**Assignment for Unit 5**

Name: DOMENIC TAGANAHAN Date: 04/21/2025

Year and Section: BSCS 3 B AI

*Note: For problems with code, create a separate python file for each question and upload them to a public repository under your GitHub account.*

*GitHub Link:* [*https://github.com/Syntax404-coder/ccs249-assign-5-taganahan*](https://github.com/Syntax404-coder/ccs249-assign-5-taganahan)

1. (20 points) Using Wikipedia as the corpus, obtain 5 different topics that will serve as your documents, and create a term-document matrix. You can use the shared code on GitHub as a reference.
   1. Term-document matrix using raw frequency.
   2. Term-document matrix using TF-IDF weights.
2. (10 points) What are the differences between using TF-IDF weights and raw frequency?

From my perspective as someone who’s been working through TF‑IDF and term‑frequency models, the core distinction between raw frequency and TF‑IDF weights lies in how they assign importance to words. Raw frequency is straightforward: it simply counts how often a term appears in a single document. While this gives a quick sense of which words dominate that text, it fails to recognize whether those words are genuinely meaningful or just ubiquitous. In contrast, TF‑IDF balances a word’s prevalence within a document against its rarity across an entire collection, thereby highlighting terms that are both frequent in one document and relatively uncommon elsewhere.

When I use raw term frequency, common words—even after stop‑word removal—can still skew the representation. For example, if I analyze biographies of five historical figures, a word like “theory” might appear often across all articles and yet offer little in distinguishing Einstein’s work from Newton’s. Raw frequency would treat “theory” and “relativity” equally if they occur with equal counts, even though “relativity” is much more distinctive to Einstein. TF‑IDF corrects for this by down‑weighting words that occur across many documents (low IDF) and up‑weighting those that are rare (high IDF). The result is a vector representation where “relativity” carries more semantic weight in Einstein’s document than a generic word does.

I’ve also noticed practical differences in downstream tasks. When computing cosine similarity between documents, raw‑frequency vectors can misleadingly cluster documents around general vocabulary, masking deeper thematic relationships. TF‑IDF vectors, by contrast, tend to surface genuine topical overlaps: two documents that both discuss “civil disobedience” will register high similarity even if their overall word counts differ. In classification tasks, TF‑IDF features often improve accuracy because they emphasize the most informative words, reducing noise from overly common terms.

In summary, while raw frequency offers simplicity and computational ease, it lacks context about a word’s informativeness. TF‑IDF introduces that context by penalizing omnipresent terms and rewarding unique ones—aligning the weighting scheme more closely with what we intuitively regard as “important” words. From my viewpoint, adopting TF‑IDF leads to richer, more discriminative text representations that better support similarity measures, clustering, and classification.

1. (10 points) Using cosine similarity, compare two documents and find out which of the documents is most similar.
2. (30 points) Using the same dataset used above, use the word2vec package to create a classifier for dense vectors.
   1. Use Logistic Regression, with the appropriate configuration for the model and dataset.
3. (20 points) What are the differences of using word2vec compared to the tf-idf in terms of:
   1. Vector Space?
   2. Vector Size?

In my experience, using **word2vec** and **TF-IDF** for text representation reveals several key differences, particularly in how they construct vector spaces and the size of those vectors. Both methods serve to represent text numerically, but they do so in fundamentally different ways, each offering unique advantages and limitations depending on the task at hand.

**A. VECTOR SPACE**

The vector space in TF-IDF is inherently sparse and high-dimensional. Each document in a collection is represented as a vector, with each term in the vocabulary corresponding to a unique dimension. This results in a large, sparse matrix, where most values are zero because most documents don’t contain every word in the entire vocabulary. The key feature of TF-IDF is that it assigns higher weights to words that are frequent within a document but rare across all documents, which emphasizes the terms that carry unique semantic meaning within the document.

In contrast, word2vec creates dense, lower-dimensional vector spaces. Each word in the corpus is represented by a vector in a continuous, low-dimensional space, and these vectors are learned by the model through predicting context words (using either the continuous bag of words or the skip-gram approach). The key advantage of word2vec is that it captures semantic relationships between words by positioning similar words close together in the vector space, often clustering words that share similar meanings. For example, “king” and “queen” would have vectors close to each other in word2vec, reflecting their shared contextual usage, even if they appear in different documents.

From my perspective, the word2vec approach creates a more semantic vector space that reflects relationships between words, while the TF-IDF vector space is more about statistical importance of words within documents and across a corpus.

**B. VECTOR SIZE**

When it comes to vector size, word2vec typically uses smaller, fixed-dimensional vectors that range from 50 to 300 dimensions, although this can vary depending on the model and corpus size. These vectors are the result of training on large text corpora and encapsulate semantic information about the words, which is why they are known as dense vectors. The compact nature of these vectors allows for efficient storage and processing, while also retaining much of the contextual and semantic information about a word's usage.

In contrast, TF-IDF vectors are typically sparse and the vector size directly corresponds to the size of the vocabulary in the corpus. If the vocabulary consists of 10,000 unique words, each document is represented by a vector of size 10,000, where each dimension corresponds to one of those words. These vectors are generally much larger and sparse, with most elements being zero because not all terms appear in every document.

From my understanding, word2vec uses smaller, dense vectors that capture more meaningful semantic relationships with fewer dimensions, making them more computationally efficient in certain tasks. On the other hand, TF-IDF vectors are larger and sparse, with each dimension representing a distinct term, and their size grows with the size of the vocabulary.

1. (10 points) How do we evaluate the performance of Semantic Models (i.e TF-IDF and Word2Vec)?

From my experience working with both TF‑IDF and Word2Vec, I’ve found that evaluating their performance really depends on the kinds of tasks they’re being used for and the insights you want to gain from your text data. With TF‑IDF, I tend to judge the model by how well it helps me distinguish documents in classification or retrieval tasks. In practice, I’ll vectorize a collection of articles using TF‑IDF, train a simple classifier like logistic regression, and then look at metrics such as accuracy or F1‑score on held‑out data. If the classifier reliably separates categories—say, topic labels or sentiment classes—that tells me my TF‑IDF weights are capturing the important terms across the corpus. I also sometimes compute cosine similarities between documents: if the most similar pairs actually share themes or keywords in the way a human would expect, that reinforces my confidence in the TF‑IDF representation.

By contrast, when I work with Word2Vec, my evaluation shifts toward semantic relationships among words and their downstream impact. I often start by examining word‑similarity and analogy benchmarks: if the classic “king – man + woman = queen” analogy holds, and words like “dog” and “puppy” cluster together under a t‑SNE plot, I know my dense vectors are learning meaningful associations. Beyond that, I’ll aggregate word vectors into document embeddings and run the same classification experiments I did with TF‑IDF; improved accuracy there suggests that the semantic information in Word2Vec is beneficial. Finally, I like to visualize both word and document vectors—if semantically related items form coherent groups in two‑dimensional space, it reassures me that the model has internalized real-world relationships.

In short, TF‑IDF evaluation centers on how well weighted term frequencies support discriminative tasks like classification and retrieval, whereas Word2Vec evaluation emphasizes semantic coherence—both at the word level through analogies and visualizations, and at the document level through classification performance using dense embeddings.