iNLP Assignment-3: Static Word Embeddings
[2023201044]

<u>Code Explanation:</u>

1. Brown Corpus Preprocessing (brown.py)

- preprocess_brown():
    - Loads the Brown Corpus from the nltk library.
    - Tokenizes sentences and words.
    - Converts words to lowercase and removes punctuation.
    - Returns a processed corpus where each sentence is represented as a list of words.

2. Continuous Bag of Words (CBOW) Model (cbow.py)

- Corpus Processing:
    - Calls preprocess_brown() to load the Brown corpus.
    - Loads a precomputed vocabulary (sorted_vocab.json).
- Vocabulary Building (build_vocab):
    - Assigns indices to words for encoding.
    - Uses frequency-based sorting.
- Training Data Generation (generate_training_data):
    - Creates context-target pairs for CBOW training.
- CBOWNegativeSampling Model:
    - Defines an embedding layer for words and a separate one for outputs.
    - Implements forward propagation using negative sampling.
- Negative Sampling (get_negative_samples):
    - Selects negative words using a weighted probability distribution.
- Training and Hyperparameter Tuning:
    - Optimizes hyperparameters (context size, negative samples, embedding dimensions).
    - Uses early stopping based on WordSim-353 evaluation.
    - The **hyperparameters** used are:
        - half-context window size (2, 3, 5)

        - number of negative samples (2, 5, 7)
        - emdedding size (300)
        - Adam optimizer with 0.001 learning rate
- Evaluation (evaluate_model):
    - Computes word similarity and evaluates embeddings.

3. Skip-gram Model (skipgram.py)

- Vocabulary Construction:
    - Builds vocabulary from the Brown Corpus.
    - Uses frequency-based sorting and filtering.
- Training Data Preparation (generate_training_data):
    - Generates (target, context) word pairs.
- Dataset Class (Word2VecDataset):
    - Implements a PyTorch dataset for Skip-gram training.
    - Generates negative samples dynamically.
- Skip-gram Model (SkipGramBinaryClassifier):
    - Uses embeddings for target and context words.
    - Computes similarity scores with a dot product.
- Training Routine (train_word2vec):
    - Loads dataset and optimizes word embeddings.
    - Uses early stopping based on WordSim-353 evaluation.
    - The **hyperparameters** used are:
        - half-context window size (2, 3)
        - number of negative samples (3, 5, 7)
        - emdedding size (300)
        - Adam optimizer with 0.001 learning rate

4. Singular Value Decomposition (SVD) Model (svd.py)

- Corpus Processing and Vocabulary Creation:
    - Filters words with minimum frequency of 3 (replacement with <UNK>).
    - Creates a co-occurrence matrix.
- Co-occurrence Matrix (build_co_occurrence_matrix):

    - Constructs a word-word matrix using a context window.
    - Optionally applies distance-based weighting and stopword removal.
- SVD Decomposition:
    - Applies Singular Value Decomposition (SVD) to extract word vectors.
    - Normalizes embeddings for better performance.
- Hyperparameter Optimization:
    - Tests different window sizes and embedding dimensions.
    - Evaluates model using WordSim-353.
    - The **hyperparameters** used are:
        - half-context window size (2, 4, 6)
        - number of negative samples (2, 5, 7)
        - emdedding size (100, 200, 300)
        - apply weighting (closer words carry more weighting in the co-occurrence matrix) (False, True)
        - remove stopwords (False, True)

5. Evaluation Scripts (wordsim.py & wordsim_score.py)

- Loading and Evaluating Embeddings (evaluate_model):
    - Loads embeddings from trained models.
    - Computes cosine similarity for word pairs.
    - Evaluates using Spearman's Rank Correlation against human scores.
- Command-line Execution (wordsim.py):
    - Loads pre-trained embeddings from user input.
    - Calls evaluate_model() to compute similarity scores.

Instructions for file execution:

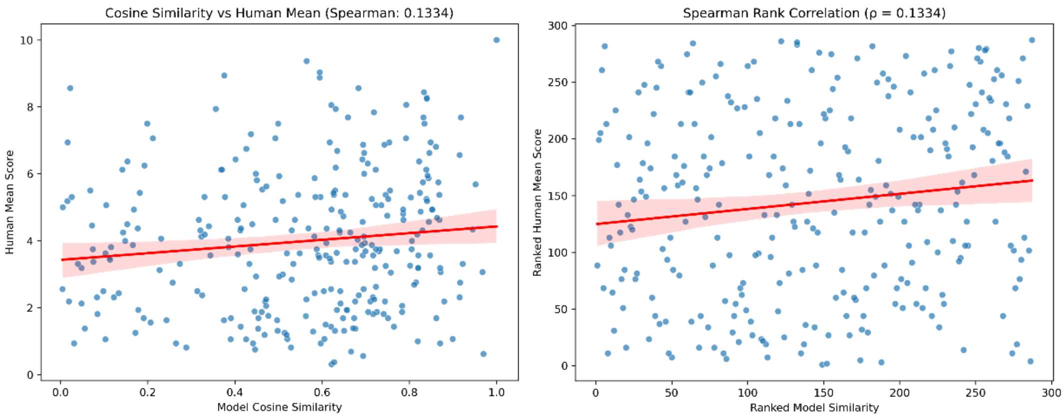
1. To train any model, run the corresponding script as:
   python model_name.py
2. To compute cosine similarities and Spearman's correlation coefficient, run the script as:
   python wordsim.py <embedding_path>

Results:

The pre-trained models can be found here.

1. SVD:
    a. Best hyper-params:
    weighting=False, remove_stopwords=True, window_size=4, embedding_dim=100
    The following are the first 5 and last 5 rows of results from hyper-parameter tuning.

Weighting	Remove Stopwords	Window Size	Embedding Dim	Spearman Correlation
False	True	4	100	0.133439
True	True	4	100	0.127971
False	True	4	300	0.126675
True	True	6	100	0.120008
False	True	6	100	0.118182
False	False	4	200	0.003220
False	False	4	100	-0.007677
False	False	6	300	-0.014319
False	False	6	200	-0.037897
False	False	6	100	-0.052195

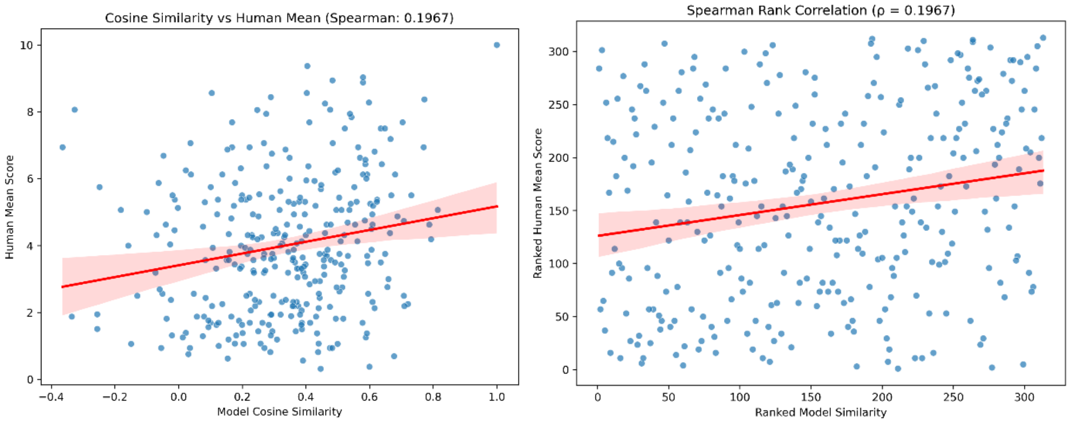


    b. Wordsim results:

Word 1	Word 2	Human (Mean)	Model Similarity
admission	ticket	5.536	0.4626487195491791
alcohol	chemistry	4.125	0.3229874074459076
aluminum	metal	6.625	0.8454661965370178
announcement	effort	2.0625	0.471233069896698
announcement	news	7.1875	0.4363973140716553

    c. Spearman's rank correlation = **0.1334**

2. CBOW:
    a. Best hyper-parameters (20 epochs):
    CONTEXT_SIZE=2, NEGATIVE_SAMPLES=7, EMBEDDING_DIM=300



b. Note that the weights in the first layer are used as embeddings. Alternately, the weights in the second layer or the averaged weights could be used as embeddings.

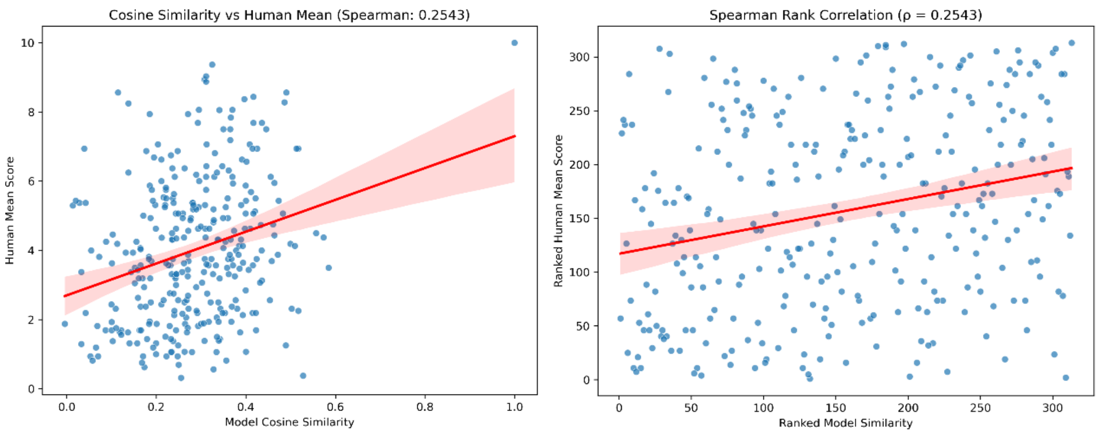
c. Wordsim results:

Word 1	Word 2	Human (Mean)	Model Similarity
admission	ticket	5.536	0.2536108195781708
alcohol	chemistry	4.125	0.5613850355148315
aluminum	metal	6.625	0.569373369216919
announcement	effort	2.0625	0.22620703279972076
announcement	news	7.1875	0.6652630567550659

d. Spearman's rank correlation = **0.1967**

3. SGNS:

a. Best hyper-params (10 epochs):
CONTEXT_SIZE=2, NEGATIVE_SAMPLES=7, EMBEDDING_DIM=300



b. Wordsim Results:

Word 1	Word 2	Human (Mean)	Model Similarity
admission	ticket	5.536	0.21598714590072632
alcohol	chemistry	4.125	0.38459086418151855
aluminum	metal	6.625	0.39151731133461
announcement	effort	2.0625	0.2668076753616333
announcement	news	7.1875	0.36513423919677734

c. Spearman's rank correlation = **0.2543**

Analysis:

The Skip-gram model with Negative Sampling (SGNS) outperforms both CBOW and SVD in the WordSim-353 task. While SVD provides a useful approximation, it struggles with less frequent words due to its reliance on a co-occurrence matrix. CBOW, though faster, is less effective in preserving rare word relationships compared to SGNS..

1. SVD:
   - Benefits: Captures global word co-occurrence relationships; computationally efficient for **offline processing**.
   - Limitations: Less effective for rare words; requires **significant memory** for storing the full co-occurrence matrix.
2. CBOW:
   - Benefits: **Fast training** due to averaging of context vectors and fewer updates; learns representation of more frequent words effectively.
   - Limitations: Loses positional context information, struggles with rare words compared to SGNS.
3. SGNS:
   - Benefits: **Best performance** on WordSim-353, better representation of **less frequent** words; produces higher quality embeddings, capturing subtle semantic nuances.
   - Limitations: Requires **more training time** and computational power due to dynamic negative sampling.