Assignment 1: Word Analogy

2024 NTHU Natural Language Processing

Hung-Yu Kao

IKM Lab TAs

What is analogy?

- Anthology helps to assess how well the word embedding model captures semantic and syntactic relationships between words.

The analogy task is often summarized by the famous phrase:

A is to B as C is to D. (e.g. **King** is to **Queen** as **Man** is to **Woman**.)

In euqation: $\overrightarrow{\textbf{King}} + \overrightarrow{\text{Queen}} - \overrightarrow{\textbf{Man}} \approx \overrightarrow{\text{Woman}}$

Dataset

Google Word Analogy (Mikolov et al., 2013)

- 19,544 entries
  - 5 sub-categories for **semantic** relationships
  - 9 sub-categories for **syntactic** relationships
- Each entry: four words, separated by a space. (e.g. **A B C D**)
- Task: **A** is to **B** in the same sense that **C** is to ?
  - The forth word (**D**) in each entry is the answer.

Dataset examples

*Answer in red

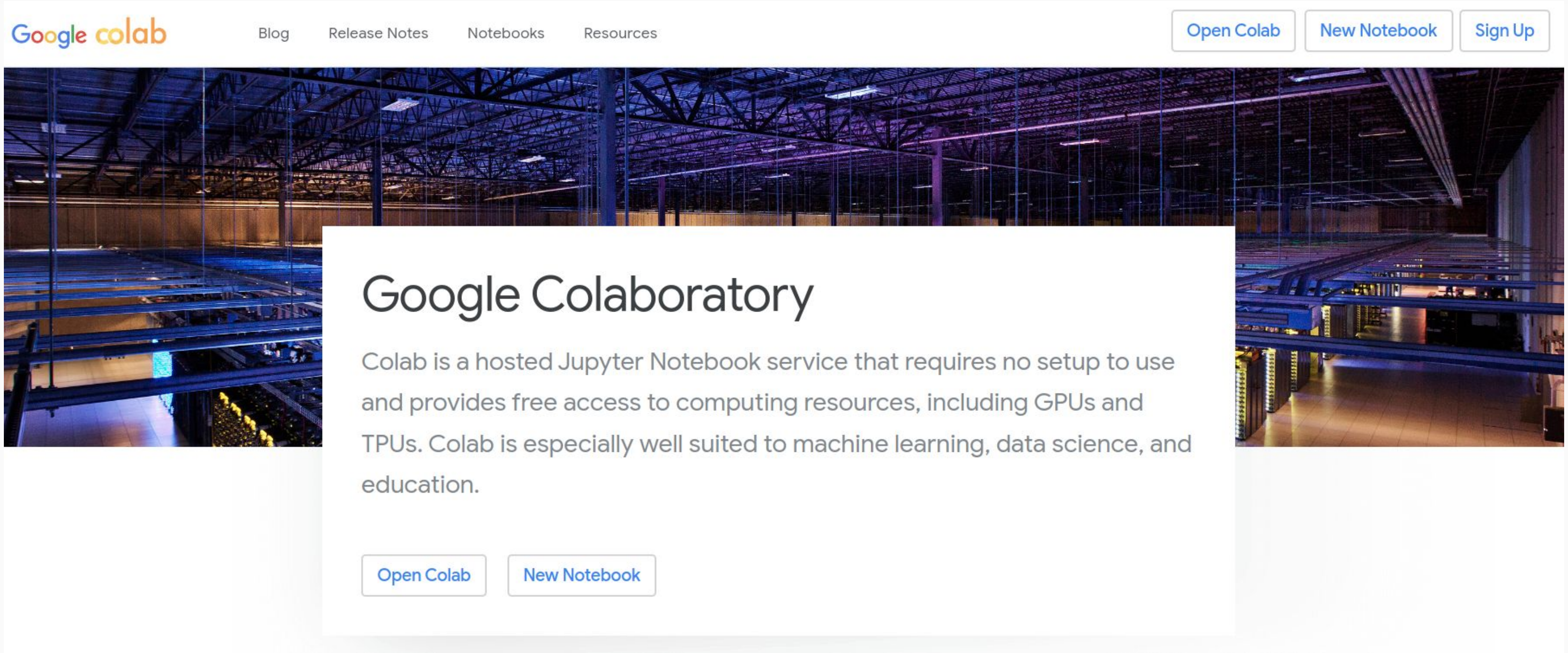
- Task: **A** is to **B** in the same sense that **C** is to ?

Category	Sub-category	Example
Semantic	family	king queen man woman
Syntactic	gram1-adjective-to-adverb	infrequent infrequently cheerful cheerfully

Code (hints only)

Coding environment

https://colab.research.google.com/



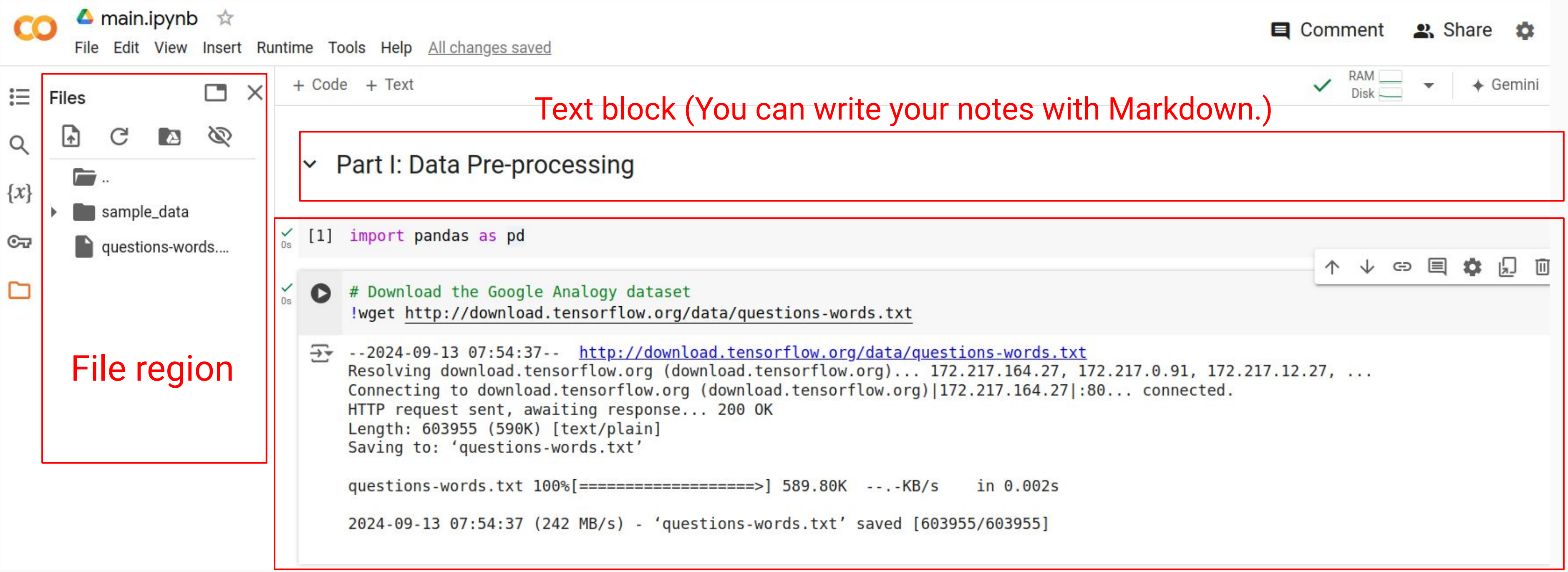
Google Colaboratory

Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.

Brief Introduction to Colab

- Cloud-Based Environment: Colab is entirely online, so no installation is required. You can write and run Python code directly in your browser.
- Free Access to Hardware: It provides access to **GPUs** and **TPUs** for free, making it ideal for machine learning, data science, and AI tasks.
- **Easy to use**: Colab uses Jupyter notebooks, a widely used format that allows combining code, output, and markdown text in a single document.

Quick view of the Colab editor



Text block (You can write your notes with Markdown.)

Part I: Data Pre-processing

```
[1] import pandas as pd
```

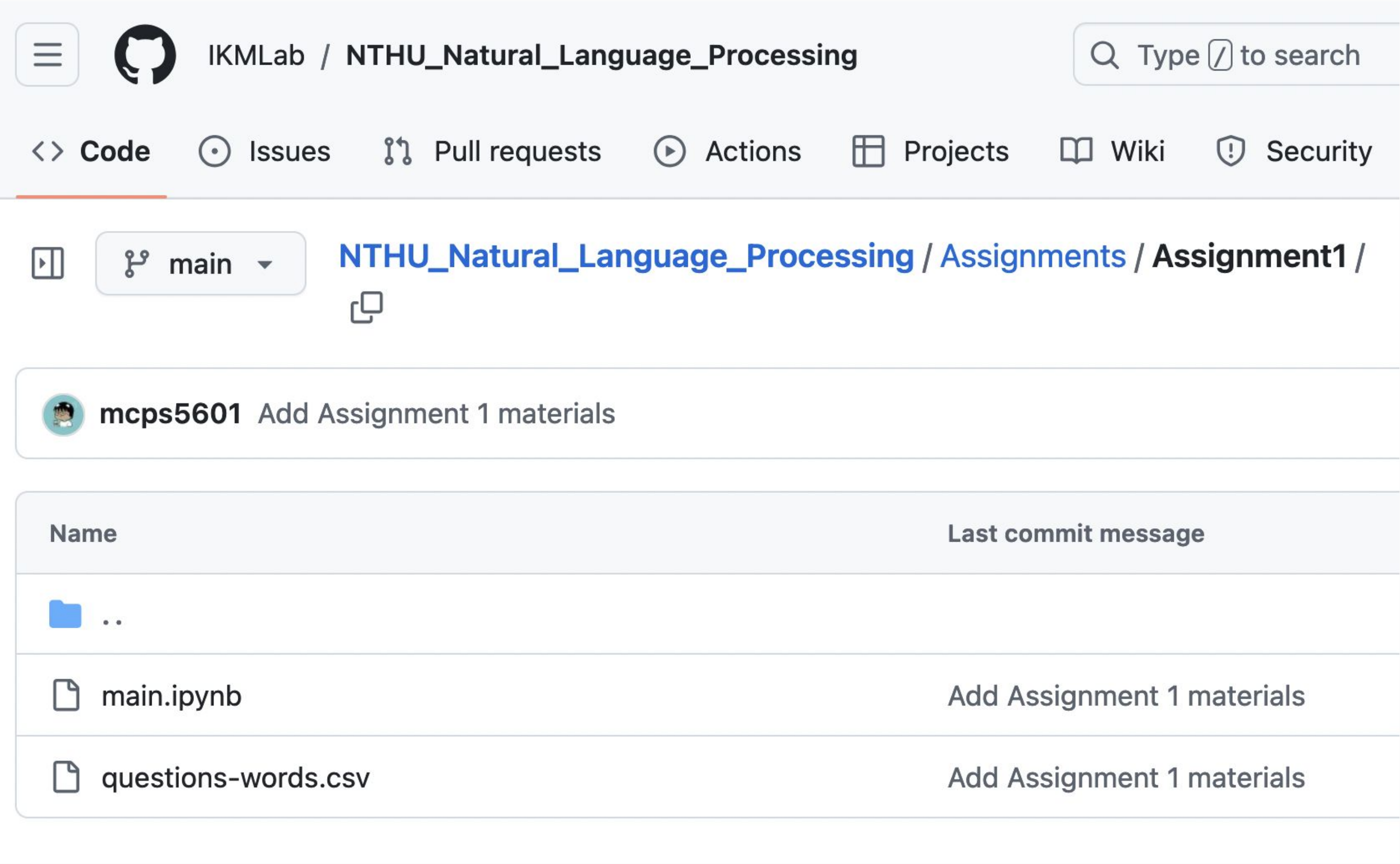
```
# Download the Google Analogy dataset
!wget http://download.tensorflow.org/data/questions-words.txt
```

```
--2024-09-13 07:54:37--  http://download.tensorflow.org/data/questions-words.txt
Resolving download.tensorflow.org (download.tensorflow.org)... 172.217.164.27, 172.217.0.91, 172.217.12.27, ...
Connecting to download.tensorflow.org (download.tensorflow.org)|172.217.164.27|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 603955 (590K) [text/plain]
Saving to: ‘questions-words.txt’

questions-words.txt 100%[===================>] 589.80K  --.-KB/s    in 0.002s

2024-09-13 07:54:37 (242 MB/s) - ‘questions-words.txt’ saved [603955/603955]
```

File region

Code blocks (with outputs)

Assignment 1 materials

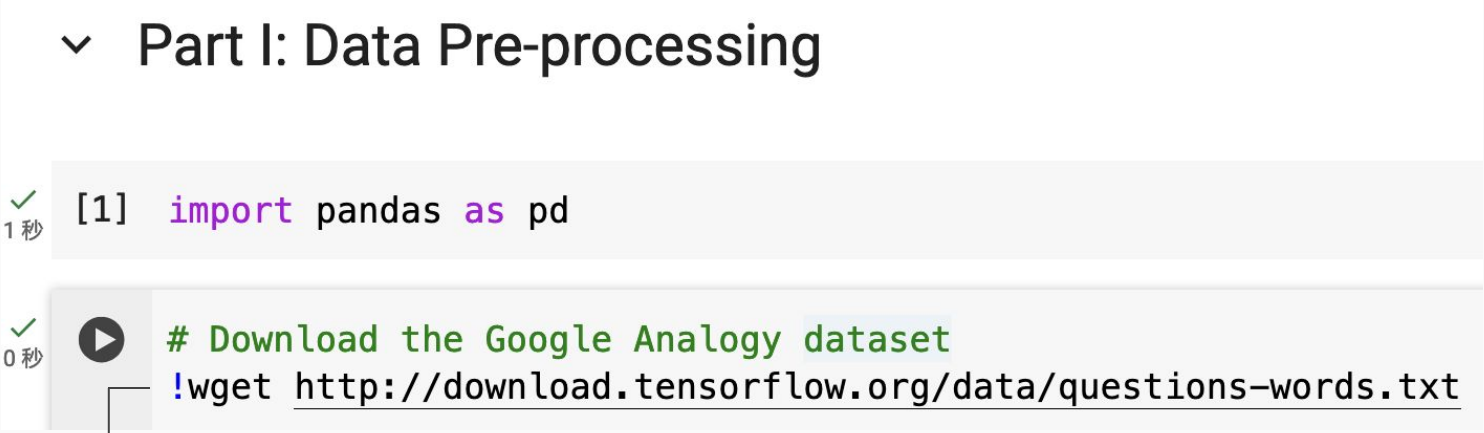


NTHU_Natural_Language_Processing / Assignments / Assignment1 /

mcps5601 Add Assignment 1 materials

Name	Last commit message
..	
main.ipynb	Add Assignment 1 materials
questions-words.csv	Add Assignment 1 materials

Download dataset



Part I: Data Pre-processing

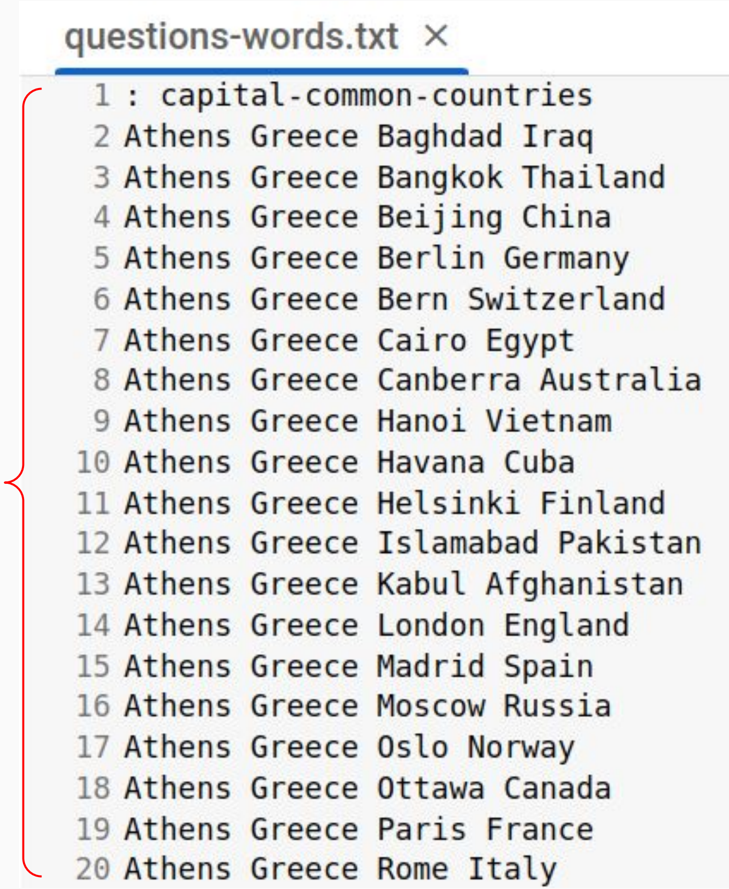
```
[1] import pandas as pd
```

```
# Download the Google Analogy dataset
!wget http://download.tensorflow.org/data/questions-words.txt
```

The downloaded data is a text file.

Check the downloaded file

- You can first browse the file on Colab!

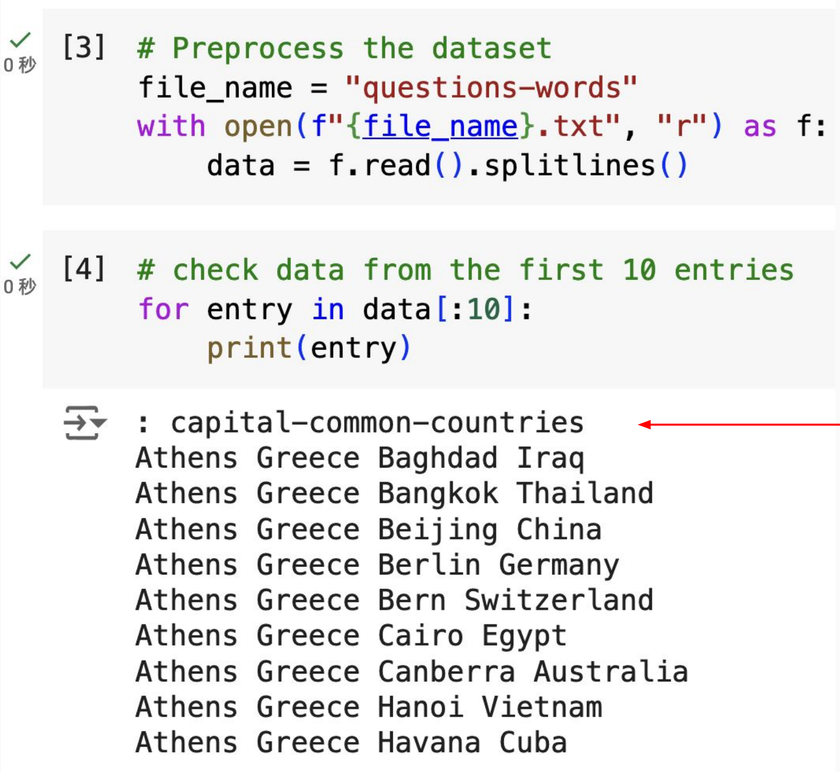


questions-words.txt ×

```
: capital-common-countries
Athens Greece Baghdad Iraq
Athens Greece Bangkok Thailand
Athens Greece Beijing China
Athens Greece Berlin Germany
Athens Greece Bern Switzerland
Athens Greece Cairo Egypt
Athens Greece Canberra Australia
Athens Greece Hanoi Vietnam
Athens Greece Havana Cuba
Athens Greece Helsinki Finland
Athens Greece Islamabad Pakistan
Athens Greece Kabul Afghanistan
Athens Greece London England
Athens Greece Madrid Spain
Athens Greece Moscow Russia
Athens Greece Oslo Norway
Athens Greece Ottawa Canada
Athens Greece Paris France
Athens Greece Rome Italy
```

Line-by-line manner

Load data and check the first ten entries



```python
# Preprocess the dataset
file_name = "questions-words"
with open(f"{file_name}.txt", "r") as f:
    data = f.read().splitlines()
```

```python
# check data from the first 10 entries
for entry in data[:10]:
    print(entry)
```

```
: capital-common-countries
Athens Greece Baghdad Iraq
Athens Greece Bangkok Thailand
Athens Greece Beijing China
Athens Greece Berlin Germany
Athens Greece Bern Switzerland
Athens Greece Cairo Egypt
Athens Greece Canberra Australia
Athens Greece Hanoi Vietnam
Athens Greece Havana Cuba
```

Sub-category

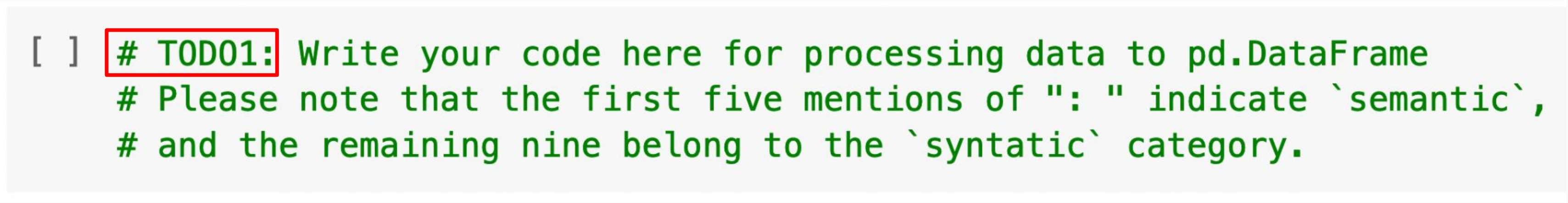
TODOs

#TODO1	Convert the analogy data to pd.DataFrame (evaluation dataset)
#TODO2	Get predictions of the analogy task using pre-trained word embeddings
#TODO3	Plot t-SNE to see word relationships in the sub-category of \`family\`
#TODO4	Sample 20% Wikipedia articles
#TODO5	Train your own word embeddings with the sampled articles
#TODO6	Get predictions of the analogy task using your trained word embeddings
#TODO7	Plot t-SNE to see word relationships in the sub-category of \`family\`

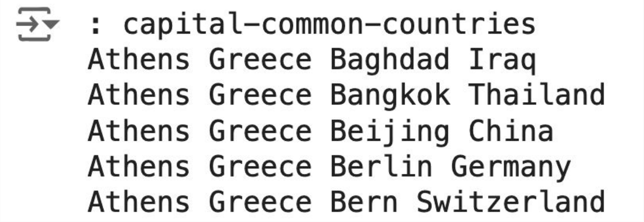
🟧 1. Use pre-trained word embeddings

🟩 2. Train your own word embeddings

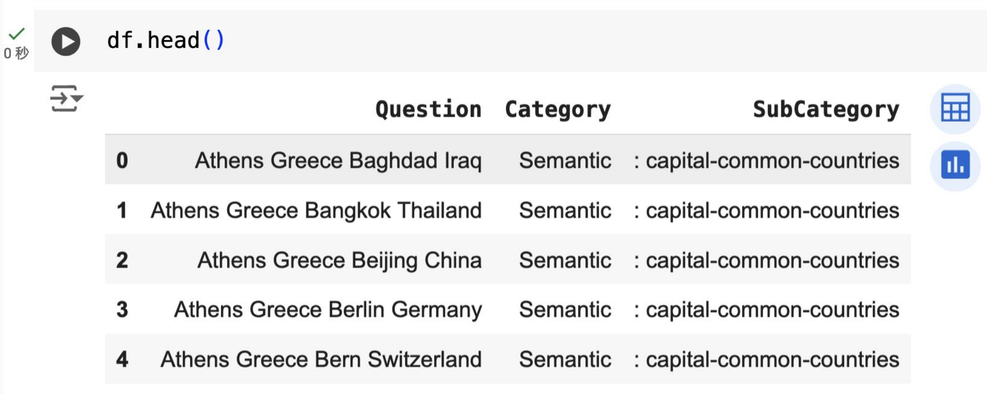
TODO1: Convert data to pd.DataFrame



```
[ ] # TODO1: Write your code here for processing data to pd.DataFrame
    # Please note that the first five mentions of ": " indicate `semantic`,
    # and the remaining nine belong to the `syntatic` category.
```



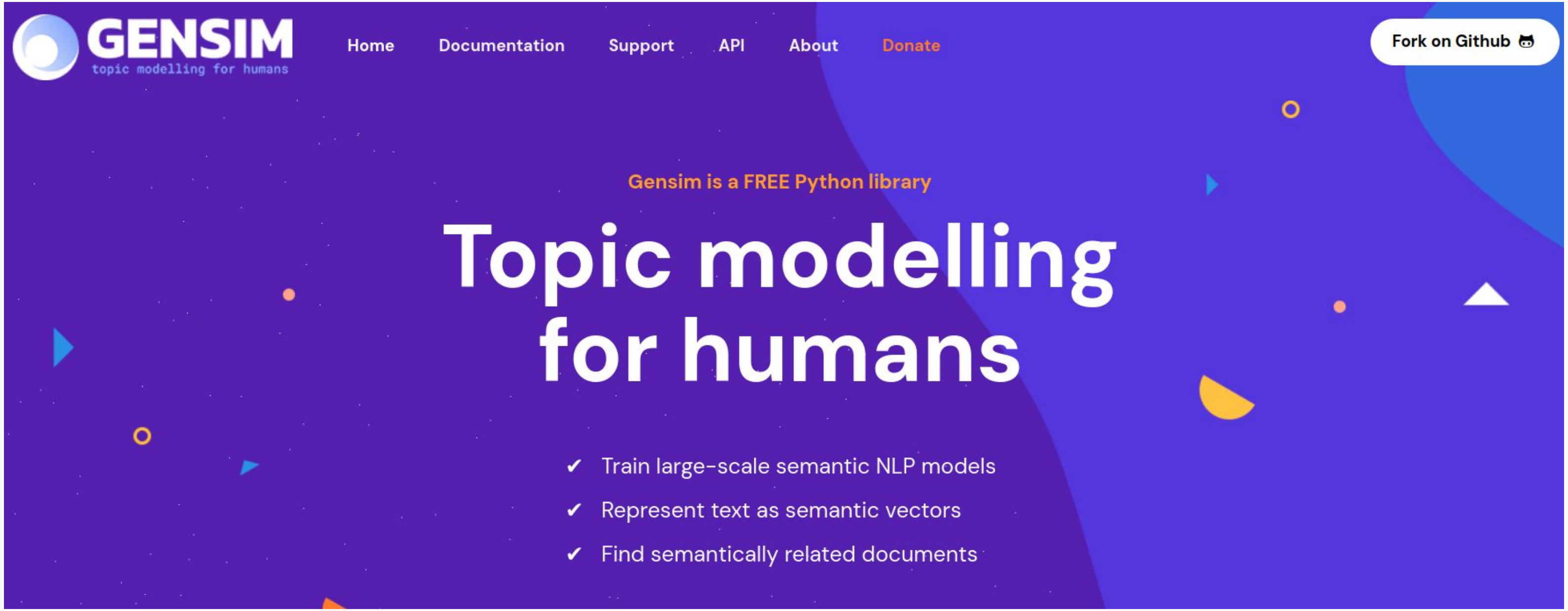
```
: capital-common-countries
Athens Greece Baghdad Iraq
Athens Greece Bangkok Thailand
Athens Greece Beijing China
Athens Greece Berlin Germany
Athens Greece Bern Switzerland
```



```
df.head()
```

	Question	Category	SubCategory
0	Athens Greece Baghdad Iraq	Semantic	: capital-common-countries
1	Athens Greece Bangkok Thailand	Semantic	: capital-common-countries
2	Athens Greece Beijing China	Semantic	: capital-common-countries
3	Athens Greece Berlin Germany	Semantic	: capital-common-countries
4	Athens Greece Bern Switzerland	Semantic	: capital-common-countries

Gensim: A popular Python package for embeddings



Gensim is a FREE Python library

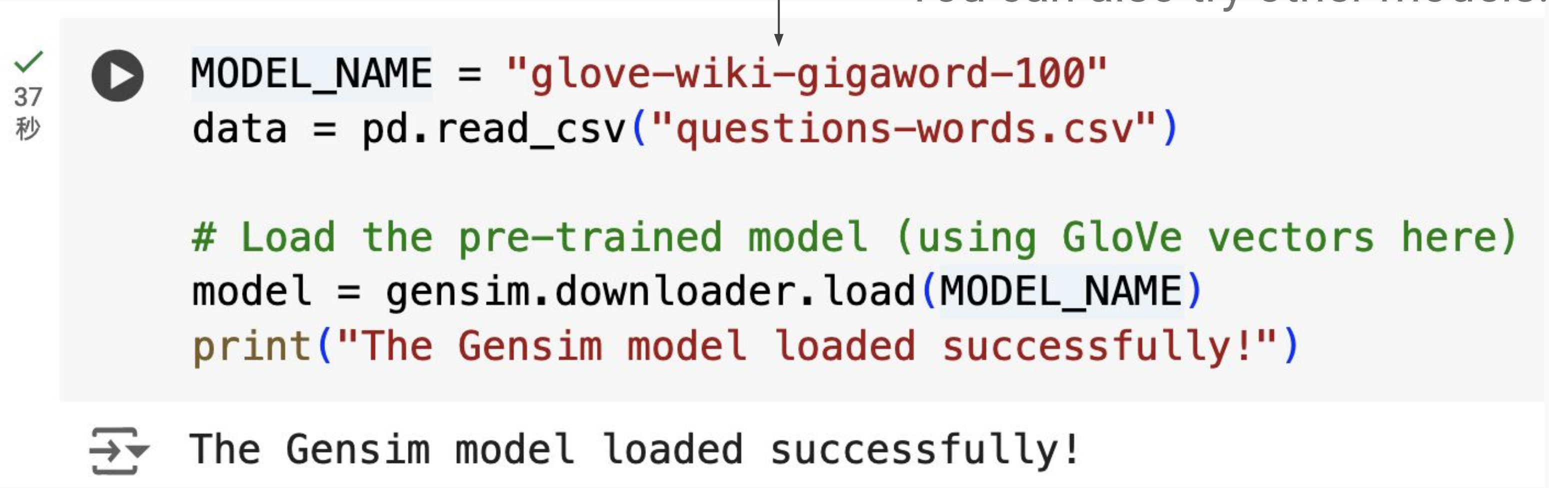
Topic modelling for humans

- ✓ Train large-scale semantic NLP models
- ✓ Represent text as semantic vectors
- ✓ Find semantically related documents

https://radimrehurek.com/gensim/index.html

Load the word embedding model with Gensim

You can also try other models!

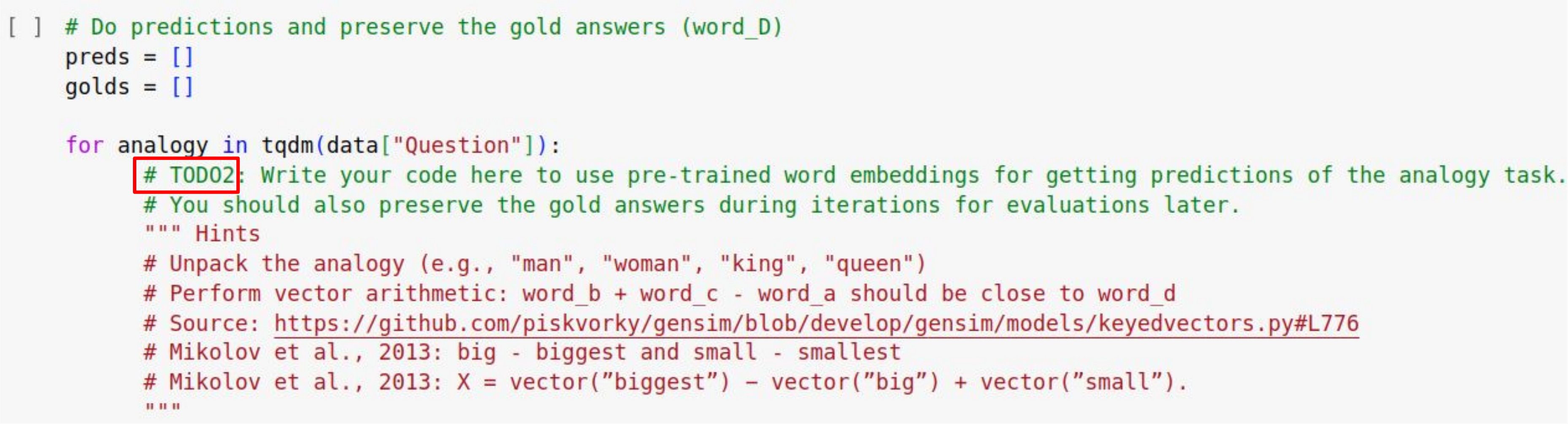


```
MODEL_NAME = "glove-wiki-gigaword-100"
data = pd.read_csv("questions-words.csv")

# Load the pre-trained model (using GloVe vectors here)
model = gensim.downloader.load(MODEL_NAME)
print("The Gensim model loaded successfully!")
```

```
The Gensim model loaded successfully!
```

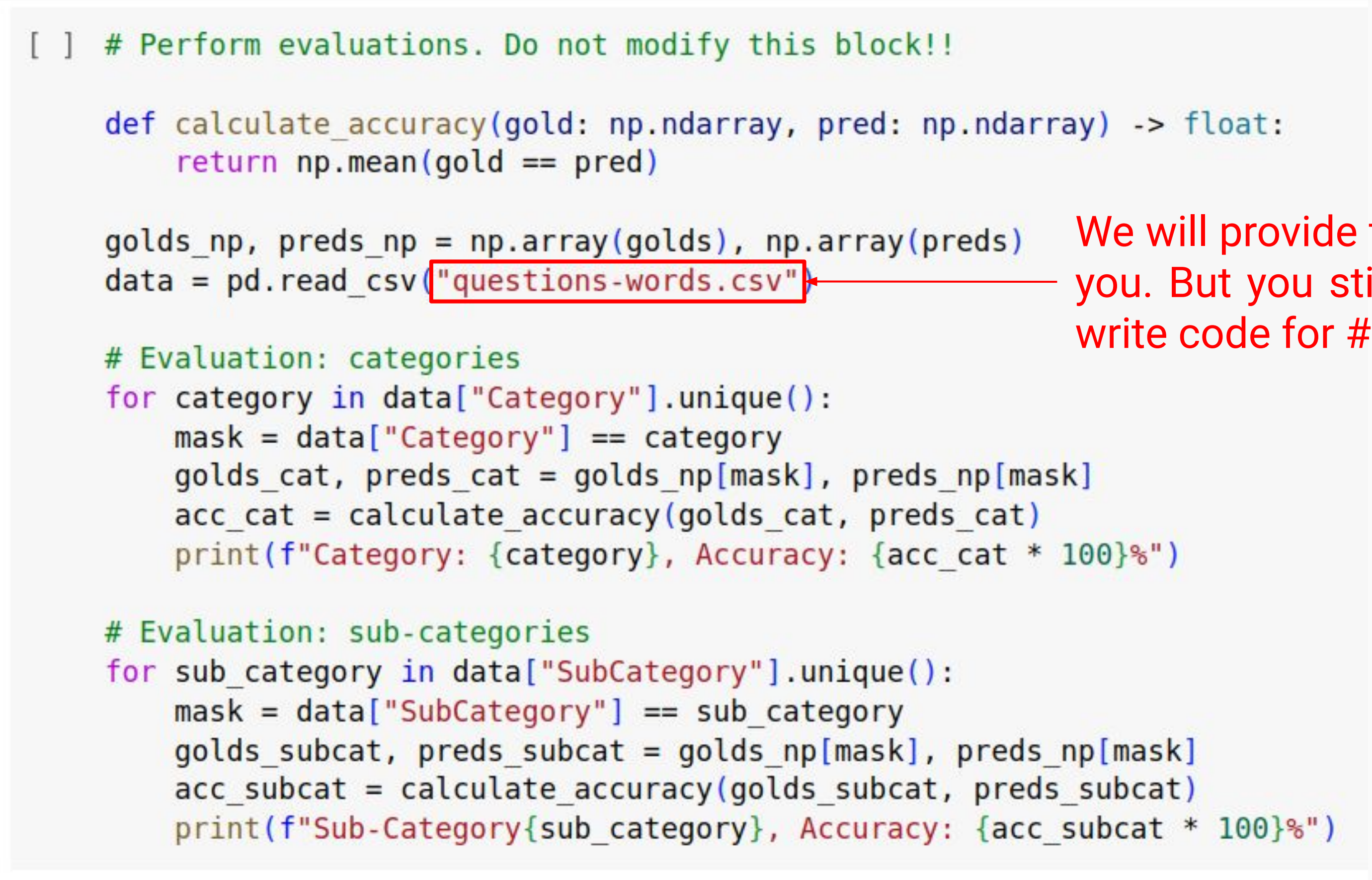
https://radimrehurek.com/gensim/models/word2vec.html#pretrained-models

TODO2: Get predictions of the analogy task using pre-trained word embeddings



```
[ ] # Do predictions and preserve the gold answers (word_D)
    preds = []
    golds = []

    for analogy in tqdm(data["Question"]):
          # TODO2: Write your code here to use pre-trained word embeddings for getting predictions of the analogy task.
          # You should also preserve the gold answers during iterations for evaluations later.
          """ Hints
          # Unpack the analogy (e.g., "man", "woman", "king", "queen")
          # Perform vector arithmetic: word_b + word_c - word_a should be close to word_d
          # Source: https://github.com/piskvorky/gensim/blob/develop/gensim/models/keyedvectors.py#L776
          # Mikolov et al., 2013: big - biggest and small - smallest
          # Mikolov et al., 2013: X = vector(”biggest”) − vector(”big”) + vector(”small”).
          """
```

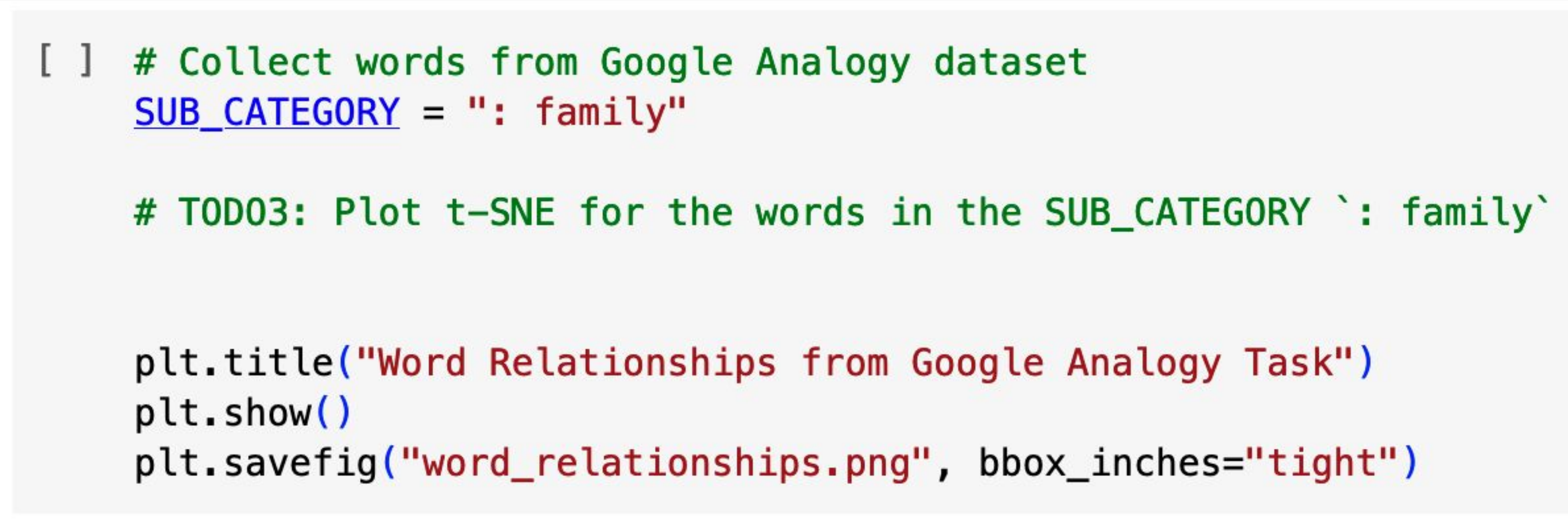
Do evaluations (We’ve finished for you to test!)



```
[ ]  # Perform evaluations. Do not modify this block!!

     def calculate_accuracy(gold: np.ndarray, pred: np.ndarray) -> float:
         return np.mean(gold == pred)

     golds_np, preds_np = np.array(golds), np.array(preds)
     data = pd.read_csv("questions-words.csv")

     # Evaluation: categories
     for category in data["Category"].unique():
         mask = data["Category"] == category
         golds_cat, preds_cat = golds_np[mask], preds_np[mask]
         acc_cat = calculate_accuracy(golds_cat, preds_cat)
         print(f"Category: {category}, Accuracy: {acc_cat * 100}%")

     # Evaluation: sub-categories
     for sub_category in data["SubCategory"].unique():
         mask = data["SubCategory"] == sub_category
         golds_subcat, preds_subcat = golds_np[mask], preds_np[mask]
         acc_subcat = calculate_accuracy(golds_subcat, preds_subcat)
         print(f"Sub-Category{sub_category}, Accuracy: {acc_subcat * 100}%")
```

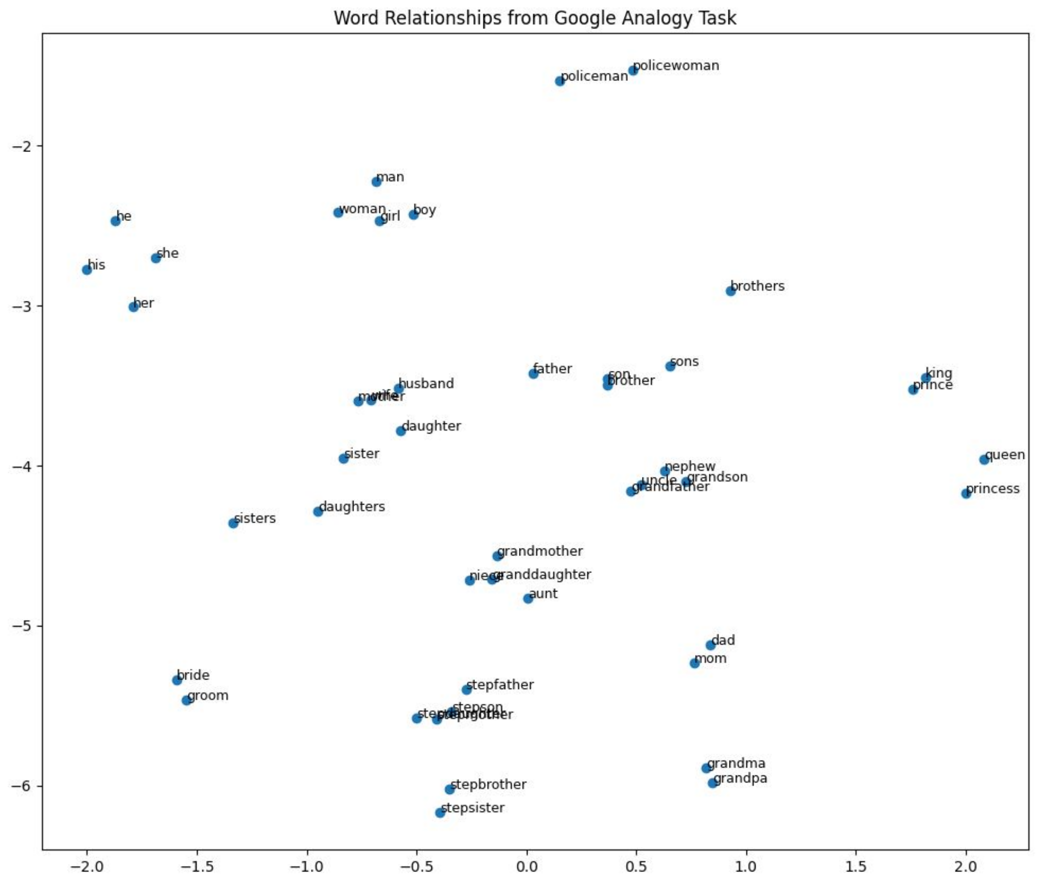
We will provide this file to you. But you still need to write code for #TODO1.

TODO3: Plot t-SNE to see word relationships in the sub-category of `family`



```
# Collect words from Google Analogy dataset
SUB_CATEGORY = ": family"

# TODO3: Plot t-SNE for the words in the SUB_CATEGORY `: family`


plt.title("Word Relationships from Google Analogy Task")
plt.show()
plt.savefig("word_relationships.png", bbox_inches="tight")
```

t-SNE


Word Relationships from Google Analogy Task
policewoman
policeman
man
woman
girl
boy
he
she
his
her
brothers
sons
father
son
brother
husband
king
prince
daughter
queen
sister
nephew
uncle
grandson
grandfather
princess
daughters
sisters
grandmother
granddaughter
aunt
dad
mom
bride
groom
stepfather
stepson
grandma
grandpa
stepbrother
stepsister
−2
−3
−4
−5
−6
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Train your own word embeddings

Training corpus: Wikipedia dump

https://dumps.wikimedia.org/wikidatawiki/20240901/

wikidatawiki dump progress on 20240901

This is the Wikimedia dump service. Please read the copyrights information. See Meta:Data dumps for documentation on the provided data formats.

Older versions of the 7zip decoder on Windows are known to have problems with some bz2-format files for larger wikis; we recommend the use of bzip2 for Windows for these cases. 7zip 20.00 alpha (x64) on Win10 LTSC 2019 and later versions should work properly.

Please report problems with these dumps on Phabricator and add the Dumps-generation tag.

See all databases list.

This dump is in progress; see also the previous dump from 2024-08-20

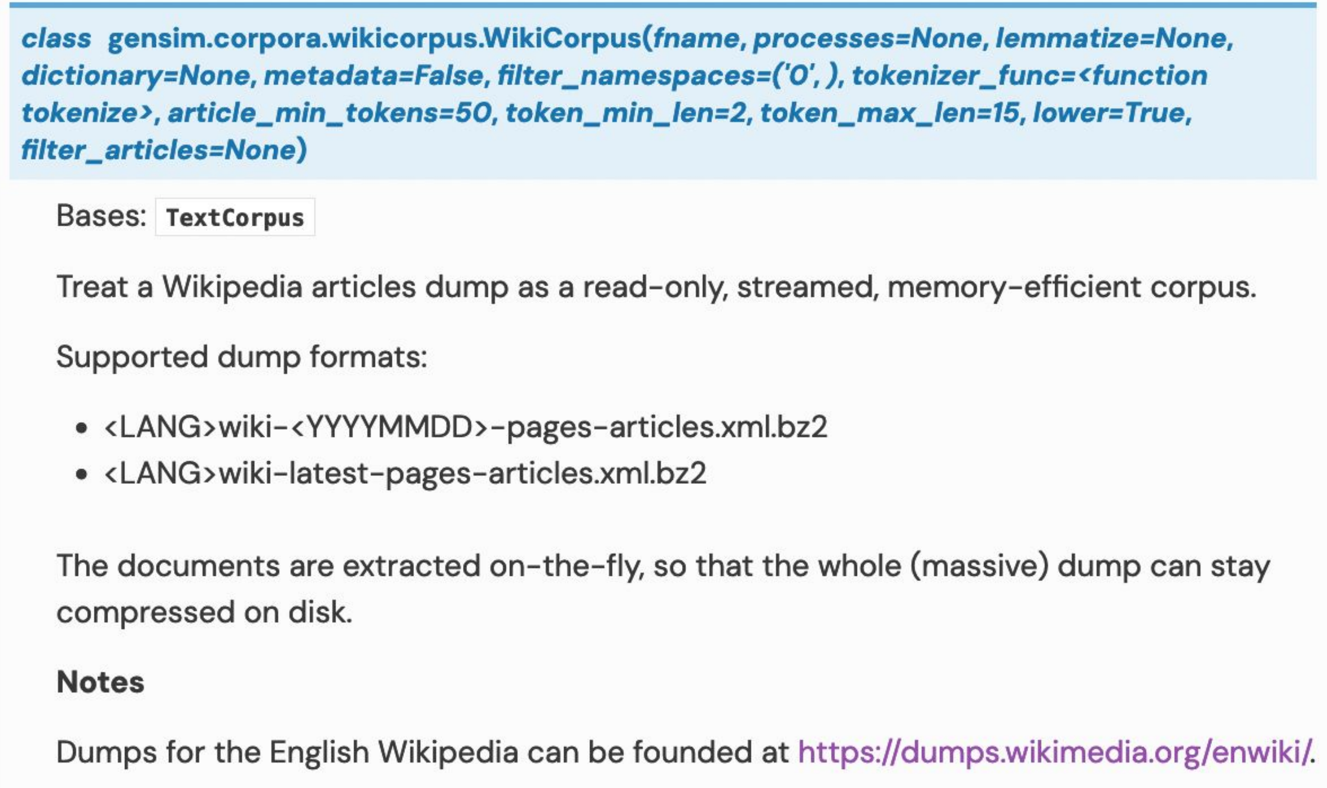
For a machine-readable version of the information on this page, see the json status file.

Currently 6M+ articles for EN Wiki

An example of raw data of Wiki dump in XML

```
 <page>
     <title>Anarchism</title>
     <ns>0</ns>
     <id>12</id>
     <revision>
     <id>1243078189</id>
     <parentid>1243073506</parentid>
     <timestamp>2024-08-30T11:06:56Z</timestamp>

'''Anarchism''' is a [[political philosophy]] and [[Political movement|movement]] that is against all forms of
[[authority]] and seeks to abolish the [[institu
tions]] it claims maintain unnecessary [[coercion]] and [[Social hierarchy|hierarchy]], typically including
the [[state (polity)|state]] and [[capitalism]]. An
archism advocates for the replacement of the state with [[Stateless society|stateless societies]] and
[[Voluntary association|voluntary]] [[Free association (c
ommunism and anarchism)|free associations]]. As a historically [[left-wing]] movement, this reading of
anarchism is placed on the [[Far-left politics|farthest
left]] of the [[political spectrum]], usually described as the [[libertarian]] wing of the [[socialist
movement]] ([[libertarian socialism]]).
```

Wiki corpus pre-processing with Gensim



***class* gensim.corpora.wikicorpus.WikiCorpus(*fname, processes=None, lemmatize=None, dictionary=None, metadata=False, filter_namespaces=('0',), tokenizer_func=<function tokenize>, article_min_tokens=50, token_min_len=2, token_max_len=15, lower=True, filter_articles=None*)**

Bases: `TextCorpus`

Treat a Wikipedia articles dump as a read-only, streamed, memory-efficient corpus.

Supported dump formats:

- <LANG>wiki-<YYYYMMDD>-pages-articles.xml.bz2
- <LANG>wiki-latest-pages-articles.xml.bz2

The documents are extracted on-the-fly, so that the whole (massive) dump can stay compressed on disk.

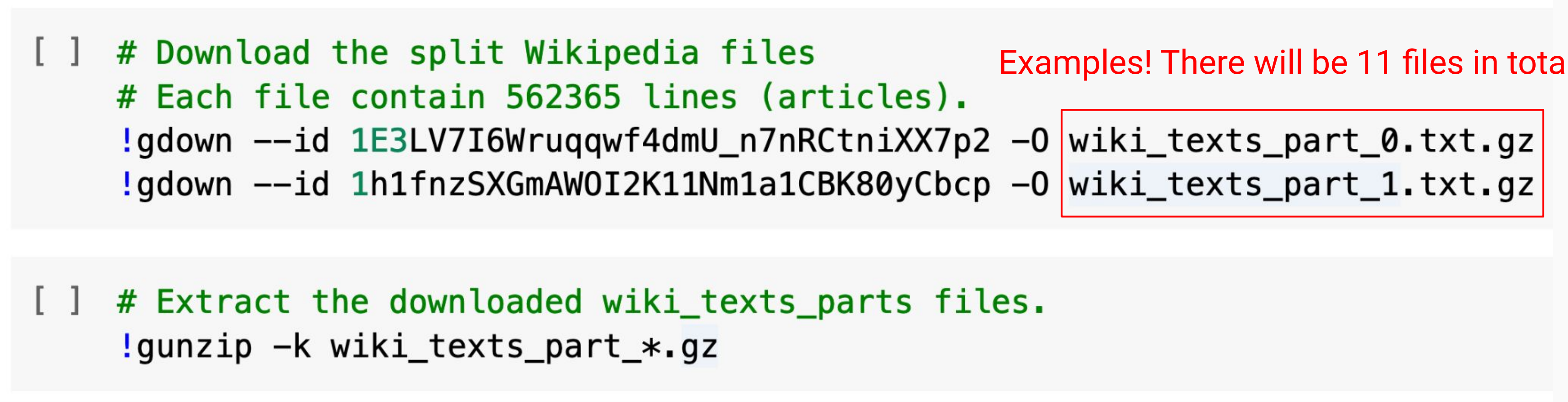
Notes

Dumps for the English Wikipedia can be founded at https://dumps.wikimedia.org/enwiki/.

https://radimrehurek.com/gensim/corpora/wikicorpus.html

We’ve done Wiki data cleaning for you!

- Downloading the raw Wiki file takes time.
- Cleaning the raw Wiki corpus takes even much longer time.

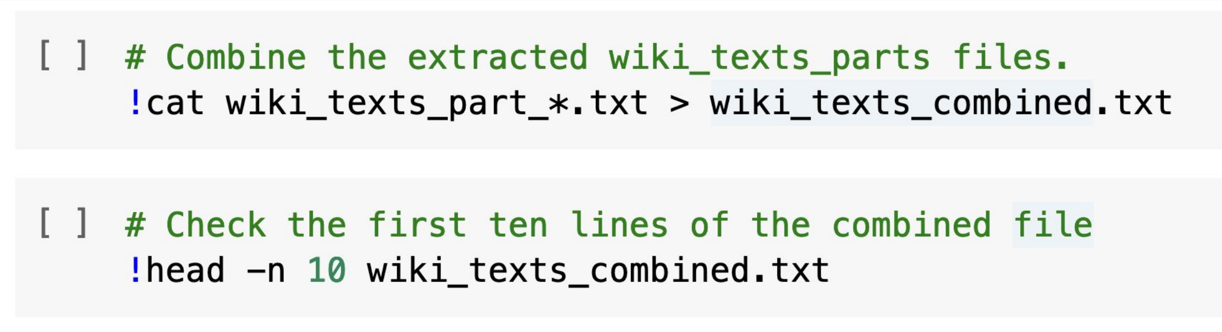


```
[ ]  # Download the split Wikipedia files
     # Each file contain 562365 lines (articles).
     !gdown --id 1E3LV7I6Wruqqwf4dmU_n7nRCtniXX7p2 -O wiki_texts_part_0.txt.gz
     !gdown --id 1h1fnzSXGmAWOI2K11Nm1a1CBK80yCbcp -O wiki_texts_part_1.txt.gz
```

Examples! There will be 11 files in total.

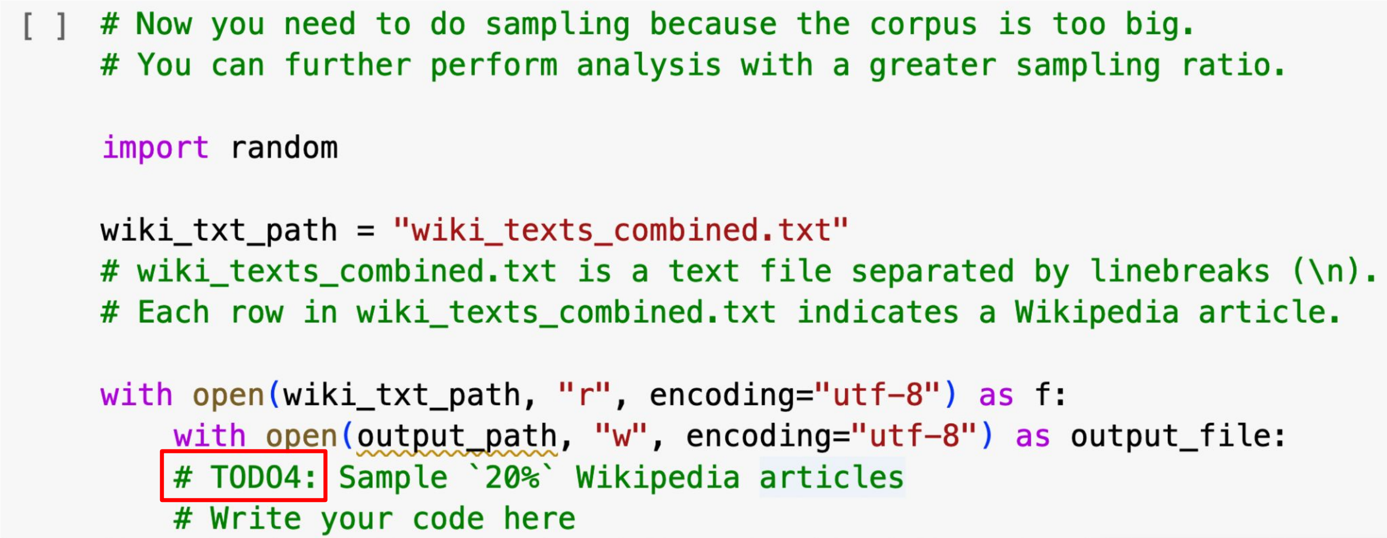
```
[ ]  # Extract the downloaded wiki_texts_parts files.
     !gunzip -k wiki_texts_part_*.gz
```

TODO4: Sample 20% Wikipedia articles



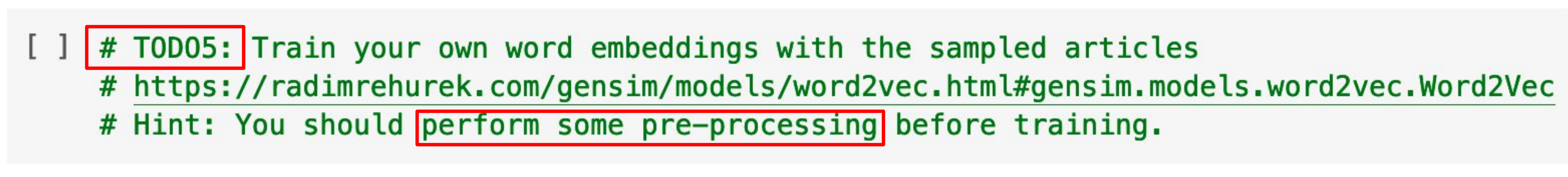
```
[ ]  # Combine the extracted wiki_texts_parts files.
     !cat wiki_texts_part_*.txt > wiki_texts_combined.txt
```

```
[ ]  # Check the first ten lines of the combined file
     !head -n 10 wiki_texts_combined.txt
```

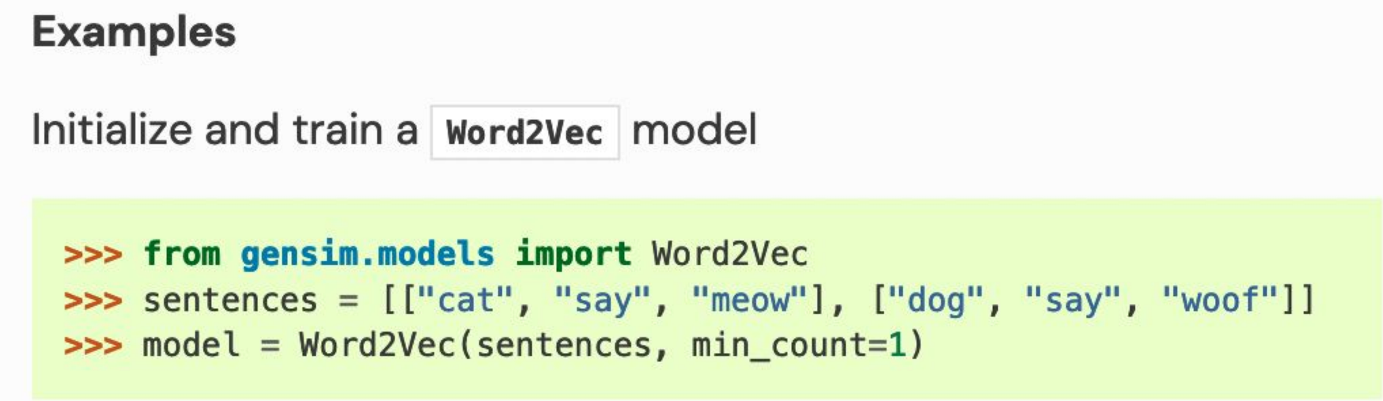


```
[ ]  # Now you need to do sampling because the corpus is too big.
     # You can further perform analysis with a greater sampling ratio.

     import random

     wiki_txt_path = "wiki_texts_combined.txt"
     # wiki_texts_combined.txt is a text file separated by linebreaks (\n).
     # Each row in wiki_texts_combined.txt indicates a Wikipedia article.

     with open(wiki_txt_path, "r", encoding="utf-8") as f:
         with open(output_path, "w", encoding="utf-8") as output_file:
         # TODO4: Sample `20%` Wikipedia articles
         # Write your code here
```

TODO5: Train your own word embeddings with the sampled articles



```
[ ] # TODO5: Train your own word embeddings with the sampled articles
    # https://radimrehurek.com/gensim/models/word2vec.html#gensim.models.word2vec.Word2Vec
    # Hint: You should perform some pre-processing before training.
```



Examples

Initialize and train a `Word2Vec` model

```
>>> from gensim.models import Word2Vec
>>> sentences = [["cat", "say", "meow"], ["dog", "say", "woof"]]
>>> model = Word2Vec(sentences, min_count=1)
```

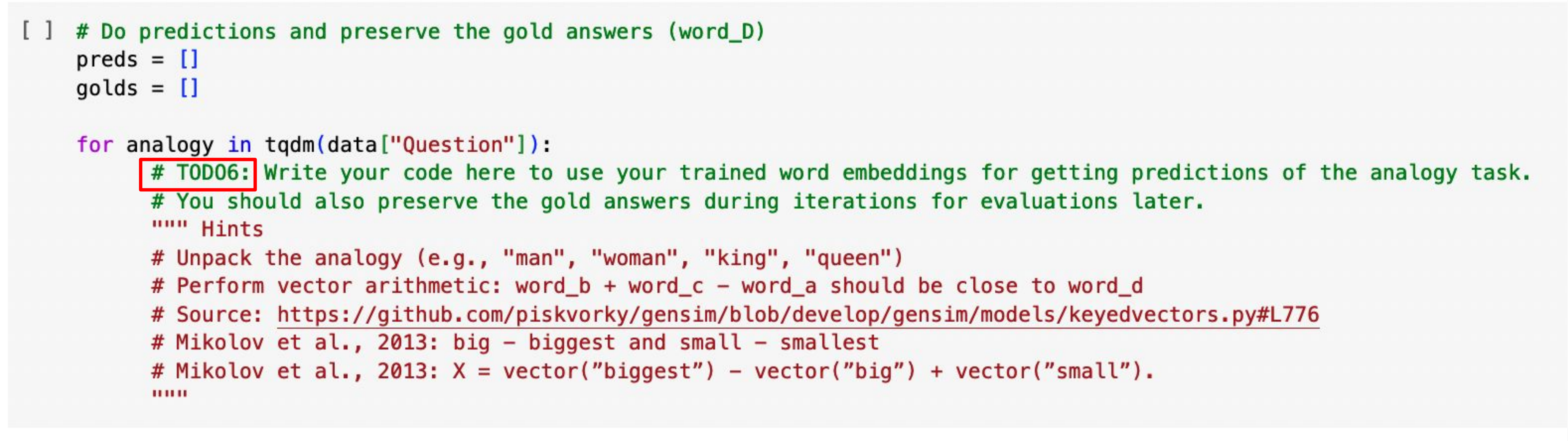
Recommended data pre-processing steps

- Remove non-English words
- Remove stop words
- Lemmatization (e.g., rocks -> rock)
- Better tokenization than simply splitting words based on whitespaces
- Retain the most frequent words for the dictionary

*Note that not every tricks yield better performance. You should try on your own.

TODO6: Get predictions of the analogy task using your trained word embeddings

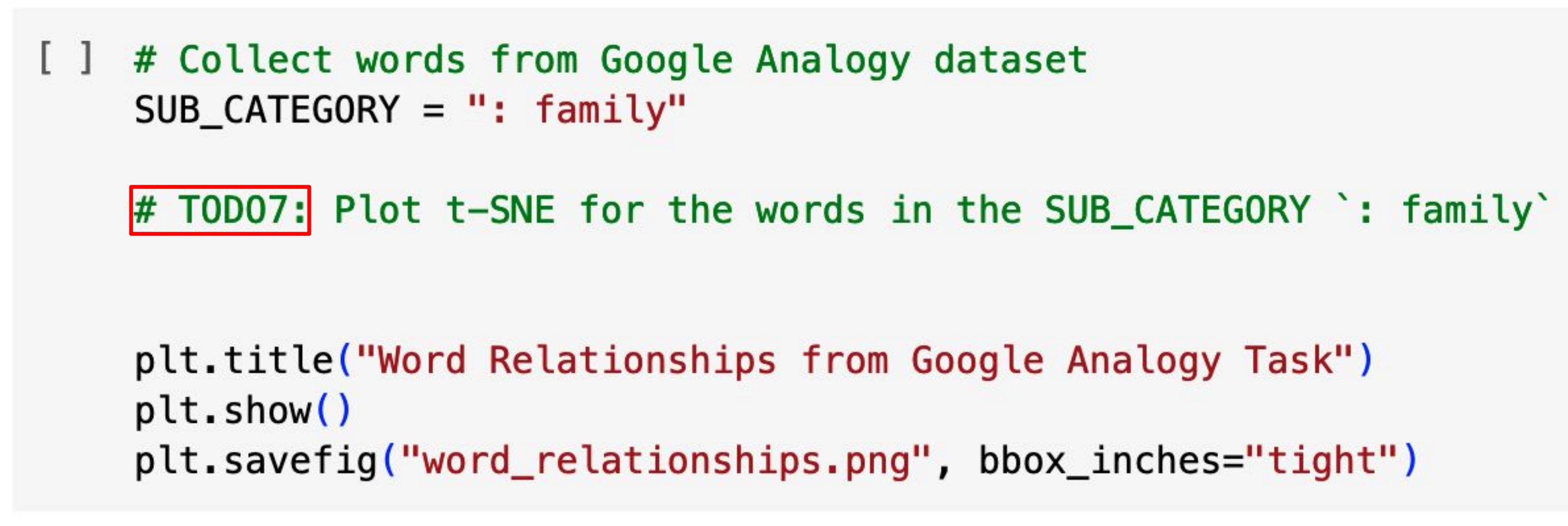
Same as #TODO2, you still need to get predictions for your trained embeddings.



```
[ ]  # Do predictions and preserve the gold answers (word_D)
     preds = []
     golds = []

     for analogy in tqdm(data["Question"]):
           # TODO6: Write your code here to use your trained word embeddings for getting predictions of the analogy task.
           # You should also preserve the gold answers during iterations for evaluations later.
           """ Hints
           # Unpack the analogy (e.g., "man", "woman", "king", "queen")
           # Perform vector arithmetic: word_b + word_c - word_a should be close to word_d
           # Source: https://github.com/piskvorky/gensim/blob/develop/gensim/models/keyedvectors.py#L776
           # Mikolov et al., 2013: big - biggest and small - smallest
           # Mikolov et al., 2013: X = vector(”biggest”) - vector(”big”) + vector(”small”).
           """
```

TODO7: Plot t-SNE to see word relationships in the sub-category of `family`

Same as #TODO3, you still need to plot t-SNE for your trained embeddings.



```
# Collect words from Google Analogy dataset
SUB_CATEGORY = ": family"

# TODO7: Plot t-SNE for the words in the SUB_CATEGORY `: family`


plt.title("Word Relationships from Google Analogy Task")
plt.show()
plt.savefig("word_relationships.png", bbox_inches="tight")
```

Scoring

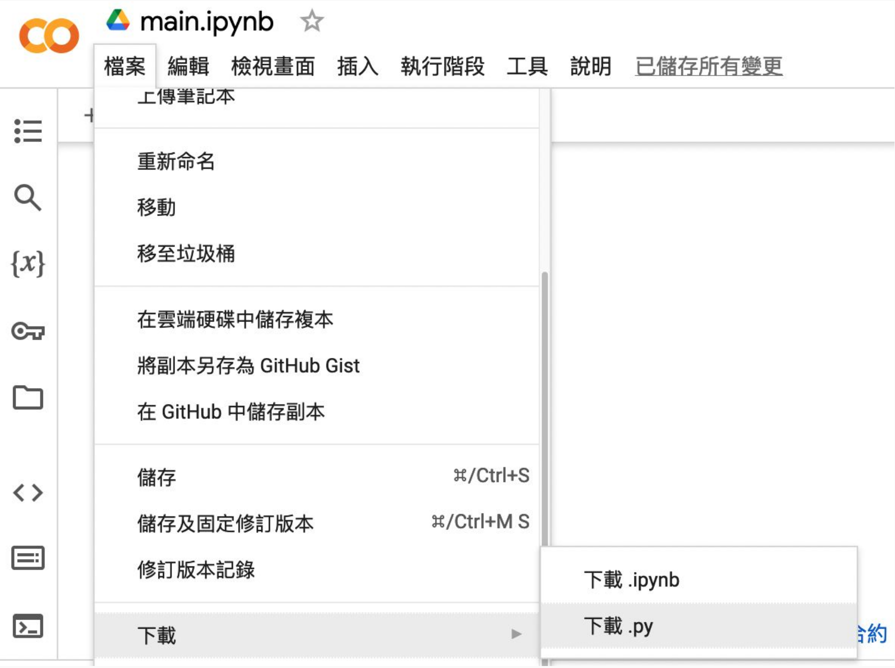
Coding work : 70% (10% for each of the seven TODOs)

Report: 30%

- Which embedding model do you use? What are the pre-processing steps? What are the hyperparameter settings?
- What is the performance for different categories or sub-categories?
- What do you believe is the primary factor causing the accuracy differences for your approach?
- What’s your discovery from your t-SNE visualization plots?
- What’s the difference in word representations if you increase the amount of training data?
- **… Anything that can strengthen your report.**

Delivery policies: File formats

- Coding work: Python file (.py)
    - Download your script via Colab.
- Package list: requirements.txt
    - E.g., gensim==4.3.3
- Report: Microsoft Word (.docx)
- No other formats are allowed.
- Zip the files above before uploading you assignment.



Delivery policies: Filenames

	Filename rule	**Filename example**
Coding work	NLP_HW1_school_student_ID.py	NLP_HW1_NTHU_12345678.py
Report	NLP_HW1_school_student_ID.docx	NLP_HW1_NTHU_12345678.docx
Package list	requirements.txt	
Zipped file	NLP_HW1_school_student_ID.zip	NLP_HW1_NTHU_12345678.zip

Delivery policies: Things You should include

- In your report:

	Example	
Environment types	**If Colab or Kaggle**	**If local**
Running environment	Colab	System: Ubuntu 22.04, CPU: Ryzen 7-7800X3D
Python version	Colab	Python 3.10.1

Delivery policies: Rules of coding

- If you use ChatGPT or Generative AI, please specify your usage **both** in:
  - **Code comments**
  - **Reports**
- **No plagiarism**. You should not copy and paste from your classmates.
- Please provide links if you take the code from the Internet as reference.
- The following behaviors will cause loss in the score of the assignment: **(1) Usage with Generative AI without specifications (2) Internet sources without specifications (3) Plagiarism.**

Uploading the zipped file

- Please upload your file to NTU COOL.
- You will have three weeks to finish this assignment.