**Lab5: Sparse Vector (embedding)**

10 August 2025

**Q1. (a) Consider the table of term frequencies for 3 documents denoted Doc1, Doc2, Doc3. Compute the tf-idf weights for the terms car, auto, insurance, best for each document, using the idf values and calculate the score for any user query q. (example queries: "car insurance", "best car")  
(b) Apply Euclidean normalization to the tf values and generate the normalized tf table.**

**Draft Plan:**

1. Ingest TF table and IDF vector (editable by user in Streamlit).
2. Compute TF×IDF for each term/document.
3. Accept a textual query; split into tokens; compute document score = sum(tf-idf of query terms per document).
4. Optionally compute Euclidean-normalized TF (vector norm per document).
5. Display clear tables and ranking of documents for query. Add input validation and helpful UI info.

**Program Description:**  
Build an interactive Streamlit tab that accepts the TF table and IDF values, computes tf-idf (per-term per-doc), computes query scores, and shows normalized TF table. Provide crisp tables and ranking so an analyst can interpret which documents best match a query — business-ready output for search relevance experiments.

**Program Logic (algorithms & libraries):**

* Libraries: pandas, numpy, streamlit (UI), optionally scipy/sklearn for vector math.
* Key formulas: tf-idf\_{t,d} = tf\_{t,d} \* idf\_t.
* Query score: Score(q, d) = sum\_{t in q} tf-idf\_{t,d}.
* Euclidean normalization: given TF vector v\_d = [tf\_{t1,d}, ..., tf\_{tn,d}], compute v\_d\_norm = v\_d / ||v\_d||\_2 where ||v\_d||\_2 = sqrt(sum tf^2).
* Input sanitization: enforce numeric TFs and IDFs >= 0; handle zero-norms gracefully

**Test cases (actual & expected I/O)**

**Input (from the lab image):**

* Terms: ['car','auto','insurance','best']
* TF table:

| **Term** | **Doc1** | **Doc2** | **Doc3** |
| --- | --- | --- | --- |
| car | 27 | 4 | 24 |
| auto | 3 | 33 | 0 |
| insurance | 0 | 33 | 29 |
| best | 14 | 0 | 17 |

* IDF values:

| **Term** | **idf** |
| --- | --- |
| car | 1.65 |
| auto | 2.08 |
| insurance | 1.62 |
| best | 1.5 |

**Expected TF-IDF table (tf \* idf):**

| **Term** | **Doc1** | **Doc2** | **Doc3** |
| --- | --- | --- | --- |
| car | 44.55 | 6.60 | 39.60 |
| auto | 6.24 | 68.64 | 0.00 |
| insurance | 0.00 | 53.46 | 46.98 |
| best | 21.00 | 0.00 | 25.50 |

**Query:** "car insurance"  
**Expected scores:**

* Doc1 = 44.55 + 0.00 = 44.55
* Doc2 = 6.60 + 53.46 = 60.06
* Doc3 = 39.60 + 46.98 = 86.58 → Doc3 is most relevant

Query: "best car"

* Doc1 = 21.00 + 44.55 = 65.55
* Doc2 = 0.00 + 6.60 = 6.60
* Doc3 = 25.50 + 39.60 = 65.10 → Doc1 slightly beats Doc3

**Euclidean-normalized TF (rounded 3 decimals):**

| **Term** | **Doc1** | **Doc2** | **Doc3** |
| --- | --- | --- | --- |
| car | 0.883 | 0.085 | 0.581 |
| auto | 0.098 | 0.705 | 0.000 |
| insurance | 0.000 | 0.705 | 0.702 |
| best | 0.458 | 0.000 | 0.412 |

**Q2. Compute the nearest neighbours of a word in a vector space using cosine score. Steps: (a) Collect N documents,**

**(b) Generate a matrix of word embeddings (simple count-based sparse vectors & tfidf vectors),**

**(c) Plot a few words to understand similarity,**

**(d) Compute the nearest word for any given word using cosine scores, (e) Plot all words nearest to a given word x. Eg: man : king = woman : X**

**Draft Plan:**

1. Allow the user to paste/upload N documents (or a sample dataset).
2. Provide vectorization choices — Count / TF-IDF.
3. Build a document-term matrix and compute cosine similarity on the word vectors (columns).
4. For a selected word, compute top-k nearest words and display scores + visualization (bar chart and a 2D projection like t-SNE or PCA).
5. Provide handling for OOV words, empty docs, and provide informative warnings.

**Program Description:**  
A Streamlit tab that creates sparse word-embeddings (count-based and tfidf-based), enables nearest neighbor lookup by cosine similarity and produces visualizations (bar charts of nearest neighbors, 2D scatter of selected words using PCA/TSNE/UMAP). Useful for exploratory analysis of semantic similarity in a small corpus.

**Program Logic:**

* Tokenize documents (sklearn vectorizers handle tokenization).
* Compute X = document-term matrix (docs × terms).
* Word vector = column j of X (size N\_docs). Cosine similarity among word vectors: cos\_sim = cosine\_similarity(X.T).
* Nearest neighbours: sort cosine scores for the target word.
* Visualize: bar chart (top-k scores) and 2D projection of selected words (PCA or t-SNE).

**Test case (small example)**

* **Docs:**
  1. "car insurance best car"
  2. "best auto insurance"
  3. "car and auto are vehicles"
* **Vectorization = Count:** Top neighbors for "car" (Count-based):
  1. best — **0.6325**
  2. insurance — **0.6325**
  3. and / are / vehicles — **0.4472**
  4. auto — **0.3162**
* **Vectorization = TFIDF:** Top neighbors for "car" (TF-IDF):
  1. best — **0.5251**
  2. insurance — **0.5251**
  3. and / are / vehicles — **0.4155**
  4. auto — **0.2255**

**Q3. Pointwise Mutual Information (PMI): Use PMI when low co-occurrences of words exists. Compute the PMI instead of tf and tfidf, and recalculate the score for similarity check.**

**Draft Plan:**

1. Take raw text input from user (paste/upload).
2. Tokenize; compute unigram counts and bigram counts (adjacent tokens).
3. Compute probabilities p(w) and p(w\_i, w\_j).
4. Compute PMI (log base 2 recommended) and optionally PPMI (max(PMI, 0)) to avoid negative values.
5. Provide a bigram table sorted by PMI, and allow user to compute PMI for a specific word pair.

**Program Description:**  
PMI highlights word-pairs that occur together significantly more than expected by chance — very useful to detect strong collocations or associations in small corpora. For sparse data, use **PPMI** or smoothing. For large corpora and semantic similarity, prefer PPMI+SVD.

**Program Logic & safety:**

* Use Counter for unigrams and bigrams.
* Probabilities: p(w) = count(w)/N; p(w1,w2) = count(w1,w2)/(N - 1) for adjacent bigrams.
* PMI can be -inf if p(w1,w2)=0; handle gracefully (display as -inf or omit). Use smoothing if desired. Use log2 for interpretability.