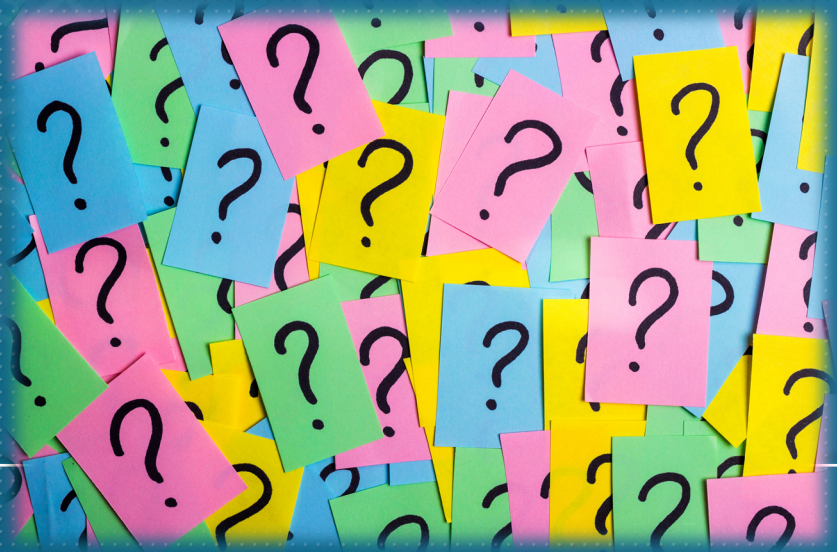
Natural Language Processing

30th November

UST 과학기술연합대학원대학교 UNIVERSITY OF SCIENCE & TECHNOLOGY

WHAT IS NLP..?

UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY



- Artificial intelligence studies by imitation of human learning methods and solves given problems based on the learned model.
- Through artificial intelligence that mimics human intelligence, what people want is a model that responds well to situations similar to humans.
- For this purpose, natural language processing is a task that converts human language into a form that the computer can understand.

Famous libraries for NLP

- Gensim
  - Provides various functions required to convert natural language into vectors.
  - Word2vec << We will learn about this today~!
- Transformers
  - BERT(Bidirectional Encoder Representations from Transformers)
  - GPT-3(Generative Pre-trained Transformer)

Terminology for NLP

- Tokenizing
  - Corpus: A corpus is a set of sample documents created for a natural language analysis task.
  - Tokenization: The division of a given corpus into units called tokens is called tokenization.
  - Simple example: Cut out based on whitespace.
    - ✓ Input: “Autumn is over and winter is coming.”
    - ✓ Output: “Autumn”, “is”, “over”, “and”, “winter”, “is”, “coming”

Terminology for NLP

- Things to consider in tokenization
  - Do not simply exclude punctuation marks or special characters.
    - ✓ Ph.D. / $45.55 / 123,456
  - When there are abbreviations and spaces within words.
    - ✓ we're == we are / I'm == I am / rock'n' roll == rock and roll
  - Standard tokenization example
    - ✓ rule 1: Keep words made up of hyphens as one.
    - ✓ rule 2: Words with 'fold' with an apostrophe, such as doesn't, are separated.

Terminology for NLP

- Things to consider in tokenization
  - Standard tokenization example
    - ✓ rule 1: Keep words made up of hyphens as one.
    - ✓ rule 2: Words with 'fold' with an apostrophe, such as doesn't, are separated.

```
from nltk.tokenize import TreebankWordTokenizer
```

```
tokenizer = TreebankWordTokenizer()
text = "FluNet is a web-based influenza surveillance platform provided by WHO. For thie year, however, Korea didn't submit data."
print(tokenizer.tokenize(text))
```

```
['FluNet', 'is', 'a', 'web-based', 'influenza', 'surveillance', 'platform', 'provided', 'by', 'WHO.', 'For', 'thie', 'year', ',', 'ho
wever', ',', 'Korea', 'did', "n't", 'submit', 'data', '.']
```

Terminology for NLP

- Cleaning & Normalization
  - Cleaning: Remove noise data from the corpus, which is an analysis target.
  - Normalization: Words with different expressions are combined to form the same word.
    - ✓ Consolidations of words with different notations based on rules: US vs USA
    - ✓ Convert upper case letters to the lower case

Terminology for NLP

- Cleaning & Normalization
  - Removing Unnecessary Words
    1) Removing Rare words
    2) Removing words with a very short length
    3) Regular Expression

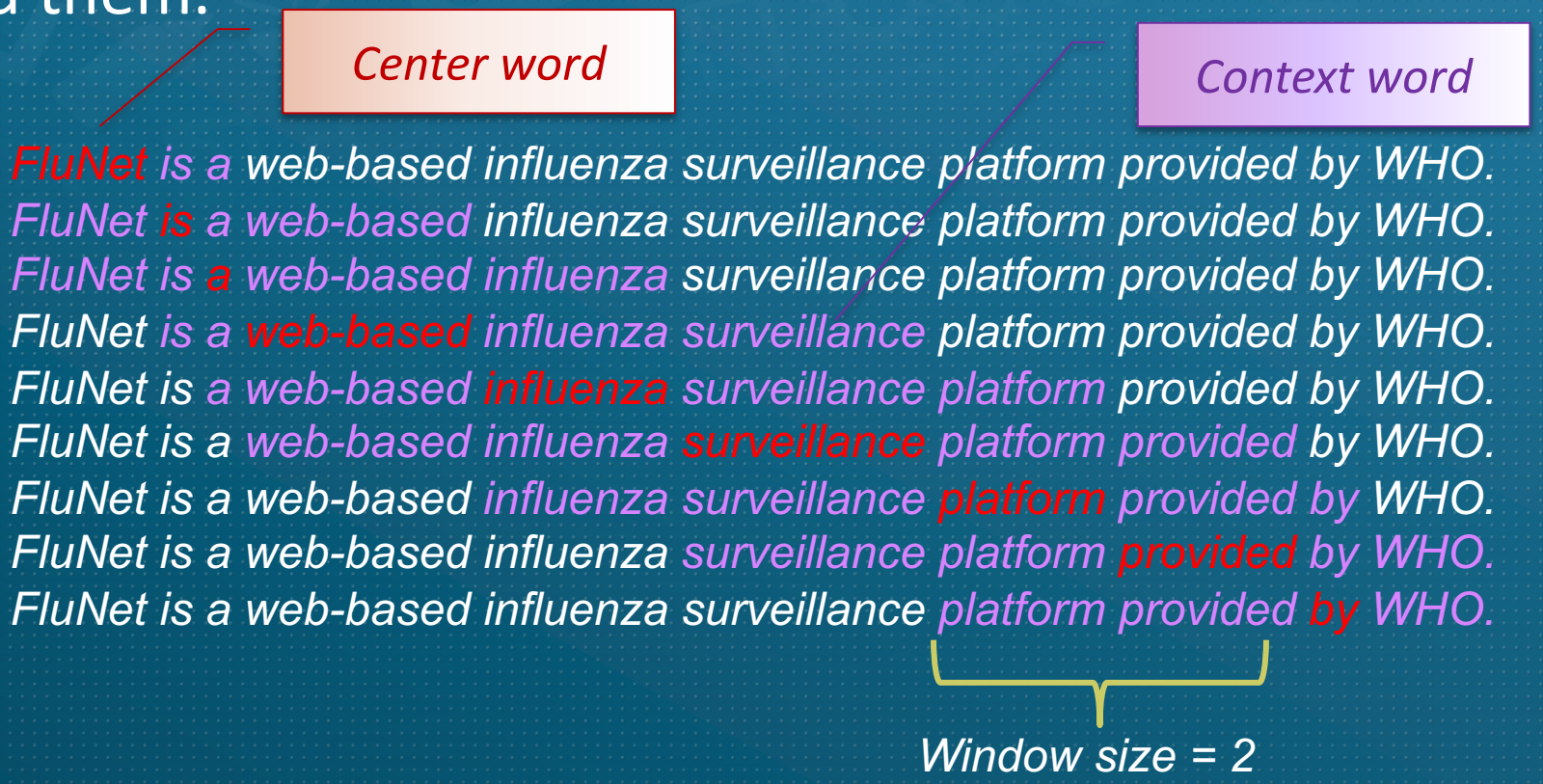
Word2vec

What is Word2vec..?

- Word2Vec was introduced in two papers between September and October 2013, by a team of researchers at Google.
- word2vec represents each distinct word with a particular list of numbers called a vector.
- Word2vec can utilize either of two model architectures to produce a distributed representation of words: continuous bag-of-words (CBOW) or continuous skip-gram.

Two Ways in Word2Vec

- CBOW(Continuous Bag of Words)
  - CBOW is a method of predicting words in the middle with words around them.



Two Ways in Word2Vec

- Continuous skip-gram
  - Skip-Gram is a method of predicting surrounding words with words in the middle.
  - Previously, in CBOW, the central word was predicted through the surrounding words, but Skip-gram tries to predict the surrounding words from the central word.

GenSim Word2Vec Example

https://www.kaggle.com/pierremegret/gensim-word2vec-tutorial

- `xlrd==1.1.0` : https://pypi.org/project/xlrd/
- `spaCy==2.0.12` : https://spacy.io/usage/
- `gensim==3.4.0` : https://radimrehurek.com/gensim/install.html
- `scikit-learn==0.19.1` : http://scikit-learn.org/stable/install.html
- `seaborn==0.8` : https://seaborn.pydata.org/installing.html

Homer Simpson

Marge Simpson

MaggieSimpson

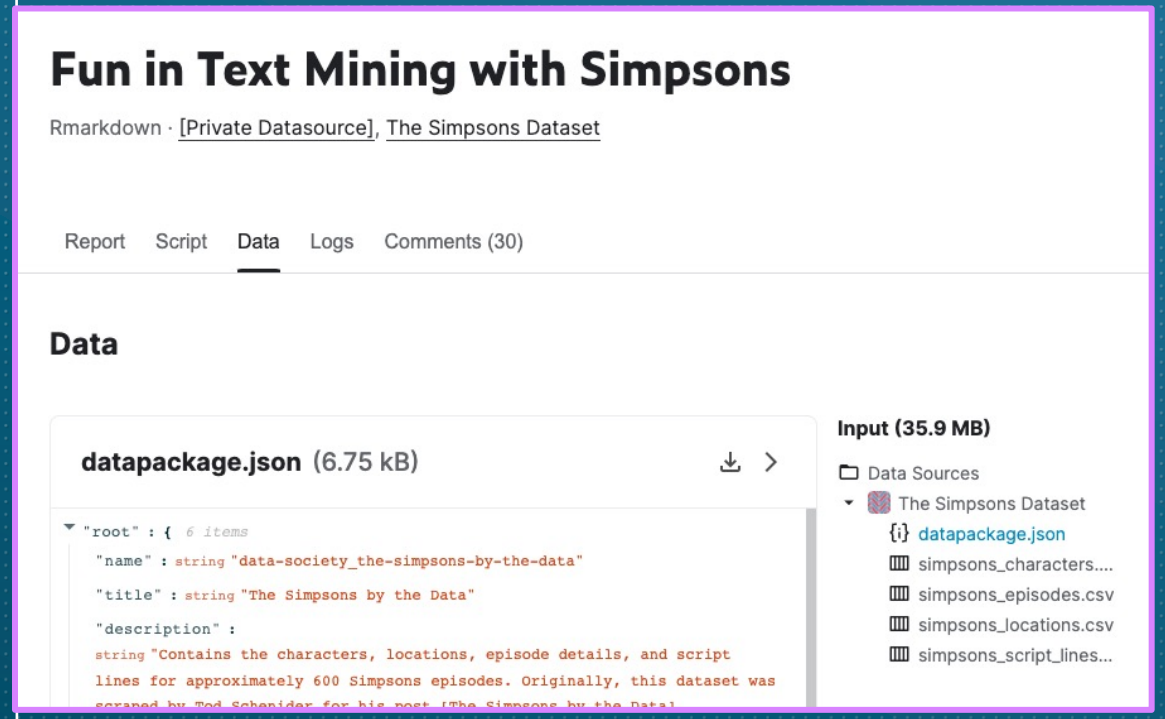
Bart Simpson

Mr. Burns

The Data

- Today's target data is 'the script from the Simpsons'.
- This dataset contains the characters, locations, episode details, and script lines for approximately 600 Simpsons episodes, dating back to 1989.
- It can be found here: https://www.kaggle.com/ambarish/fun-in-text-mining-with-simpsons/data (~25MB)

The Data



➢ DESCRIPTION

Contains the characters, locations, episode details, and script lines for approximately 600 Simpsons episodes.

➢ SUMMARY

Originally, this dataset was scraped by Tod Schenider for his post The Simpsons by the Data, for which he made the scraper available on GitHub. Kaggle user William Cukierski used the scraper to upload the data set, which has been rehosted here.

1. Preprocessing

- raw_character_text: the character who speaks (can be useful when monitoring the preprocessing steps)
- spoken_words: the raw text from the line of dialogue

```
import re  # For preprocessing
import pandas as pd  # For data handling
from time import time  # To time our operations
from collections import defaultdict  # For word frequency

import spacy  # For preprocessing

import logging  # Setting up the loggings to monitor gensim
logging.basicConfig(format="%(levelname)s - %(asctime)s: %(message)s", datefmt= '%H:%M:%S', level=logging.INFO)
```

```
df = pd.read_csv('./simpson_data/simpsons_dataset.csv')
df.shape
```

```
(158314, 2)
```

```
df.head()
```

	raw_character_text	spoken_words
0	Miss Hoover	No, actually, it was a little of both. Sometim...
1	Lisa Simpson	Where's Mr. Bergstrom?
2	Miss Hoover	I don't know. Although I'd sure like to talk t...
3	Lisa Simpson	That life is worth living.
4	Edna Krabappel-Flanders	The polls will be open from now until the end ...

1. Preprocessing

- The missing values comes from the part of the script where something happens, but with no dialogue.

```
df.isnull().sum()
```

```
raw_character_text     17814
spoken_words           26459
dtype: int64
```

```
df = df.dropna().reset_index(drop=True)
df.isnull().sum()
```

```
raw_character_text     0
spoken_words           0
dtype: int64
```

2. Cleaning

- Lemmatize and remove the stopwords and non-alphabetic characters for each line of dialogue.
  - ✓ Lemmatize means “sort words by grouping inflected or variant forms of the same word.”

```
nlp = spacy.load('en_core_web_sm', disable=['ner', 'parser']) # disabling Named Entity Recognition for speed

def cleaning(doc):
    # Lemmatizes and removes stopwords
    # doc needs to be a spacy Doc object
    txt = [token.lemma_ for token in doc if not token.is_stop]
    # Word2Vec uses context words to learn the vector representation of a target word,
    # if a sentence is only one or two words long,
    # the benefit for the training is very small
    if len(txt) > 2:
        return ' '.join(txt)
```

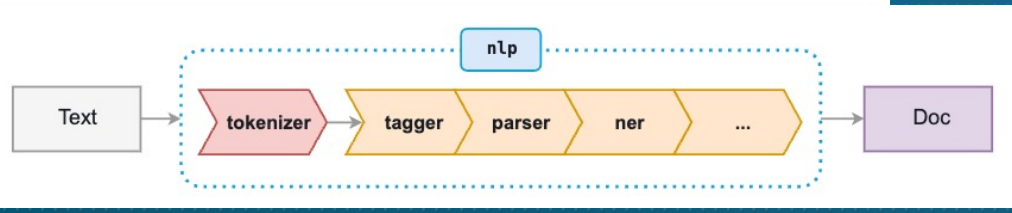
```
brief_cleaning = (re.sub("[^A-Za-z']+", ' ', str(row)).lower() for row in df['spoken_words'])
```

```
t = time()

txt = [cleaning(doc) for doc in nlp.pipe(brief_cleaning, batch_size=5000, n_process = -1)]

print('Time to clean up everything: {} mins'.format(round((time() - t) / 60, 2)))
```

```
Time to clean up everything: 1.5 mins
```

```
df_clean = pd.DataFrame({'clean': txt})
df_clean = df_clean.dropna().drop_duplicates()
df_clean.shape
```

```
(85952, 1)
```



3. Using Gensim

- We are using Gensim Phrases package to automatically detect common phrases (bigrams) from a list of sentences.
- The main reason we do this is to catch words like "mr_burns" or "bart_simpson" .

```
from gensim.models.phrases import Phrases, Phraser
```

```
sent = [row.split() for row in df_clean['clean']]
```

```
phrases = Phrases(sent, min_count=30, progress_per=10000)
```

```
INFO - 17:48:56: collecting all words and their counts
INFO - 17:48:56: PROGRESS: at sentence #0, processed 0 words and 0 word types
INFO - 17:48:56: PROGRESS: at sentence #10000, processed 63557 words and 52763 word types
INFO - 17:48:56: PROGRESS: at sentence #20000, processed 130938 words and 99744 word types
INFO - 17:48:56: PROGRESS: at sentence #30000, processed 192959 words and 138351 word types
INFO - 17:48:56: PROGRESS: at sentence #40000, processed 249832 words and 172387 word types
INFO - 17:48:56: PROGRESS: at sentence #50000, processed 311271 words and 208254 word types
INFO - 17:48:56: PROGRESS: at sentence #60000, processed 373576 words and 243325 word types
INFO - 17:48:56: PROGRESS: at sentence #70000, processed 436427 words and 278322 word types
INFO - 17:48:56: PROGRESS: at sentence #80000, processed 497902 words and 311462 word types
INFO - 17:48:56: collected 330189 token types (unigram + bigrams) from a corpus of 537083 words and 85952 sentences
INFO - 17:48:56: merged Phrases<330189 vocab, min_count=30, threshold=10.0, max_vocab_size=40000000>
INFO - 17:48:56: Phrases lifecycle event {'msg': 'built Phrases<330189 vocab, min_count=30, threshold=10.0, max_vocab_size=40000000>
in 0.53s', 'datetime': '2021-11-26T17:48:56.594874', 'gensim': '4.1.2', 'python': '3.9.4 (tags/v3.9.4:1f2e308, Apr  6 2021, 13:40:21)
[MSC v.1928 64 bit (AMD64)]', 'platform': 'Windows-10-10.0.19041-SP0', 'event': 'created'}
```

```
bigram = Phraser(phrases)
sentences = bigram[sent]
```

```
INFO - 17:49:32: exporting phrases from Phrases<330189 vocab, min_count=30, threshold=10.0, max_vocab_size=40000000>
INFO - 17:49:33: FrozenPhrases lifecycle event {'msg': 'exported FrozenPhrases<127 phrases, min_count=30, threshold=10.0> from Phrase
s<330189 vocab, min_count=30, threshold=10.0, max_vocab_size=40000000> in 0.60s', 'datetime': '2021-11-26T17:49:33.356641', 'gensim':
'4.1.2', 'python': '3.9.4 (tags/v3.9.4:1f2e308, Apr  6 2021, 13:40:21) [MSC v.1928 64 bit (AMD64)]', 'platform': 'Windows-10-10.0.190
41-SP0', 'event': 'created'}
```

```
word_freq = defaultdict(int)
for sent in sentences:
    for i in sent:
        word_freq[i] += 1
len(word_freq)
```

```
29651
```

3. Using Gensim

- Parameters
  - min_count = int - Ignores all words with total absolute frequency lower than this
  - window = int - The maximum distance between the current and predicted word within a sentence.

```
w2v_model = Word2Vec(min_count=20,
                     window=2,
                     size=300,
                     sample=6e-5,
                     alpha=0.03,
                     min_alpha=0.0007,
                     negative=20,
                     workers=cores-1)
```

3. Using Gensim

- Building the Vocabulary Table
  - Word2Vec requires us to build the vocabulary table (simply digesting all the words and filtering out the unique words, and doing some basic counts on them)

```
t = time()

w2v_model.build_vocab(sentences, progress_per=10000)

print('Time to build vocab: {} mins'.format(round((time() - t) / 60, 2)))
```

3. Using Gensim

- Training of the model
  - total_examples = int - Count of sentences
  - epochs = int - Number of iterations (epochs) over the corpus - [10, 20, 30]

```
t = time()

w2v_model.train(sentences, total_examples=w2v_model.corpus_count, epochs=30, report_delay=1)

print('Time to train the model: {} mins'.format(round((time() - t) / 60, 2)))
```

4. Exploring the model..!

- Here, we will ask our model to find the word most similar to some of the most iconic characters of the Simpsons!

```
w2v_model.wv.most_similar(positive=["homer"])
```

```
[('marge', 0.7565736174583435),
 ('crummy', 0.7183721065552124),
 ('sweetheart', 0.6857509613037109),
 ('bartender', 0.6811666488647461),
 ("y'ello", 0.6794908046722412),
 ('good_friend', 0.6785166263580322),
 ('rude', 0.6734878420829773),
 ('sorry', 0.6732467412948608),
 ('gee', 0.6692392826080322),
 ('hammock', 0.667172431945808)]
```

4. Exploring the model..!

- Here, we will ask our model to find the word most similar to some of the most iconic characters of the Simpsons!

```
w2v_model.wv.most_similar(positive=["homer_simpson"])
```

```
[('pleased', 0.7021192908287048),
 ('select', 0.6916783452033997),
 ('recent', 0.6788554191589355),
 ('council', 0.6750441789627075),
 ('fellow', 0.6576398015022278),
 ('governor', 0.655640423297882l),
 ('montgomery_burn', 0.654712438583374),
 ('easily', 0.6516457796096802),
 ('erotic', 0.6494141221046448),
 ('judgment', 0.639093279838562)]
```

4. Exploring the model..!

- What about Marge..?

```
w2v_model.wv.most_similar(positive=["marge"])
```

```
[('homer', 0.7565736174583435),
 ('becky', 0.6908849477767944),
 ('badly', 0.689024806022644),
 ('hammock', 0.6791144013404846),
 ('spoil', 0.67778921127319 34),
 ('homie', 0.6770768165588379),
 ('honey', 0.6731526255607605),
 ('attract', 0.6684098839759827),
 ('grownup', 0.6673101186752319),
 ('rapture', 0.6593933701515198)]
```

4. Exploring the model..!

- Let’s check Bart!

```
w2v_model.wv.most_similar(positive=["bart"])
```

```
[('lisa', 0.8418701291084291),
 ('hearing', 0.7701248526573181),
 ('convince', 0.7359577417373657),
 ('mom', 0.7326207160949707),
 ('dr_hibbert', 0.7111763954162598),
 ('maggie', 0.7051558494567871),
 ('substitute', 0.7045503854751587),
 ('mom_dad', 0.7034925818443298),
 ('surprised', 0.7028061151504517),
 ('homework', 0.7024934291839)]
```

4. Exploring the model..!

- Similarities!

```
w2v_model.wv.similarity('maggie', 'baby')
```

```
0.6789742
```

4. Exploring the model..!

- Odd-One-Out
  - Here, we ask our model to give us the word that does not belong to the list!

```
w2v_model.wv.doesnt_match(["nelson", "bart", "milhouse"])
```

```
'nelson'
```

4. Exploring the model..!

- Odd-One-Out
  - Here, we ask our model to give us the word that does not belong to the list!

```
w2v_model.wv.doesnt_match(['homer', 'patty', 'selma'])
```

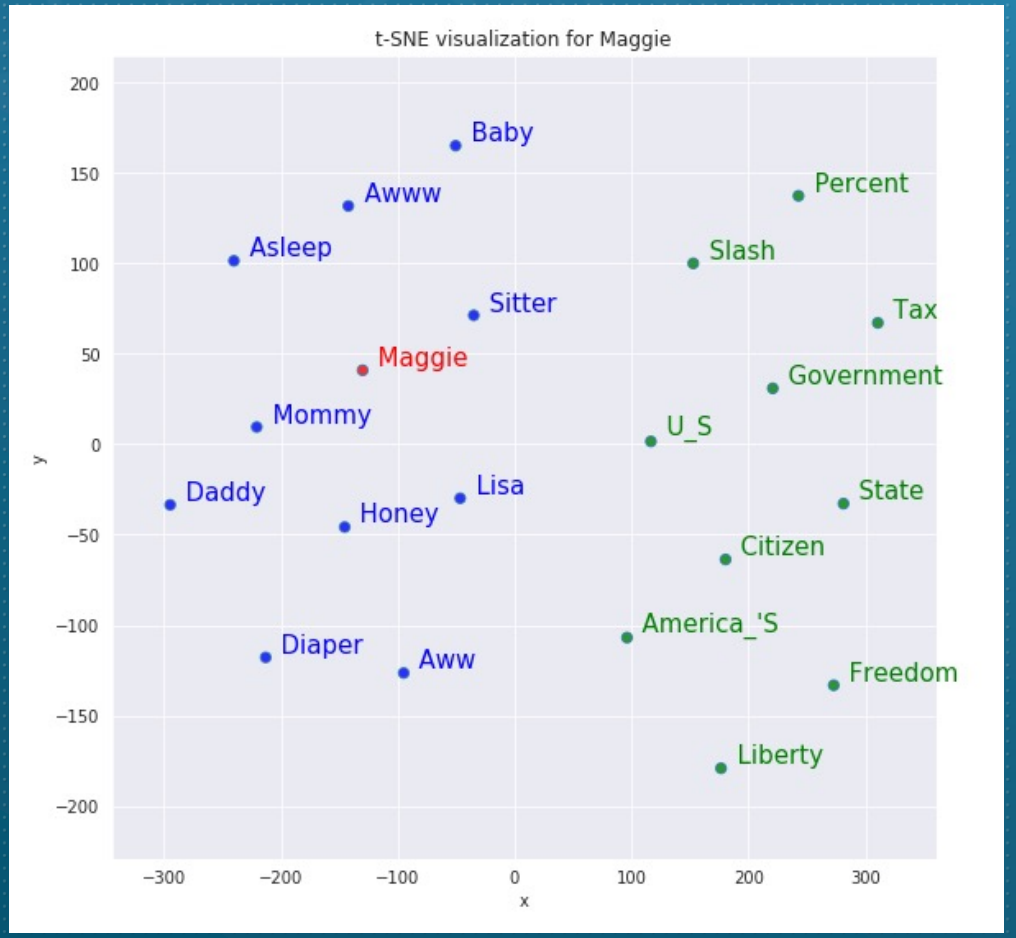
```
'homer'
```

5. t-SNE visualization

- t-SNE is a non-linear dimensionality reduction algorithm that attempts to represent high-dimensional data and the underlying relationships between vectors in a lower-dimensional space.
- Our goal in this section is to plot our 300 dimensions vectors into 2 dimensional graphs, and see if we can spot interesting patterns.

5. t-SNE visualization

- 10 Most similar words vs. 10 Most dissimilar



Assignments

Assignments

1. **Visit** the Kaggle's Gensim Tutorial page, and just try to create your model.
2. If possible, try to create model using documents related with disease .
3. Athiruj Poositaporn from KISTI and Changlun Sun from KICT Prepare a 5-minute presentation of what these people have done.

THANK YOU