NLP Practice in Medical Texts

Jiawei Xiong

August 10, 2022

Abstract

At present, NLP is widely used in practice, especially in the biomedical field. Their combined use has continuously raised the level of research in the field of natural science and stimulated the desire of human exploration. Therefore, it is essential for researchers to better understand and use the NLP model. The purpose of this project is to analyze the data provided on the basis of different NLP models. In the research process, we first extracted the text from the data, and marked the text with NLTK, SciSpaCy and BPE methods. Then, N-gram, Skip-gram and MLM models were used to establish Word presentation, so that we could better understand the relationship between biological terms. Finally, we use T-SNE to visualize the matrix, which more clearly reflects the relationship between words and improves the readability of the data. According to the experimental results, we have explored a variety of NLP functions that can be used in medical texts in the process of research, and found that the model has good performance in practice. Finally, we believe that this model still needs a long time to test and improve, and we believe that more NLP models need to be put into the medical field.

1 Introduction

CORD-19 is a corpus of academic papers about COVID-19 and related coronavirus research. It's curated and maintained by the Semantic Scholar team at the Allen Institute for AI to support text mining and NLP research. The final release of this dataset was in 2022-06-02 so it contains latest data about COVID-19. On March 16, 2020, the Allen Institute for AI (AI2), in collaboration with our partners at The White House Office of Science and Technology Policy (OSTP), the National Library of Medicine (NLM), the Chan Zuckerburg Initiative (CZI), Microsoft Research, and Kaggle, coordinated by Georgetown University's Center for Security and Emerging Technology (CSET), released the first version of CORD-19. This resource is a large and growing collection of publications and preprints on COVID19 and related historical coronaviruses such as SARS and MERS.

CORD-19 sourced from PubMed Central (PMC), PubMed, the World Health Organization's Covid-19 Database,4 and preprint servers bioRxiv, medRxiv, and arXiv. They processed the metadata by clustering papers using paper identifiers, selecting canonical metadata for each cluster, filtering clusters to remove unwanted entries. And then processed the full text using PDF parsing pipeline creating two sets of full text JSON parses associated with the papers in the collection, one set originating from PDFs (available from more sources), and one set originating from JATS XML (available only for PMC papers).

In summary, this project offers a paradigm of how the community can use machine learning to advance scientific research.

2 Parse the Data

Unzip the zip file and using "json" and "jsonpath" module to parse data from 425k json files.

Using the function jsonpath.jsonpath() to get the title and text in json files, then using the os.writelines() to write filtered data in txt for next part.

3 Tokenization

Tokenization is a very important step in NLP technology, and it is a necessary step before data can be entered into the model for calculation. For English and other Latin language word size Tokenization

is very easy, we can directly follow the space can be cut out naturally. The following are three ways to tokenization the source text:

3.1 split()

The split method is one of the most basic functions in the Python library. This method can split a string into a list of strings after breaking a given string by the specified separator. After the preprocess, now it is convenient for us to use

```
file.read().split()
```

to get a list of English words directly.

3.2 NLTK

The second method we use NLTK. NLTK stands for Natural Language ToolKit. NLTK is a Python library used for human natural language processing. The biggest advantage of NLTK is that it provides a programmatical interface to over 100 lexical resources and corpora. This means that from within your python program, you can use those corpora.

```
from nltk.tokenize import word_tokenize
for text in file.readlines():
  a=word_tokenize(text)
  b.append(a)
```

then we get word list "b".

3.3 ScispaCy

ScispaCy is an NLP library dedicated to the processing of biomedical texts that builds on the strength of the spatial library. Due to the specialty of SPACE in biomedical field, there are a large number of biomedical words in en_core_sci_sm model. Therefore, the direct use of the en_core_sci_sm model is a good way to label the words of biomedical articles.

But we just learn general concept of ScispaCy, and we did not use it in our project.

3.4 BPE

The the former tokenization methods are not conducive to the model learning the relationship between affixes. For example, The relations learned by the model between "old", "older", and "oldest" could not be generalized to the "smart", "smarter", and "Penelope" of the models. Also, the former methods give too many different tokens, e.g. "dog" and "dogs" are different words. However, BPE uses subwords to control the number of the tokens, which makes next step more convenient.

In the previous algorithm, we have obtained the word list, then we append "¡/w¿" to each word and add " " to split each word. For example, if the frequency of "low" is 5, then we rewrite it as "L ow ¡/ w¿" : 5

```
def get_vocab(filename):
vocab = collections.defaultdict(int)
with open(filename, 'r', encoding='utf-8') as fhand:
    for line in fhand:
        words = line.strip().split()
        for word in words:
            vocab[' '.join(list(word)) + ' </w>'] += 1
return vocab
```

The frequency of each consecutive byte pair is counted, and the one with the highest frequency is selected and merged into a new subword. Repeat above step until it reached the subword list size or the next highest frequency byte pair occurs at frequency 1.

```
    num_merges = 1000
for i in range(num_merges):
    pairs = get_stats(bpe)
    if not pairs:
        break
    best = max(pairs, key=pairs.get)
    bpe = merge_vocab(best, bpe)
    tokens = get_tokens(bpe)
```

In the end, we put the token into a list and save it to file. It look like this. In former method we get 308637 tokens, and this time we cut it into 1127 tokens.

Figure 1: skip-gram model

3.5 The advantages and disadvantages

Let's make a brief summary of the 4 methods. The *split()* ,NLTK and ScispaCy belong to word granularity tokenization, which is consistent with the natural segmentation that we humans do when we read. While BPE belongs to sub-word granularity tokenization.

For word granularity tokenization, the advantages is like humans to read. On the one hand, it can keep the boundary information of words well, on the other hand, it can keep the meaning of words well.

But the disadvantage is obvious, too. For the method of word granularity, the dictionary needed to be constructed is too large, which seriously affects the computational efficiency and consumes memory. Second, even if using this large dictionary does not affect efficiency, it still causes OOV(out of the vocabulary) problems. Because human language is constantly developing, the vocabulary is also increasing in development. For example: needle doesn't poke, Niubility, Sixology, etc. Moreover, different forms of a word can produce different words, such as "looks" and "looking," derived from "look," but with similar meanings, it is not necessary to train them all.

For subword granularity tokenization, like BPE, the above problems are handled. To sum up, the word list should be as small as possible, while covering the vast majority of words as little as possible OOV words. In addition, each token in this list has a certain meaning, that is, for a word cut out of the subwords, each one is meaningful. This saves memory and preserves semantics

4 Build Word Representations

4.1 Use N-gram Language Modeling

4.1.1 what is N-gram

In order to obtain the word vectors in biomedical papers, we first adopt the N-gram model, which is the simplest and most basic word representation method we have mentioned. Since the term "n-gram" refers to a sequence of consecutive words, an n-gram represents a word by how often it occurs together with other words in the n-gram range.

N-gram embedding models can be generated in a variety of ways. The traditional method is to count the frequency of co-occurrence words: $P(w_i|w_{i-1}, w_{i-2}, ..., w_{i-n+1})$ and construct embedding vectors according to the frequency of co-occurrence words.For example, in the sentence of "I like deep learning and I like machine learning!", the count of "I like" is 2 and so the frequency of this co-occurring words is 2.However,most of the co-occurring words only appears once even do not exist in the dataset so the final Matrix is sort of sparse.

4.1.2 How to train our data?

In the next step, we build a neural network with the Pytorch package to construct a more reliable word embedding based on the n-gram model.

To train our data,first we use the Pytorch package to build our N-gram model and construct a more reliable word embedding based on the n-gram model:

```
ngrams = [
    (
        [words[i - j - 1] for j in range(CONTEXT_SIZE)],
        words[i]
    )
    for i in range(CONTEXT_SIZE, len(words))
]

class NGramLanguageModeler(nn.Module):

    def __init__(self, vocab_size, embedding_dim, context_size):
        super(NGramLanguageModeler, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.linear1 = nn.Linear(context_size * embedding_dim, 256)
        self.linear2 = nn.Linear(256, vocab_size)

    def forward(self, inputs):
        embeds = self.embeddings(inputs).view((1, -1))
        out = F.relu(self.linear1(embeds))
        out = self.linear2(out)
        log_probs = F.log_softmax(out, dim=1)
        return log_probs
```

Then,we use the neural network to train the data on the gpu for below 5 steps:

```
    # Step 1. Prepare the inputs to be passed to the model
    context_idxs = torch.tensor([word_to_ix[w] for w in context], dtype=torch.long)
    context_idxs = context_idxs.to(device)

    # Step 2. Recall that torch *accumulates* gradients
    model.zero_grad()

    # Step 3. Run the forward pass, getting log probabilities over next words
    log_probs = model(context_idxs)

    # Step 4. Compute your loss function
    loss = loss_function(log_probs, torch.tensor([word_to_ix[target]], dtype=torch.
    long).to(device))
    loss = loss.to(device)

    # Step 5. Do the backward pass and update the gradient
    loss.backward()
    optimizer.step()

    # Get the Python number from a 1-element Tensor by calling tensor.item()
    total_loss += loss.item()
    losses.append(total_loss)
```

The neural network is built on 100 sample texts, with the gram size set to be 2,the dimension of word vectors to be 150 and the learning rate to be 0.01. After 20 epochs, the loss of the neural network falls from 1106327.5956125814 to 304868.8174783171.Though the final loss is still too large but it do decrease and because of the poor computing capability of computer added with insufficient time we didn't make the loss smaller.

4.2 Use Skip-gram with Negative Sampling

4.2.1 what is Skip-gram

Skip grammar models can be used to classify words in the same sentence. Specifically, we use a central word to predict its contextin condition possibility: $P(w_{c-m}, ...w_{c-1}, w_{c+1}, ..., w_{c+m}|w_c)$. The model is shown in the figure. The central word is in the input layer, and the hidden layer matrix is

```
tensor([-0.8510, -0.1359, -0.7707, -0.7427, -0.0471,  0.6664, -1.4849,  0.4169,
        -0.2116,  0.3628, -1.3500, -0.7138,  1.0303, -0.3507, -2.6210, -0.2662,
        -0.0844,  0.8304, -1.1483,  1.3839,  0.0807,  0.5981, -0.8081, -0.1998,
        -0.7374,  0.2797, -0.3512,  1.0412, -0.1756, -0.0588,  0.9035, -0.3060,
        -0.3872, -0.1782, -0.0584,  1.0126,  0.0321,  1.3045,  0.6667, -0.2805,
         0.7046,  1.2902, -0.1708,  2.4073,  0.8876,  0.3608, -1.8774, -0.2549,
         0.1937, -2.5291, -0.5177,  0.3115, -0.3948, -0.6315, -0.4802,  1.7012,
        -1.1637, -0.2635,  1.0025,  0.7738, -1.4146,  0.3690,  1.4632, -0.2564,
        -0.3267,  0.7538, -0.4850, -0.3144,  0.1895,  0.1726,  1.0245, -0.5556,
         0.0796, -0.6657, -0.2091,  1.0561,  0.0654, -1.0643,  0.6178, -0.6142,
         0.0942, -1.1601, -0.0921,  1.0310,  0.2709,  0.3443, -0.4494, -1.0616,
        -2.1098, -1.4405,  0.0729, -0.5273, -0.8806,  0.5944,  0.0952,  0.6025,
         0.0646, -0.7821,  0.7528,  0.3759, -0.0143, -1.6883,  1.7341,  0.4814,
        -0.1689,  1.5591, -0.4909, -2.0122, -1.8928,  0.6904, -0.2652,  0.1471,
        -0.8230,  0.4079, -0.6071, -0.8101,  0.7550,  0.1732, -0.2964, -1.0095,
         0.4790, -0.3106, -0.7892, -0.1879, -0.4557, -1.5149,  0.0964,  0.1721,
        -0.9636,  0.2076, -0.5320, -0.4735,  1.3955, -2.0571, -0.5601, -0.8534,
        -0.9998, -2.3204,  0.2660, -0.1542,  0.3898, -0.7445, -0.4968,  0.9975,
         0.6732, -0.2138,  0.8193, -0.2253, -0.1970,  0.4206], device='cuda:0',
```

Figure 2: result

obtained by multiplying a hidden weight matrix W. In the output layer, multiple multinomial matrices are computed using the same hidden_to_output weight matrix W (note that the input_to_output weight matrix is different from the hidden_to_output weight matrix).

For skip-gram model, calculating and updating the weight matrix is the key problem. Similar to the implementation of the traditional neural network, the performance of the model is evaluated by adjusting the loss function between the output layers, and the parameters in the weight matrix are adjusted by applying the specified learning rate, so the real predicted value and weight matrix can be updated and adjusted.

The skip-model use softmax, a log-linear classification model, to obtain the posterior distribution of words, which is a multinomial distribution, to ensure the probability of each dimension can be sum to one.

$$y_j = softmax(u_j) = \frac{exp(u_j)}{\sum_{j=1}^{V} exp(u_j)}$$

where u_j stand for the similarity for each words.

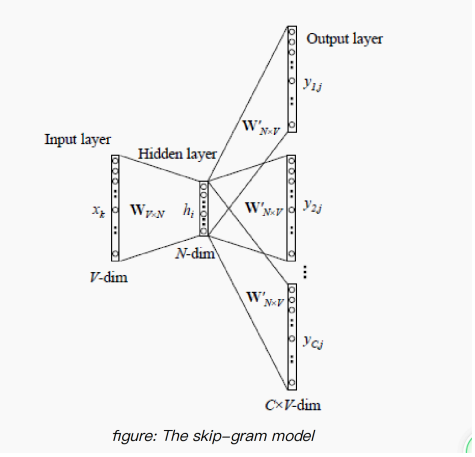


Figure 3: skip-gram model

4.2.2 How to train our data?

First, construct a dictionary, count the frequency of each word, and convert each word to an integer ID based on the frequency. Then convert the text into ID, which helps model to handle. And next, we construct the data and set max_window_size to 3. The program scans the entire corpus from left to right according to Max window size.

Data is ready, We use *paddlepaddle* to build the neural network. We constructed Mini-batch, and we train the model, put different kinds of data into different kinds of tensor for the neural network to deal with, and construct different kinds of tensor using the array function of Numpy, and then train them into the neural network(the network structure is clear in our code, so we are not going to say too much here.) We set the *batch_size=256, embedding_size=200, learning_rate=0.0002*

4.2.3 Optimization

First, in this project, negative sampling is introduced. In negative sampling, the loss can be approximated from the softmax layer by only updating a small subset of all the weights at once. In negative sampling, the method is used to update the weights for the correct example, but only a small number of incorrect examples. Also, we use a modified loss function where we only care about the true example and a small subset of incorrect examples.

$$E = -log(\sigma(u_t^T v_c)) - \sum(log(\sigma((u_k)^T v_c)))$$

where is the u_t vector embedding of positive sample from context, u_k are the vector embeddings of negative samples, V_c is the vector embedding of the central word, σrefers to the sigmoid function.

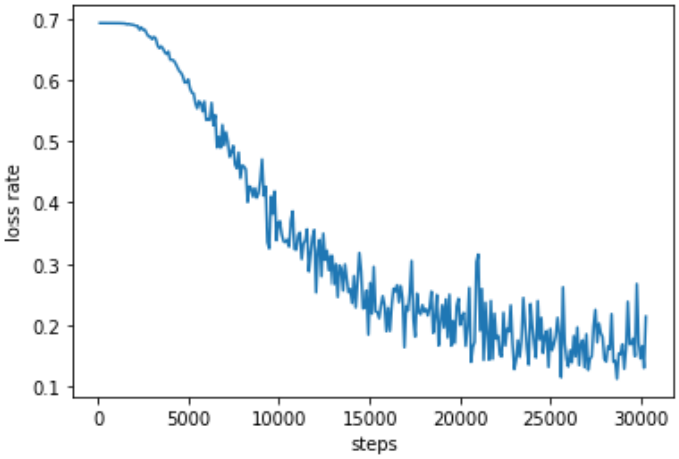
Another optimization is using subsampling. In the sampling process, some common words are discarded with a certain probability (too common may only indicate modification, but not specific meaning). The word "the", for example, usually has no effect on what we really want to express but occurs a lot. In this case, we can delete it in the sampling.

$$P(w_i) = (\sqrt{\frac{z(w_i)}{0.001}} + 1) \times \frac{0.001}{z(w_i)}$$

Z (wi): the probability that a word appears in the thesaurus, where wi represents a specific word. 0.001 is the value of parameter sample, which controls the degree of subsampling. 0.001 is the value of general settings. The smaller the sample value, the greater the probability of throwing away words

4.2.4 Result

During the training, We can see a steady decline in the loss rate, which means the training works. And we get a embedding matrix for each word. Also, we can use the inner product of two word matrices to see how similar two words are.



loss rate in 30000 steps

```
The shape of matrix (308637, 200)
Each word matrix is like this:
[-0.18947142  0.09229063 -0.00893008  0.10956005 -0.07930114  0.09757648
  0.08177182  0.16020983  0.21913262  0.1730393   0.10774501 -0.12106024
  0.10627557 -0.15698189 -0.07507355  0.04720854 -0.21621738 -0.08814045
  0.1938731   0.13763106  0.16293682  0.00602511 -0.19595166 -0.05220388
 -0.11653194 -0.11894817  0.12377495  0.03462756  0.19205248 -0.10328148
  0.22269039 -0.06005666 -0.01413714  0.15967815 -0.02119645 -0.14629708
  0.09175205  0.10705189  0.14249326 -0.19424689 -0.0937048   0.22999078
 -0.02023458  0.02513173 -0.23077449  0.0935494   0.16783723  0.05467445
  0.07258544  0.13666178  0.16073073  0.19265023 -0.13528003  0.02263115
  0.07156504 -0.06115059  0.08173417  0.18331586  0.1241238  -0.02353617
 -0.15460601  0.09289493  0.22547014  0.16470698  0.13796286 -0.11902513
  0.01819592 -0.16365427  0.0938042   0.10539988 -0.09137388  0.22280188
```

emmbeding matrix sample

```
get_cos("dog","dogs",a)
get_cos('topic','theme',a)
get_cos('covid','medicine',a)
```

```
单词1 dog 和单词2 dogs 的cos结果为 0.935383
单词1 topic 和单词2 theme 的cos结果为 0.991170
单词1 covid 和单词2 medicine 的cos结果为 0.944105
```

similarity of some words

Figure 4: result of skip-gram

4.3 Masked Language Model

MLM involves masking some of the words in the input prediction during training and then predicting the words from the context. Just as traditional language model algorithms match RNN, this feature of MLM matches well with the structure of Transformer. In the BERT experiment, the training data generator randomly selects 15% of the marker positions for prediction. If the ith token is selected, we replace the ith token with the [MASK] token, which is random 80% of the time, and constant ith token 10% of the time, which is constant 10% of the time. The Transformer encoder does not know which words will be asked to predict or which words have been replaced by random words, so it is forced to maintain a distributed contextual representation of each input token. Moreover, since random substitutions occur in only 1.5% of all tags (that is, 10% of 15%), this does not seem to impair the language understanding of the model.

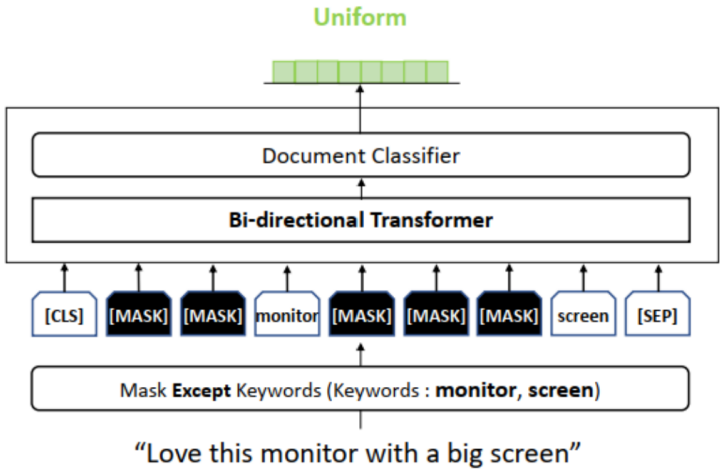


Figure 5: MLM model

5 Explore the Word Representations

5.1 Visualise the word representations by t-SNE

t-SNE, also known as T-distributed Stochastic Neighbor Embedding, is a Embedding model, which is used to map data in a high-dimensional space to a low-dimensional space and retain the local characteristics of data sets.It is mainly used for high dimensional data reduction and visualization. t-SNE, is one of the most effective methods for data reduction and visualization. t-SNE transforms the similarity between data points into conditional probabilities, and the similarity of data points in the original space is represented by Gaussian joint distribution. We write functions, nested models to achieve data dimensionality reduction, and improve the degree of data visualization. However, the disadvantages of t-SNE are obvious: high memory footprint and long running time.

5.2 Visualise the Word Representations of Biomedical Entities by t-SNE

We used the websites provided in the document to classify diseases, and we used NLTK to assign different colors to different entities to visualize biomedical entities.

K-means method is a kind of unsupervised learning algorithm, which solves the clustering problem. Its basic principle is to randomly determine k (artificially specified) initial points as the cluster centroid, and then calculate the distance between each point in the data sample and each cluster centroid, according to which the samples are allocated. The centroid of each cluster is then changed to the average of all points in the cluster. Therefore, we use k-means method to cluster English text. After clustering, words of the same category obviously become the same color and cluster together. However, the effect after clustering is not as expected, so we need to carry out subsequent research and adjustment.

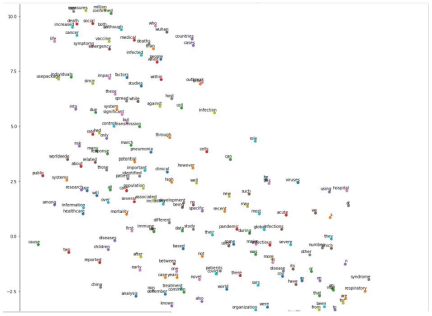


Figure 6: 5.1: t-sne

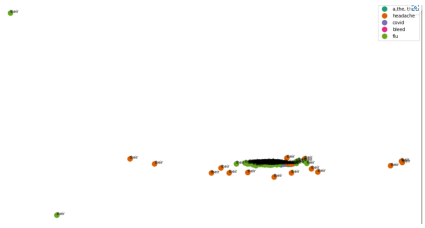


Figure 7: 5.2:k-means

word	$u^T v_j$
disease	0.970910
health	0.975594
coronavirus	0.999999
respiratory	0.971383
virus	0.970299
severe	0.977721
human	0.973321
viral	0.974989
influenza	0.980652

Table 1: 5.3:close word to "coronavirus"

5.3 Co-occurrence

We already know that the embedding matrix we get from the skip-gram stands for each word vectors. The idea is to go through each word vector, calculating the value of $u^T v_j$, where u stands for vector of "coronavirus", and v_j stands for other word vector.The bigger $u^T v_j$ is, the meaning of word v_j is more close to the target.

First, we set a function to calculate $u^T v_j$:

```
def get_cos(query1_token, query2_token, embed):
    W = embed
    x = W[word2id_dict[query1_token]]
    y = W[word2id_dict[query2_token]]
    cos = np.dot(x, y) / np.sqrt(np.sum(y * y) * np.sum(x * x) + 1e-9)
    flat = cos.flatten()
```

Then we pick the word whose $u^T v_j > 0.97$, and then we get the following words: disease, health, respiratory, virus, severe, human, viral, influenza

5.4 Semantic Similarity

In order to find words and entities that have similar semantic meanings to the designated entity (e.g. COVID-19 or coronavirus), we then train the model to represent the word, after that, compare the word vector of COVID-19 and other biomedical entities. In order to compare the word vector, we used the formula below:

$$Similarity = \frac{vex[w_1] \cdot vex[w_2]}{|vex[w_1]||vex[w_2]}$$

We apply the method of cosine similarity to compare the similarity between words:

```
from scipy import spatial
def similarity(m,n):
    if(m,n <= EMBEDDING_DIM):
        #vex is the word embedding
        return 1 - spatial.distance.cosine(vex[m], vex[n])
```

```
    else:
        print("ERROE!")
```

6 Open Challenge: Mining Biomedical Knowledge

There are explicit relationships between biomedical entities based on textual descriptions. We can more intuitively understand their internal connections by building knowledge graph. We import NetworkX to complete this task. A and B are two different biological entities(nodes). The line(edge) between them indicates that they are related to each other, while the X on the line indicates the reason for their association. Here's a basic picture:

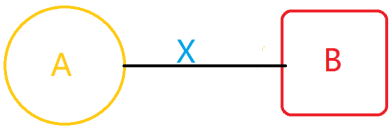


Figure 8: relation

We first used NLTK to extract biological entities and discover the relationship between two biological entities.

Based on the result of 5.3 and 5.4, the resulting knowledge graph tells us that not all biological entities are clearly connected to each other. Some biological entities have specific associations, but they are also isolated from some biological entities.

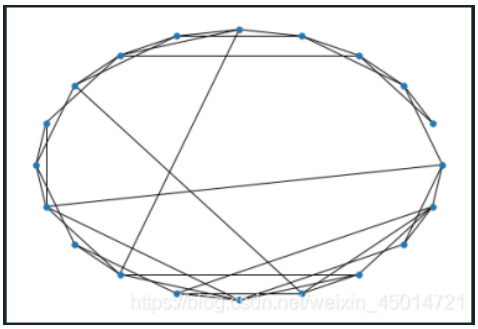


Figure 9: Network X

7 Summary

It is undeniable that our experiment still has some limitations, because of time and computer hardware issues, the test data is not wide enough. Therefore, we will make more perfect modifications and tests on this model in the future, and master more skills of using NLP model.

In the end, thanks for Weihang's guidance, and thanks to all the teachers for their instruction. It was an unforgettable summer school, and we learned a lot.