Natural Language Processing

Corpus Collection and Data Extraction

Ramaseshan Ramachandran

WIKI COLLECTION

Wiki dumps are available at these locations for various languages

- Bengali
- Hindi
- Kannada
- Malayalam
- Marathi
- Tamil
- Telugu
- Gujarati
- English
- Odia

Sample code to extract text from wiki dump

```
from wiki_dump_reader import Cleaner, iterate

def write_corpus():
    corpus_file = 'corpus.txt'
    page_count = 0
    cleaner = Cleaner()
    with open(corpus_file, 'w', encoding='utf-8') as output:
        for title, text in iterate('wikidump.xml'):
            text = cleaner.clean_text(text)
            cleaned_text, links = cleaner.build_links(text)
            output.write(title + '\n' + cleaned_text + '\n')
            page_count += 1
            if page_count % 1000 == 0:
                print(f'Pages dumped = {page_count}', end='\r')

                print(title + '\n' + cleaned_text + '\n')
    print(f"\npage count = {page_count}")

#
```

For other languages, replace XX with the ISO code of the language of your choice:
https://dumps.wikimedia.org/XXwiki/latest/XXwiki-latest-pages-articles.xml.bz2

This very small corpus contains many budget speeches made by our Finance ministers

COVID-19 CORPUS

1. Research papers ≈5GB (when expanded)
2. ≈10GB COVID-19 dataset

```
def extract_all_text(filename:str)->str:
    file = open(filename)
    paper_content = json.load(file)
    body_text = ""
    abstract = ""
    title = ""

    #get the paper_id
    #paper_id = paper_content['paper_id']

    if 'title' in paper_content:
        title = paper_content['title']

    #get abstract.text
    if 'abstract' in paper_content:
        for abs in paper_content['abstract']:
            abstract = abstract + abs['text']

    if 'body_text' in paper_content:
        for bt in paper_content['body_text']:
        body_text = body_text + bt['text']
```

COMMON LAYERS OF NLP APPLICATIONS

- Preprocessing layer
- **Data extraction layer**
- Analysis of extracted information
- Semantic understanding
- Human/automatic evaluation of word meaning, sentence structure using the content obtained from the previous layers

WHAT ARE N-GRAMS?

- A contiguous sequence of n items from a given sample of text
- Items can be characters, words, or syllables.
- Common types:
  - Unigrams ($n = 1$): Single item
  - Bigrams ($n = 2$): Pairs of items
  - Trigrams ($n = 3$): Triplets of items.
  - ...and so on.
- Used in various NLP tasks:
  - Language modeling
  - Text classification
  - Spell checking
  - Sentiment analysis
  - Building co-occurrence windows for word embeddings (word2vec)
  - ...

N-GRAM EXAMPLE (WORD LEVEL)

- Text: "The quick brown fox jumps over the lazy dog."
- Unigrams:
  - "The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"
- Bigrams:
  - "The quick", "quick brown", "brown fox", "fox jumps", "jumps over", "over the", "the lazy", "lazy dog"
- Trigrams:
  - "The quick brown", "quick brown fox", "brown fox jumps", "fox jumps over", "jumps over the", "over the lazy", "the lazy dog"

N-GRAM EXAMPLE (CHARACTER LEVEL)

- Text: "hello"
- Character Unigrams: h, e, l, l, o
- Character Bigrams: he, el, ll, lo
- Character Trigrams: hel, ell, llo

N-GRAM EXTRACTION

```
import nltk
from nltk.util import ngrams

text = "The quick brown fox jumps over the lazy dog."
tokens = nltk.word_tokenize(text)

bigrams = list(ngrams(tokens, 2))
trigrams = list(ngrams(tokens, 3))

print("Bigrams:", bigrams)
print("Trigrams:", trigrams)
```

N-GRAM EXTRACTION IN PYTHON (SCIKIT-LEARN)

```
from sklearn.feature_extraction.text import CountVectorizer

text = ["The quick brown fox jumps over the lazy dog."]
vectorizer = CountVectorizer(ngram_range=(2, 2)) # Bigrams
X = vectorizer.fit_transform(text)
print(vectorizer.get_feature_names_out())

vectorizer = CountVectorizer(ngram_range=(3, 3)) # Trigrams
X = vectorizer.fit_transform(text)
print(vectorizer.get_feature_names_out())
```

APPLICATIONS: LANGUAGE MODELING

- Predict the next word in a sequence.
- Calculate the probability of a sentence.
- Used in:
  - Autocomplete
  - Machine translation
  - Speech recognition
- N-gram models are a simple form of statistical language models.

APPLICATIONS: TEXT CLASSIFICATION

- Use n-grams as features for classifiers.
- Capture word order and context.
- Useful for:
  - Sentiment analysis
  - Spam detection
  - Topic classification

CHALLENGES AND CONSIDERATIONS

- Data Sparsity: Higher n-grams are less frequent.
- Computational Cost: Larger n-grams increase memory and processing requirements.
- Smoothing Techniques: Used to handle unseen n-grams (e.g., add-one smoothing, Kneser-Ney smoothing).
- Choice of n: Depends on the task and data.
- Preprocessing of the text is critical.

COMMON LAYERS OF NLP APPLICATIONS

- Preprocessing layer
- Data extraction layer
- **Analysis of extracted information**
- Semantic understanding
- Human/automatic evaluation of word meaning, sentence structure using the content obtained from the previous layers

CO-OCCURRENCE MATRIX CREATION

```
import nltk
from nltk.util import ngrams
import numpy as np
import pandas as pd

def create_word_cooccurrence_matrix(text):
    tokens = nltk.word_tokenize(text.lower())
    bigrams = list(ngrams(tokens, 2))
    unique_words = sorted(list(set(tokens)))
    num_words = len(unique_words)
    # Initialize the cooccurrence matrix
    matrix = np.zeros((num_words, num_words), dtype=int)
    word_to_index = {word: i for i, word in enumerate(unique_words)}

    # Populate the matrix
    for bigram in bigrams:
        word1, word2 = bigram
        index1 = word_to_index[word1]
        index2 = word_to_index[word2]
        matrix[index1, index2] += 1

    # Create a pandas DataFrame for better visualization
    df = pd.DataFrame(matrix, index=unique_words, columns=unique_words)
    return df

if __name__ == '__main__':
    text = '''It is a guide to action which ensures that the military always obeys the commands of the party
     It is to insure the troops forever hearing the activity guidebook that party direct
     It is a guide to action that ensures that the military will for ever heed Party commands
     If many words and phrases are shared between the candidate and the reference translations,
     then it a good choice'''.lower()

    cooccurrence_matrix = create_word_cooccurrence_matrix(text)
```

ANALYSIS OF EXTRACTED INFORMATION - CO-OCCURRENCE MATRIX

	,	a	guide	guidebook	hearing	heed	to	translations	troops	which	will	words
a	0	0	2	0	0	0	0	0	0	0	0	0
action	0	0	0	0	0	0	0	0	0	1	0	0
activity	0	0	0	1	0	0	0	0	0	0	0	0
always	0	0	0	0	0	0	0	0	0	0	0	0
direct	0	0	0	0	0	0	0	0	0	0	0	0
ensures	0	0	0	0	0	0	0	0	0	0	0	0
ever	0	0	0	0	0	1	0	0	0	0	0	0
for	0	0	0	0	0	0	0	0	0	0	0	0
forever	0	0	0	0	1	0	0	0	0	0	0	0
good	0	0	0	0	0	0	0	0	0	0	0	0
guide	0	0	0	0	0	0	2	0	0	0	0	0
if	0	0	0	0	0	0	0	0	0	0	0	0
is	0	2	0	0	0	0	1	0	0	0	0	0
it	0	1	0	0	0	0	0	0	0	0	0	0
military	0	0	0	0	0	0	0	0	0	0	1	0
obeys	0	0	0	0	0	0	0	0	0	0	0	0
the	0	0	0	0	0	0	0	0	1	0	0	0
translations	1	0	0	0	0	0	0	0	0	0	0	0
troops	0	0	0	0	0	0	0	0	0	0	0	0

CONCLUSION

- N-grams are fundamental in NLP.
- Simple yet powerful for various tasks.
- Practical implementation is straightforward with libraries like nltk and scikit-learn.
- Understanding the trade-offs is crucial.