NLP 2025 Lab 2 Report: Word Embeddings and Information Retrieval

Marta Adamowska[*] **Katarina Dvornák**[†] **Pavels Sorocenko**[‡]

Exercise 1: Clean function

1 shows a code snippet used to preprocess the text before further analysis. First, all characters are converted to lowercase and punctuation is removed. Then the text is split into separate words. Words that are not stop words are lemmatized and added to the cleaned version of the text.

Exercise 2: Tokenize function

To perform tokenization, we utilize the NLTK library. This process is done by executing the following line of code:

```
tokens = word_tokenize(text)
```

Exercise 3: Extracting vocabulary counts

The function *extract_vocabulary_counts* returns a Counter object of the form {word: count}. Based on this function we will create a vocabulary containing 10 000 most frequent words.

Exercise 4: Bag of Words

The Bag of Words method is implemented in the *bag_of_words* function. It iterates through all words in a given sentence. For each word found in the vocabulary, it retrieves its index and increments the corresponding count in a NumPy array at the appropriate position. The function returns a NumPy array of size vocab_size, where each element represents the count of the corresponding word in the vocabulary.

Exercise 5: Cosine Similarity between two vectors

Cosine Similarity between two vectors is calculated by *cosine_similarity* function, using the following formula:

$$\text{cosine\_similarity}(v_1, v_2) = \frac{v_1 \cdot v_2}{\|v_1\| \|v_2\|} \quad (1)$$

We make sure that the denominator is non zero. In case the magnitude of vector 1 or 2 is 0, then we return 0.

Exercise 6: Cosine Similarity between a vector and an array of vectors

Cosine Similarity a vector and an array of vectors in a function *cosine_similarity_1_to_n* is calculated using the same formula as the Cosine Similarity between two vectors. However noe we return a NumPy array containing the cosine similarity between the vector and all the other vectors.

Exercise 7: Analyzing and improving BOW search results

Performance Evaluation

We evaluated the performance of a Bag-of-Words (BoW)-based sentence retrieval system on a dataset of news headlines and compressions. The system was tested using a set of queries, and for each query, we computed the cosine similarity between its BoW vector and the BoW vectors of the sentences in the test set. The top-3 most similar sentences were retrieved and analyzed.

Successful Queries (Expected Results)

The BoW model performed well when there was high lexical overlap between the query and dataset sentences. These included:

- **"man died"** – retrieved multiple headlines with exact phrasing such as "A Bamfield man died in a fight...". The words "man" and "died" are frequent and literal.
- **"police arrest"** – matched well with crime-related headlines, e.g., "Police arrest man after

[*] m.adamowska@student.maastrichtuniversity.nl
[†] k.dvornak@student.maastrichtuniversity.nl
[‡] p.sorocenko@student.maastrichtuniversity.nl

chase". These terms are highly representative of standard reporting patterns.

- **“house fire”** – matched directly with phrases like "house fire in Noble" or "crews respond to house fire". High overlap with frequent phrases ensured success.

These successes highlight that BoW excels when the query contains high-frequency, literal terms with clear overlap.

Unsuccessful Queries (Unexpected Results)

BoW failed in several cases due to known limitations, which we categorize as follows:

1. Homonyms: **“river bank”** and **“money bank”** returned the same results, all referring to financial institutions. BoW treats "bank" as a single token without understanding its meaning in context. It cannot distinguish between geographic and financial senses of the word.

2. Word Ordering and Role Confusion: The queries **“suspect shoots”** and **“shoots suspect”** produced identical results, despite the fact that the roles of subject and object are reversed. BoW ignores word order, so both are treated as unordered bags containing the same words.

3. Rare Words: The query **“he discussed neutrinos”** performed poorly, returning generic discussion-related headlines while ignoring the domain-specific term "neutrinos". This occurred because "neutrinos" is rare in the dataset and possibly absent or downweighted in the vocabulary, leading to low influence on similarity scores.

4. Contextual Failures: The query **“a woman reading a book”** returned results like "book club meeting", missing the core action and the subject. BoW matched on the word "book" alone and lacked understanding of who is doing what.

Improvements and Robustness of BoW Search

Despite several optimizations to the Bag-of-Words (BoW) pipeline, the overall retrieval performance did not significantly improve. To improve relevance in queries involving rare or important terms, we implemented a custom `bag_of_words_boost()` function. This variant of BoW boosts the counts of rare words using inverse word frequency:

```
if vocab_counts:
    word_freq = vocab_counts.get(word, 1)
   boost = 1 + np.log10(1 + 1 / word_freq)
     bow[idx] += boost
```

However, this change had only a marginal effect on similarity scores and ranking. This is likely because cosine similarity is direction-sensitive but not scale-sensitive, so boosting magnitudes did not strongly alter vector alignment.

We also refactored the `clean()` function to initialize the stopword list and lemmatizer only once globally, reducing redundant computation during batch processing. The `cosine_similarity()` function was also updated to guard against numerical instability and zero-vector issues.

Remaining Limitations of BoW

Even with improvements, BoW has inherent limitations:

- No semantics
- Word order ignored
- No disambiguation
- OOV issues

Why More Advanced Fixes Were Not Pursued

Addressing the above limitations would require switching from BoW to more advanced models such as:

- **TF-IDF:** Adds importance weighting to rare terms and downweights frequent words.
- **Word Embeddings (e.g., GloVe):** Captures semantic similarity even between different words.
- **Contextual Embeddings (e.g., BERT):** Understands context, syntax, and paraphrases.

Despite its limitations, the implemented BoW pipeline is robust for its intended purpose:

- It consistently retrieves relevant headlines for keyword-rich, literal queries such as `"man died"` or `"house fire"`, demonstrating strong lexical matching.
- Edge cases are handled : the system avoids crashes or instability even when queries contain out-of-vocabulary terms or produce zero vectors.

- The preprocessing (including tokenization, stopword removal, and lemmatization) is cleanly structured, consistent across the dataset.
- Cosine similarity calculations are numerically stable, with protections against divide-by-zero errors and normalization issues.

Exercise 8: Inverse Document Frequency (IDF)

In this exercise, we implemented Inverse Document Frequency (IDF) part of the TF-IDF algorithm. The IDF measures how informative a word is across a dataset by emphasizing rare words and under-weighing frequent ones. The IDF formula:

$$\text{IDF}(w) = \log_{10}\left(\frac{N}{df_w + 1}\right) \tag{2}$$

IDF Implementation

```
def calculate_idf(bows):
    N = bows.shape[0]
    df = np.sum(bows > 0, axis=0)
    idf = np.log10(N / (df + 1))
    return idf
```

This function computes the IDF for each word in the vocabulary using NumPy.

Exercise 9: TF-IDF

We implemented TF-IDF, a weighted representation of words in documents, which combines term frequency (TF) and inverse document frequency (IDF). It is computed as:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t) \tag{3}$$

TF-IDF Implementation

```
def calculate_tfidf(bows, idf):
    tf = np.log10(1 + bows)
    tfidf = tf * idf
    return tfidf
```

TF-IDF Search Analysis

We tested the Tf-IDF based search engine using various new headline queries, and comparing TF-IDF vectors via cosine similarity.

Expected Results

TF-IDF performed well when queries included rare and specific terms:

- `"house explosion"` — correctly retrieved articles with building-related explosions.
- `"earthquake damage"` — matched reports describing structural impact from earthquakes.
- `"wildfire"` — surfaced regional fire-related updates and statistics.
- `"woman killed"` — returned fatality-focused articles effectively.

Unexpected Results

The following limitations highlight, that while TF-IDF is effective in certain tasks, it may not be sufficient for applications requiring deeper understanding of language.

- **Out-of-Vocabulary (OOV):** For example, the query `"ebola"` returned completely irrelevant results. This occurred because the word `"ebola"` was not present in the vocabulary extracted from the training data. Since TF-IDF relies on exact word matches, any term not seen during training is ignored, resulting in a zero vector and causing similarity scores to be zero for all documents.
- **Spelling Sensitivity:** The query `"volcano erruption"` failed to retrieve articles about volcanic eruptions. This was due to a spelling mistake in the word `"eruption"`. TF-IDF does not include any error tolerance or fuzzy matching, so even a minor typo renders the query term unmatchable with the correct word in the corpus. This limitation can be especially problematic in real-world applications where users frequently make typos or spelling variations.
- **Lack of Synonym and Paraphrase Understanding:**
  - The query `"child kidnapped"` returned different results from `"child abducted"`, even though the two phrases have the same meaning. TF-IDF treats each word as an independent token and lacks any knowledge of synonymy.
  - Similarly, `"female dies"` produced weaker results compared to `"woman killed"`, although both describe the same event. This shows the limitations of

TF-IDF in capturing linguistic variability and paraphrasing.

Since TF-IDF has no semantic understanding, it cannot recognize that different words or phrases may represent the same concept. This makes it brittle in tasks where natural language diversity is high.

- **Word Order Insensitivity:** TF-IDF, like Bag-of-Words, is completely order-agnostic. The queries `"shoots suspect"` and `"suspect shoots"` yield similar results, despite representing different roles of the entities involved. In many cases, word order can change the meaning (e.g., subject vs object), but TF-IDF cannot capture such syntactic nuances.

Comparison: TF-IDF vs. Bag-of-Words Representation

In this section, we evaluate how the TF-IDF representation compares to the Bag-of-Words (BoW) model for headline similarity and search relevance. While both are sparse vector-based representations, TF-IDF incorporates term frequency and document rarity, giving it a significant edge in many scenarios.

Observed Improvements with TF-IDF

TF-IDF often outperformed BoW by boosting the influence of rare, informative words while down-weighting generic or frequent terms. This helped improve ranking in several query categories:

- **Query: `police arrest`**
  BoW Top Match Similarity: 0.5883
  TF-IDF Top Match Similarity: 0.4827
  Although TF-IDF had slightly lower top similarity scores numerically, the retrieved results were more focused (e.g., "Ocean City police report arrests") vs. off-topic fatal incidents in BoW results.

- **Query: `house fire`**
  TF-IDF yielded more focused matches (e.g., "Dryer sparks house fire") than BoW, which occasionally favored unrelated results with high-frequency terms like "firefighters".

- **Query: `dog barking`**
  BoW returned highly ranked but weakly related results such as "Why do dogs howl?" due to keyword overlap. TF-IDF preferred more semantically meaningful matches like "Dog wakes owner during fire".

Similarities and Shared Limitations

Despite improvements, TF-IDF shares many limitations with BoW:

- **Lack of Semantics:** Queries like "`man died`" and "`man killed`" yield different results due to vocabulary mismatch.

- **Word Order Ignorance:** Queries "`shoots suspect`" and "`suspect shoots`" return nearly identical matches in both models.

- **No Tolerance for Typos:** Misspellings like "`volcano erruption`" lead to failure in both models.

- **Out-of-Vocabulary (OOV):** Words like "`ebola`", "`neutrinos`" produce vectors of all zeros, yielding irrelevant results.

While BoW sometimes gives higher cosine similarities, the relevance and specificity of TF-IDF results are often more meaningful. TF-IDF provides a more context-aware retrieval by prioritizing rare and informative terms. It often retrieves more relevant headlines for keyword-rich queries and penalizes overly generic ones. However, it still lacks semantic awareness, synonym handling, and robustness to noise. These issues that could be addressed with modern embedding models.

Exercise 10: Plotting similarities between words

To explore the semantic properties of GloVe word embeddings, we computed cosine similarity matrices for various sets of related and unrelated words. The resulting matrices help visualize how similar the embeddings of words are in the vector space, revealing both meaningful associations and limitations. For each group of words, we used the pretrained GloVe embeddings with dimension 100 to compute a square cosine similarity matrix, where each value ranges from 0 (no similarity) to 1 (identical vectors). We experimented with different semantic domains including emotions, geography, professions, animals, etc.

Similarity Matrix Implementation

```
def get_similarity_matrix(word_list, model):
    size = len(word_list)
    sim_matrix = np.zeros((size, size))
    for i in range(size):
        for j in range(size):
```

```
        try:
            sim_matrix[i, j] =
       model.similarity(word_list[i],
            word_list[j])
        except KeyError:
            sim_matrix[i, j] = np.nan
return sim_matrix
```

Results and Interpretation

- **Emotions and Concepts:** `love`, `hate`, `life`, `equal`, `alive`, `dead`
  - Semantic opposites such as `alive-dead` still show high similarity (0.71), suggesting embeddings reflect topic proximity, not just sentiment.
  - Pairs like `love-life` (0.73) and `love-hate` (0.57) illustrate related emotional or conceptual fields.
  - `equal-dead` (0.18) has low similarity, reflecting distinct meanings and contexts.
- **Countries and Capitals:** `france`, `paris`, `italy`, `rome`, `germany`, `berlin`
  - Strong relations like `france-paris` (0.75), `germany-berlin` (0.73) validate that embeddings capture geopolitical associations.
  - Cross-country similarity is also notable (e.g., `france-germany`: 0.73), likely due to shared context in geopolitical discourse.
- **Technology Terms:** `computer`, `keyboard`, `internet`, `email`, `phone`, `screen`
  - `computer-internet` (0.73) and `internet-phone` (0.73) show meaningful connectivity in the digital domain.
  - Lower values for `email-screen` (0.25) reflects their functional difference: communication vs. display.
- **Food and Produce:** `apple`, `banana`, `orange`, `fruit`, `vegetable`, `cucumber`, `carrot`
  - High similarity between `fruit` and fruits/vegetables (`fruit-banana`: 0.65, `fruit-vegetable`: 0.69) reflects the categorical nature.
  - `vegetable-apple` (0.32) is lower, suggesting some separation between categories.
- **Polysemy – Word Sense Ambiguity:** `bank`, `river`, `loan`, `water`, `money`, `shore`
  - `bank-loan` (0.55), `bank-money` (0.57): financial sense.
  - `bank-river` (0.33), `river-shore` (0.62): geographic/physical sense.
  - Shows how embeddings capture both meanings in the same vector, showing that ambiguity is a limitation.
- **Animals:** `dog`, `cat`, `mouse`, `lion`, `tiger`, `wolf`
  - `dog-cat` (0.88) shows high similarity due to common domestic context.
  - Wild animals like `lion-tiger` (0.57), `lion-wolf` (0.59) are also well-clustered.
- **Emotions:** `happy`, `sad`, `angry`, `joyful`, `depressed`, `excited`
  - Positives like `happy-excited` (0.76), `joyful-sad` (0.59) cluster together.
  - `joyful-depressed` (0.21) is low, aligning with psychological contrast.
- **Occupations and Gender:** `doctor`, `nurse`, `teacher`, `housekeeper`, `scientist`, `engineer`, `man`, `woman`
  - `man-woman` (0.83): very strong similarity, suggesting gender relation is prominent in embeddings.
  - Gender-stereotyped roles: `nurse-woman` (0.61), `housekeeper-woman` (0.46), possibly reflecting corpus bias.
  - Low similarity between `engineer-housekeeper` (0.13) reveals occupational segregation.

Insights and Conclusion

- Embeddings effectively encode topical similarity (e.g., `bank-loan`, `dog-cat`) and even oppositional relationships (e.g., `alive-dead`).
- They struggle with polysemy (e.g., `bank`) and semantic roles (e.g., `engineer-woman`), showing that they represent surface-level co-occurrence rather than deep understanding.
- Observed patterns also reveal biases in the training data (e.g., gender-profession associations).

The similarity matrices for all word sets are shown in the Appendix.

Exercise 11: Other pre-trained word embeddings

For comparison, we implemented GloVe embeddings with dimensions of 200 and 300. We computed and visualized new cosine similarity matrices using the same algorithm as in exercise 10.

The overall cosine similarity values between word vectors tend to decrease as the embedding dimension increases, resulting in a lower average similarity across matrices. This occurs because higher-dimensional vectors become more orthogonal, more spread out in space. Consequently, the similarity matrices generated in exercise 11 have a lower contrast and a less distinct visual of colors. Lower-dimensional models are computationally lighter and highlight broader patterns, often providing higher similarity values even for loosely related words. In contrast, higher-dimensional embeddings capture more nuanced and selective relationships, making them better suited for capturing precise relationships.

The similarity matrices for all word sets are shown in the Appendix.

Exercise 12: Sentence Embedding

Sentence embedding worked as follows: if a word of a sentence is present in a pretrained model GloVe, it is being added to the embeddings array. If at least one word was present in GloVe, the mean of the array was calculated, giving us an embedded sentence. Otherwise, if all the words were OOV, the zero vector of appropriate length is being returned.

This is a good method to deal with the unknown sentences, it prevents any hard errors from happening and it is not hard to implement, additionally it is predictable. On the other hand there exist more sophisticated techniques, like <UNK> token vector. It encodes the average of all vectors in the model. Is similar to zeroes, but it has some semantics left in it. A BPE model can be trained, and it can greatly generalize the unknowns, but the runtime significantly increases.

Exercise 13: Analyze sentence embeddings

The sentence embedding for queries showed us that it works well with High semantic similarity queries ("police arrest", "house fire", "man died", "money bank"), worse with low semantic similarity, the results were especially bad if the words in there are rare ("neutrinos"), and word-sense disambiguation-wise model performed quite well.

In general, the model works as expected, since it is based on an unknown dataset model, captured generality, so more sophisticated queries that do not contain unknown words can be used on the model, but it struggled with specificity of the queries, or at least performed worse than BoW/TFIDF in that sence. It can be easily explained, because the generality of a model helps in this case find "synonims" that is impossible with BoW/TFIDF model, it somewhat capturees context due to vector representation, whereas BoW/TFIDF tends to return more literal matches.

To sum up: word embeddings show great performance for general, semantically rich queries, but BoW/TF-IDF may outperform in domain-specific or literal match cases where context is less ambiguous and/or limited vocabulary is used.

Exercise 14: Cosine similarity between two sets of vectors

It has been implemented based on the previous Cosine similarity module. The only difference is that both vectors need to be normalized, and to perform the dot product operation on two matrices, we need to transpose one of them twice.

Exercise 15: Evaluating retrieval methods

Clearly, recall rises with k, there is a correlation. The results capture that embeddings are the worst in terms of recall, which is easily explainable, since embeddings tend to capture semantic similarity, which can lead to lack of exactness, and this lowers recall. BoW and TF-IDF differ slightly, TF-IDF being a better model because of more layers of complexity, like reduction of impact of common words and more weight on rare ones.

Exercise 16: Improving retrieval

"The most promising" retrieval method is a very ambiguous statement, but for me, since we are dealing with plenty of new data everyday, and the ambiguity of many of them is unavoidable, to my mind, the embedding model is the most promising one.

The bigger models than glove, such as fasttext or conceptnet, increase the recalletK scores since it captures less unknown words and the model becomes more "exact"

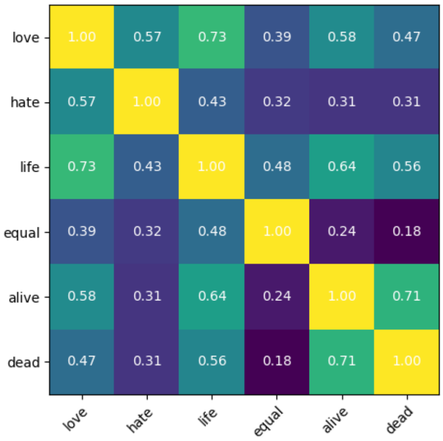
References

.1 Exercise 1: Clean function

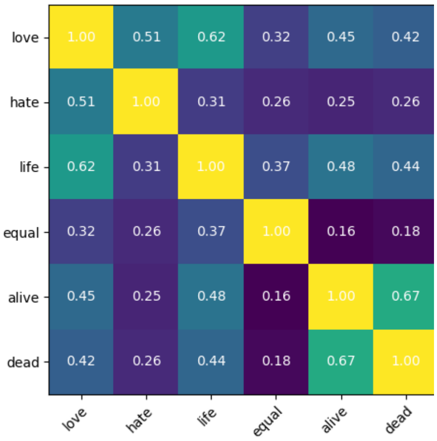
.2 Exercises 10 & 11: Side-by-Side Similarity Matrices with different GloVe Dimensions

```
# make all letters lowercase
text = text.lower()

# remove punctuation
for char in text:
    if char in string.punctuation:
        text = text.replace(char,"")

# split text into words
text_words = text.split()

# remove stop words
cleared_words = []
lemmantizer = WordNetLemmatizer()
for word in text_words:
    if word not in stopwords.words('english'):
        cleared_words.append(lemmantizer.lemmatize(word))

text = ' '.join(cleared_words)
```

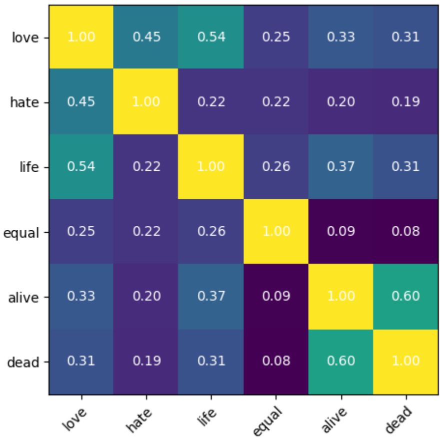
Figure 1: Clean function implementation.



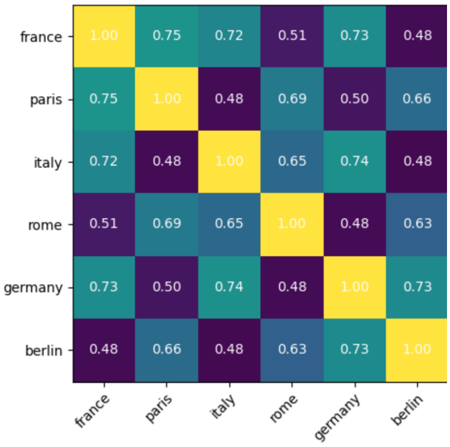
Ex.10 GloVe dimension=100



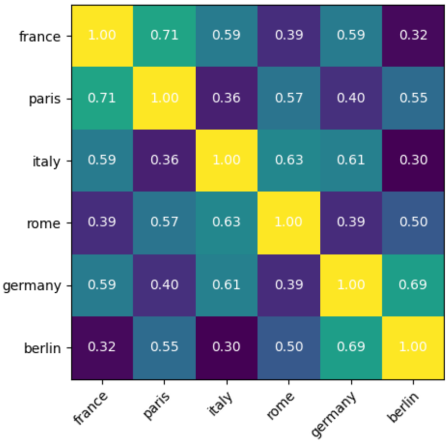
Ex.11 GloVe dimension=200



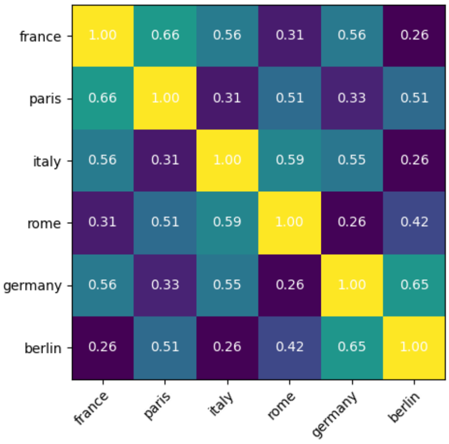
Ex.11 GloVe dimension=300



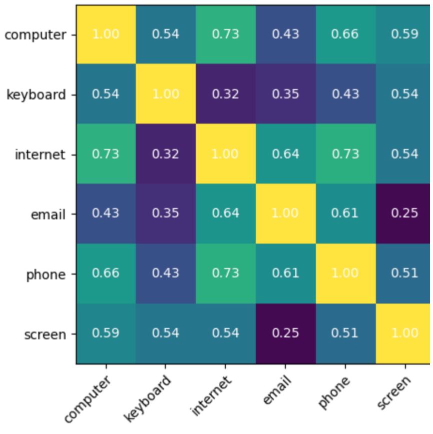
Ex.10 GloVe dimension=100



Ex.11 GloVe dimension=200

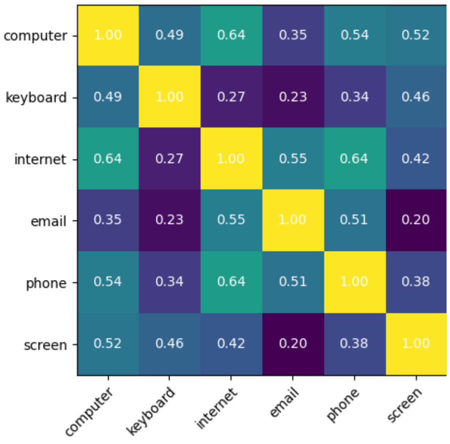


Ex.11 GloVe dimension=300



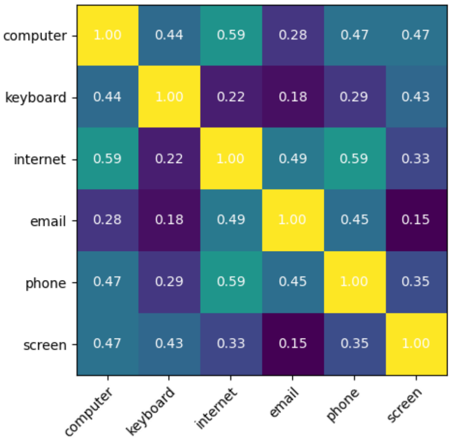
	computer	keyboard	internet	email	phone	screen
computer	1.00	0.54	0.73	0.43	0.66	0.59
keyboard	0.54	1.00	0.32	0.35	0.43	0.54
internet	0.73	0.32	1.00	0.64	0.73	0.54
email	0.43	0.35	0.64	1.00	0.61	0.25
phone	0.66	0.43	0.73	0.61	1.00	0.51
screen	0.59	0.54	0.54	0.25	0.51	1.00

Ex.10 GloVe dimension=100



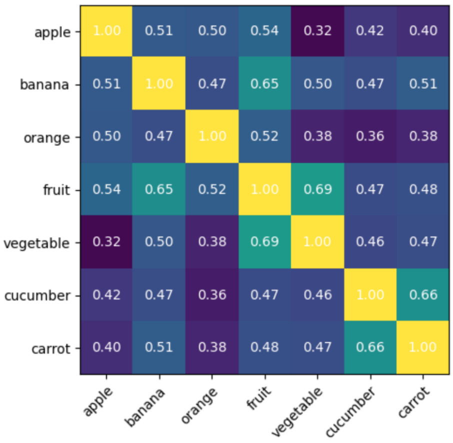
	computer	keyboard	internet	email	phone	screen
computer	1.00	0.49	0.64	0.35	0.54	0.52
keyboard	0.49	1.00	0.27	0.23	0.34	0.46
internet	0.64	0.27	1.00	0.55	0.64	0.42
email	0.35	0.23	0.55	1.00	0.51	0.20
phone	0.54	0.34	0.64	0.51	1.00	0.38
screen	0.52	0.46	0.42	0.20	0.38	1.00

Ex.11 GloVe dimension=200



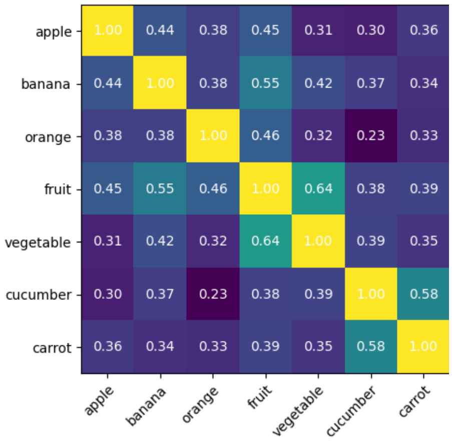
	computer	keyboard	internet	email	phone	screen
computer	1.00	0.44	0.59	0.28	0.47	0.47
keyboard	0.44	1.00	0.22	0.18	0.29	0.43
internet	0.59	0.22	1.00	0.49	0.59	0.33
email	0.28	0.18	0.49	1.00	0.45	0.15
phone	0.47	0.29	0.59	0.45	1.00	0.35
screen	0.47	0.43	0.33	0.15	0.35	1.00

Ex.11 GloVe dimension=300



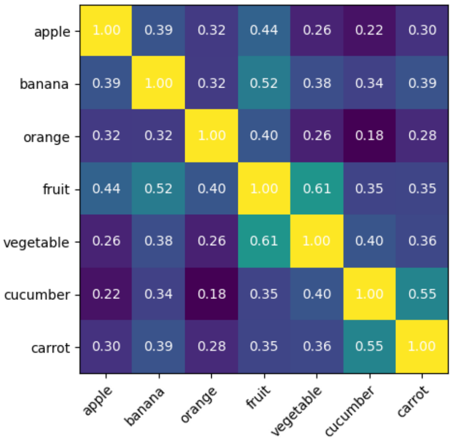
	apple	banana	orange	fruit	vegetable	cucumber	carrot
apple	1.00	0.51	0.50	0.54	0.32	0.42	0.40
banana	0.51	1.00	0.47	0.65	0.50	0.47	0.51
orange	0.50	0.47	1.00	0.52	0.38	0.36	0.38
fruit	0.54	0.65	0.52	1.00	0.69	0.47	0.48
vegetable	0.32	0.50	0.38	0.69	1.00	0.46	0.47
cucumber	0.42	0.47	0.36	0.47	0.46	1.00	0.66
carrot	0.40	0.51	0.38	0.48	0.47	0.66	1.00

Ex.10 GloVe dimension=100



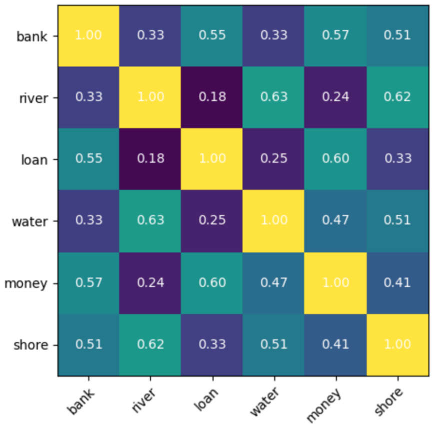
	apple	banana	orange	fruit	vegetable	cucumber	carrot
apple	1.00	0.44	0.38	0.45	0.31	0.30	0.36
banana	0.44	1.00	0.38	0.55	0.42	0.37	0.34
orange	0.38	0.38	1.00	0.46	0.32	0.23	0.33
fruit	0.45	0.55	0.46	1.00	0.64	0.38	0.39
vegetable	0.31	0.42	0.32	0.64	1.00	0.39	0.35
cucumber	0.30	0.37	0.23	0.38	0.39	1.00	0.58
carrot	0.36	0.34	0.33	0.39	0.35	0.58	1.00

Ex.11 GloVe dimension=200



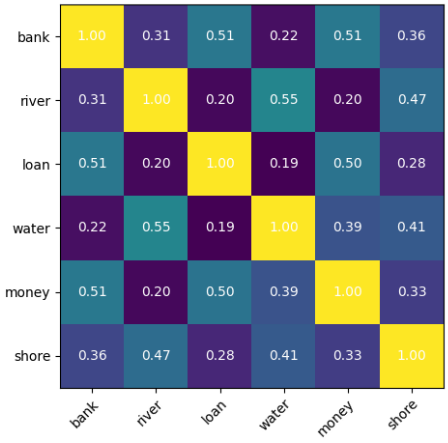
	apple	banana	orange	fruit	vegetable	cucumber	carrot
apple	1.00	0.39	0.32	0.44	0.26	0.22	0.30
banana	0.39	1.00	0.32	0.52	0.38	0.34	0.39
orange	0.32	0.32	1.00	0.40	0.26	0.18	0.28
fruit	0.44	0.52	0.40	1.00	0.61	0.35	0.35
vegetable	0.26	0.38	0.26	0.61	1.00	0.40	0.36
cucumber	0.22	0.34	0.18	0.35	0.40	1.00	0.55
carrot	0.30	0.39	0.28	0.35	0.36	0.55	1.00

Ex.11 GloVe dimension=300



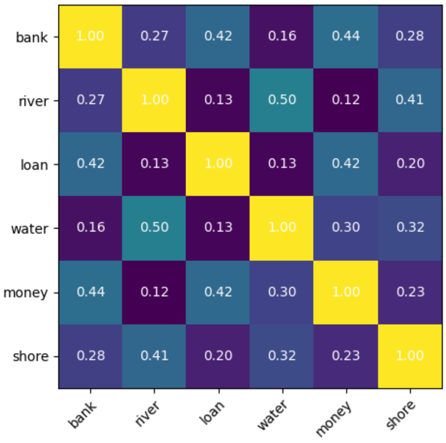
	bank	river	loan	water	money	shore
bank	1.00	0.33	0.55	0.33	0.57	0.51
river	0.33	1.00	0.18	0.63	0.24	0.62
loan	0.55	0.18	1.00	0.25	0.60	0.33
water	0.33	0.63	0.25	1.00	0.47	0.51
money	0.57	0.24	0.60	0.47	1.00	0.41
shore	0.51	0.62	0.33	0.51	0.41	1.00

Ex.10 GloVe dimension=100



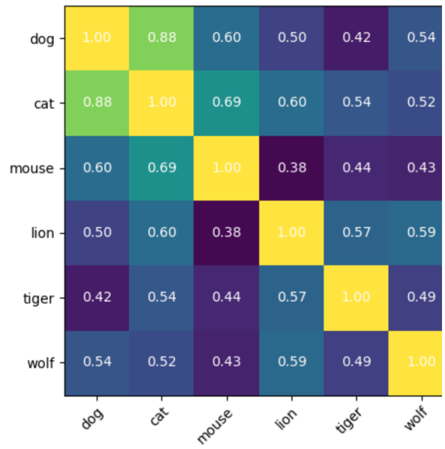
	bank	river	loan	water	money	shore
bank	1.00	0.31	0.51	0.22	0.51	0.36
river	0.31	1.00	0.20	0.55	0.20	0.47
loan	0.51	0.20	1.00	0.19	0.50	0.28
water	0.22	0.55	0.19	1.00	0.39	0.41
money	0.51	0.20	0.50	0.39	1.00	0.33
shore	0.36	0.47	0.28	0.41	0.33	1.00

Ex.11 GloVe dimension=200



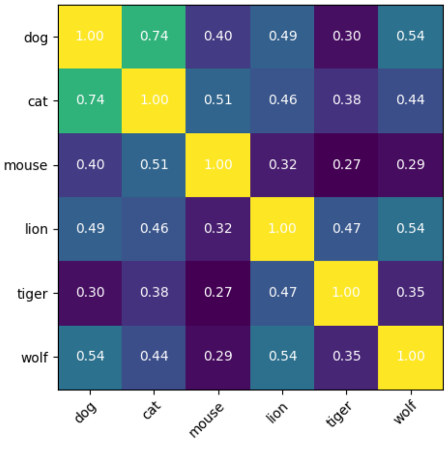
	bank	river	loan	water	money	shore
bank	1.00	0.27	0.42	0.16	0.44	0.28
river	0.27	1.00	0.13	0.50	0.12	0.41
loan	0.42	0.13	1.00	0.13	0.42	0.20
water	0.16	0.50	0.13	1.00	0.30	0.32
money	0.44	0.12	0.42	0.30	1.00	0.23
shore	0.28	0.41	0.20	0.32	0.23	1.00

Ex.11 GloVe dimension=300



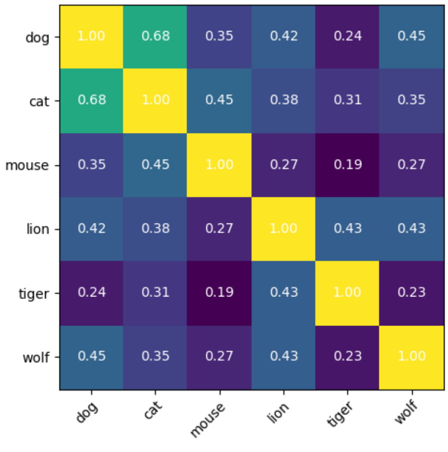
	dog	cat	mouse	lion	tiger	wolf
dog	1.00	0.88	0.60	0.50	0.42	0.54
cat	0.88	1.00	0.69	0.60	0.54	0.52
mouse	0.60	0.69	1.00	0.38	0.44	0.43
lion	0.50	0.60	0.38	1.00	0.57	0.59
tiger	0.42	0.54	0.44	0.57	1.00	0.49
wolf	0.54	0.52	0.43	0.59	0.49	1.00

Ex.10 GloVe dimension=100



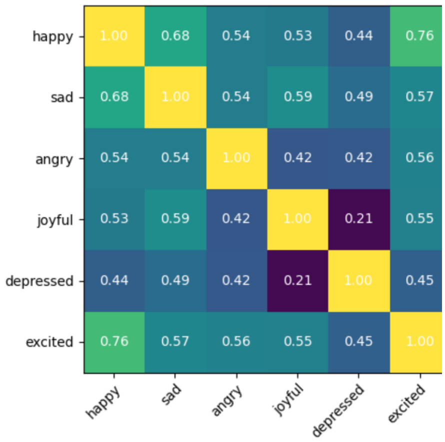
	dog	cat	mouse	lion	tiger	wolf
dog	1.00	0.74	0.40	0.49	0.30	0.54
cat	0.74	1.00	0.51	0.46	0.38	0.44
mouse	0.40	0.51	1.00	0.32	0.27	0.29
lion	0.49	0.46	0.32	1.00	0.47	0.54
tiger	0.30	0.38	0.27	0.47	1.00	0.35
wolf	0.54	0.44	0.29	0.54	0.35	1.00

Ex.11 GloVe dimension=200



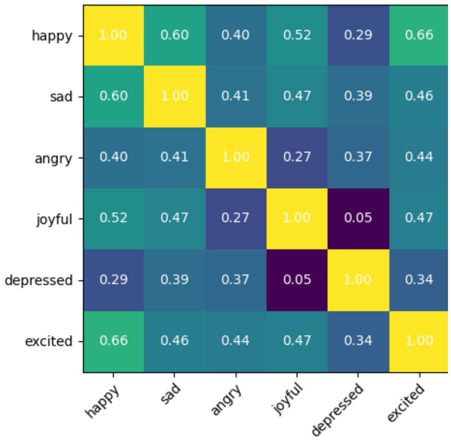
	dog	cat	mouse	lion	tiger	wolf
dog	1.00	0.68	0.35	0.42	0.24	0.45
cat	0.68	1.00	0.45	0.38	0.31	0.35
mouse	0.35	0.45	1.00	0.27	0.19	0.27
lion	0.42	0.38	0.27	1.00	0.43	0.43
tiger	0.24	0.31	0.19	0.43	1.00	0.23
wolf	0.45	0.35	0.27	0.43	0.23	1.00

Ex.11 GloVe dimension=300



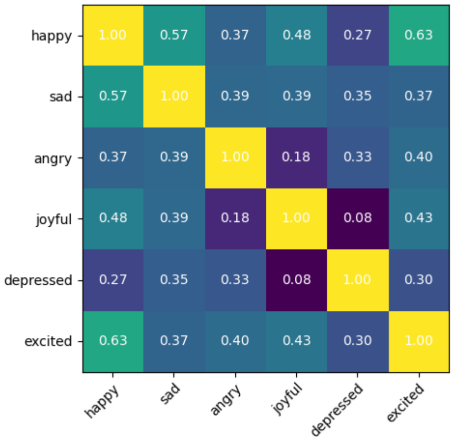
	happy	sad	angry	joyful	depressed	excited
happy	1.00	0.68	0.54	0.53	0.44	0.76
sad	0.68	1.00	0.54	0.59	0.49	0.57
angry	0.54	0.54	1.00	0.42	0.42	0.56
joyful	0.53	0.59	0.42	1.00	0.21	0.55
depressed	0.44	0.49	0.42	0.21	1.00	0.45
excited	0.76	0.57	0.56	0.55	0.45	1.00

Ex.10 GloVe dimension=100



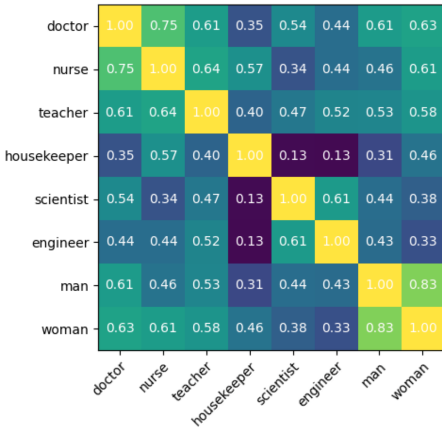
	happy	sad	angry	joyful	depressed	excited
happy	1.00	0.60	0.40	0.52	0.29	0.66
sad	0.60	1.00	0.41	0.47	0.39	0.46
angry	0.40	0.41	1.00	0.27	0.37	0.44
joyful	0.52	0.47	0.27	1.00	0.05	0.47
depressed	0.29	0.39	0.37	0.05	1.00	0.34
excited	0.66	0.46	0.44	0.47	0.34	1.00

Ex.11 GloVe dimension=200



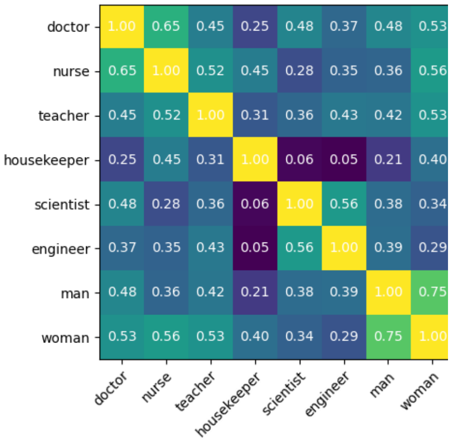
	happy	sad	angry	joyful	depressed	excited
happy	1.00	0.57	0.37	0.48	0.27	0.63
sad	0.57	1.00	0.39	0.39	0.35	0.37
angry	0.37	0.39	1.00	0.18	0.33	0.40
joyful	0.48	0.39	0.18	1.00	0.08	0.43
depressed	0.27	0.35	0.33	0.08	1.00	0.30
excited	0.63	0.37	0.40	0.43	0.30	1.00

Ex.11 GloVe dimension=300



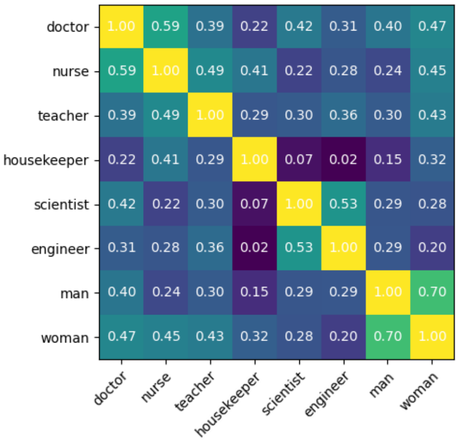
	doctor	nurse	teacher	housekeeper	scientist	engineer	man	woman
doctor	1.00	0.75	0.61	0.35	0.54	0.44	0.61	0.63
nurse	0.75	1.00	0.64	0.57	0.34	0.44	0.46	0.61
teacher	0.61	0.64	1.00	0.40	0.47	0.52	0.53	0.58
housekeeper	0.35	0.57	0.40	1.00	0.13	0.13	0.31	0.46
scientist	0.54	0.34	0.47	0.13	1.00	0.61	0.44	0.38
engineer	0.44	0.44	0.52	0.13	0.61	1.00	0.43	0.33
man	0.61	0.46	0.53	0.31	0.44	0.43	1.00	0.83
woman	0.63	0.61	0.58	0.46	0.38	0.33	0.83	1.00

Ex.10 GloVe dimension=100



	doctor	nurse	teacher	housekeeper	scientist	engineer	man	woman
doctor	1.00	0.65	0.45	0.25	0.48	0.37	0.48	0.53
nurse	0.65	1.00	0.52	0.45	0.28	0.35	0.36	0.56
teacher	0.45	0.52	1.00	0.31	0.36	0.43	0.42	0.53
housekeeper	0.25	0.45	0.31	1.00	0.06	0.05	0.21	0.40
scientist	0.48	0.28	0.36	0.06	1.00	0.56	0.38	0.34
engineer	0.37	0.35	0.43	0.05	0.56	1.00	0.39	0.29
man	0.48	0.36	0.42	0.21	0.38	0.39	1.00	0.75
woman	0.53	0.56	0.53	0.40	0.34	0.29	0.75	1.00

Ex.11 GloVe dimension=200



	doctor	nurse	teacher	housekeeper	scientist	engineer	man	woman
doctor	1.00	0.59	0.39	0.22	0.42	0.31	0.40	0.47
nurse	0.59	1.00	0.49	0.41	0.22	0.28	0.24	0.45
teacher	0.39	0.49	1.00	0.29	0.30	0.36	0.30	0.43
housekeeper	0.22	0.41	0.29	1.00	0.07	0.02	0.15	0.32
scientist	0.42	0.22	0.30	0.07	1.00	0.53	0.29	0.28
engineer	0.31	0.28	0.36	0.02	0.53	1.00	0.29	0.20
man	0.40	0.24	0.30	0.15	0.29	0.29	1.00	0.70
woman	0.47	0.45	0.43	0.32	0.28	0.20	0.70	1.00

Ex.11 GloVe dimension=300

A Collaborators outside our group

B Use of genAI

During the completion of this assignment, the following external resources were used:

Generative AI Tools

We used ChatGPT and prompts included queries such as:

- "Give me examples of synonyms, homonyms, etc."
- "Examples of news headline search queries"
- "Explain the error: . . ."
- "Put this code into LaTeX"
- "Format this as a two-column LaTeX layout"

Other Resources

- Lecture slides and materials provided on Canvas.
- Official documentation for libraries and packages used, such as `NumPy`, `GloVe`, and `Gensim`.