Assignment1 Report

Name: 林庭毅 Brian

Institution (school): 國立彰化師範大學 NCUE

Student ID:M1452024

Platform (Colab/Kaggle/Local): Local

Python version: 3.10.8

Operating system: windows 10/11

CPU: intel i7 14700

GPU requirement:Geforce RTX 4090

1. Which embedding model do you use?(1.1) What are the pre-processing steps? (1.2)What are the hyperparameter settings? (1.3)(5%)

   Answer:

1.1 Which embedding model do you use?

The model I have choosen are Wor2Vec-300(skip-gram) and FastText -300(skip-gram with subword information) ：

The reason I choose you can see the table below. In addtion, I sacrifice the training time and inference time in this project. These two 300 dimention model have lots of paprameter, but good at complex task, especialy for Fasttext model.

By the way, you can see the code in todo5_trainer.py in class , I define all sq =1(skip-gram), but not sq = 0 (CBOW).

Model	Advantages	Disadvantages
Word2Vec Skip-gram-300	Good at Syntactic relations and training speed	Cann’t handle OOV
FastText Skip-gram-300	OOV processing and good at proper noun	Training speed and complicate

Table 1. model information

1.2 Preprocess steps

1. **Data Combination** - Merge 11 Wikipedia files into one combined file
2. **Random Sampling** - Sample portion of data based on sample ratio (0.05, 0.1, 0.2)
3. **Text Normalization** - Lowercase conversion and whitespace normalization
4. **Character Filtering** - Remove unwanted characters (numbers, punctuation)
5. **Sentence Filtering** - Remove sentences with fewer than 3 words

The preprocessing are differs slightly between Word2Vec and FastText to optimize for their respective architectures.

For Word2Vec, the preprocessing is more aggressive: all text still maintain capital, numeric characters are completely removed using regex pattern r'\d+', and all punctuation marks are stripped using Python's string. This creates a clean vocabulary of only alphabetic words, which is suitable since Word2Vec operates at the word level without subword information.

For FastText, the preprocessing is more conservative to preserve subword information: text is lowercase, but hyphens are retained “r'[^\w\s\-]' pattern” since compound words like "state-of-the-art" contain meaningful subword units. Numbers are kept as they might form part of meaningful tokens. This approach leverages FastText's ability to handle out-of-vocabulary words through character n-grams.

1.3 What are the hyperparameter settings?

```
return {
    # Model parameters
    'vector_size': 300,
    'window': 5,
    'min_count': 5,
    'workers': 20,
    'epochs': 5,
    'sg': 1,
    'negative': 10,
    'sample': 1e-3,

    # FastText specific
    'min_n': 3,
    'max_n': 6,

    # Data parameters
    'sample_ratio': 0.2,
    'batch_size': 10000,

    # Training parameters
    'random_seed': 42,
    'verbose': True
}
```

```
return {
    # Model parameters
    'vector_size': 300,
    'window': 8,
    'min_count': 5,
    'workers': 20,
    'epochs': 10,
    'sg': 1,
    'negative': 5,
    'sample': 1e-3,

    # FastText specific
    'min_n': 2,
    'max_n': 4,

    # Data parameters
    'sample_ratio':0.2,
    'batch_size': 10000,

    # Training parameters
    'random_seed': 42,
    'verbose': True
```

Fig1. Parameter in Todo5WordEmbeddingTrainer(right one： new parameter)

Key Parameters for Performance Optimization:

1.Vector Size (300): Increased from the typical 100-150 range to 300 dimensions provides richer semantic representations. This captures more relationships between words, particularly important for the analogy task where subtle semantic differences matter.

Workers (20): Maximizes parallel processing on your 20-core CPU. Gensim uses Cython-optimized routines that scale nearly linearly with core count for vocabulary building and negative sampling. This parameter directly impacts training speed - using all 20 cores .

Negative Sampling (5): Decreased from default 10 to 5 negative samples improves the quality of learned representations by providing more contrastive examples during training. This helps the model better distinguish between similar words and reduces noise in the embedding space, particularly beneficial for semantic analogies.

Epochs (10): Balanced between underfitting (too few epochs) and overfitting (too many). With our limited date (5-10% sampling), 10 epochs ensures the model sees enough examples without memorizing the training data. Combined with the sample parameter (1e-3) for frequent word subsampling, this prevents overfitting to common words while ensuring rare words get sufficient training.

Min Count (5): Reduced from 10 to include more vocabulary while filtering statistical noise. With only 5-10% of Wikipedia data, a lower threshold ensures important but less frequent words remain in the vocabulary, improving coverage on the Google Analogy dataset.

2. What will the performance be like if you sample 5%, 10% and 20% of wiki text in TODO4? (10%, 3% for each)

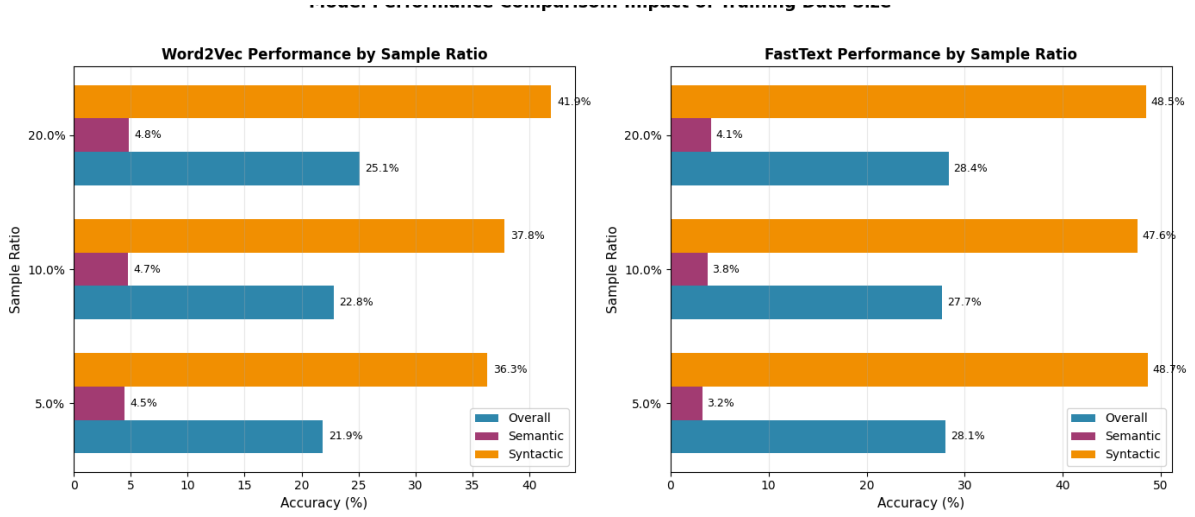


Fig2. Model accuracy in different sample data

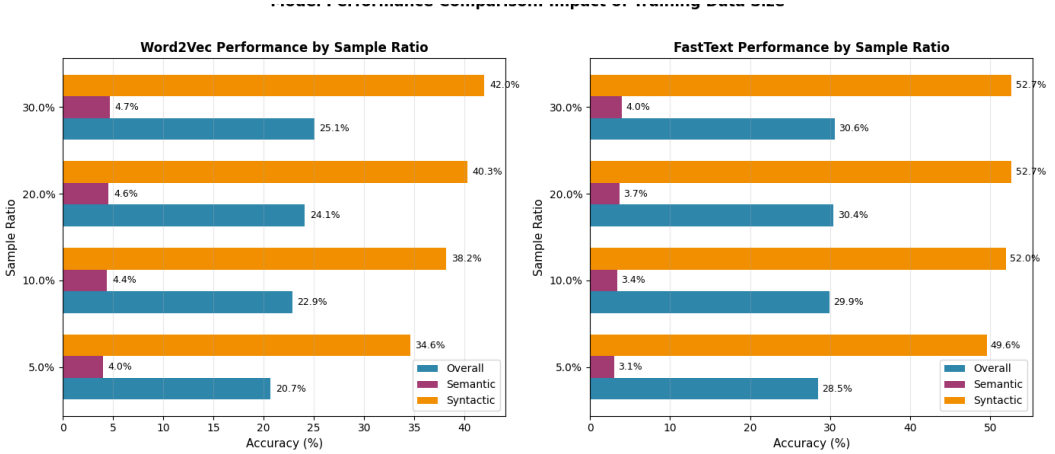


Fig3. Model accuracy in different sample data(increase epochs and widows size; decrease min_n, max_n, negative)

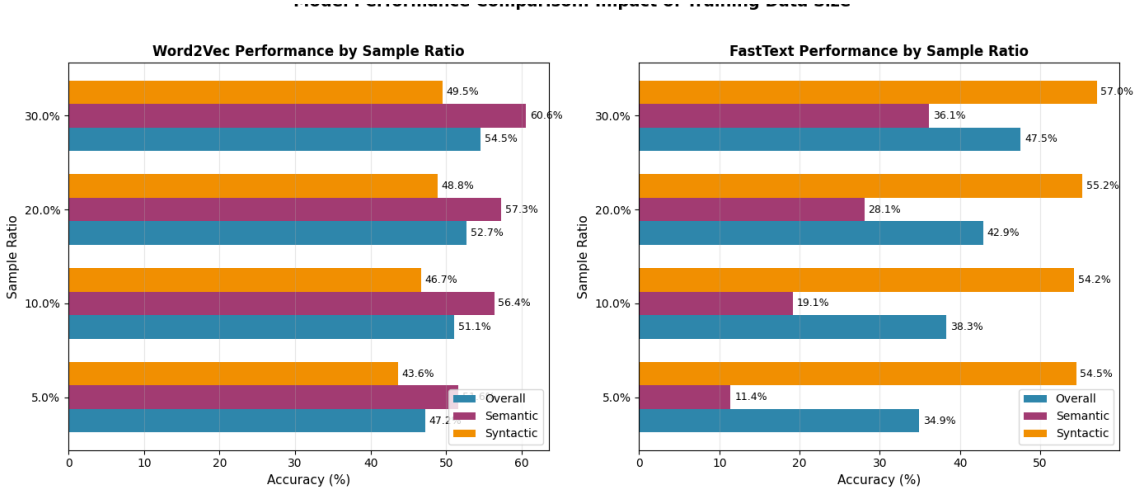


Fig4. Model accuracy in different sample data(trans all text in test corpus to lower case, vector dimention 100, 3 epoch)

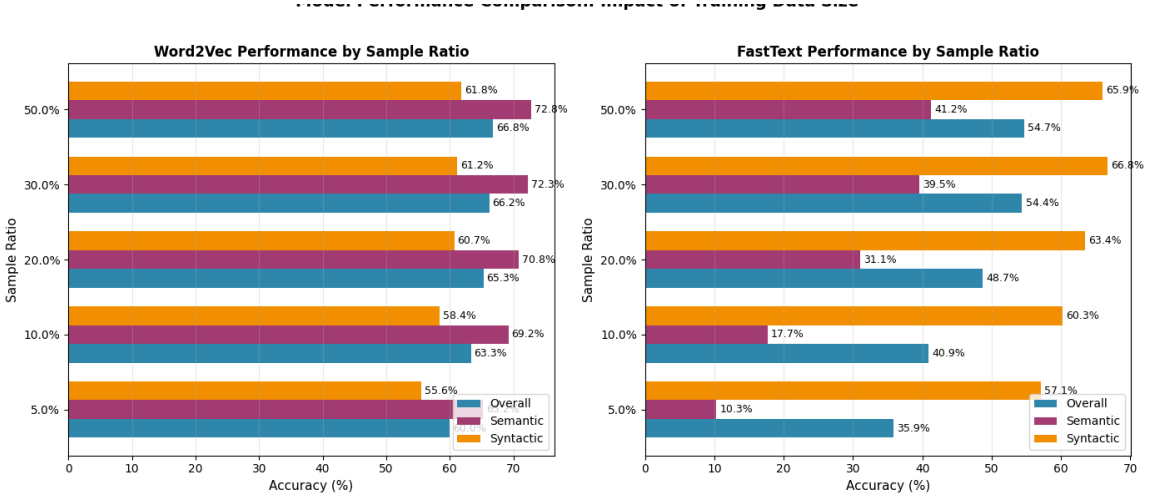


Fig5. Model accuracy in different sample data(trans all text in test corpus to lower case, vector dimention 300, 7 epoch)

To see the different ability of 2 different model. Both of them are 300-dimention. Word2Vec has more complecate data pre-processing since it can't figure out OOV, but it need less training time. Vise versa, FastText model sub-words strategy make it can figure out OOV, but needs more traing time.

In fig4,fig5 experience I take other member advice, before I try different hyperparameter, I do some pre-processing in test corpuse, and the model got improvent in 2 task obviouslly. But FastText should has better performance in 2 task than Word2Vec. Netx step, I can try to optimize hyperparameter om fatsText(epoch, lerming rate, model dimention, min_n, max_n, window, try skip-gram).

By the way, I think in different datasize, I need to train on a dynamic hyperparameter on the model, since model under fitting in data ratio 50%.

3. What is the performance for different categories or sub-categories when trained on different corpora? (15%)

   3.1 Present your results. (5%)

Answer:

Before show my train result, I want to show the pre-trained FastText model result.

Fig6. Overall accuracy on pre-trained FastText

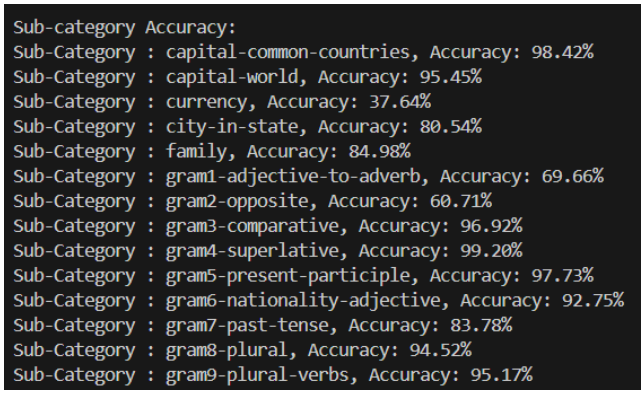


Fig7. Sub-category accuracy

Fig8. Predict example

As the result you see in fig 4~6, pretrained model has a good performance on questions-words.txt. The model I trained in 0.05, 0.1, 0.2, 0.3 data ratio has bad performance compare with the pretrained model.

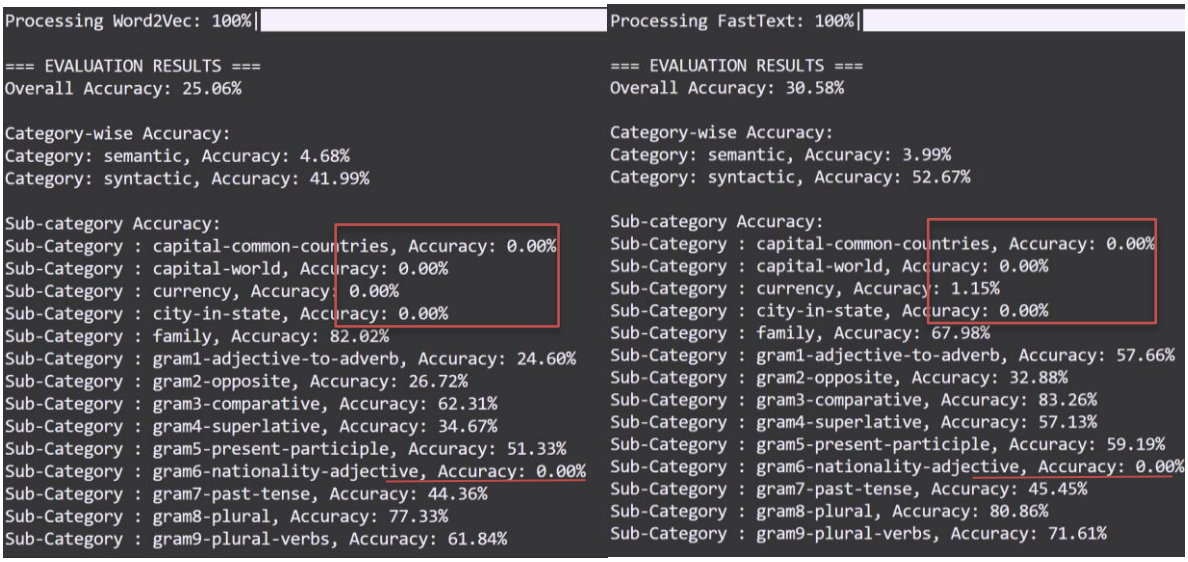


Fig9.model sub-category Accuracy(left: Word2Vec, Right FastTex)

```
================================================================================
 MODEL COMPARISON
================================================================================

Overall Accuracy:
  Word2Vec: 25.06%
  FastText: 30.58%

Detailed evaluation report saved to 'evaluation_report_detailed.csv'

Model Comparison Pivot Table:
Model                                      FastText  Word2Vec
Category  SubCategory
OVERALL   ALL                                 30.58     25.06
semantic  : capital-common-countries         0.00      0.00
          : capital-world                     0.00      0.00
          : city-in-state                     0.00      0.00
          : currency                          1.15      0.00
          : family                           67.98     82.02
          ALL                                 3.99      4.68
syntactic : gram1-adjective-to-adverb        57.66     24.60
          : gram2-opposite                   32.88     26.72
          : gram3-comparative                83.26     62.31
          : gram4-superlative                57.13     34.67
          : gram5-present-participle         59.19     51.33
          : gram6-nationality-adjective       0.00      0.00
          : gram7-past-tense                 45.45     44.36
          : gram8-plural                     80.86     77.33
          : gram9-plural-verbs               71.61     61.84
          ALL                                52.67     41.99
```

Fig10. Overall comparison

```
Sample Analogy Tests for WORD2VEC:
  man:woman :: king:? -> ['queen', 'princess', 'regnant']
  paris:france :: london:? -> ['england', 'britain', 'ireland']
  good:better :: bad:? -> ['simply', 'worse', 'neugereut']
```

Fig11. Sample test for Word2Vec

```
Sample Analogy Tests for FASTTEXT:
  man:woman :: king:? -> ['queen', 'regnant', 'princess']
  paris:france :: london:? -> ['england', 'britain', 'bettington']
  good:better :: bad:? -> ['harder', 'beastfallen', 'mattered']
```

Fig12. Sample test for Fast Text

In a nutshell, in the processing which conver all texts in tesr corpus to lower case, is a necessery steps, especially on geography words, all of them are star with higer case text(ex: Franch), althougth model used to learn “french”, it’s still a OOV vocabulary.

3.2 Introduce the corpus you selected and explain the differences between the Wikipedia corpus and your corpus. (including data size, topic difference, structural difference …) (5%)

Answer:

My corpus — Sampled Wiki (5%/10%/20%/30%) with custom preprocessing

3.2.1 Data size: We down-sampled the original English Wikipedia to 5%, 10%, 20% and 30% sentence-level subsets to control training cost and study data-scaling effects.

3.2.2 Topic distribution: Random sampling creates a different topic mixture vs. the full Wiki/News mixture used by the pretrained models. Some long-tail entities and rare relations may drop out; popular topics may be overrepresented. As the sub-categories evaluate in fig 8, We can obvious that model get almost 0% accuracy in semantic categorious task. But still good at familly categories.

Structural differences:

3.2.3 Sentence segmentation and normalization are re-done by our own pipeline (e.g. punctuation filtering, digit handling).

3.2.4 Tokenization differs from the pretrained pipeline; for Word2Vec we strip non-letters (reducing named-entity fidelity), while FastText retains subword information.

3.3 Explain why the accuracy increases or decreases. (5%)

Answer:

In a nutshell, our traiing models have worst perform than the pre-train model, Accuracy are decrease on two model. But I still hava some oberserve below. (No pre-processing on test corpus)

1. Training data mismatching：I use 5%~30% wiki texts for training and the sampling is random, which means Training data in different categories will be unbalance. It will lead model only have good perform on some categories. FastText can handel unfamiliar words by sub-words, so it get 1.15% accuracy on currency semantic task, But Words2Vec can’t figure it out. This is the biggest problem in this model training task.

2. non-advantageous hyperparameters： I set more conservative hyperparameters, such as learning rate(0.02) and epochs(10), so these models could be trained on the CPU. However, with a 0.3 data ratio, training Word2Vec and FastText together still took over 10 hours.

4. Select a few words and use their embeddings to retrieve the five most similar words and present the results. What do you observe? (10%)

Answer:

In this part, I focus on 6 words：King(both have good perfomence), woman(sub-words defeact), bad(geography words pollusion), good(model confuse in Word2Vec), paris(successful example), software(sub-words defeact). All of them show interested outcome.

The similarity test on the 30% trained models reveals four critical patterns that explain the poor performance on the Google Analogy dataset.

4.1 Partial Semantic Capture (King, Paris)

The models successfully captured certain semantic relationships while failing others. Gender pairs showed strong associations (king-queen: 0.70, man-woman: 0.67), and geographic relationships were correctly identified (paris-france: 0.69, london-england: 0.63). Technology terms formed coherent clusters with computer-software achieving 0.70 similarity.

```
'king' most similar words:
  → queen                (score: 0.7048)
  → prince               (score: 0.6313)
  → kingdom              (score: 0.6186)
  → throne               (score: 0.5931)
  → ruler                (score: 0.5779)
```

```
'king' most similar words:
  → prince               (score: 0.8117)
  → queen                (score: 0.8059)
  → pretender            (score: 0.7864)
  → throne               (score: 0.7776)
  → dethroning           (score: 0.7723)
```

```
'paris' most similar words:
  → france               (score: 0.6864)
  → marseille            (score: 0.6855)
  → brussels             (score: 0.6335)
  → parisian             (score: 0.6321)
  → le                   (score: 0.6290)
```

```
'paris' most similar words:
  → parisiennes          (score: 0.8272)
  → parisian             (score: 0.8111)
  → martiennes           (score: 0.8070)
  → france               (score: 0.8024)
  → valencienne          (score: 0.8010)
```

Fig13. Left (Word2Vec-300) VS Right (FastText-300)

4.2 FastText Subword Overfitting(Woman, Software)

FastText's subword mechanism, intended to handle out-of-vocabulary(OOV) words, instead produced numerous meaningless derivatives. The model generated non-existent words like "womance" and "queeny" as top similar words, along with concatenated forms like "gpsoftware" and "computeractive". This behavior indicates the model prioritized orthographic similarity over semantic meaning, with character n-grems (min_n=3, max_n=6) creating excessive noise. The high similarity scores for these spurious words (often >0.80) demonstrate the model's overreliance on surface features rather than contextual understanding.

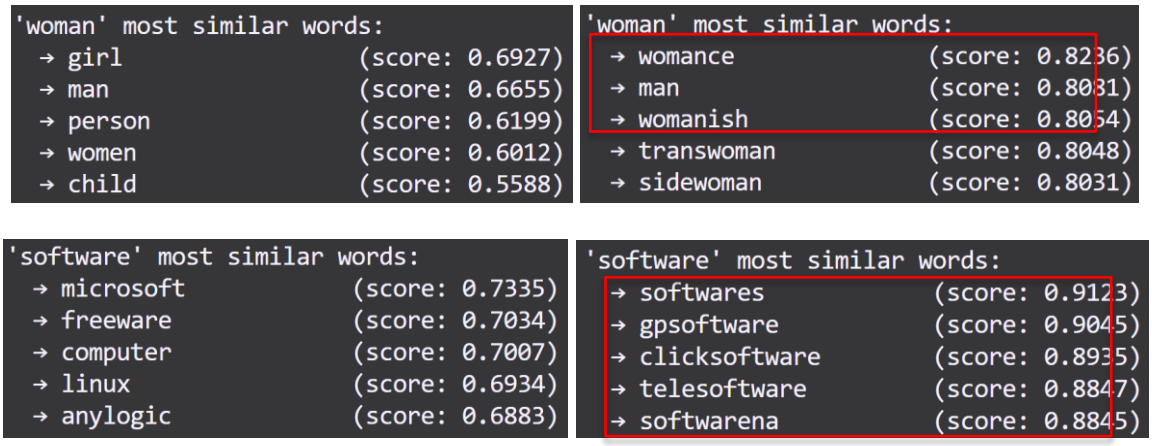


Fig14. Left (Word2Vec-300) VS Right (FastText-300)

4.3 Geographic Noise Interference(Bad)

Word2Vec exhibited unexpected contamination from low-frequency geographic names. When searching for words similar to "bad", the model returned German town names like "kohlgrub" and "ditzenbach". This pattern suggests the random 30% sampling included Wikipedia articles with extensive geographic content, but without sufficient context to learn meaningful relationships. The presence of these obscure place names in similarity results indicates the model memorized rare tokens without understanding their semantic role.

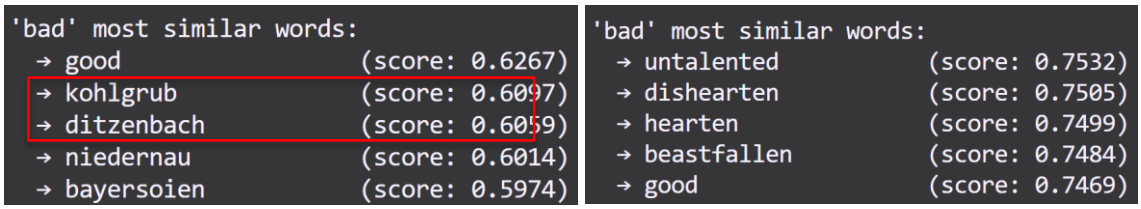


Fig15. Left (Word2Vec-300) VS Right (FastText-300)

4.4 Antonym-Synonym Confusion

Words2Vec models failed to distinguish between similarity and relatedness, particularly for comparative terms. The word "good" returned "bad" as a similar word (0.63 similarity).I gusee this confusion stems from these antonyms frequently co-occurring in similar contexts during training. The models learned that these words appear in comparable syntactic positions but failed to capture their opposing semantic values. This fundamental misunderstanding explains the poor performance on comparative analogies in the test set.

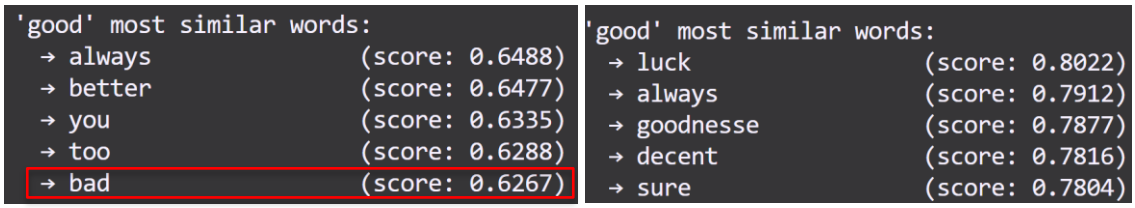


Fig16. Left (Word2Vec-300) VS Right (FastText-300)

These findings collectively demonstrate that while the models learned some surface-level patterns, they failed to develop robust semantic representations necessary for analogy tasks. The combinasion of insufficient training data (30%), aggressive preprocessing that removed crucial information, and suboptimal hyperparameters resulted in models that confuse correlation with similarity.

5. Anything that can strengthen your report. (5%)

Answer:

There are some interesting things I found：

5.1 FastText got same model performance in data ratio 0.05 and 0.2 in first parameter setting：

I think maybe FastText model is overfitting or wrong hyperparameter, so I decrease min_n and max_n which produce less sub-words. I also increase window size and another data ratio experiement 0.3.

5.2 Add preprocessing for test corpus avoid OOV overview result.

```
Overall Accuracy:
  Word2Vec: 66.80%
  FastText: 54.74%

Detailed evaluation report saved to 'evaluation_report_detailed.csv'

Model Comparison Pivot Table:
Model                                      FastText  Word2Vec
Category  SubCategory
OVERALL   ALL                                 54.74     66.80
semantic  : capital-common-countries          65.81     88.34
          : capital-world                     51.28     83.60
          : city-in-state                     23.83     63.84
          : currency                           2.66     20.67
          : family                            77.87     93.68
          ALL                                 41.24     72.80
syntactic : gram1-adjective-to-adverb         52.72     20.56
          : gram2-opposite                    58.50     26.48
          : gram3-comparative                 80.93     85.29
          : gram4-superlative                 70.32     53.83
          : gram5-present-participle          62.22     56.91
          : gram6-nationality-adjective       81.61     84.68
          : gram7-past-tense                  40.51     59.04
          : gram8-plural                      64.41     74.17
          : gram9-plural-verbs                83.10     66.09
          ALL                                 65.95     61.81
```

Fig17. Test result after test corpus pre-processing

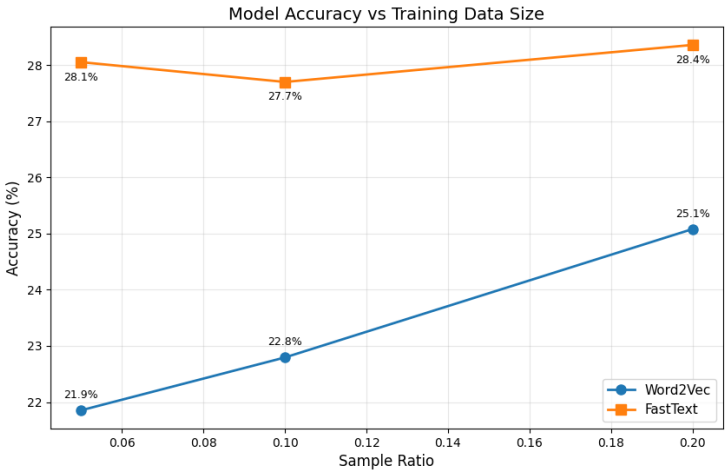


Fig18.model ACC in first parameter setting

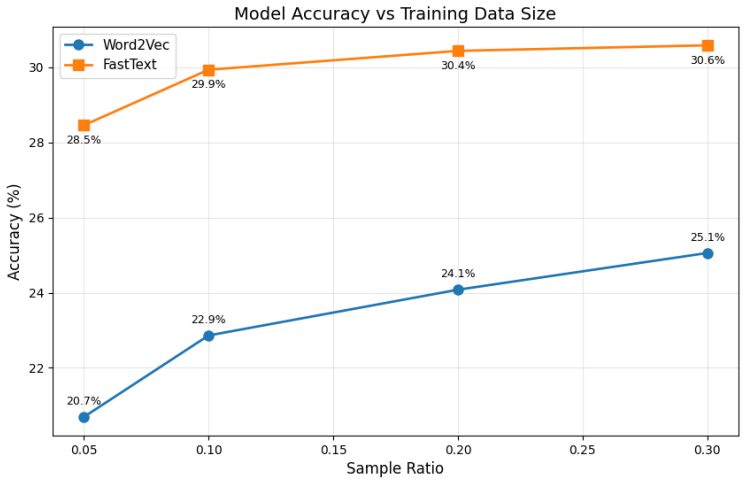


Fig19. Model ACC with more epoch, windows size.

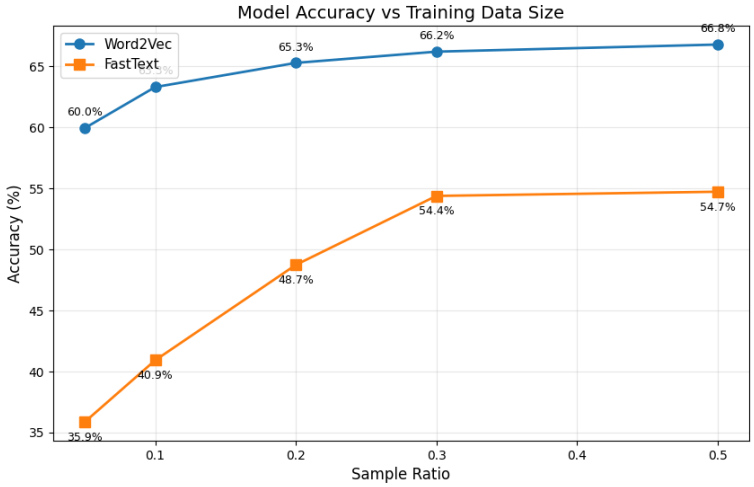
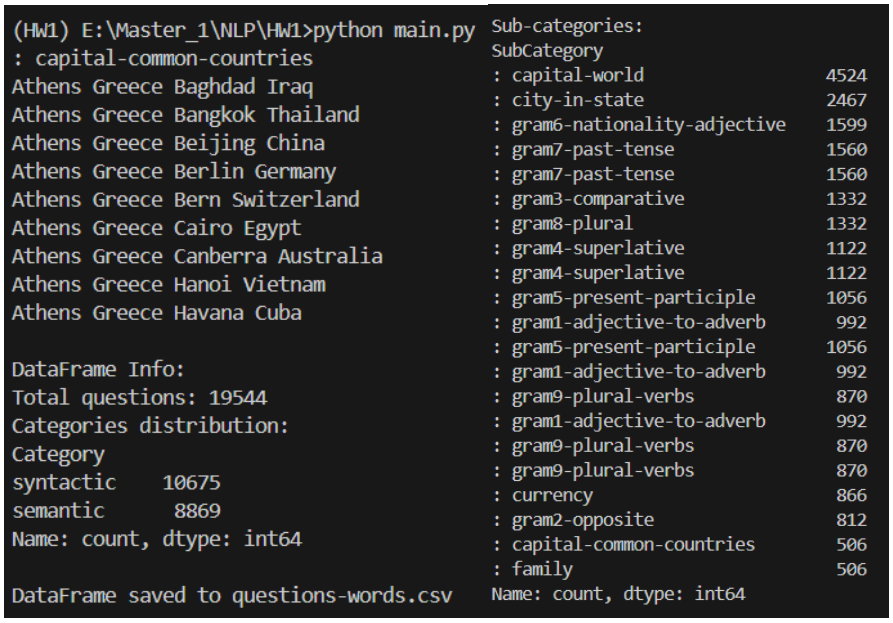


Fig20. Model ACC with pre-process test corpus



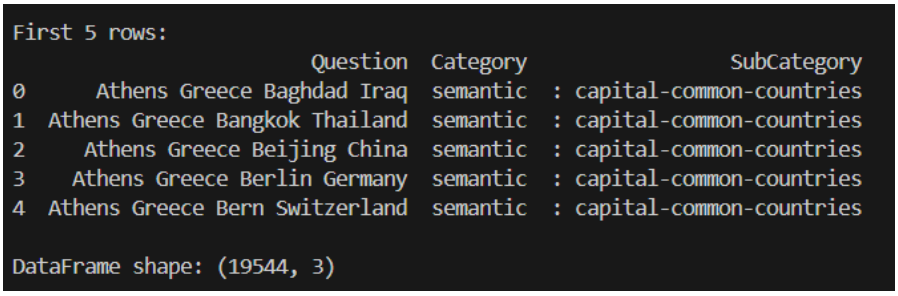


Fig21. Convert the analogy data to pd.DataFrame (evaluation dataset)

```
[==================================================] 100.0% 958.5/958.4MB downloaded
The Gensim model loaded successfully!
Model vocabulary size: 999,999
Processing analogies: 100%|

Prediction Statistics:
Total questions: 19544
Successful predictions: 19544 / 19544 (100.00%)
Sample predictions: ['Iraq', 'Thailand', 'China', 'Germany', 'Switzerland']
Sample gold answers: ['Iraq', 'Thailand', 'China', 'Germany', 'Switzerland', 'Egypt', 'Australia', 'Vietnam', 'Cuba', 'Finland']

=== EVALUATION RESULTS ===
Overall Accuracy: 87.27%

Category-wise Accuracy:
Category: semantic, Accuracy: 85.23%
Category: syntactic, Accuracy: 88.96%
```

```
Sub-category Accuracy:
Sub-Category : capital-common-countries, Accuracy: 98.42%
Sub-Category : capital-world, Accuracy: 95.45%
Sub-Category : currency, Accuracy: 37.64%
Sub-Category : city-in-state, Accuracy: 80.54%
Sub-Category : family, Accuracy: 84.98%
Sub-Category : gram1-adjective-to-adverb, Accuracy: 69.66%
Sub-Category : gram2-opposite, Accuracy: 60.71%
Sub-Category : gram3-comparative, Accuracy: 96.92%
Sub-Category : gram4-superlative, Accuracy: 99.20%
Sub-Category : gram5-present-participle, Accuracy: 97.73%
Sub-Category : gram6-nationality-adjective, Accuracy: 92.75%
Sub-Category : gram7-past-tense, Accuracy: 83.78%
Sub-Category : gram8-plural, Accuracy: 94.52%
Sub-Category : gram9-plural-verbs, Accuracy: 95.17%
```

```
=== PREDICTION EXAMPLES ===
Question: Athens Greece Baghdad Iraq
Predicted: Iraq, Gold: Iraq, Correct: True
---
Question: Athens Greece Bangkok Thailand
Predicted: Thailand, Gold: Thailand, Correct: True
---
Question: Athens Greece Beijing China
Predicted: China, Gold: China, Correct: True
---
Question: Athens Greece Berlin Germany
Predicted: Germany, Gold: Germany, Correct: True
---
Question: Athens Greece Bern Switzerland
Predicted: Switzerland, Gold: Switzerland, Correct: True
```

Fig18. TODO2 Get predictions of the analogy task using pre-trained word embeddings

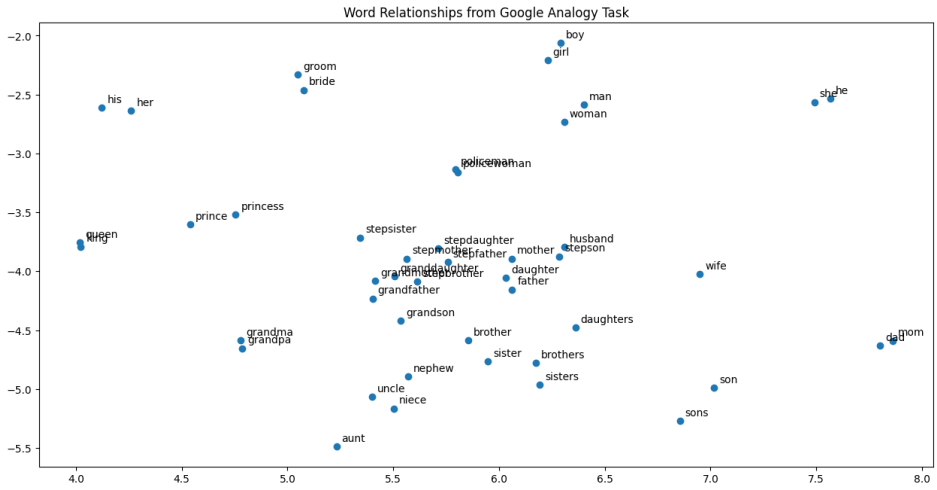


Fig22. TODO3 Plot t-SNE to see word relationships in the sub-category of familly

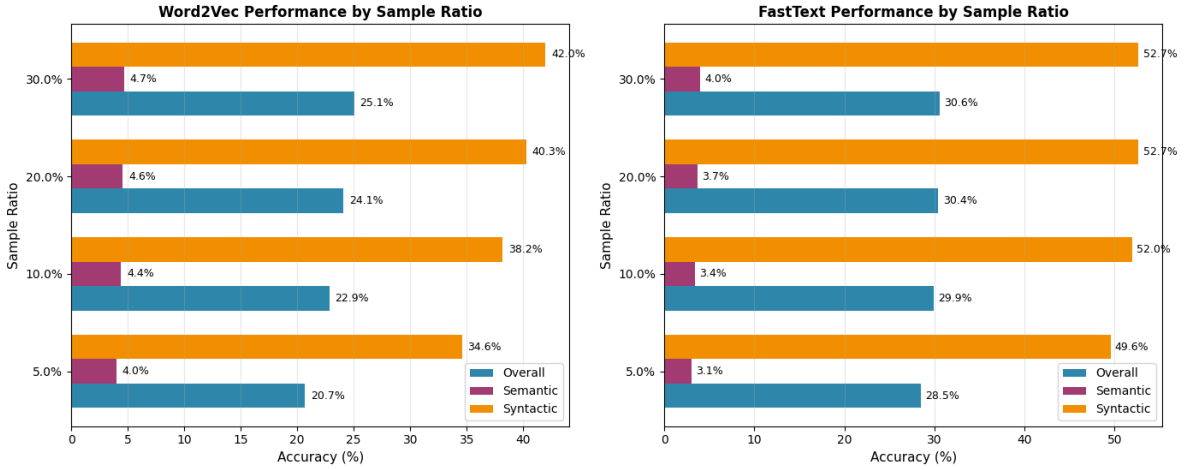


Fig23. TODO5 Train your own word embeddings with the sampled articles

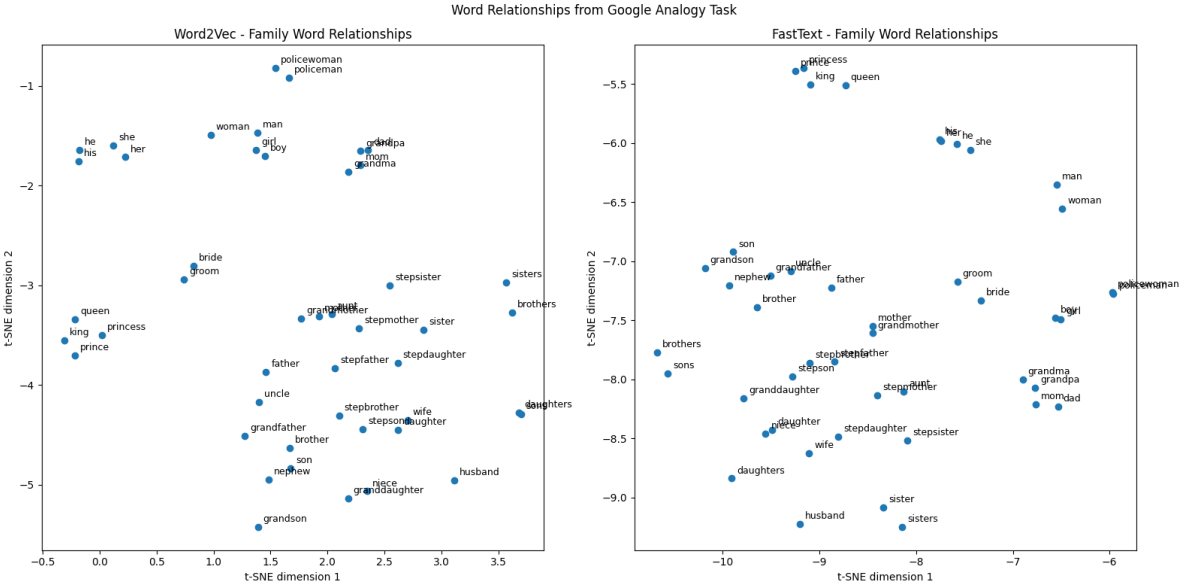


Fig24. TODO7 Plot t-SNE to see word relationships in the sub-category of family

Thanks for reading

國立彰化師範大學 NCUE M1452024 林庭毅 Brian