SQLite :-

→ `pd.read_sql_query("""SELECT name FROM sqlite_master WHERE type='table' ORDER BY name; """, con)` → Prints list of tables.

	name
0	Reviews

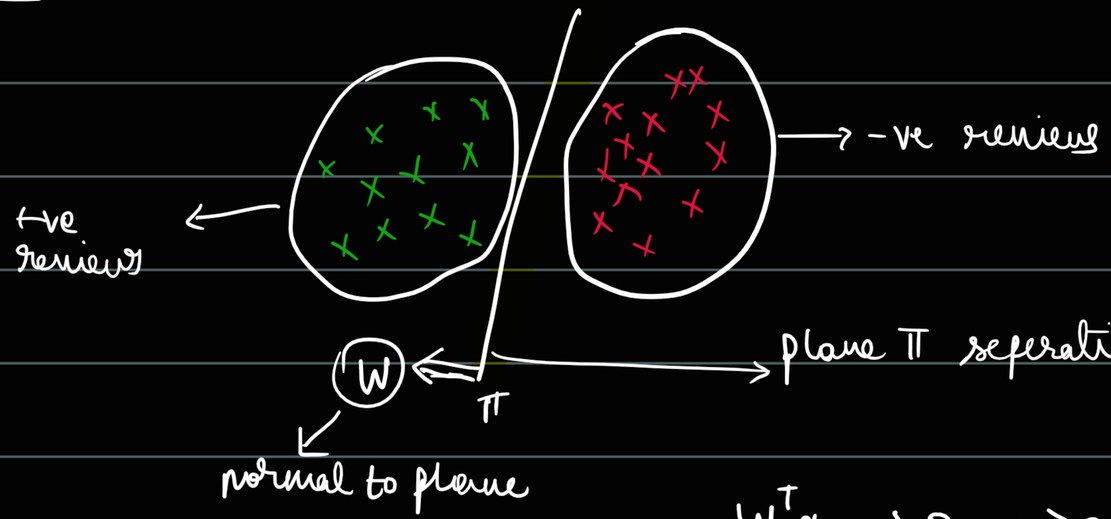
→ df.drop_duplicates(subset={" ", " ",... "}, keep="first") (Pandas)

↳ drops duplicate values with these column name & keeps the first occurance.

→ Use common sense to clean data. No general technique.

→ how to convert text to numerical vectors?

ex:- Review text → n-dim vector



(reviews are represented by 'X' after converting to vector)

$W^T x_1 > 0$ ⇒ review x_1 is a positive review (on same side)

$W^T x_2 < 0$ ⇒ review x_1 is a negative review (on diff side)

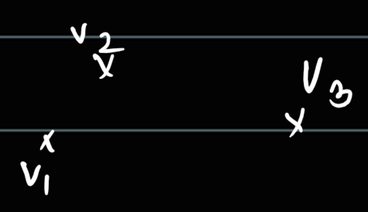
→ Rules for conversion from text to vector :-

r_1 r_2 r_3

assume semantic similarity b/w $(r_1, r_2) > (r_1, r_3)$

then distance b/w $(v_1, v_2) <$ dist b/w (v_1, v_3)

ie: if they are similar, vectors must be close.



Bag of words :-

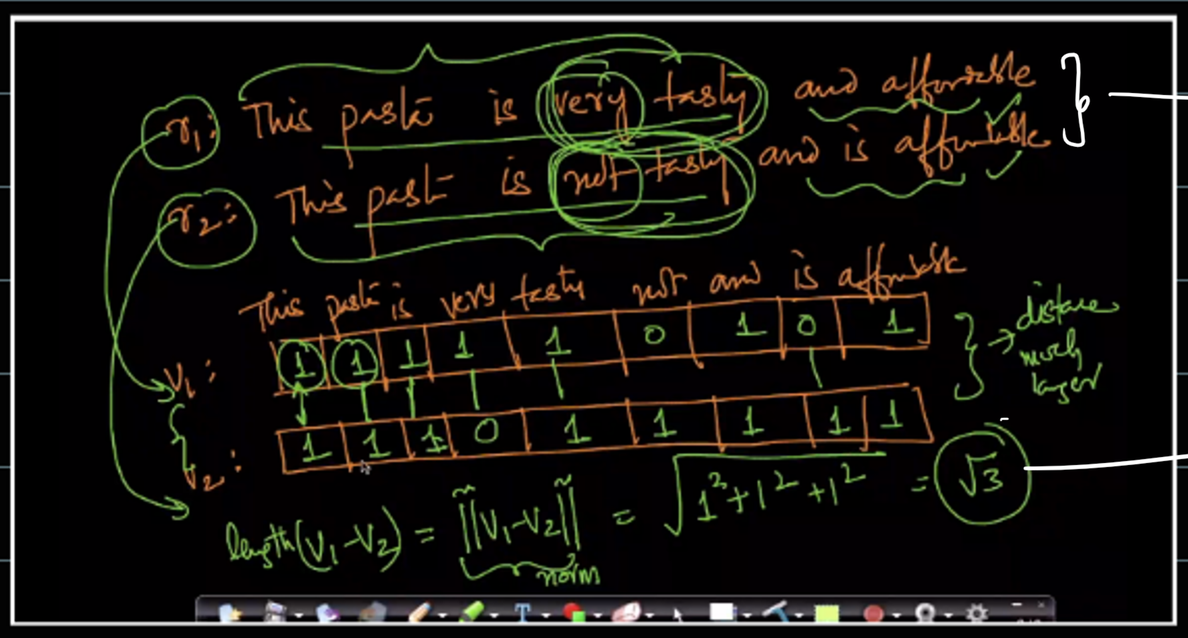
step ① :- Constructing dictionary (set of all unique words)

step ② :- Assume there are d unique words across all 'n' documents/reviews. [Collection of documents is called a corpus]

For each review (d dimensional vector)

The number in pos represents the number of times that word occured in the review.

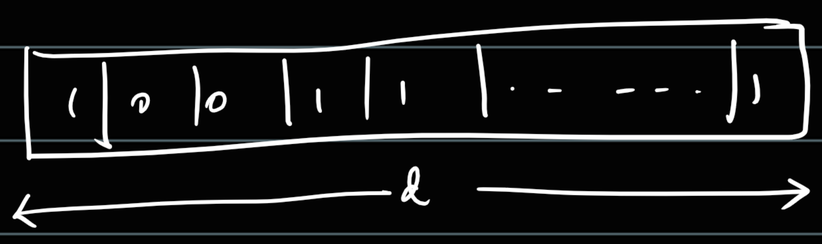
Each word is a different dimensions.



→ Diff sentences but distance is less compared to length 'd' (9)

Boolean bag of words / binary bag of words :-

Instead of counting number of occurances, it only counts occurances -



$\|v_1 - v_2\| \approx \sqrt{\#\text{ of differing words}}$

→ We can notice that some of the words in BoW don't matter (is, and, a, the etc.,) → Stopwords

→ Bag of words is still used for creating baselines with Generalized Linear Models.

→ Removing stop word = Smaller dimensions data.

These are called text preprocessing steps.

→ from nltk.corpus import stopwords

→ We also make everything lowercase.

→ Stemming :- tasty, tasteful, tasteY all have same meaning.

converting all into one word 'taste'. This is called stemming.

PorterStemmer, Snowball Stemmer } two techniques for stemming.

Snowball Stemmer → more powerful.

→ tokenization :- Breaking sentences into words. eg:- if we use space as delimiter, New York are seperate words.

→ In BoW we are not taking semantic meaning.

eg:- delicious tasty convey same meaning. We use word2Vec to solve this.

Lemmatisation is the algorithmic process of determining the lemma of a word based on its intended meaning. Unlike stemming, lemmatization depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document.

→ Algorithm combines inflected words into a common word called lemma.

{go, going, went} => 'go' is the lemma given by algo.

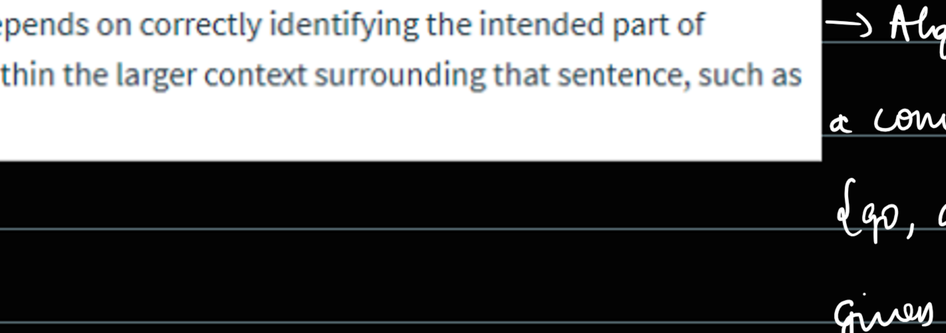
→ NLTK makes use of parts of speech. This is called tagging. Makes sense of context and helps pick the right lemma.

→ Stopwords are not removed in stemming. All words are reduced to base words and the tails are removed.

unigram / Bi-gram / n-gram :-

unigram :- each word is considered as a dimension

bigrams :- pair of words



n = 3 → trigrams

⋮

→ unigram BoW completely discards sequence information

Bigrams, trigrams, ... retains partial sequence informations.

→ But the catch is $n(\text{bigrams}) > n(\text{unigrams})$

→ As n increases, dimensionality increases drastically.

→ But if there are no repeated words in text, #unigrams > #bigrams > #trigrams, ...

TF-IDF:- (term frequency inverse document frequency) (industrialized version of BoW)

r_1: w_1 w_3 w_2 w_5 w_1

N docs { r_2: w_3 w_1 w_4 w_1 w_6 w_2

⋮

⋮

w_1	w_2	w_3	w_4	w_5
2	1	1	0	1

⋮

$$TF(w_i, r_j) = \frac{\#\text{ of times } w_i \text{ occurs in } r_j}{\text{total number of word in } r_j}$$

→ TF-IDF was originally meant for informations retrieval.

→ TF gives how often word occurs. (Probability)

$$0 \leq TF(w_i, r_j) \leq 1$$

IDF:-

D_c = Document corpus { r_1: — w_1 —, r_2: —, ⋮ — w_1 —, ⋮ — w_1 —

$$\rightarrow IDF(w_i, D_c) = \log\left(\frac{N}{n_i}\right)$$

n_i → Number of docs that contains w_i

$$D_c = \{r_1, r_2, \ldots, r_n\}$$

$n_i \leq N \Rightarrow \frac{N}{n_i} \geq 1$

$\Rightarrow \log\left(\frac{N}{n_i}\right) \geq 0$

⇒ As n_i increases, N/n_i decrease. As N/n_i decrease, $\log(N/n_i)$ also decreases as log is monotonic function.

If a word is more frequent in corpus, then its IDF is less. If it's less frequent in corpus, then it's IDF is high

→ Combining both

D_c { r_1 :- ——, r_2 :- ——, ⋮, r_n :- ——

V_1 :- [| | ..... | | ...]

$$w_i = TF(w_i, r_1) * IDF(w_i, D_c)$$

→ Advantage is that we give more importance if → word is more frequent (from TF)
→ word is less frequent (from IDF)

→ Drawback:- Still no semantic importance.