Contents

Part 1: Corpus processing (legal text): tokenization and word counting

Tokenization

```
"\b\w+(?:'\w+)?\b"
```

The tokenizer relies on a regular expression (ReGex) as shown above, to break down a large piece of text into a list of words, as known as “tokens”. Consider the expression syntactically:

- \b is a word boundary anchor. It asserts a position at the start or end of a word. It ensures that the match occurs at the beginning or end of a word.
- \w+ matches one or more-word characters. Word characters include small and capital alphabets (a-z, A-Z), digits (0-9), and underscores (_).
- (?:'\w+)? is a non-capturing group. It matches an optional sequence of a single quote followed by one or more-word characters. The “?” at the end makes this group optional, meaning it will match zero or one occurrence of the pattern inside the group.
- \b at the end of the expression is another word boundary anchor. It asserts a position at the end of a word.

In short, this ReGex is useful for tokenizing text into words, including contractions like "it's" or possessives like "example's".

Pre-processing

Lemmatization was originally considered, implemented and unit tested:

```
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

class Lemmatizer:
    def __init__(self):
        self.lemmatizer = WordNetLemmatizer()
        nltk.download('wordnet')
        nltk.download('averaged_perceptron_tagger')
        nltk.download('punkt', quiet=True)

    def get_wordnet_pos(self, word):
        """Map POS tag to first character lemmatize() accepts"""
        tag = nltk.pos_tag([word])[0][1][0].upper()
        tag_dict = {"J": wordnet.ADJ,
                    "N": wordnet.NOUN,
                    "V": wordnet.VERB,
                    "R": wordnet.ADV}
        return tag_dict.get(tag, wordnet.NOUN)

    def lemmatize(self, corpus):
        lemmatized_corpus = []
        for sentence in nltk.sent_tokenize(corpus):
            lemmatized_sentence = []
            for word in nltk.word_tokenize(sentence):
                lemmatized_word = self.lemmatizer.lemmatize(word, self.get_wordnet_pos(word))
```

```
            lemmatized_sentence.append(lemmatized_word)
        lemmatized_corpus.append(' '.join(lemmatized_sentence))
    return ' '.join(lemmatized_corpus)
```

However, due to the large corpus size, the above process is running very slow and thus not applied. This is because our tokenizer splits a piece of text into words. To optimize the performance, we could have split the corpus into sentences for easier lemmatization and batch processing of words.

Text concatenation is done by simply joining texts from multiple files with a space.

```
tokenizer.corpus = ' '.join(corpus)
```

Results

Sample Output from Word Tokenization (a)

```
1. CO
2. BRANDING
3. AND
4. ADVERTISING
5. AGREEMENT
6. THIS
7. CO
8. BRANDING
9. AND
10. ADVERTISING
11. AGREEMENT
12. the
13. Agreement
14. is
15. made
16. as
17. of
18. June
19. 21
20. 1999
```

Sample Output of Token Frequencies (c)

```
1. the: 239999
2. of: 151815
3. and: 128998
4. to: 127311
5. or: 106443
6. in: 74269
7. any: 58853
8. shall: 48424
9. a: 46609
10. by: 42050
11. be: 39165
12. Agreement: 37020
13. for: 35480
14. this: 35216
15. such: 34815
16. with: 32574
17. as: 31637
18. that: 27281
19. other: 25063
20. 1: 23056
```

Analytical Results

# of tokens (b)	4087261	
# of types (b)	45597	
type/token ratio (b)	0.011155881652774315	
tokens appeared only once (d)	15297	
# of words (excluding punctuation) (e)	4087261	
type/token ratio (excluding punctuation) (e)	0.8345288397539471	
List the top 3 most frequent words and their frequencies (e)	**Word**	**Frequency**
	'the'	239999 occurrences
	'of'	151815 occurrences
	'and'	128998 occurrences
type/token ratio (excluding punctuation and stopwords) (f)	0.5398918738000827	
List the top 3 most frequent words and their frequencies (excluding stopwords) (f)	**Word**	**Frequency**
	'Agreement'	37020 occurrences
	'1'	23056 occurrences
	'Party'	19216 occurrences
List the top 3 most frequent bigrams and their frequencies (g)	**Bigram**	**Frequency**
	'Confidential Information'	2869 occurrences
	'written notice'	2369 occurrences
	'Effective Date'	2264 occurrences

Discussion of Results

First, it is observed the **same number of tokens regardless of punctuations**. With an effective ReGex, it is confident that our word tokenizer which splits an entire corpus text into individual words effectively splits based on punctuations as well.

Second, it is observed the **type/token ratio ≈ 0.8 when only punctuations are disregarded**, whereas the type/token ratio ≈ 0.01 in the original corpus. The ratio improved a lot. This means most words in the corpus after disregarding punctuations become unique.

Third, by excluding stopwords or by applying the bigram analysis method, **only meaningful words remain in the corpus**. From the original corpus (as well as the filtered corpus which excludes only punctuations), the top 3 most frequent words found in the corpus are "the", "of", and "and", which are connectives or prepositions without a contextual meaning. From the filtered corpus which excludes stopword, the top 3 most frequent words found in the corpus are "Agreement", "1" and "Party". Words in those filtered corpora have an actual contextual meaning. Keeping only those types of words is beneficial to our analysis because in the domain of natural language processing, we usually care only about the context and the meaning of a particular text, while words without an actual definition just provides noises in the corpus. To further enhance the analysis, lemmatization could have been applied such that words with the same base form are considered the same word. In such an enhancement, we could effectively find actual frequent words which have different meaning.

Last but not least, bigram analysis gives the most meaningful words, in the form of a phrase. Comparatively, the original corpus contains noises which are meaningless words. The filtered corpus without particular stopwords extracts only single words, which may also contain numbers or digits like “1”. Meanwhile, the top 3 most frequent bigrams (i.e., pairs of consecutive words) are ‘Confidential Information’, ‘written notice’, and ‘Effective Date’. These are commonly used phrases. Concluding the corpus with words in those phrases makes the results of the analysis more trustable. Like we tend to believe the accuracy of how the bigram analysis tells us about the corpus. Therefore, bigram analysis is the best method comparatively.

Part 2: Evaluation of Pre-trained Sentence Embedding Models

Introduction

In this evaluation, we examined five pre-trained sentence embedding models to evaluate how well they predict the semantic similarity of sentence pairs. The models were tested using the Semeval 2016 Task 1 Semantic Textual Similarity (STS) Core dataset. The main evaluation metric was the Pearson correlation between the similarity scores produced by the models and the gold-standard labels included in the dataset.

Methodology

The dataset for sentence similarity evaluation was derived from the Semeval 2016-Task1 Semantic Textual Similarity (STS) dataset. The raw data files (STS2016.input.answer-answer.txt,STS2016.input.headlines.txt,STS2016.input.plagiarism.txt,STS2016.input.postediting.txt, STS2016.input.question-question.txt) were concatenated to form a unified test dataset. Stopwords were retained during preprocessing to ensure compatibility with pre-trained sentence embedding models.

Model Selection

The following pre-trained sentence embedding models were selected:

1. **SBERT (Sentence-BERT):** A model fine-tuned for semantic textual similarity tasks.
2. **Universal Sentence Encoder (USE):** A lightweight and efficient model designed for sentence-level tasks.
3. **GloVe with averaged word embeddings:** Pre-trained word vectors aggregated to represent sentences.
4. **FastText-based sentence embeddings:** Word embeddings aggregated from subword information.
5. **Hugging Face Model**: text-embedding-ada-002 (OpenAI's Embedding Model)

- Source: Integrated via Hugging Face's OpenAI API compatibility.
- Mechanism: This model provides dense sentence embeddings designed to capture semantic meanings. It is based on OpenAI's large language models and optimized for text similarity tasks.
- Parameters:
  Embedding dimension: 1536
  Token limit: 8192 tokens per input

Experimental Procedure

1. **Loading Data:** The preprocessed dataset was loaded into a Jupyter Notebook for experimentation.
2. **Embedding Sentences:** Sentences from the test dataset were encoded using the above pre-trained models.
3. **Similarity Computation:** Pairwise cosine similarity scores were calculated between sentence pairs in the dataset.
4. **Evaluation:** The Pearson correlation between the model-generated scores and the gold standard scores provided in the dataset was computed to evaluate model performance.
5. **Output Storage:** Results were exported to the Part 2 - Output/ directory for further analysis.

Results

Sentence Similarities

Dataset	SBERT	USE	GloVe	FastTest	HuggingFace	Best Score
STS Test Data	-0.0574	-0.0244	-0.0435	-0.0631	-0.0438	-0.0244

Challenges and Observations

During the evaluation of sentence embeddings using the specified models and the provided STS dataset, I observed that the Pearson correlation coefficients between the normalized similarities and the gold labels were negative for some models. Despite extensive validation and debugging of the code, the data preprocessing steps, and the integrity of the dataset itself, the negative correlation persists.

Several measures were taken to verify the correctness of the results:

1. **Data Validation**: Ensured that there were no missing values or misaligned entries in the dataset. All sentence pairs and gold labels were correctly aligned.
2. **Code Debugging**: Reviewed the data processing code to confirm that the normalized similarity scores were calculated correctly for each model.
3. **Statistical Distribution**: Examined the distribution of both the gold labels and the normalized similarity scores to ensure sufficient variance in the data.
4. **Alternative Testing**: Tested the Pearson correlation calculation with subsets of the data to rule out errors caused by outliers or specific data ranges.

Despite these validations, the Pearson correlation for some models remained negative, suggesting that the sentence embeddings produced by these models may not align with the human-annotated

similarity scores in the dataset. This finding could indicate that the models used are not well-suited for this specific dataset or task.

While negative Pearson correlations are unusual in this context, they accurately reflect the statistical relationship observed in the data and the embeddings generated by the models.