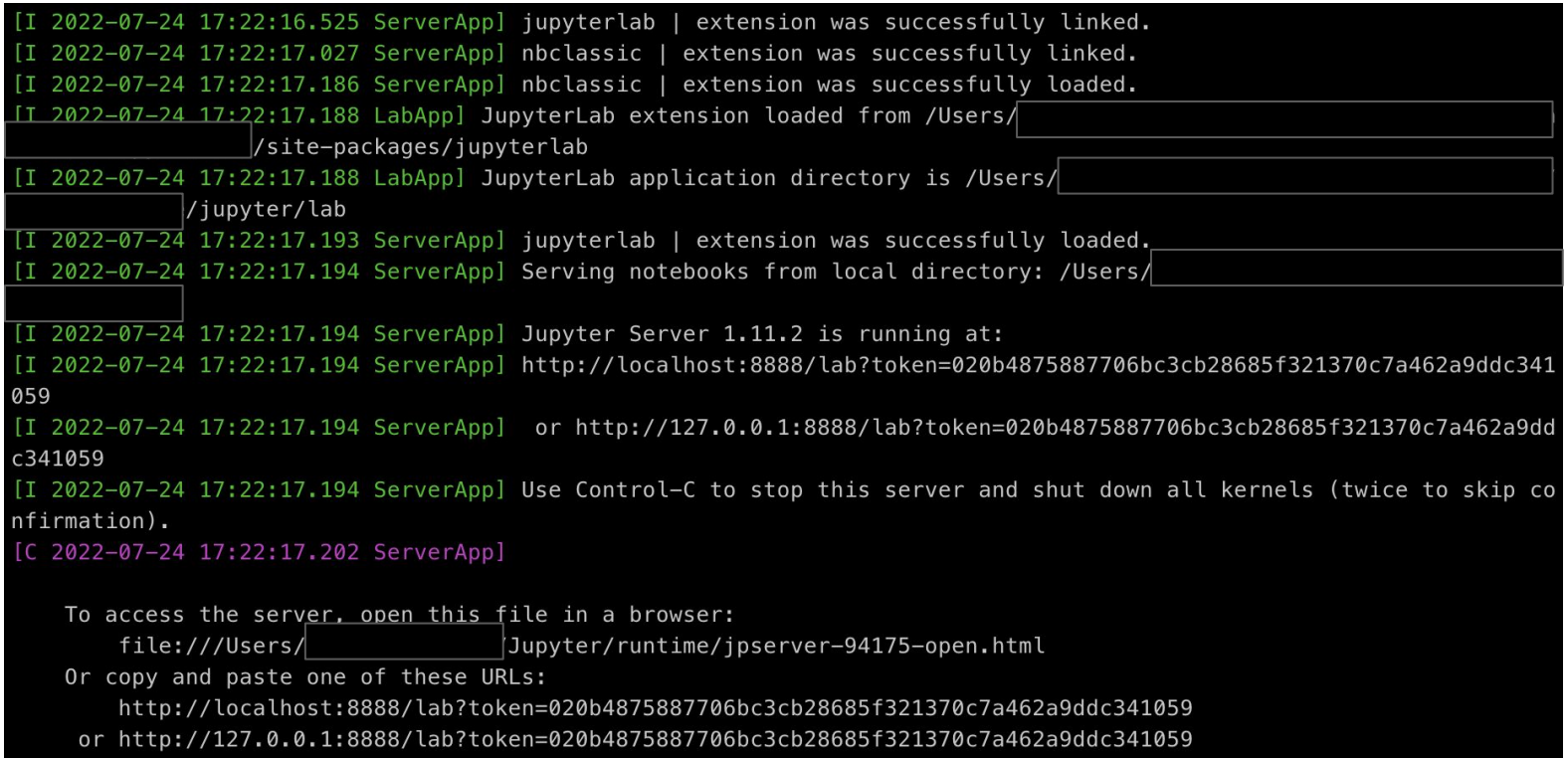
In that case, try:

```
python -m jupyter notebook
python -m jupyter-lab
```

Reference: https://jupyter.org/install

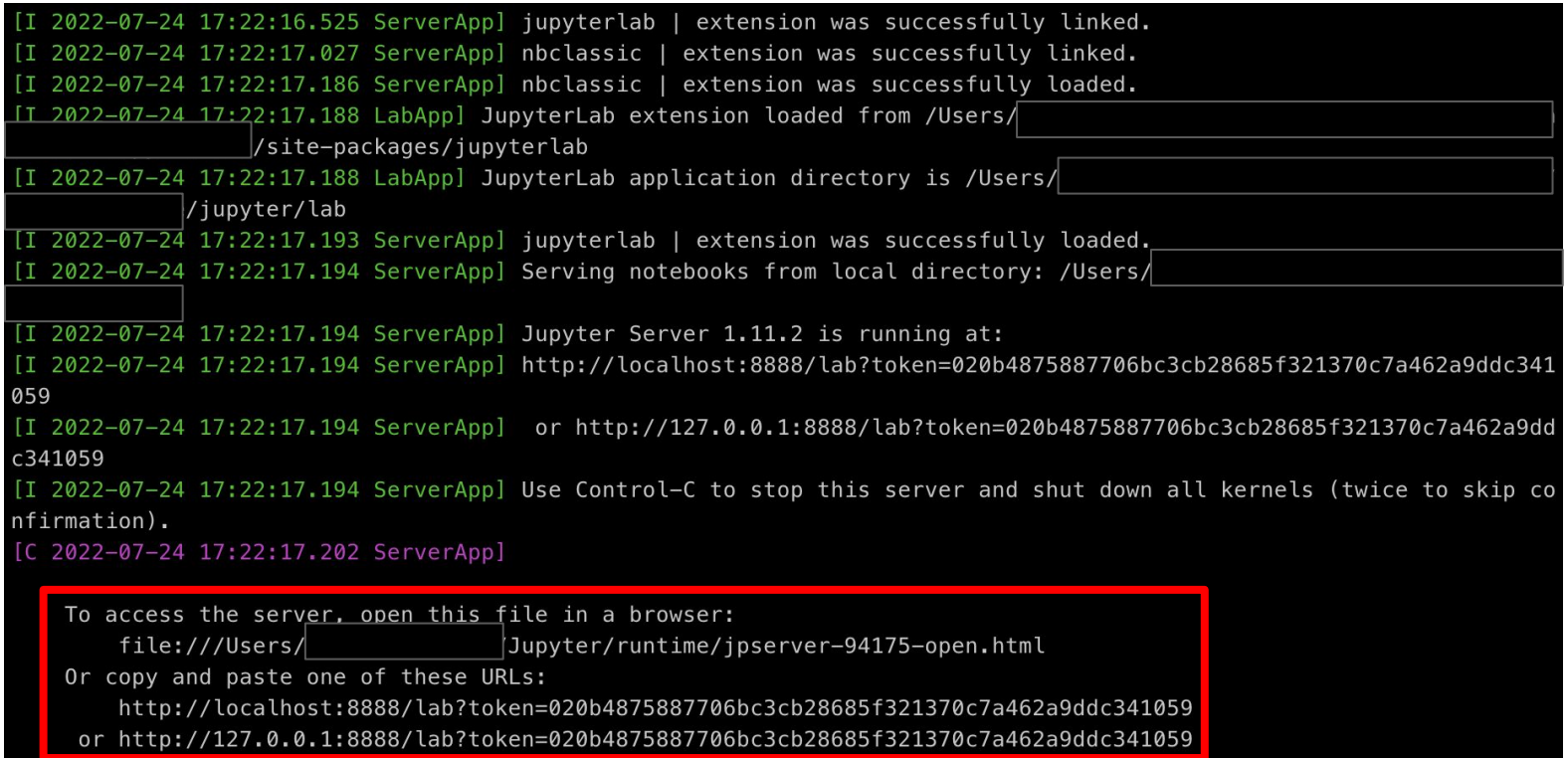
Using Jupyter notebook/lab

Once you launched your notebook, you should see something like this in your terminal:



Using Jupyter notebook/lab

If a window doesn't open in your browser, copy either of these links to open the notebook in your preferred browser.



To close Jupyter notebook/lab

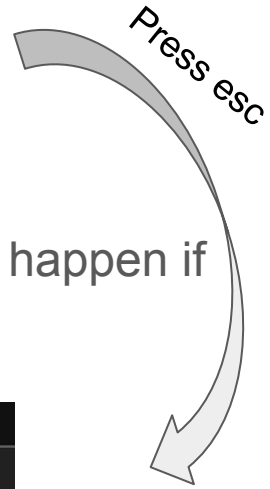
In your terminal where you opened your notebook, press **control/ctrl + C**.

- The program will ask you whether you are sure to close your notebooks etc. Follow the prompts to close it.

Some jupyter notebook commands

Your cursor is in the cell ⇒ you can safely edit your code

```
[2]: from matplotlib import pyplot as plt
```

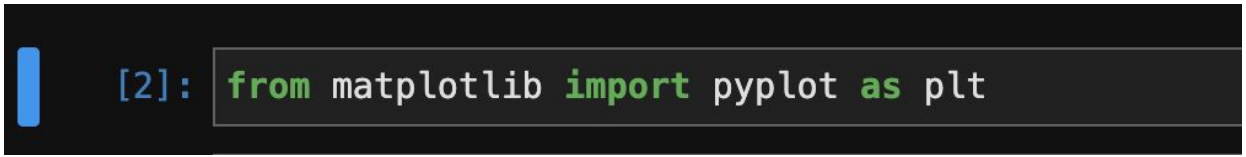


Your cursor is not in the cell but the cell is selected ⇒ ✨things✨ will happen if you accidentally pressed any of the magic buttons

```
[2]: from matplotlib import pyplot as plt
```

Some jupyter notebook commands

Your cursor is not in the cell but the cell is selected ⇒ ✨things✨ will happen if you accidentally pressed some magic buttons



- M ⇒ your cell will go from whatever mode to **Markdown** mode
- Y ⇒ changes cell from whatever mode back to **coding** mode
- A/B ⇒ adds an empty cell above/below current cell
- D ⇒ **deletes** current cell 😱
- Z ⇒ restores previous action 😌
- … and there are tons more of these commands out there. Google “jupyter notebook shortcuts” for more!

Checking that our Jupyter notebook/lab is using the right environment…

1. Open a notebook. In the cell, type

```
import sys
sys.path
```

2. Run the cell
3. The output should include

```
<your username>/<your project file>/<your environment name>/lib/python<version, e.g. 3.8>/site-packages
```

If not, ask the TAs for help!

tmux

tmux

What does **tmux** do?

- Opens several terminals on one screen
  - Allows you to multitask or monitor your training process
  - **iTerm** (MacOS) or **mobaXterm** (Windows) can also do this
- ❗ Keeps your “session” running when you close the terminal
  - Useful when you’re training something a server, which would usually take a prolonged amount of time
  - `nohup` also does the this (more info: https://en.wikipedia.org/wiki/Nohup)
- **tmux** and `nohup` are Unix-like systems only
  - If you want to use tmux on a Windows system, you’ll have to install WSL ~~or Cygwin ([English tutorial] [中文教學])~~ first.

Using tmux

- Start a session:
  `tmux [new -s mysession]`
- Detach from a session (i.e. keep the code running in the background):
  press keys **control (ctrl)** **b** **]**
- Attach to a previous session:
  `tmux attach -t mysession/<session number>`
- Kill a session (i.e. terminate everything open in the terminal)
  press keys **control (ctrl)** **x** ⇒ **y** to confirm

More: https://tmuxcheatsheet.com/

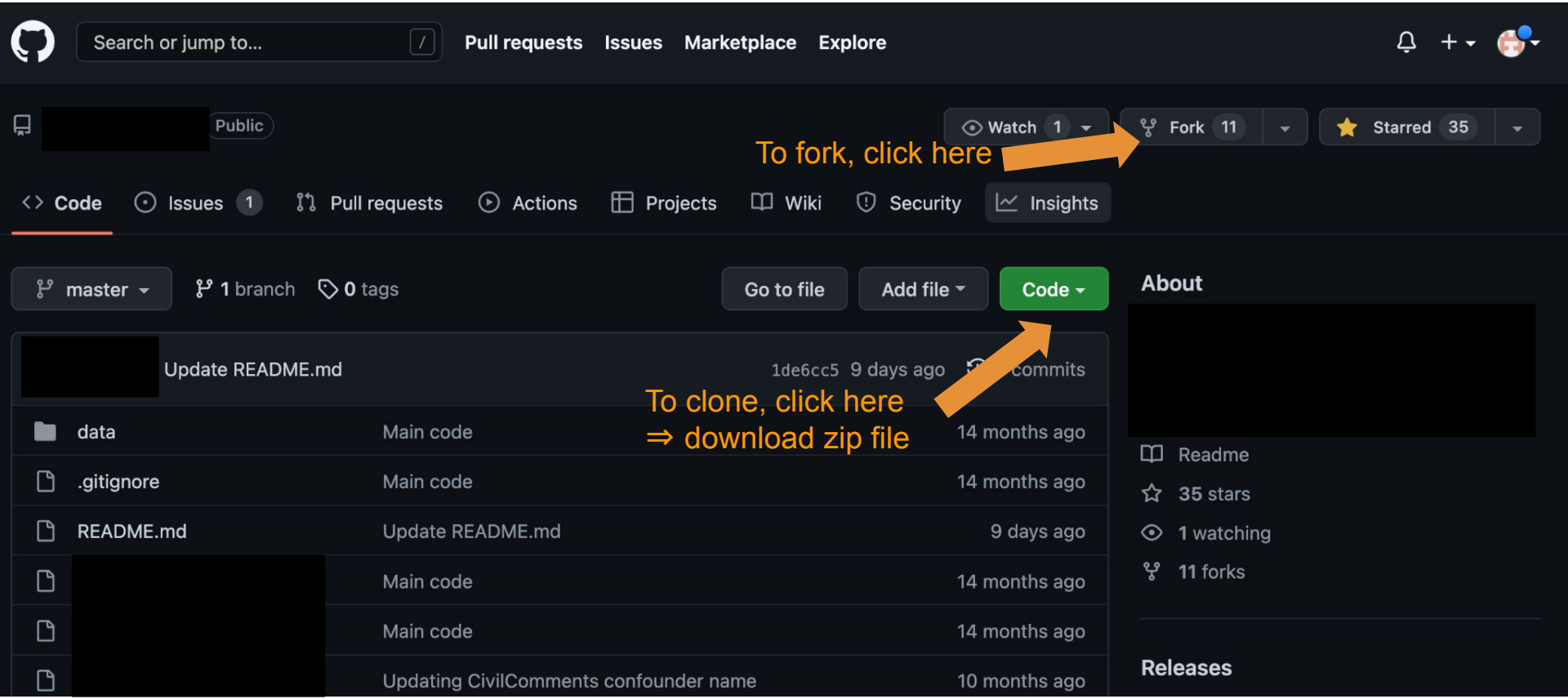
GitHub

What is GitHub? What does it do?

- Version control
- **Code sharing** – we will be distributing your assignments via GitHub in some weeks

GitHub basics

- Register for an account if you haven’t
- Glossary
  - **Repository**: “repo” for short. a place where users place their code
  - **Fork**ing: making a copy of someone else’s repository to your account. Changes made on your copy won’t affect the original.
  - **Cloning**: also making a copy of someone else’s repo, but you download it to your computer instead.
- You will be forking *or* cloning the assignment repo to obtain the starter code.


Search or jump to...
Pull requests
Issues
Marketplace
Explore
Public
Watch 1
Fork 11
Starred 35
To fork, click here
Code
Issues 1
Pull requests
Actions
Projects
Wiki
Security
Insights
master
1 branch
0 tags
Go to file
Add file
Code
About
Update README.md
1de6cc5 9 days ago
commits
To clone, click here
⇒ download zip file
data
Main code
14 months ago
.gitignore
Main code
14 months ago
README.md
Update README.md
9 days ago
Main code
14 months ago
Main code
14 months ago
Updating CivilComments confounder name
10 months ago
Readme
35 stars
1 watching
11 forks
Releases

Assignment 1:
Peter Norvig's **Spellchecker**

How to build your spellchecker

- Based on https://norvig.com/spell-correct.html (2007) (15 years ago..!)
- Probability + some clever algorithms + data

First, some prerequisites…

- Be sure you are familiar with Python string operations (tutorial 1, tutorial 2)
  (or just google “python string operations tutorial”)
- Familiarize yourselves with basic Python data structures (list, dictionary, string, class, …) and syntax
- Good to know
  - Python list comprehension
  - Nested for loops
  - Regular expression

First, some prerequisites…

- Bayes theorem

$$P(B|A) = \frac{P(A|B) \cdot P(B)}{P(A)}$$ “Probability of B given A”

⇒ We want the most likely correction candidate *c* , given (mistyped) word *w*

Applying Bayes' theorem to spell-checking

- Bayes' theorem

$$P(B|A) = \frac{P(A|B) \cdot P(B)}{P(A)}$$ "Probability of B given A"

⇒ We want the most likely correction candidate *c* , given (mistyped) word *w*

$$P(c|w) = \frac{P(w|c) \cdot P(c)}{P(w)}$$

Applying Bayes' theorem to spell-checking

- Bayes' theorem

$$P(B|A) = \frac{P(A|B) \cdot P(B)}{P(A)}$$ "Probability of B given A"

⇒ We want the most likely correction candidate *c*, given (mistyped) word *w*

$$\underset{c \in candidates}{\operatorname{argmax}} P(c|w)$$

$$= \underset{c \in candicates}{\operatorname{argmax}} \frac{P(w|c) \cdot P(c)}{P(w)}$$

Applying Bayes' theorem to spell-checking

- Bayes' theorem

$$P(B|A) = \frac{P(A|B) \cdot P(B)}{P(A)}$$ "Probability of B given A"

⇒ We want the most likely correction candidate *c* , given (mistyped) word *w*

$$\underset{c \in candidates}{\operatorname{argmax}} P(c|w) = \underset{c \in candicates}{\operatorname{argmax}} \frac{P(w|c) \cdot P(c)}{\cancel{P(w)}}$$

P(w) is the same for every c
⇒ we may ignore it

4 things to implement

We want the most likely correction candidate *c* , given (mistyped) word *w*

$$\underset{c \in candidates}{\operatorname{argmax}} P(c|w) = \underset{c \in candicates}{\operatorname{argmax}} \frac{P(w|c) \cdot P(c)}{P(w)}$$

P(w) is the same for every c
⇒ we may ignore it

1. argmax: Python's `max(candidates, key=func)`

`func` decides the order in which we want to rank the candidates

4 things to implement

We want the most likely correction candidate *c* , given (mistyped) word *w*

$$\underset{c \in candidates}{\operatorname{argmax}} P(c|w) = \underset{c \in candicates}{\operatorname{argmax}} \frac{P(w|c) \cdot P(c)}{\cancel{P(w)}}$$

P(w) is the same for every c
⇒ we may ignore it

1. argmax: Python's `max(candidates, key=func)`
2. c ∈ candidates : which candidates to choose from?

4 things to implement

We want the most likely correction candidate *c* , given (mistyped) word *w*

$$\underset{c \in candidates}{\operatorname{argmax}} P(c|w) = \underset{c \in candicates}{\operatorname{argmax}} \frac{P(w|c) \cdot P(c)}{P(w)}$$

P(w) is the same for every c
⇒ we may ignore it

1. argmax: Python's `max(candidates, key=func)`
2. c ∈ candidates : which candidates to choose from?

For example:

- Mistyped word = "*thew*"
- All possible candidates? thaw, them, the, …

4 things to implement – $c \in candidates$

- Define: all possible edits as candidates
- Possible edit operations
  - Delete: thew ⇒ the
  - Insert: thew ⇒ threw
  - Replace: thew ⇒ thaw
  - Transpose (a fancy way of saying “swap”): thew ⇒ thwe
- Not all edited combinations are words…!
  - Build our “known” vocabulary
- How many edits do you need?
  - 1? 2?
  - Consider the number of letter combinations generated!

4 things to implement

We want the most likely correction candidate *c* , given (mistyped) word *w*

$$\underset{c \in candidates}{\operatorname{argmax}} P(c|w) = \underset{c \in candicates}{\operatorname{argmax}} \frac{P(w|c) \cdot P(c)}{\cancel{P(w)}}$$

P(w) is the same for every c
⇒ we may ignore it

1. argmax: Python's `max(candidates, key=func)`
2. c ∈ candidates : which candidates to choose from?
3. P(c) : probability of of each candidate?

4 things to implement – $P(c)$ (probability of each candidate c)

- Some words appear more often than others.
- Remember our “known” words dictionary?
  - It might look something like this:
    {“the”: 2594732, “you”: 8763, “I”: 3423, …, “knoll”: 1}
  - The simplest way of calculating P(c):

$$P(c) = \frac{\#c}{\sum_c \#c}$$

4 things to implement

We want the most likely correction candidate *c* , given (mistyped) word *w*

$$\underset{c \in candidates}{\operatorname{argmax}} P(c|w) = \underset{c \in candicates}{\operatorname{argmax}} \frac{P(w|c) \cdot P(c)}{\cancel{P(w)}}$$

P(w) is the same for every c
⇒ we may ignore it

1. argmax: Python's `max(candidates, key=func)`
2. c ∈ candidates : which candidates to choose from?
3. P(c) : probability of of each candidate?
4. P(w|c) : probability of *w* given *c*

4 things to implement – $P(w|c)$ (probability of *w* given *c*)

- The probability that *w* would be typed in a text when the author meant *c*.
  - For example, P(teh|the) is relatively high, but P(theeexyz|the) would be very low
- What do you do when you don't have the data for what you mistyped before?
  ⇒ Make the data!
  - ★ $\text{Prob}(\text{known word of edit distance 0 to } w)$
    $> \text{Prob}(\text{known word of edit distance 1 to } w) > \ldots.$
  - ★ Generate words of edit distance n from step (2)
    - Again, keep the compute in mind!

      `get_candidates(word=w, dist=1):` $O(n)$

      `get_candidates((get_candidates(w,1)), dist=1):` $O(n^2)$

      …

Let's recap

We want the most likely correction candidate *c* , given (mistyped) word *w*

$$\underset{c \in candidates}{\operatorname{argmax}} P(c|w) = \underset{c \in candicates}{\operatorname{argmax}} \frac{P(w|c) \cdot P(c)}{\cancel{P(w)}}$$

P(w) is the same for every c
⇒ we may ignore it

1. argmax
   ⇒ Python's **`max(candidates, key=func)`**
2. c ∈ candidates : which candidates to choose from?
   ⇒ Generate candidates with edit operations
3. P(c) : probability of of each candidate?
   ⇒ Calculate probabilities from word-count dictionary
4. P(w|c) : probability of *w* given *c*
   ⇒ Probabilities of generated candidates from 2.

Discussion

We want the most likely correction candidate *c* , given (mistyped) word *w*

$$\underset{c \in candidates}{\operatorname{argmax}} P(c|w) = \underset{c \in candicates}{\operatorname{argmax}} \frac{P(w|c) \cdot P(c)}{\cancel{P(w)}}$$

P(w) is the same for every c
⇒ we may ignore it

1. argmax
   ⇒ Python's `max(candidates, key=func)`
2. c ∈ candidates : which candidates to choose from?
   ⇒ Generate candidates with edit operations
3. P(c) : probability of of each candidate?
   ⇒ Calculate probabilities from word-count dictionary
4. P(w|c) : probability of *w* given *c*
   ⇒ Probabilities of generated candidates from 2.

Any other ways to generate candidates and their probabilities?

Discussion

We want the most likely correction candidate *c* , given (mistyped) word *w*

$$\underset{c \in candidates}{\operatorname{argmax}} P(c|w) = \underset{c \in candicates}{\operatorname{argmax}} \frac{P(w|c) \cdot P(c)}{\cancel{P(w)}}$$

P(w) is the same for every c
⇒ we may ignore it

1. argmax
   ⇒ Python's `max(candidates, key=func)`
2. c ∈ candidates : which candidates to choose from?
   ⇒ Generate candidates with edit operations
3. P(c) : probability of of each candidate?
   ⇒ Calculate probabilities from word-count dictionary
4. P(w|c) : probability of *w* given *c*
   ⇒ Probabilities of generated candidates from 2.

Any better ways to generate + calculate the probabilities?

Your turn

- View and run the implementation at https://norvig.com/spell-correct.html
  - Hint: you might want to ignore or fix some parts of the `unit_tests()` function
- Expand the spell-checking algorithm:
  - Can you make it smarter? Faster? Better? More personalized?
  - There are also some other nice resources in the above website…!

Your turn (cont.)

Grading criteria

- Hand in Norvig's implementation + pass development set (75±2% correct rate) ⇒ +60
- Pass test set (≥ 65% correct rate of TA's version) ⇒ +20
  - "correct rate": `65% of 421 correct` of the program's `spelltest()` output
- Your own additional feature to the original Norvig code
  - Higher correct rate ⇒ +10 * (rank % among all students)
  - A brief report explaining what you added, and why it worked or didn't work ⇒ +10 max
    (Your score will vary based on what you added to the code and your analysis in the report)

TA’s notes

Code requirements:

- Keep Norvig’s `spelltest()` and `Testset()` functions for TAs to grade your code.

- Hand in **your code** `{student id}.py` and **report** `{student id}.pdf` (if available) on **elearn**.
- Deadline for assignment 1: 11:59 a.m. Sep 22 (Thu)