NLP Assignment 3

Joshua Brook

An NYT Connections-style game, generated using word embeddings and played in the terminal. Given 16 words in a table, the objective of the game is to categorise them into 4 groups of 4, each with a semantic connection.

1. How to Play

1.1 Setup

- Have Python 3.x installed on your system.
- The repo / source code cloned on your PC
- Install required packages via pip:

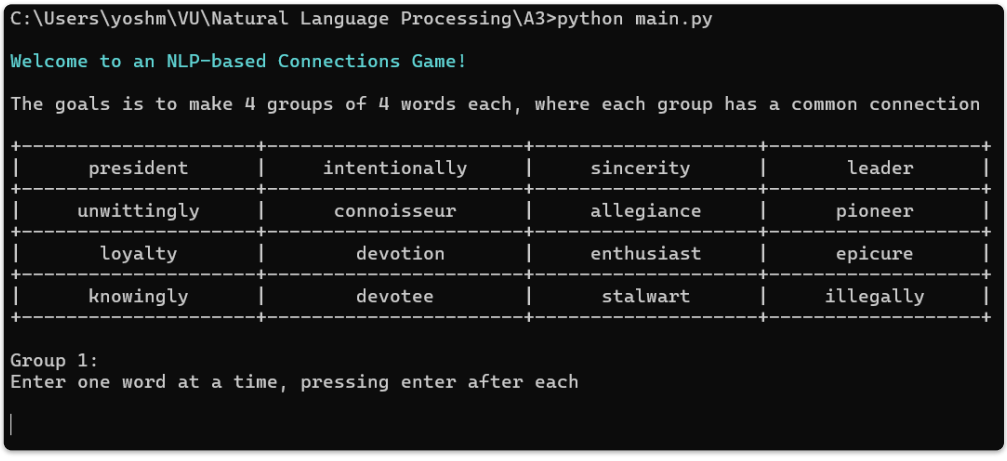
```bash
pip install -r requirements.txt
```

- Execute the Python script.

```bash
python main.py
```

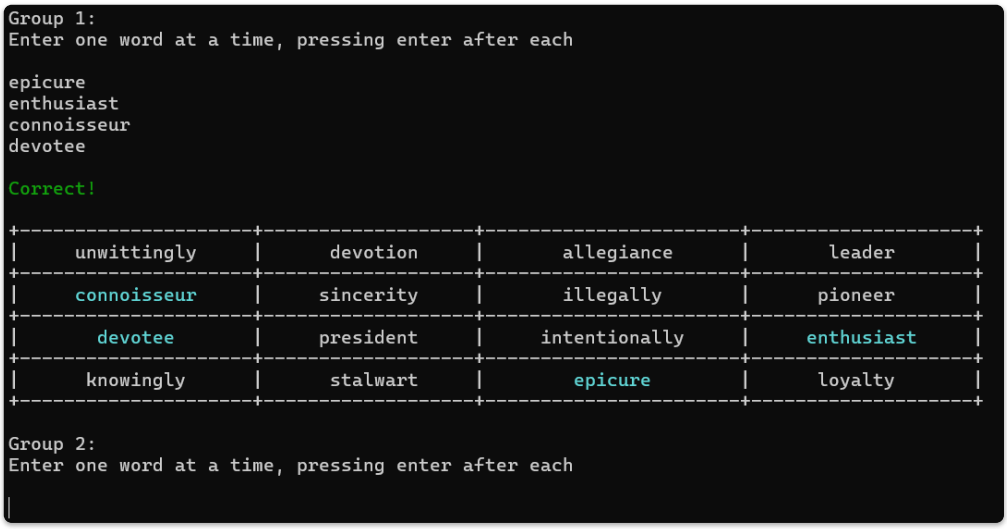
1.2 Gameplay

- Upon running the script, a set of connections will be generated and you will be to guess the connections between words in each group.

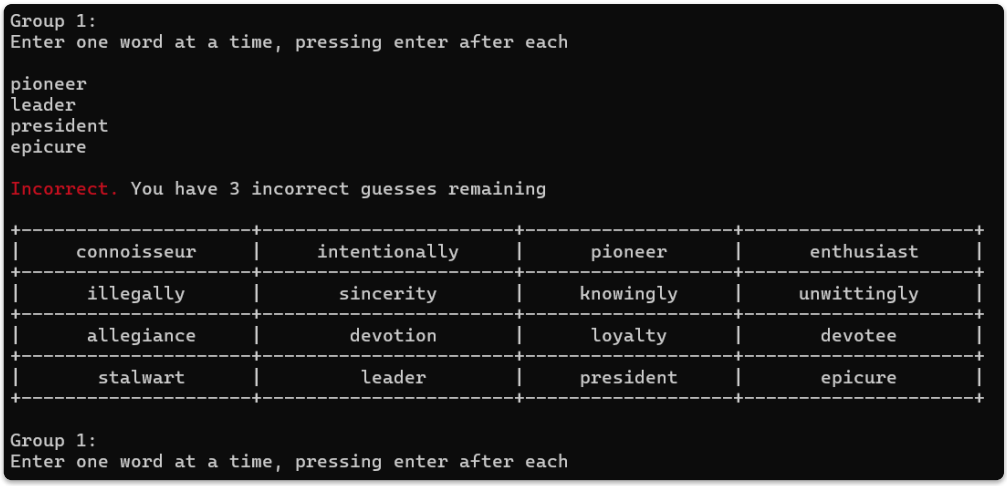


- Enter one word at a time for each group, pressing enter after each entry.

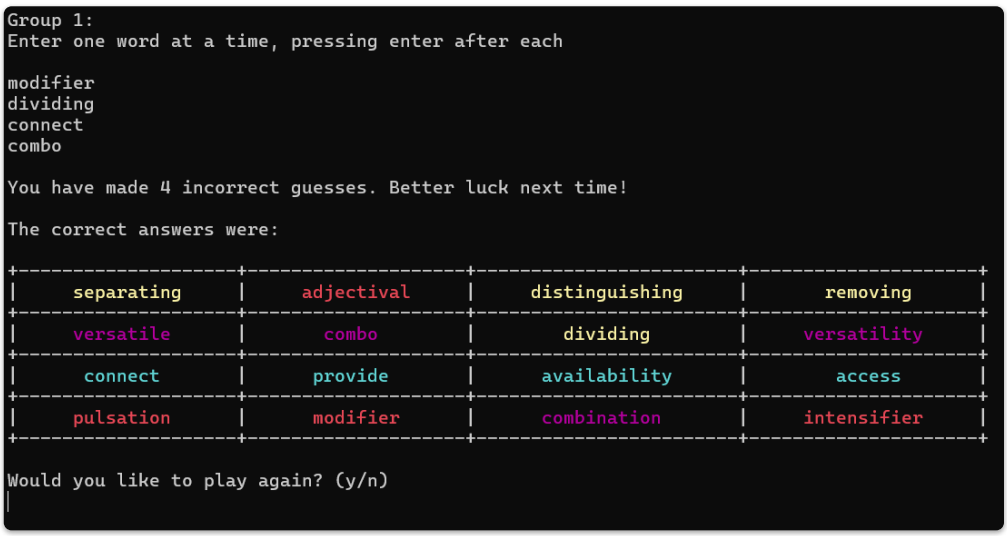
- If your guesses correctly form a group, you will be informed and the table will be reprinted with the given group highlighted



- If your guess is incorrect, you will be told so and prompted to have another go.



- You are allowed 4 incorrect guesses, after which the game will exit, displaying the correct answers and asking whether you would like to play again.



- If you correctly guess all four connections, you'll be congratulated and also given the option to play again.

2. How it Works

2.1 Overview

A pre-generated wordlist is used to pick random words, which are converted to word embeddings and similar words are found. Some filtering is done to make sure that these similar words aren't too similar or too different. This is done to create 4 groups of 4 connected words, which are then shuffled and presented to the player in a table. The player then has to decide through intuition how to categorise the 16 available words into the correct 4 groups.

2.2 Word Embeddings

A wordlist has been generated using the vocabulary from the Brown corpus, accessed through NLTK. This word list excludes most stopwords by simply dropping many of the most common words from the dataset. As the Brown corpus is nicely formatted and is comprised of "proper" grammar, we can drop most proper nouns and acronyms by simply checking if `word.islower()`.

```language-python
def get_words():
    """Extract words from Brown corpus to create wordlist and word
    embeddings"""

    vectors = {}
    lem = WordNetLemmatizer()
    wv = api.load('word2vec-google-news-300')
```

```
        # simple method to drop most proper nouns, acronyms, and
numbers
    words = [word for word in brown.words() if word.isalpha()
                          and word.islower()]

        # lemmatise all words which match our parameters
    wordlist = {lem.lemmatize(word) for (word, count) in
            Counter(words).items() if 1 < count < 300
                and len(word) > 2}

        # save vectors and wordlist
    with open('wordlist.txt', 'w') as f:
        for word in wordlist:
            try:
                vectors[word] = wv[word]
                f.write(word + '\n')
            except KeyError:
                pass

    save_word2vec_format('vectors.bin', vectors, 300)
```

Included in the repo is a `vectors.bin` file which matches Gensim's format for importing `KeyedVectors`. This set of vectors is reduced to just the vocabulary size extracted from the Brown wordlist - around 16,000 potential words for now. This makes it much faster to load and generate connections than using the full `word2vec-google-news-300` embeddings which were originally employed to generate the new reduced set of embeddings.

2.3 Connection Generation

For each of 4 groups, a word is randomly chosen from the wordlist and converted to a word embedding. A selection of the most similar words are chosen and each is checked for validity. These checks are meant to exclude different versions of the same root, while also making sure the chosen word is present in our wordlist, and hence, our word embeddings. These include:

- If the word is in the wordlist
- If the part of speech is the same as the original word
- If the Levenshtein distance between the chosen word and the original is less than 4

```language-python
def make_groups():
    """Generate 4 groups of 4 words each, with a common connection
    between words in each group"""

    # Load word vectors and word list
    wv = KeyedVectors.load_word2vec_format('vectors.bin',
            binary=True, unicode_errors='ignore')

    with open('wordlist.txt', 'r') as f:
            wordlist = f.read().splitlines()
                g = 0
        groups = []
```

```language-python
# Generate groups
while g < 4:
        choices = [random.choice(wordlist).strip()]
        wvs = wv.most_similar(choices[0], topn=30,
                restrict_vocab=16000)
```

```language-python
for word, _ in wvs:
        word = word.lower().strip()

# Check if new word is a valid choice
if (pos_tag([word])[0][1] == pos_tag([choices[0]])[0][1]) and
                lev(choices[0], word) > 4 and
                        word in wordlist and
                                word not in choices and
                                        word not in choices[0]:


        choices.append(word)

if len(choices) == 4:
        g += 1
        groups.append(choices)
        break
```

If a set of 4 connected words can be made, it is added to the list of groups and once we have 4, the set is shuffled and returned to the player's terminal in a nicely-printed table, designed to match the original style of the NYT Connections game.

```language-python
def generate_table(groups, guessed=[]):
    """Generate a table of words in groups, with guessed groups
```

```
    coloured in terminal output"""

    # Add colour to guessed groups
    for i, g in enumerate(groups):
        if g in guessed:
            for j, word in enumerate(g):
                g[j] = colour(word, i+1)

    # Merge groups and shuffle words
    words = [word for group in groups for word in group]
    random.shuffle(words)

    # Define table settings
    table = PrettyTable()
    table.header = False
    table.padding_width = 5

    # Add words to table
    for i in range(4):
        row = []
        for j in range(len(groups)):
            word_index = i + 4 * j
            row.append(words[word_index])
        table.add_row(row, divider=True)

    return table
```