Features of n-gram algorithm and its application in sentence/words generation

COMP40020 Assignment 2

Yang Yinfang

22204768

University College Dublin

Similarity between words

A natural language model is built in this report based on the previous corpus and n-gram algorithm. With Word2Vec, words with similar meanings will be mapped to similar positions in the vector space. In Task 1, we will use this feature of word2Vec to observe the similarity between words in our corpus. This technique has the potential to yield valuable insights into the structure and meaning of language, which can have a wide range of applications.

A pre-trained Word2Vec model is adopt to find the most similar words for ten words selected from the corpus I created in Assignment 1, i.e., *design.txt*. The ten words are: design, color, cover, arrow, digital, architecture, poster, style, brick, and administration. The outcomes are interesting and provide contextual perspectives on the given corpus. Take for example the words 'architecture', 'brick' and 'poster'. The top ten similar words for "architecture" are: *architectural, design, architectures, architect, designs, architects*, *Architect, urbanism, Borromini,* and *Design.* This output indicates that "architecture" is closely related to the fields of design and construction, as well as related disciplines such as urbanism and urban planning. Such relationships are intuitive, indicating that the Word2Vec model is able to capture the semantic relationships among words.

The top ten similar words for "poster" are: *posters, Poster, billboard, banner, flier, flyer, banners, photograph, picture,* and *mural*. Together, these outcomes suggest that "poster" is closely related to other forms of advertising and promotional materials, such as billboards, banners, flyers and leaflets. Once again, these relationships are expected and intuitive, and demonstrate the ability of the Word2Vec model to capture semantic relationships between words.

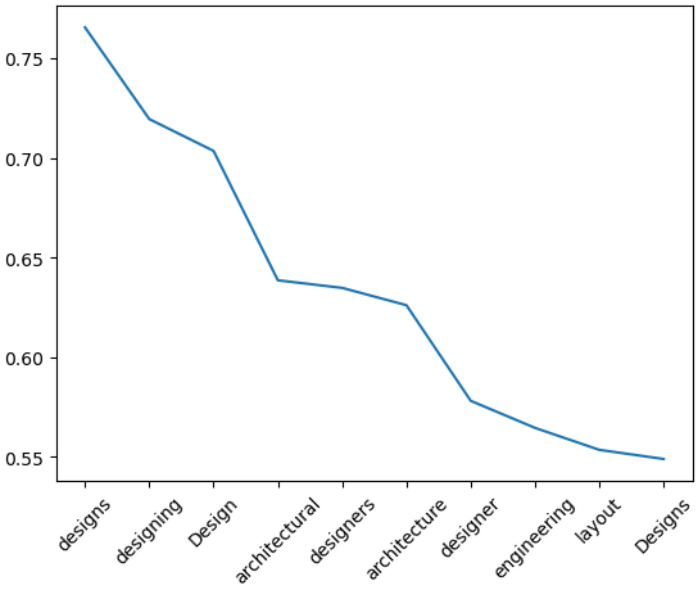

Fig. 1 The similarity to the word "design"

The most similar words for "brick" are *bricks, stucco, masonry, fieldstone, stone, building, tile, facade, granite,* and *Bricks*, all of which are related to various building materials that are interoperable building materials, and it is not surprising that the classifications are very similar. These semantic relationships are valuable insights in the fields of architecture, engineering and construction.

In summary, the Word2Vec model returns a list of the most similar words for each given word in the corpus. It has demonstrated the utility of the pre-trained Word2Vec model in capturing the semantic relationships between words in a given corpus. Through analysis, it is shown that this model can effectively identify words with common classifications or attributes, and that these words can have semantic relationships in a variety of domains.

These examples show that in Word2Vec a word can be defined by the words around it, and that the semantics of a word can be inferred from the information represented by the words in its context. A pre-trained Word2Vec model can effectively capture meaningful semantic relationships between words. This is a way for computers to understand natural language, including understanding semantics as well as contextual relationships. Central and contextual words predict each other. For a given input word, it provides a good connection to related words that can be used

for tasks such as natural language understanding or search engine/information retrieval. This leads to a more nuanced understanding of natural language and the contextual relationships between words.

However, it is important to note that Word2Vec similar words also capture a lot of useless information, such as plural forms of the same words. This may make it difficult to identify truly meaningful semantic relationships between words and may require additional filtering or processing to remove noise from the data.

Generating sentences by n-gram

In this part, the corpus *design.txt* is used to build the word-based model. The data is first read and tokenized. The punctuations are removed and separated. The type of the input data used to generate the model is a list of lists (2-d list), each sublist is a sentence containing all the words assigned to that sentence.

N-gram algorithm is used to generate the model based on the above data. Here the parameter n is set to 3 (*i.e.*, 3-gram). Maximum Likelihood Estimator is used estimate the probability of 3-gram model. The model is trained and the "sentences" can be generated based on that model. For example, if the text_seed *tell* is adopted, the sentence generated is "tell you these things I can buy". The sentence composed of 5 words is "looks like 911 is this".

There are two features one can get from this sentence generator. First, the sentence as a whole does not make much sense. One reason that the generated sentences do not make much sense is that the corpus *design.txt* is taken from reddit forum. Compared with scientific papers and novels, the sentences here are informal and not very coherent. Hence, it is important to take into account the nature of the corpus used when evaluating the quality of the sentences generated. Second, each word is closely related to one or two words next to it. Adjacent words form meaningful language

segments, like “tell you”, “I can buy”, “looks like” *etc.* This is one of the most important features of n-gram algorithm, i.e., an n-gram model predicts the word x_i based on the previous n words. It is also important to take an appropriate value for n, which is crucial for building an effective model. the value of n affects the range of words captured by the model, which also determines the model's inability to capture more complex or distant relationships.

Generating words by n-gram

In this part, we use the novel *Dracula* as the corpus. All the words which initiated with the letter ‘Y’ are collected and stored for later use. The reason we use *Dracula* instead of *design.txt* is that the there are less than 5 words that begin with Y for *design.txt*. *Dracula* has a richer vocabulary thus better to for the word-based model training.

We did exactly the same cleaning procedure to the previous section like tokenization, punctuation removal and separation of words. We use an iteration to pickup all the words begin with Y, and bundled them into a list which can be further used as the input for the padded_everygram_pipelin function. The model is optimized by MLE algorithm. Here we use ‘y’ as the text_seed argument for the function lm.generate, that is to say, we will generate some words starting with y. We limit the length of the word to 5 and iterate 10 times. The Dummy characters like </s> and <s> are removed to keep the generated words clean.

The ten words generated just now are *yes, you, yard, you, yard, you, you, you, you, your*. We find all the generated words are within the bounds of the training data (here is the list ‘words’ in our Jupyter code). In other words, we are not *creating* new vocabulary. This is different from the sentence generator shown in the previous section. Each training ***sentence*** data set in the previous section is unique, while here there are some duplicated words in the ***word*** data set. Hence in the

later case, the connection between the elements (here the letters) are strongly coupled, and it is less likely to create some new combinations. For example, the combination y+m is almost impossible.

We also find that the word '*you'* occurs most frequently, which is same to the training data. This indicates that the generated words by n-gram algorithm are highly affected by the training data.

Some limitations of N-gram algorithm

One limitation is that they are highly dependent on the input data. For instance, I replaced the corpus in task3. The forum corpus design.txt was not able to generate meaningful enough results. After switching to the book corpus, it still did not produce satisfactory results because the words associated with the initial Y of my name were not abundant. If the input data is not diverse or representative enough, the language model may not be able to generate high-quality sentences, which means that n-gram language model relies heavily on the source database or the corpus used to train the model. The quality and diversity of the data directly influence the model's ability to generate coherent and meaningful sentences. Additionally, the seed or the initial text provided to the model for generating sentences also plays a crucial role in determining the quality of the output. If the input data or seed text is not representative of the target domain or is too limited, the generated sentences may be irrelevant or incoherent.

In order to set a suitable seed text for the sentences generated using the language model, we should choose a sequence of words that represents the style and content of the corpus used. Ideally, seed_text should be a complete sentence or a single meaningful fragment, and it should contain some unique or salient features of the corpus. For example, the corpus I am using here is a corpus of design reviews, and a good seed text might be something like "this design is". In general, we should choose a seed text that reflects the patterns and themes that are prominent in the corpus.

Another limitation is that when generating longer sentences, the model may produce sentences that lack coherence or have grammatical errors. This limitation arises from the fact that the model can only generate sequences based on the probability of occurrence of n-grams in the input data. The model may not have the ability to capture the complex syntactic or semantic relationships between words in a sentence, resulting in generated sentences that lack coherence or have syntactic errors. This is especially true when generating longer sentences, where the likelihood of generating incorrect or incoherent sentences increases. A similar limitation is the degree of understanding based on the n-gram, which only considers n-word contexts, which limits it to understanding longer distance contexts, especially when n is low.

word count: 1657