Day - 3

06 October 2024 22:01

OpenAI - chat model endpoint - settings -
Tokenizers
RNN/LSTM
Vectors and embeddings

Topics for today

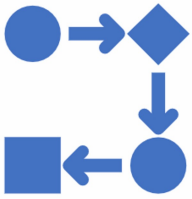
1. LLM settings.
2. Semantic Search (concepts).
3. Prompt engg. (Advanced concepts).
4. Vectors & embeddings.
5. How is/are embeddings a used (DL/Keras).
6. RNN/LSTM.
7. Tokenizers.

Vectors in NLP

1. .....
2. DL
3. LLM / create.
4. Vector databases - emb -

Vectorization

- is a transformative procedure.
- serves as the bridge between the rich, human-readable world of text and the numerical realm of machine learning algorithms.
- vectorization is the art and science of transforming the intricate patterns of language into structured numerical representations, or vectors,



Example

- Consider the following 3 documents:
  1. Document 1: "Machine learning is fascinating machine"
  2. Document 2: "Natural language processing is a key technology in AI."
  3. Document 3: "AI is shaping the future of technology."
- Tokenization
  - Document 1: ["machine", "learning", "is", "fascinating"]
  - Document 2: ["natural", "language", "processing", "is", "a", "key", "technology", "in", "ai"]
  - Document 3: ["ai", "is", "shaping", "the", "future", "of", "technology"]

Example

- Vocabulary Construction
  - Vocabulary: ["machine", "learning", "is", "fascinating", "natural", "language", "processing", "a", "key", "technology", "in", "ai", "shaping", "the", "future", "of"]

- Represent each document as a vector,
  - Document 1: [2, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    - { represent the input doc -> vocab}
  - Document 2: [0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
  - Document 3: [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1]

review doc 1 : [A, B A]
doc2 : [B, C, D]

doc1 → bag
doc2 → bag

doc1 → [A B C D]
[1 2 0 0] vector.
" bm25 "

doc 2 → [

Sorted
[A B C D]
dict / bag.

- vectors

Embeddings

Embedding refers to the process of mapping words or phrases to continuous vector spaces.

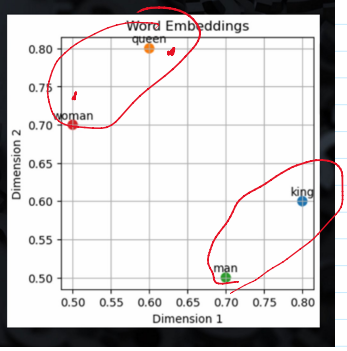
capture semantic relationships between words and are often learned from large corpora using neural network models

vector → [1, 5, 0, 2, 0 0 ..... 0] len = len of bag dic.
emb → [.25 .35 .0) ....]

} len = user specified
OpenAI emb = 1536.

Example

- Consider an example using a hypothetical 2-dimensional embedding space.
- words are represented as vectors with two values:
  - Word "king": [0.8, 0.6]
  - Word "queen": [0.7, 0.9]
  - Word "man": [0.6, 0.2]
  - Word "woman": [0.4, 0.7]
- words related in meaning or context are closer in the embedding space.
- For instance, "king" and "man" are closer to each other than "king" and "woman."
- Additionally, vector arithmetic can capture relationships such as "king - man + woman = queen."



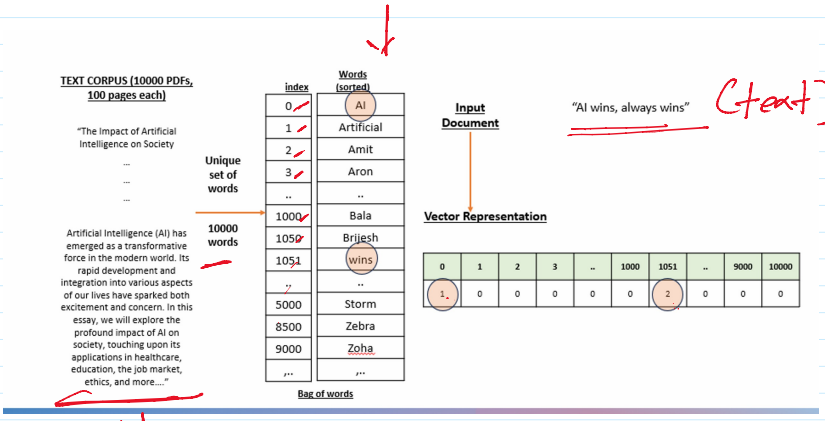
vector [5 0 1 0 ... 0]
emb [x x x

comparison between vectorization and embeddings

Aspect	Vectorization	Embeddings
Definition	Converts text into numerical vectors based on word frequencies or counts.	Converts text into dense vectors that capture semantic meaning.
Dimensionality	Typically, high-dimensional and sparse.	Typically, lower-dimensional and dense.
Semantic Information	Does not capture semantic relationships or context.	Captures semantic relationships and context.
Context Awareness	Context-independent.	Context-aware.
Complexity	Simpler and easier to compute.	More complex and computationally intensive.
Example Techniques	Bag-of-Words (BoW), TF-IDF, Hashing	Word2Vec, GloVe, BERT, GPT....

dense vector (ML/DL)
Vectors = len (corpus words)
15,0000
[...]
150k

emb →
150k
wx1
1536
[...]
1536



(text)
100k pages

Key characteristics of BOW

Word Frequency: represents a document as a vector where each element corresponds to a unique word in the vocabulary and the value in each element represents the count (or sometimes a binary indication of presence/absence) of that word in the document.

Order Ignored: BoW completely ignores the order of words within a document.

Key characteristics of BOW

Vocabulary: BoW requires a pre-defined vocabulary, which is created by collecting all unique words across the entire corpus of documents.

The vocabulary size determines the dimensions of the BoW vectors.

Sparse Representation: BoW vectors are typically sparse, especially in large vocabularies, because most documents only contain a subset of the words in the vocabulary.

This sparsity can be computationally efficient but may require specialized data structures for storage.

sparsity high/low?

Key characteristics of BOW

Scalability: BoW is scalable to large datasets and can be applied to a wide range of text analysis tasks, including document classification, sentiment analysis, and information retrieval.

Text Preprocessing: Before applying BoW, text preprocessing steps like tokenization (splitting text into words or tokens), stop word removal (excluding common and uninformative words), and stemming (reducing words to their base form) are often performed to improve the quality of representations.

** emb

medical feat

xyz-abc

collocation.

ML Journey

classical ML

KNN

boosting

svm

NN → RNN → LSTM → enc-dec

lstm

emg - fr.

2015.

** attention. 2015/16

vanilla.

--- self-att.

vanilla.
--- self-attn

Limitations - Ignores Word Order and Structure

CountVectorizer treats each document as a bag of words, disregarding the order and structure of words in a sentence.

limitation may lead to a loss of important information, especially in tasks where word order matters, such as sentiment analysis or language modeling.

... more limitations

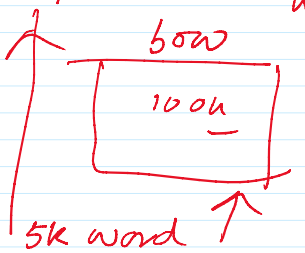
Fixed Size Vocabulary

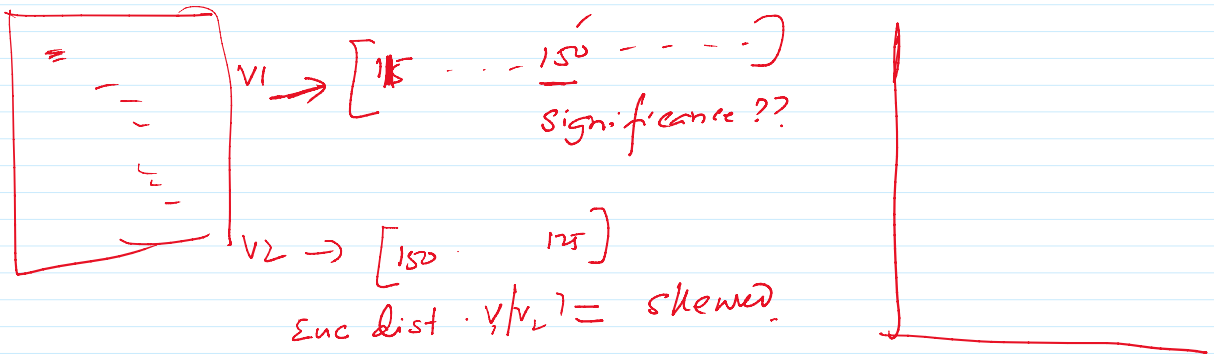
- The vocabulary created by CountVectorizer has a fixed size determined by the unique words in the training data.
- New words in test or unseen data won't be represented unless the vocabulary is updated, potentially leading to information loss.

Sensitivity to Stop Words

- Commonly used words (stop words) are often ignored by CountVectorizer based on the stop_words parameter.
- While this can be beneficial for some tasks, it may lead to the exclusion of important context-carrying words.

Corpus: 100k words.





how high counts in count vect poses problem

High counts in count vectorization, while seemingly informative, can pose several problems

very high word counts can dominate the representation and introduce noise

← similarities/distance

bow / count vect

problem

mitigate the problems associated with high counts

- **Term Frequency-Inverse Document Frequency (TF-IDF):** TF-IDF is one way to address the dominance of common words by assigning lower weights to them based on their inverse document frequency. This reduces the impact of high counts for common words.
- **Text Preprocessing:** Techniques like stop word removal and stemming or lemmatization can help reduce the frequency of common words and variations of words, improving the quality of vector representations.

bow / count vect

TF → word count

IDF → inverse doc freq

trick.

TF * IDF = Score

high freq → ↓ score

low freq → ↑ score

Term Frequency (TF)

The first part of TF-IDF, the Term Frequency (TF), measures how frequently a term (word) appears in a document.

It quantifies how often a word occurs within the document.

The TF of a term in a document is calculated using ...

TF(t, d) = (Number of times term t appears in document d) / (Total number of terms in document d)

Inverse Document Frequency (IDF):

- The second part of TF-IDF, the Inverse Document Frequency (IDF), measures the importance of a term across a collection of documents (corpus).
- It quantifies how common or rare a word is in the entire corpus.
- The IDF of a term is calculated using the following formula:

IDF(t) = log((Total number of documents in the corpus) / (Number of documents containing term t))

100k.

Example

- Consider a corpus with three documents:
  - Document 1: "The quick brown fox jumps over the lazy dog."
  - Document 2: "A brown cat plays with a dog."
  - Document 3: "The lazy cat sleeps."

- Let's calculate the TF-IDF score for the term "dog" in Document 1:
  - TF(dog, Document 1) = 1 (since "dog" appears once in Document 1)
  - IDF(dog) = log(3 / 2) ≈ 0.176 (since "dog" appears in two out of three documents)
  - TF-IDF(dog, Document 1) = TF(dog, Document 1) * IDF(dog) = 1 * 0.176 ≈ 0.176

count

$TF(dog, d_1) = 1$

$IDF(dog) = \frac{N}{n}$

$= \log\left(\frac{3}{2}\right)$

$= 0.176.$

$TF\text{-}IDF = 1 * 0.176 = .176$

$N = 10000$

$n = 9999$

$idf = \frac{N}{n}$

$= \log\left(\frac{10000}{9999}\right)$

$= 0.0000001$

$TF = 15 *$ ↙ ↓

$N = 10000$

$n = 5$

use cases.

① medical data for logistic.

② count / TF-IDF (text)

$idf = \frac{N}{n} = \frac{10000}{5}$ ✓

$= \log(\text{large no.})$

$= \text{large no.}$

TF ✗ ← ↑ boost.

✓ use case for count vect: technical archive.

count / TF-IDF

{ (- - - x:) (- - -) . . . } → count [- - - -] [- - - -] → SVM LogReg.

TF/IDF

↑ incrementally update.

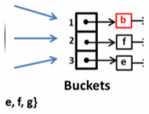
" online ML model".

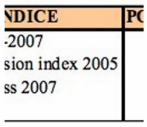
↑ delta

Hashing vectorizer

Well-suited for scenarios involving streaming data, where it can efficiently handle new text instances on the fly.

Basic concepts





Hash Function: Hashing Vectorizer uses a hash function, which is an algorithm that maps input data of arbitrary size to fixed-size values (hashes). These hashes are usually integer values.

Mapping Words to Indices: Hashing Vectorizer directly converts each word to a hash value using the hash function. This hash value becomes the index of the word in the feature vector.

Example

text = "apple banana apple orange"

hash("apple") -> 2
hash("banana") -> 0
hash("orange") -> 3

assume our feature vector's size is 5 (n_features = 5)

Vector: [?, ?, ?, ?, ?]

Binary Representation	Frequency-based Representation
"apple" hashes to index 2 -> Set position 2 to 1 "banana" hashes to index 0 -> Set position 0 to 1 "orange" hashes to index 3 -> Set position 3 to 1 Resultant Binary Vector: Vector: [1, 0, 1, 1, 0]	First occurrence of "apple" -> Increment position 2 by 1 "banana" -> Increment position 0 by 1 Second occurrence of "apple" -> Increment position 2 again by 1 "orange" -> Increment position 3 by 1 Resultant Frequency Vector : Vector: [1, 0, 2, 1, 0]

Drawbacks of hashing vect

01

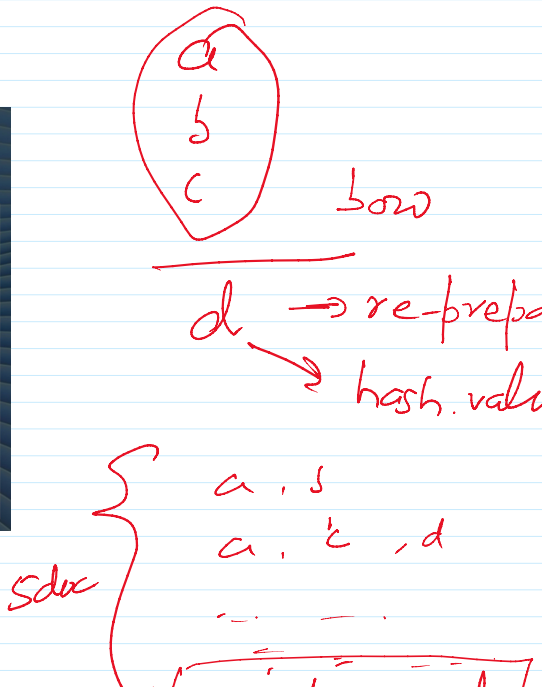
Collisions: prominent drawback of the Hashing Vectorizer is the possibility of collisions.

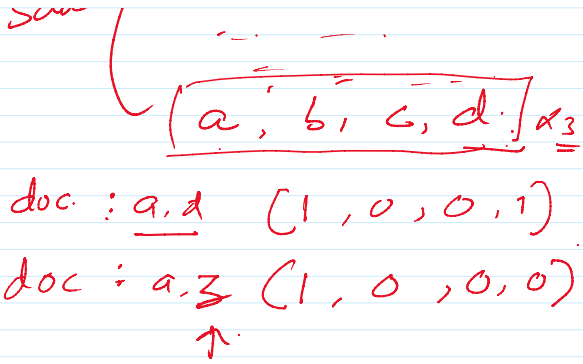
Different words might be hashed to the same index, which means they will be indistinguishable in the resulting vector representation.

02

No Inverse Mapping: Once words are hashed, there's no direct way to determine which word a particular hash corresponds to.

This lack of inverse mapping can make models harder to interpret and debug.





Basics of embedding (NN)
Word2vec
Doc2vec
Sent2vec

Open source emb models (Glove....)