Part A: Creation and Training of a Word2Vec Model on the Knesset Corpus

Step 1: Load and Tokenize the Corpus

The Hebrew sentences from the corpus are tokenized using a regular expression to extract only Hebrew words. These tokenized sentences are stored in a list, with each sentence represented as a list of words.

```
Loading and tokenizing the corpus...
Example tokenized sentences: [['וגא', 'וימזמ', 'וישכע', 'תא', 'רבמ', 'תסנכה', 'ולימינב', 'והינתנ'], ['ויהיאר', 'רבמ', 'תסנכה', 'ולולובד', 'רמה']]
```

Step 2: Train the Word2Vec Model

this step is to train the Word2Vec model using the following parameters, chosen to best represent the semantic and contextual relationships between Hebrew words in the corpus:

1. **vector_size=50**:
    - Each word is represented as a vector in a 50-dimensional space.
    - This size provides a balance between capturing enough semantic detail and computational efficiency. Larger sizes might capture more details but increase memory and processing requirements.
2. **window=5**:
    - This defines the number of context words to consider on either side of the target word.
    - A window size of 5 ensures that the model captures both local context and broader relationships without overly diluting the specific meaning of the target word.
3. **min_count=1**:
    - This ensures that all words in the corpus, even those appearing only once, are included in the vocabulary.
    - Since the corpus is relatively clean and focused on Hebrew, retaining all words allows us to capture unique and rare words that might still hold significant meaning in the context of the data.

These parameter choices aim to optimize the model's ability to generate meaningful word embeddings while remaining computationally manageable.

```
Training the Word2Vec model...
Vocabulary size: 77642
```

Step 3: Save the Model

The trained Word2Vec model is saved to a file (knesset_word2vec.model) for future use, ensuring the embeddings can be reused without retraining.

Step 4: Load the Saved Model

Once the model is saved, it can be reloaded using the Word2Vec.load() function. This allows immediate access to the trained embeddings without needing to retrain the model, saving time and computational resources.

Step 5: Test the Model

A sample word ("ישראל") is tested to verify the model. its 50-dimensional vector is retrieved and printed, ensuring the embeddings are correctly trained and functional.

The vector represents the word's learned semantic meaning based on its context in the corpus:

```
Testing the model with the word: לארשי
Vector for 'לארשי':
[-1.611242   -0.289285   -5.650178    1.9325953   0.34847736  0.6457238
  3.925033    0.28157422 -1.1391962   5.352613    1.6018765   0.9968889
  1.4827744   4.6771517   3.4235566  -1.0183513   5.2459517   2.8327777
  1.010476   -0.6181233  -0.76977444 -0.72547144 -3.87682     2.6622343
  5.6214385   2.7166126  -1.5239255   2.4569736   1.1472204  -2.2942517
  4.0518885  -3.4279134   3.2719793   1.9267967   2.3726625   5.160647
  4.134178    2.4729831  -0.34555942 -4.1952477   6.278238    3.907385
  2.411302    0.25909573 -1.6485372  -4.716001    3.9587681   0.76404107
 -5.9036794   5.5001726 ]
```

Questions:

Question 1 :

The vector_size parameter in Word2Vec defines the number of dimensions for the word embeddings. Each word in the vocabulary is represented as a vector in a continuous space with vector_size dimensions. These dimensions capture the semantic and contextual meaning of the word, based on how it appears in the corpus.

Choosing the appropriate vector_size involves a trade-off between semantic richness and computational efficiency. Larger vectors can capture more detailed relationships between words, providing deeper semantic understanding, but they require more memory and computational resources. On the other hand, smaller vectors are more efficient and faster to train but may lose critical semantic information, especially in complex datasets.

Advantages of Increasing vector_size

1. **More Semantic Detail**: A larger vector_size allows the model to capture more nuanced relationships between words. Words with similar meanings or contexts will be positioned closer in the high-dimensional space.
2. **Improved Performance on Complex Tasks**: Tasks requiring deeper semantic understanding (e.g., word similarity, clustering) benefit from higher-dimensional vectors, as they can encode more information.
3. **Better Generalization**: Larger vectors can represent words more flexibly, reducing overfitting for complex corpora.

Disadvantages of Increasing vector_size

1. **Higher Computational Cost**: Larger vectors require more memory and processing power, increasing both training and inference time.
2. **Risk of Overfitting**: With limited data, a larger vector_size may overfit to specific contexts, reducing generalization.
3. **Diminishing Returns**: Beyond a certain point, increasing vector_size adds minimal improvement in semantic quality, especially for smaller or simpler datasets.

Advantages of Decreasing vector_size

1. **Faster Training and Inference**: Smaller vectors reduce computational requirements, making the model more efficient.
2. **Simplicity**: For smaller datasets, reducing vector_size prevents overfitting and keeps the embeddings concise.
3. **Lower Resource Consumption**: Useful when working on systems with limited memory or processing capabilities.

Disadvantages of Decreasing vector_size

1. **Loss of Semantic Detail**: A smaller vector_size may not fully capture the relationships between words, especially in large or complex datasets.
2. **Reduced Task Performance**: Tasks requiring nuanced understanding (e.g., analogy-solving) may perform worse with smaller vectors.
3. **Poor Generalization**: The model might struggle to represent less frequent or ambiguous words effectively.

Question 2 :

When using a model trained on the Knesset corpus, several challenges may arise due to the unique structure, content, and scope of the data. While the corpus provides valuable domain-specific insights, its limitations could impact the model's ability to generalize, create unbiased embeddings, and remain flexible for broader applications. Below are the key issues that could emerge:

1. Domain-Specific Vocabulary Bias

The Knesset corpus includes terms that are heavily tied to political and legislative contexts, which can lead to an overrepresentation of domain-specific vocabulary. This overrepresentation affects the diversity of word embeddings and their applicability to tasks outside the corpus' focus.

2. Overgeneralization of Morphological Forms

The corpus does not distinguish between morphological forms of words (e.g., singular/plural, masculine/feminine, past/present, or the use of prefixes and suffixes in Hebrew). For example, the model may treat "בא", "באו", "באה" as interchangeable, failing to recognize their distinct meanings. Similarly, while words like "וכיושב", "יושב", "יושבי", "יושבת", "ויושב" share the same root, the model may treat them as entirely different words. This lack of granularity in handling morphological variations can lead to inaccuracies, particularly in tasks that require precise semantic distinctions.

3. Temporal and Speaker-Specific Bias

Language evolves over time, and the Knesset corpus reflects a specific period and set of speakers. This creates a bias toward their speaking styles, vocabulary, and linguistic preferences, which may not reflect the broader use of Hebrew. As a result, the embeddings might not capture emerging linguistic trends or modern terminology.

4. Corpus Size and Limited Generalization

Compared to large-scale corpora used for state-of-the-art models, the Knesset corpus is relatively small. This affects the model's ability to generalize across different contexts and tasks. Subtle word associations that require diverse data may be underrepresented, reducing the richness of the embeddings.

Part B: Word Similarity

Section a:

The trained Word2Vec model was utilized to explore semantic relationships between words by leveraging the most_similar() function. This function retrieves the top 5 words most similar to a given word, along with their corresponding similarity scores.

Cosine similarity was employed as the metric to measure the closeness of word vectors in the embedding space. Words with higher similarity scores reflect stronger semantic relationships, indicating their frequent co-occurrence or similar context in the corpus.

The results of this analysis were saved in a structured format within a file named knesset_similar_words.txt .

Section b:

The sentence embedding was computed as the mean of all word vectors in the sentence:

1. **Calculation Method**:
   - If the sentence contained at least one valid word (i.e., present in the vocabulary), the average vector was calculated using NumPy:

     $$Sentence\ Embedding = \frac{\sum_{i=1}^{k} v_i}{k}$$

     Here, is the vector of the word, and k is the number of valid words in the sentence.
   - For sentences with no valid words (e.g., containing only out-of-vocabulary words), a **zero vector** with the same dimensionality as the Word2Vec vectors was assigned.
2. **Output**:
   - The computed sentence embeddings were stored in a list, with each element representing the embedding of a sentence.

These embeddings provide a dense vector representation of each sentence.

Section c:

In this section, we aimed to analyze sentence similarities within the corpus by comparing randomly selected sentences to others based on their semantic embeddings.

1. Random Sentence Selection:

- Initially, we implemented a process to randomly select 10 sentences from the corpus that contain at least 4 tokens.
- Each random selection was based on a filtered list of sentences, ensuring only those meeting the token count condition were considered.

2. Manual Sentence Selection:

- After observing the results of the random selection, we manually chose 10 specific sentences (using their indices) to ensure meaningful and representative examples from the corpus were analyzed.

3. Calculating Cosine Similarity:

- For each manually selected sentence, we computed the cosine similarity between its embedding and all other sentence embeddings in the corpus.
- The similarity calculation used the semantic embeddings generated earlier. These embeddings capture the contextual and semantic meaning of the words in the sentences.

4. Finding the Most Similar Sentence:

- For each chosen sentence, the model identified the most similar sentence by finding the highest cosine similarity score (excluding the sentence itself).
- This allowed us to determine which sentences in the corpus were most semantically related.

5. Saving the Results:

- The results were saved in a file named knesset_similar_sentences.txt.

Section d:

In the replace_red_words_for_sentences function, we process a list of sentences, each containing specific "red words" to be replaced. For each sentence, the function iterates over the list of red words and attempts to find the most similar word using the Word2Vec model. If positive and/or negative contexts are provided, these are incorporated into the most_similar function to refine the search for a semantically appropriate replacement. The red word is then replaced with the similar word in the sentence. If a red word is not found in the model's vocabulary, it is flagged as such. The function write the original sentence, the modified sentence, and the mapping of replaced words to a the red_words_sentences.txt file in a structured format as expected.

The process was divided into two main stages:

1. **Initial Attempt Without Context (Positive/Negative Words)**:
   Initially, we attempted to find the most similar word for each "red word" without providing any positive or negative words for contextual guidance. This allowed the model to use its trained embeddings to suggest the closest match based solely on general semantic similarity. However, the results obtained during this phase were not accurate, often leading to replacements that were only marginally similar to the original meaning and frequently resulted in sentences with poor grammar or incorrect semantics.

2. **Refining the Replacements with Positive/Negative Words**:
   After analyzing the results from the first attempt, we identified cases where the replacements could be improved by providing additional context. We incorporated positive words to emphasize the desired semantic relationship and negative words to exclude undesired meanings. This refinement ensures that the replacements are more aligned with the intended meaning of the sentences.

Below is a snapshot of the sentences, along with their respective "red words" and the context we defined for them. These context definitions allowed us to experiment with various replacement strategies and improve the accuracy and relevance of the replacements.

```
sentences = [
("בעוד מספר דקות נתחיל את הדיון בנושא השבת החטופים.",
 [
    ("דקות", ["זמן","יחידות","רגעים"], []),
    ("הדיון", ["השיחה", "הישיבה"], [])
 ]),
("בתור יושבת ראש הוועדה, אני מוכנה להאריך את ההסכם באותם תנאים.",
 [
    ("הוועדה", ["הממשלה","השיחה"], []),
    ("אני", ["עצמי","אישי","אחרי","הרגע"], []),
    ("ההסכם", ["חוזה"], ["סכסוך"])
 ]),
("בוקר טוב, אני פותח את הישיבה.",
 [
    ("בוקר", ["יום"], ["לכולם"]),
    ("פותח", ["אקרא","אמשיך","אפעיל"], ["סוף"])
 ]),
("שלום, אנחנו שמחים להודיע שחברינו היקר קיבל קידום.",
 [
    ("שלום", ["אדוני","גברתי"], []),
    ("שמחים", ["מרוצים","מאושרים","מתרגשים","כולנו"], []),
    ("היקר", ["האלוף","הגדול","הטוב"], []),
    ("קידום", ["שיפור","התקדמות"], [])
 ]),
("אין מניעה להמשיך לעסוק בנושא.",
 [
    ("מניעה", ["התנגדות","מגבלה"], ["הרשאה","תמיכה"])
 ])]
```

Below is a snapshot of the results:

```
1: בעוד מספר דקות נתחיל את הדיון בנושא השבת החטופים.: בעוד מספר ספורות נתחיל את המליאה בנושא השבת החטופים.
replaced words: (דקות: ספורות), (הדיון: המליאה)
2: בתור יושבת ראש הוועדה, אני מוכנה להאריך את ההסכם באותם תנאים.: בתור יושבת ראש המפלגה, בהתחלה מוכנה להאריך את הדוח באותם תנאים.
replaced words: (הוועדה: המפלגה), (אני: בהתחלה), (ההסכם: הדוח)
3: בוקר טוב, אני פותח את הישיבה.: רביעי טוב, אני מודיע את הישיבה.
replaced words: (בוקר: רביעי), (פותח: מודיע)
4: שלום, אנחנו שמחים להודיע שחברינו היקר קיבל קידום.: כבוד, אנחנו צפויים להודיע שחברינו חוסיין קיבל יעילות.
replaced words: (שלום: כבוד), (שמחים: צפויים), (היקר: חוסיין), (קידום: יעילות)
5: אין מניעה להמשיך לעסוק בנושא.: אין ספק להמשיך לעסוק בנושא.
replaced words: (מניעה: ספק)
```

Based on the results, we successfully completed the task. While some words posed challenges, we made our best effort to ensure they matched the sentence's grammar and meaning as accurately as possible. The replacement process effectively captured the context and intent of most sentences, demonstrating significant improvement in generating meaningful and grammatically correct substitutions.

Questions:

Question 1:

In evaluating the closest words generated by the model, we observed that the results did not always align with our expectations. The model sometimes struggled to suggest accurate words within specific contexts, revealing challenges in handling complex linguistic nuances or subtle meanings.

However, in certain cases, the generated words showed promise, providing outputs that fit well within the given context. This suggests that the model has potential but requires further tuning and refinement to ensure more consistent and accurate performance across a wider range of scenarios.

Question 2:

In Word2Vec embeddings, antonyms like "אהבה" and "שנאה" or "קל" and "כבד" often appear closer in the vector space than expected, despite their opposite meanings. This behavior arises because Word2Vec captures **contextual similarity** rather than direct semantic relationships. Words frequently used in similar contexts tend to be embedded closer together, even if their meanings are contrasting or opposing.

To validate this, we measured the **cosine similarity** between antonyms using the model.wv.similarity() function. As anticipated, we observed high similarity scores. This confirms that Word2Vec embeddings place antonyms closer due to their shared contextual usage in the training corpus, rather than their semantic differences.

```
Cosine similarity between 'הבהא' and 'שנאש': 0.9519
Cosine similarity between 'לק' and 'דבכ': 0.8418
```

Thus, while antonyms are semantically opposite, their proximity in the embedding space reflects their **shared contexts** in language, emphasizing how Word2Vec represents meaning based on usage patterns.

Question 3:

In our Word2Vec model, we examined three pairs of antonyms:

1. "אהבה" and "שנאה" .
2. "קל" and "כבד" .
3. "חם" and "קר" .

Using cosine similarity, the distances between these antonyms were as expected from the previous section, where we discussed the proximity of antonyms in Word2Vec embeddings. Below are the results of the similarity measurements:

```
Cosine similarity between הבהא and שנאש: 0.9518803954124451
Cosine similarity between לק and דבכ: 0.8418188095092773
Cosine similarity between חם and רק: 0.9294009804725647
```

These high cosine similarity values confirm the hypothesis from the previous section.

Question 4:

Based on the results, the closest words generated by the model generally met expectations, successfully matching the grammatical structure and intended meaning in most cases. This success can be attributed to the use of positive and negative context parameters, which helped refine the model's ability to choose semantically appropriate replacements. Additionally, the Word2Vec model's training on a domain-specific corpus allowed it to capture the contextual relationships and syntactic patterns relevant to the text. However, there were some instances where the replacements were less accurate. These challenges arose particularly with words requiring nuanced understanding of the sentence structure or semantic intent.

Part C: classification

K-Nearest Neighbors (KNN) classification model was built to classify sentences from a given corpus into two categories based on their speaker. The two primary speakers were identified as "ראובן ריבלין" and "א' בורג" .Using pre-trained Word2Vec embeddings, sentence embeddings were calculated and used as input features for the KNN classifier.

Firstly, the corpus was loaded, and the primary speakers were defined based on the findings from Assignment 3. Sentences attributed to these two speakers were filtered for binary classification. Tokenization and sentence embeddings were then calculated, leveraging the same approach used in the previous task. Finally, the KNN model was trained using these embeddings.

The following classification report summarizes the performance of the KNN model:

```
Classification Report:
              precision    recall  f1-score   support

   Speaker 1       0.84      0.94      0.89      2800
   Speaker 2       0.88      0.73      0.80      1761

    accuracy                           0.86      4561
   macro avg       0.86      0.83      0.84      4561
weighted avg       0.86      0.86      0.85      4561
```

Question 1:

Sentence embedding achieved an accuracy of 86%, as shown in the classification report. The performance metrics highlight that the model performed better for Speaker 1, achieving a higher recall (94%) compared to Speaker 2 (73%).

In the previous task, we used two feature sets: TF-IDF features and custom features. When evaluated on the KNN classifier:

- TF-IDF Features resulted in an accuracy of 85%, with balanced recall and precision for both classes (Class 1 and Class 2).
- Custom Features performed the best, achieving a high accuracy of 90%, with precision, recall, and F1-scores of 90% across both classes.

Sentence Embeddings vs. TF-IDF Features:

- Sentence embeddings yielded slightly better performance (86% vs. 85%) than TF-IDF features. This is likely due to embeddings capturing semantic relationships between words and sentences better than word frequency-based TF-IDF.

Sentence Embeddings vs. Custom Features:

- The custom features outperformed embeddings significantly, achieving an accuracy of 90%. Custom features likely included domain-specific metadata, which provided more contextual information relevant to speaker classification.

Part D: Using DictaBERT

We began by loading the DictaBERT model and tokenizer to process sentences containing masked tokens. The sentences with [*] were read from the input file, and the placeholders were replaced with [MASK]. After tokenizing the sentences, they were passed through the DictaBERT model to predict the most likely tokens for each [MASK]. The predicted tokens were decoded into Hebrew words and iteratively replaced back into the original sentences to reconstruct them. Finally, we saved the results, including the original masked sentences, reconstructed sentences, and the predicted tokens, in the required format to the output file.

Questions:

Question 1:

Yes, the results confirm that the model successfully generated sentences that maintain contextual coherence and grammatical accuracy. By leveraging the DictaBERT model's ability to predict and generate masked tokens, the reconstructed sentences align with the original context while adhering to linguistic conventions.

Question 2:

Yes, the model provided completions that closely align with the missing words' intended meanings. For instance:

1. **Original Sentence**: "אי אפשר להטיל את זה על אדם יחיד."
    - **Masked Sentence**: "אי אפשר להטיל את [*] על אדם יחיד."
    - **DictaBERT Sentence**: "אי אפשר להטיל את האחריות על אדם יחיד."
    - **Generated Token**: "האחריות"

    This example illustrates how the model accurately predicted the word "האחריות," which is semantically consistent and contextually appropriate for the masked sentence.

2. **Original Sentence**: " יש לנו קנים במושבים , בקיבוצים , בערים , בכפרים הערביים , בכפרים הדרוזיים. "
    - **Masked Sentence**: " יש לנו קנים במושבים , בקיבוצים , בערים [*] בכפרים [*] , בכפרים הדרוזיים. "

- **DictaBERT Sentence**: " יש לנו קנים במושבים , בקיבוצים , בערים , בכפרים הערביים , בכפרים הדרוזיים."
- **Generated Tokens**: ",, הערביים"

In this case, the model perfectly reconstructed the original sentence by correctly predicting the exact tokens.

And so on for all the sentences, the model consistently generated appropriate predictions based on the given context, highlighting its effectiveness in handling masked language tasks within this corpus.

Question 3:

Yes, the results obtained in this task show improvement compared to those from assignment 2. The use of DictaBERT, allowed for enhanced understanding of the legal and formal context, leading to more accurate predictions of missing words.

For example:

original_sentence: "לא היה לי ספק בתשובה ."

- **assignment 2 result:** "לא היה אפשר ספק בתשובה ."
- **current task result:** "לא היה לי ספק בתשובה ."

original_sentence: " על , 1968 - ב כנסת בכנסת שהוחלט לכך שמה להסכים לא וחובתנו זכותנו מדיניות הרפורמה , לא יבוטל שלא בדרך של החלטה לאומית ."

- **assignment 2 result:** " - ב כנסת בכנסת שהוחלט לכך שמה להסכים לא וחובתנו אני על מדיניות הרפורמה , לא רק שלא בדרך של החלטה לאומית , 1968 ."
- **current task result:** " - ב כנסת בכנסת שהוחלט לכך שמה להסכים לא וחובתנו זכותנו על מדיניות הרפורמה , לא ייעשה שלא בדרך של החלטה לאומית , 1968 ."

These examples highlight how **DictaBERT** excels in both grammatical accuracy and contextual understanding, producing results that are not only more syntactically precise but also semantically aligned with the original intent of the sentences.

Question 4:

Yes, there were sentences where the model performed less accurately. For example, sentences with ambiguous contexts or rare terms posed a challenge for the model. The primary reason could be the limitations of the training corpus of DictaBERT, which may lack sufficient examples or context for these specific scenarios. Additionally, if the surrounding context did not provide strong clues, the predictions tended to be less precise, leading to weaker results for such sentences.

Question 5:

The model is more likely to fail or provide suboptimal results in the following cases:

1. **Ambiguous Contexts:** Sentences where the masked word could have multiple plausible completions, making it harder for the model to pinpoint the most contextually relevant prediction.
2. **Rare or Domain-Specific Terms:** If the corpus used to train DictaBERT did not include sufficient examples of these terms, the model may struggle to predict accurately.
3. **Grammatical Complexity:** Sentences with uncommon syntactic structures may confuse the model, leading to errors.
4. **Short Sentences:** When sentences are too short, there is less context available for the model to make an informed prediction, increasing the likelihood of errors.
5. **Continuous Masked Tokens:** Sentences where multiple consecutive tokens are masked are particularly challenging, as the model must

In such scenarios, the reliance on broader, more diverse training data or additional fine-tuning could help reduce these risks.