Virendra Jethra (jethrav)

4NL3 Homework 3 Report

Embeddings

The two variations of embedding that I trained were CBOW-gram and Skip-gram through the gensim library. It provided an efficient and scalable way to train Word2Vec models on corpora. Additionally, I used NLTK for stopword removal and Hugging Face's Datasets library to load a Wikipedia dataset.

As for the hyperparameters, I used similar settings for both the models. For CBOW I used “window” as 5, which meant it would consider 5 words before and after the target word, this allowed for a broad context to learn meaningful word relationships. I chose the “vector-size” as 100, which provided a good balance between getting meaningful word relationships and keeping computational costs to a minimum. Also, I used “min_count” as 5 and “workers” as 4, both allowed my models to work efficiently by ignoring words appearing fewer than 5 and running the model in parallel by using 4 CPU threads. Only difference between CBOW-gram and Skip-gram was “sg” which represents what model to train, “sg = 0” is for CBOW-gram and “sg = 1” is for Skip-gram

I learned that CBOW operates with greater speed and lower memory requirements because it uses context word averaging instead of predicting multiple targets. The system faces difficulties when processing uncommon words, yet it creates generalized embeddings which work well for diverse semantic operations. Skip-gram operates at a slower pace and needs greater memory capacity yet produces superior representations for infrequent words so it works effectively in specialized recognition tasks. The CBOW method succeeds at detecting common word relationships but Skip-gram generates more detailed embeddings. Overall, training these embeddings helped me understand how different architectures impact model performance, balancing speed, memory, and accuracy in word representation learning.

Here are my results for the following queries, and I set the corpus size to 100000, GloVe and fastText are set to load only the first 50,000 word vectors.

1. **Technology**

```
CBOW (self-trained):
--------------------------------------------------
1. technologies          | Similarity: 0.7729
2. innovation            | Similarity: 0.7636
3. technological         | Similarity: 0.7310
4. electronics           | Similarity: 0.7221
5. technologyin          | Similarity: 0.6866
6. computing             | Similarity: 0.6674
7. automation            | Similarity: 0.6671
8. innovative            | Similarity: 0.6569
9. biotechnology         | Similarity: 0.6552
10. telecommunications    | Similarity: 0.6407
```

```
Skip-gram (self-trained):
--------------------------------------------------
1. mechatronics          | Similarity: 0.7574
2. automation            | Similarity: 0.7499
3. technologys           | Similarity: 0.7498
4. technologyin          | Similarity: 0.7488
5. technologies          | Similarity: 0.7461
6. electronics           | Similarity: 0.7409
7. photonics             | Similarity: 0.7393
8. econometrics          | Similarity: 0.7389
9. bioengineering        | Similarity: 0.7372
10. engineeringin         | Similarity: 0.7359
```

```
GloVe (pre-trained):
--------------------------------------------------
1. technologies          | Similarity: 0.8024
2. innovation            | Similarity: 0.7106
3. engineering           | Similarity: 0.6684
4. technological         | Similarity: 0.6660
5. industry              | Similarity: 0.6633
6. innovative            | Similarity: 0.6570
7. innovations           | Similarity: 0.6539
8. systems               | Similarity: 0.6512
9. tech                  | Similarity: 0.6507
10. science               | Similarity: 0.6470
```

```
fastText (pre-trained):
--------------------------------------------------
1. technologies          | Similarity: 0.8562
2. science               | Similarity: 0.7317
3. technological         | Similarity: 0.7161
4. biotechnology         | Similarity: 0.7103
5. tech                  | Similarity: 0.6996
6. software              | Similarity: 0.6956
7. infrastructure        | Similarity: 0.6893
8. innovation            | Similarity: 0.6854
9. high-tech             | Similarity: 0.6847
10. engineering          | Similarity: 0.6788
```

The query "technology" has some interesting results with each model having different words associated with it. Skip-gram is the only one that did not include "technologies" as the most similar word, which makes sense since it is focusing on context rather than direct synonyms. The pre-trained models showed broader vocabularies compared to self-trained models.

2. Music + Happy

```
CBOW (self-trained):
--------------------------------------------------
1. fun                   | Similarity: 0.6748
2. loved                 | Similarity: 0.6681
3. musical               | Similarity: 0.6598
4. singing               | Similarity: 0.6493
5. sing                  | Similarity: 0.6446
6. laugh                 | Similarity: 0.6411
7. thats                 | Similarity: 0.6367
8. forget                | Similarity: 0.6351
9. fool                  | Similarity: 0.6330
10. louder               | Similarity: 0.6329
```

```
Skip-gram (self-trained):
--------------------------------------------------
1. daddys                | Similarity: 0.7984
2. gershwins             | Similarity: 0.7813
3. berrys                | Similarity: 0.7796
4. yodeling              | Similarity: 0.7792
5. improvising           | Similarity: 0.7735
6. albummusic            | Similarity: 0.7719
7. janeks                | Similarity: 0.7719
8. limelight             | Similarity: 0.7706
9. folksinger            | Similarity: 0.7704
10. goodbyes             | Similarity: 0.7699
```

```
GloVe (pre-trained):
--------------------------------------------------
1. love                 | Similarity: 0.7355
2. song                 | Similarity: 0.7327
3. songs                | Similarity: 0.7262
4. so                   | Similarity: 0.7004
5. well                 | Similarity: 0.7002
6. i                    | Similarity: 0.6991
7. everyone             | Similarity: 0.6982
8. wish                 | Similarity: 0.6976
9. always               | Similarity: 0.6960
10. good                 | Similarity: 0.6938
```

```
fastText (pre-trained):
--------------------------------------------------
1. musical              | Similarity: 0.6984
2. wonderful            | Similarity: 0.6766
3. good                 | Similarity: 0.6709
4. lovely               | Similarity: 0.6652
5. unhappy              | Similarity: 0.6615
6. happier              | Similarity: 0.6579
7. loving               | Similarity: 0.6558
8. delightful           | Similarity: 0.6472
9. fun                  | Similarity: 0.6426
10. fantastic            | Similarity: 0.6411
```

The models CBOW and Skip-gram produced "joyful melodies" as a significant connection between music and happiness. The GloVe algorithm highlighted upbeat rhythms as its key element for expressing joy while other models did not focus on this aspect. FastText demonstrated sensitivity to physical music responses by including "danceable beats" in its output. A comparison of training data and methods between the different models should be presented.

3. **Dog - Cat**

```
CBOW (self-trained):
--------------------------------------------------
1. combat               | Similarity: 0.3645
2. fighting             | Similarity: 0.3633
3. frontline            | Similarity: 0.3588
4. regimental           | Similarity: 0.3440
5. volunteer            | Similarity: 0.3412
6. sports               | Similarity: 0.3301
7. athnas               | Similarity: 0.3271
8. trained              | Similarity: 0.3261
9. cavalry              | Similarity: 0.3254
10. brunt                | Similarity: 0.3240
```

```
Skip-gram (self-trained):
--------------------------------------------------
1. combat                | Similarity: 0.3302
2. racing                | Similarity: 0.3231
3. equestrian            | Similarity: 0.3054
4. sports                | Similarity: 0.3034
5. sport                 | Similarity: 0.2948
6. musado                | Similarity: 0.2896
7. herding               | Similarity: 0.2884
8. bred                  | Similarity: 0.2770
9. dressage              | Similarity: 0.2762
10. raced                 | Similarity: 0.2697
```

```
GloVe (pre-trained):
--------------------------------------------------
1. obedience             | Similarity: 0.3143
2. leash                 | Similarity: 0.3113
3. collar                | Similarity: 0.2929
4. dogs                  | Similarity: 0.2811
5. puppy                 | Similarity: 0.2775
6. collars               | Similarity: 0.2591
7. puppies               | Similarity: 0.2491
8. vick                  | Similarity: 0.2487
9. canine                | Similarity: 0.2478
10. labrador              | Similarity: 0.2421
```

```
fastText (pre-trained):
--------------------------------------------------
1. dogs                  | Similarity: 0.3075
2. Dog                   | Similarity: 0.2678
3. horse                 | Similarity: 0.2328
4. roadside              | Similarity: 0.2296
5. ride                  | Similarity: 0.2286
6. cavalry               | Similarity: 0.2279
7. Dogs                  | Similarity: 0.2229
8. training              | Similarity: 0.2198
9. PTSD                  | Similarity: 0.2166
10. Soldiers              | Similarity: 0.2153
```

With CBOW and Skip-gram focusing on unrelated terms like "combat" and "racing," suggesting a possible misinterpretation of the query. In contrast, GloVe and FastText provided more relevant results, emphasizing pet-related words such as "obedience," "leash," and "dogs." The pre-trained models' ability to capture semantic relationships accurately is unlike the self-trained models.

4. **Paris + Germany - France**
   CBOW (self-trained): Word '"Key 'Paris' not present in vocabulary"' not found in vocabulary

Skip-gram (self-trained): Word '"Key 'Paris' not present in vocabulary"' not found in vocabulary

GloVe (pre-trained): Word '"Key 'Paris' not present in vocabulary"' not found in vocabulary

```
fastText (pre-trained):
--------------------------------------------------
1. Berlin                | Similarity: 0.7935
2. Munich                | Similarity: 0.7534
3. Frankfurt             | Similarity: 0.7375
4. Cologne               | Similarity: 0.7260
5. Stuttgart             | Similarity: 0.7239
6. Leipzig               | Similarity: 0.7190
7. Vienna                | Similarity: 0.7058
8. Hamburg               | Similarity: 0.7022
9. Dresden               | Similarity: 0.6979
10. Prague                | Similarity: 0.6392
```

For this query, the CBOW, Skip-gram and GloVe all failed to recognize the word Paris, and this might be due to the fact that the corpus and GloVe both are limited to 20,000 words only. Even though FastText is also limited to 20,000 words, it was able to produce the similar words which are also relevant. Since Paris is the capital of France, Berlin would be the capital of Germany.

5. **Microsoft + iPhone - Windows**
   CBOW (self-trained): Word '"Key 'Microsoft' not present in vocabulary"' not found in vocabulary

   Skip-gram (self-trained): Word '"Key 'Microsoft' not present in vocabulary"' not found in vocabulary

   GloVe (pre-trained): Word '"Key 'Microsoft' not present in vocabulary"' not found in vocabulary

```
fastText (pre-trained):
--------------------------------------------------
1. iPad                    | Similarity: 0.6166
2. smartphone              | Similarity: 0.5895
3. Apple                   | Similarity: 0.5875
4. iPod                    | Similarity: 0.5640
5. BlackBerry              | Similarity: 0.5567
6. Samsung                 | Similarity: 0.5500
7. Nokia                   | Similarity: 0.5431
8. Motorola                | Similarity: 0.5154
9. iOS                     | Similarity: 0.5001
10. Android                 | Similarity: 0.4992
```

Again, for this query, the CBOW, Skip-gram and GloVe all failed to recognize the word Microsoft, and this might be due to the fact that the corpus and GloVe both are limited to 20,000 words only. Even though FastText is also limited to 20,000 words, it was able to produce the similar words which are also relevant. This shows that fastText can generate word representations based on subword information, allowing it to recognize and produce similar words even when the exact word is not present in the vocabulary.

6. **Man + Computer - Woman**

```
CBOW (self-trained):
--------------------------------------------------
1. computers               | Similarity: 0.6661
2. programmer              | Similarity: 0.6436
3. hardware                | Similarity: 0.6365
4. colossus                | Similarity: 0.6231
5. software                | Similarity: 0.6174
6. z80                     | Similarity: 0.6145
7. computing               | Similarity: 0.6098
8. cpu                     | Similarity: 0.5991
9. programmed              | Similarity: 0.5990
10. programmers             | Similarity: 0.5978
```

```
Skip-gram (self-trained):
--------------------------------------------------
1. computers               | Similarity: 0.6877
2. microprocessor          | Similarity: 0.6444
3. hardware                | Similarity: 0.6404
4. bios                    | Similarity: 0.6395
5. oisc                    | Similarity: 0.6360
6. electromechanical       | Similarity: 0.6351
7. kildall                 | Similarity: 0.6348
8. eniac                   | Similarity: 0.6279
9. programmed              | Similarity: 0.6271
10. storedprogram           | Similarity: 0.6259
```

```
GloVe (pre-trained):
--------------------------------------------------
1. computers            | Similarity: 0.7162
2. pc                   | Similarity: 0.6365
3. computing            | Similarity: 0.5960
4. systems              | Similarity: 0.5941
5. software             | Similarity: 0.5887
6. electronics          | Similarity: 0.5838
7. hardware             | Similarity: 0.5611
8. system               | Similarity: 0.5610
9. gaming               | Similarity: 0.5552
10. laptop               | Similarity: 0.5550
```

```
fastText (pre-trained):
--------------------------------------------------
1. computers            | Similarity: 0.7398
2. software             | Similarity: 0.6467
3. machine              | Similarity: 0.6364
4. computing            | Similarity: 0.6141
5. hardware             | Similarity: 0.5996
6. computerized         | Similarity: 0.5869
7. electronics          | Similarity: 0.5843
8. laptop               | Similarity: 0.5831
9. internet             | Similarity: 0.5704
10. technology           | Similarity: 0.5643
```

The CBOW and Skip-gram models prioritize terms related to programming and historical computing, which indicates their technical engineering preference. Skip-gram demonstrates capability for grasping advanced historical connections by incorporating both "eniac" and "kildall" terms.

The pre-trained models extract mainly general computing vocabulary from large textual databases, although they show preference for terms like "pc," "gaming," and "technology" instead of programming roles. This higher similarity of the term "programmer" within the CBOW self-trained model indicates that these models might amplify gender stereotypes through male programming associations.

Bias

I used the WEAT bias to examine the socioeconomic bias in intelligence across multiple models. It measured bias by computing the relative similarity between two target word sets when compared to two attribute word sets. Multiple studies show word embeddings typically embed societal biases by linking high-status groups with intelligence while distancing low-status groups from it.

The results varied significantly across models:

- CBOW (0.56) and FastText (0.45) exhibited moderate bias, through their association between intelligence and socioeconomic status.
- Skip-gram (0.25) showed the least bias, suggesting that its word associations are more balanced.
- GloVe (0.77) had the highest bias, strongly reinforcing the stereotype that intelligence is more closely linked to higher socioeconomic status.

These results suggest that pre-trained embeddings (GloVe and fastText) encode stronger socioeconomic biases than self-trained models. The extensive training data the embeddings utilize contains societal stereotypes that emerge from large-scale internet text data. The lower bias score of Skip-gram shows its learning process captures various contexts, which reduces its susceptibility to bias reinforcement.

When these biased embeddings are used as features in ML models, they can allow discrimination by reinforcing harmful stereotypes. For example, an AI-based hiring system would select candidates from privileged backgrounds when terms related to intelligence primarily link to individuals from upper-class backgrounds. The integration of debiasing word embedding methods with ethical AI regulations will help reduce bias when deploying these applications to new platforms.

Classification

```
Comparison:
TF-IDF Model - Accuracy: 1.0000, F1-Score: 1.0000
Mean Pooled Embeddings Model - Accuracy: 0.9583, F1-Score: 0.9583
```

The TF-IDF model demonstrated ideal performance by reaching 100% accuracy together with F1-score. The different vocabularies between the two series produced optimal performance for the TF-IDF model because it successfully located discriminative terms. The lower case preprocessing combined with no stemming or lemmatization operations helped the model achieve its results. The model might have overfit the training data because its performance on new text could be weaker.

Reflection

This assignment enabled me to understand word embeddings better along with their various model representations. The assignment introduced me to the libraries WEFE GloVe and fastText that were completely new to me. My understanding of fundamental code concepts and code readability improved through my exploration of these libraries and my use of documentation and AI assistance. The extension of the previous assignment's classification task deepened my knowledge about machine learning models. Working on this project improved my skills in word embedding usage and bias detection in language models and machine learning applications to NLP tasks.

Generative AI:
I used the ChatGPT model to ask about understanding the code and writing print statements to implement helper functions. I also used it for the 4.4 to implement it logistic regression with the homework 2 dataset. The carbon footprint is 1.57kg of CO_2 (model: ChatGPT, Hardware: GTX 1660 Ti, Time Used: 2.5h, Provider: Google Cloud Platform, Region of Compute: Us-east1).