Part 1(LSA):

Step 2, extracting 5000 most common words:

```
print("The most frequent five words in the most frequent 5000 words in the Brown Corpus:")

W[:5]
```

```
The most frequent five words in the most frequent 5000 words in the Brown Corpus:

[('one', 3292),
 ('would', 2714),
 ('said', 1961),
 ('new', 1635),
 ('could', 1601)]
```

```
print("The least frequent five words in the most frequent 5000 words in the Brown Corpus:")

W[-5:]
```

```
The least frequent five words in the most frequent 5000 words in the Brown Corpus:

[('advances', 18),
 ('applicable', 18),
 ('humble', 18),
 ('defended', 18),
 ('spectacle', 18)]
```

Step 8, calculate the Pearson Correlation:

Pearson Correlation for M1:

```
S_list_M1 = []
for key in S_M1:
    S_list_M1.append(S_M1[key][0][0])
```

```
corr_M1, p_value_M1 = pearsonr(S_list_M1, S)
```

```
print(corr_M1)
```

```
0.11843435038906105
```

Pearson Correlation for M1_plus:

```
S_list_M1_plus = []
for key in S_M1_plus:
    S_list_M1_plus.append(S_M1_plus[key][0][0])
```

```
corr_M1_plus, p_value_M1_plus = pearsonr(S_list_M1_plus, S)
```

```
print(corr_M1_plus)
```

```
0.11993626872007863
```

Pearson Correlation for M2_10:

```
S_list_M2_10 = []
for key in S_M2_10:
    S_list_M2_10.append(S_M2_10[key][0][0])
```

```
corr_M2_10, p_value_M2_10 = pearsonr(S_list_M2_10, S)
```

```
print(corr_M2_10)
```

```
0.07028427881086727
```

Pearson Correlation for M2_100:

```
S_list_M2_100 = []
for key in S_M2_100:
    S_list_M2_100.append(S_M2_100[key][0][0])
```

```
corr_M2_100, p_value_M2_100 = pearsonr(S_list_M2_100, S)
```

```
print(corr_M2_100)
```

```
0.1143811990634159
```

Pearson Correlation for M2_300:

```
S_list_M2_300 = []
for key in S_M2_300:
    S_list_M2_300.append(S_M2_300[key][0][0])
```

```
corr_M2_300, p_value_M2_300 = pearsonr(S_list_M2_300, S)
```

```
print(corr_M2_300)
```

```
0.14137103020616631
```

Part 2(Word2Vec):

Step 3, calculate the cosine distance and the Pearson correlation:

Cosine Similarities for w2v model:

```
S_w2v = cal_cos_sim(P_dict, w2v_matrix, new_words)
```

Pearson Correlation for w2v model:

```
S_list_w2v = []
for key in S_w2v:
    S_list_w2v.append(S_w2v[key][0][0])
```

```
corr_w2v, p_value_w2v = pearsonr(S_list_w2v, S)
```

```
print(corr_w2v)
```

```
0.7521151672039903
```

```
print(p_value_w2v)
```

```
7.849322751325408e-13
```

The Pearson correlation of the Word2Vec model is significantly larger than the one from the LSA model. I suspect this is mainly the result of a relatively small context dimension(around 5000) leading to the result matrix lacking information.

What is interesting, however, is that before I removed all the stop words, the Pearson correlation for LSA for M2_300 was around 0.33 whereas afterwards it dropped to around 0.14. Similar phenomena was observed in all other matrices to different degrees. I suspect this is because with a small context dimension, some of the stop words acted as grammatical indicators which helped with distinguishing word meanings.

Step 4, semantic analogy test and syntactic analogy test on Word2Vec:

```
def analogy_test_w2v(list_of_lines, w2v_model):
    successful = 0

    for line in list_of_lines:
        words = line.split()
        prediction = w2v_model.most_similar(positive=[words[3], words[0]], negative=[words[2]])[0]

        if prediction[0] == words[1]:
            successful += 1

    return successful

print(analogy_test_w2v(semantic_pairs, model))
print(analogy_test_w2v(syntactic_pairs, model))
```

```
51
1441
```

The accuracy rate for semantic analogy test is approximately 91.07%(51/56), whereas the one for syntactic analogy test is approximately 69.28%(1441/2080).

Step 5, semantic analogy test and syntactic analogy test on LSA:

```
def analogy_test_lsa(list_of_lines, M_lsa, vocab_list):
    successful = 0

    for line in list_of_lines:
        words = line.split()
        prediction_vec = M_lsa[vocab_list.index(words[3])] - M_lsa[vocab_list.index(words[2])] + M_lsa[vocab_list.index
        cos_sim_list = []
        for word in vocab_list:
            i_w = vocab_list.index(word)
            cos_sim_list.append(cosine_similarity(sparse.csr_matrix(prediction_vec), sparse.csr_matrix(M_lsa[i_w])))

        if vocab_list[vocab_list.index(words[1])] == vocab_list[cos_sim_list.index(max(cos_sim_list))]:
            successful += 1

    return successful

print(analogy_test_lsa(semantic_pairs, M2_300, vocab))
print(analogy_test_lsa(syntactic_pairs, M2_300, vocab))
```

```
0
0
```

The accuracy rate for both semantic analogy test and syntactic analogy test are approximately 0%.

Of course, it could be caused by some problem in my programming. I examined my approach, which is to find the highest cosine similarity between the prediction vector and all the vectors in the model matrix, and compare it with gensim's implementation most_similar(), which makes use of a weighted matrix and the doc product between the model matrix and this weighted matrix. However, I think the idea behind both approaches are similar.

On the other hand, I discovered a pattern in the prediction error. In the case of, for example, [brother, sister, boy, girl], vec(brother) - vec(boy) + vec(girl) produces the result similar to vec(brother), which is not vec(sister), the desired result. I think this is suggesting that according to the LSA model, the word boy and the word girl has very similar vector representation, yet the word brother is significantly different from the word boy(since the result is brother rather than girl). This indicates that while the model is able to identify some differences in word meanings, the lack of information lead it to its failure to identify some similar words apart(presumably words often used in similar context). Further investigations could be guided towards the comparison of cosine similarity between all the four words and try to find what information was missed out in the model's insufficient context dimension(see step 3).

GitHub repo for both parts:

https://github.com/JieruiYang/CSC2611Lab1