Amazon Fine Food Reviews Analysis

Real world dataset

positive   negative

* Between all the attributes given, text (text of the review) is the most important attribute to know if a review is positive or negative

* Data cleaning :→ 20-30% time of machine learning is applied on data cleaning & pre-processing. Otherwise without cleaning, garbage in $\xrightarrow[\text{algo}]{\text{ML}}$ garbage out

⇒ To remove duplicate entries.

Steps of Data cleaning :→

1. Sort the data (sort_values)
2. Drop duplicates values (drop_duplicates), 'keep' → imp
   keep = 'first' → it will keep the first one & remove other duplicates.
3. Check how much % of data still remains.

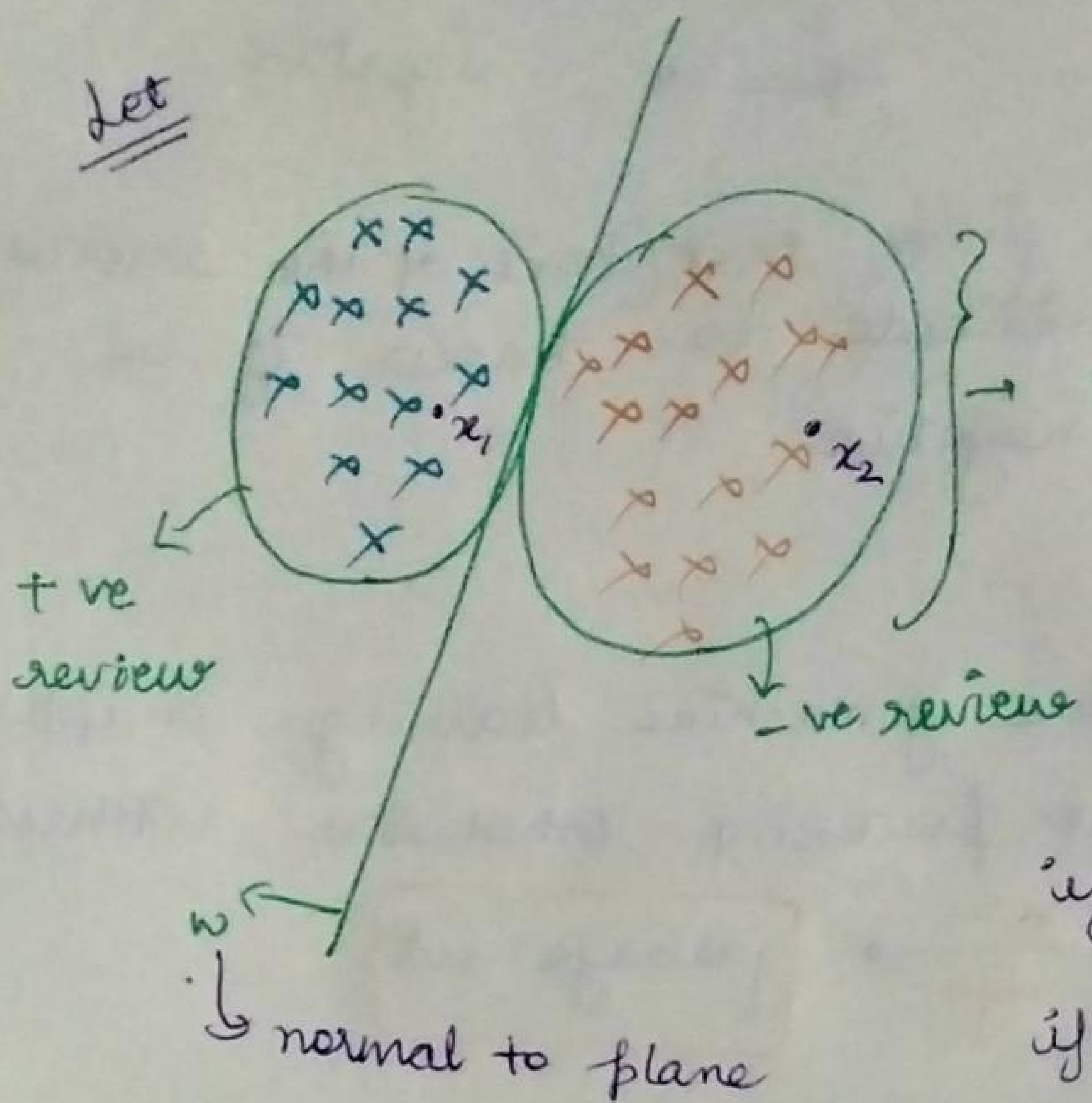
⇒ To remove invalid entries.

* As the most important attribute is text here, so as we know by converting data points into vectors, we can use whole thing of linear algebra.

So, Question is How to convert text to vectors?

d-dimension representation of a vector

Let



d-dimensional space

if $w^T x_i = +ve$, +ve review

if $w^T x_i = -ve$, -ve review

Step 1: Convert review-text $\xrightarrow[\text{rules}]{}$ d-dim vector

$\overbrace{r_1, r_2}^{\text{similar}}, r_3$ semantically

$r_1 \to v_1$, $r_2 \to v_2$, $r_3 \to v_3$

If English $Sim(r_1, r_2) >$ english $sim(r_1, r_3)$

then, $dist(v_1, v_2) < dist(v_1, v_3)$

* r_1 & r_2 are more similar then v_1 & v_2 must be close.

Now Question is How to convert text to d-dimen-vector such that similar text must be closer geometrical

Techniques to convert text to d-dim vector :→

1. Bag of words (BOW) :→

Steps :→ (i) Constructing a dictionary :- Set of all the unique words in your reviews.

(ii) for each review r_i make a vector v_i

for ex :→ r_1 : This pasta is very tasty and affordable ~~effecti~~

v_1 :

1	2	3	4	5				9		…	d-1	d
			1		1		1		1			
a	an	the	pasta		This		tasty		is			

no. of times a word occurs in r_i.

each word is a different dimension.

$v_1 \rightarrow$ sparse vector

Similar text must form closer vectors.

for ex :→

r_1 :→ This pasta is very tasty and affordable.

r_2 :→ This pasta is not tasty and is affordable.

	This	pasta	is	very	tasty	not	and	is	affordable
v_1 :→	1	1	1	1	1	0	1	0	1
	↕	↕	↕	↕ 1	↕	↕ 1	↕	↕ 1	↕
v_2 :→	1	1	1	0	1	1	1	1	1

$$\text{length}(v_1 - v_2) = \underbrace{\|v_1 - v_2\|}_{\text{norm}} = \sqrt{1^2+1^2+1^2} = \sqrt{3}$$

BOW :→

pasta	
2	. . . — — — -

no. of occurrences

$w_1 \quad w_2 \quad w_3 \quad \ldots \ldots \quad w_d$

binary BOW → | 1 | | | ... | |

boolean BOW

↓ Stores boolean value(true or false)
1 or 0

1 :→ if w_1 occurs atleast once

→ $||v_1 - v_2|| \approx$ # no. of differing words

Now if we notice carefully, works like this, is, and etc are not important. So, we will how we can improve BOW.

These words (this, is, ...) are called stop words.

- After removing stop words our BOW will be smaller & more meaningful.

This is called text pre-processing. (✓)

- Convert all words into lowercase so that they are considered to be different.

- Stemming is to convert similar meaning words into a base word.

For ex :→ tastes, tasty, tasteful ⟶ taste.

beautiful, beauty ⟶ beauty.

Porter Stemmer & snowball stemmer are types of Stemmer.

Lemmitization :→ breaking a sentence into words.

for ex :→ This pasta is very tasty. This is the best in $\boxed{\text{New York}}$ → single word.

Semantic meaning of words

{ tasty ⟷ delicious → very similar in meaning

↓

BOW do not take care of this.

* Uni-gram / Bi-gram / n-gram :→

Unigram :→ Single words → in 1-D

r_1 :→ This pasta is very tasty and affordable.

r_2 :→ This pasta is not tasty and is affordable.

→ stop words.

removing stopwords → v_1 & v_2 are exactly same

⇒ conclude r_1 & r_2 are very similar

↓

but this conclusion is not right at all.

Unigram :→

						- - -
This	is	pasta	very	tasty	affordable	

↓

each word is considered a dimension.

Bigram :→

Thispasta	ispasta					
						- - -
					not tasty	

↓

pair of words is considered a dimension = 2-2 words trigrams :→ 3 consequtive words → in a dimension

n-grams :→ n = 1 → unigram

n = 2 → bigram

n = 3 → trigrams.

⇒ Unigram in BOW :→ discards the sequence info like :→ "very tasty", "not tasty"

⇒ Bigrams / tri-grams :→ retains some of the sequence info.

$\boxed{\text{no. of bigrams} >= \text{no. of unigrams}}$

⋮ so on so forth.

n-grams ($n>1$) ↑ ⟶ dimensionality 'd' ↑

* TF-IDF :→

TF → Term frequency

IDF → Inverse document frequency

document → (review)

N documents / reviews:

		w_1	w_2	w_3	w_4	w_5	w_6	
r_1 : w_1 w_2 w_3 w_4 w_5	→ 5	1	2	1	0	1	0	BOW
r_2 : w_1 w_3 w_4 w_5 w_6 w_2	→ 6	1	1	1	1	1		BOW

⋮

r_N

$$TF(w_i, r_j) = \frac{\text{no. of times } w_i \text{ occurs in } r_j}{\text{Total no. of words in } r_j}$$

$$TF(w_2, r_1) = \frac{2}{5}$$

$$0 \le TF(w_i, r_j) \le 1 \rightarrow \text{probability}$$

TF/IDF $\longrightarrow$ Information Retrieval techniques (NLP)
$\downarrow$
natural language processing

TF says how often w_i occurs in r_j

$TF \propto$ (more w_i occurs in r_j)

* <u>IDF</u> :→

$D = \begin{cases} r_1 ; \\ r_2 ; \\ r_3 ; \\ \vdots \\ r_N ; \end{cases}$

$\downarrow$ Data corpus

no. of docs/reviews ←

$IDF(w_i, D_c)$

$D_c = \{ r_1, r_2 \dots r_N \}$

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

N → no. of documents

n_i → no. of documents which contain w_i

$n_i \le N \rightarrow$ Always

$$\frac{N}{n_i} \ge 1$$

$$\log\left(\frac{N}{N_i}\right) \ge 0 \rightarrow (1) \; IDF \ge 0$$

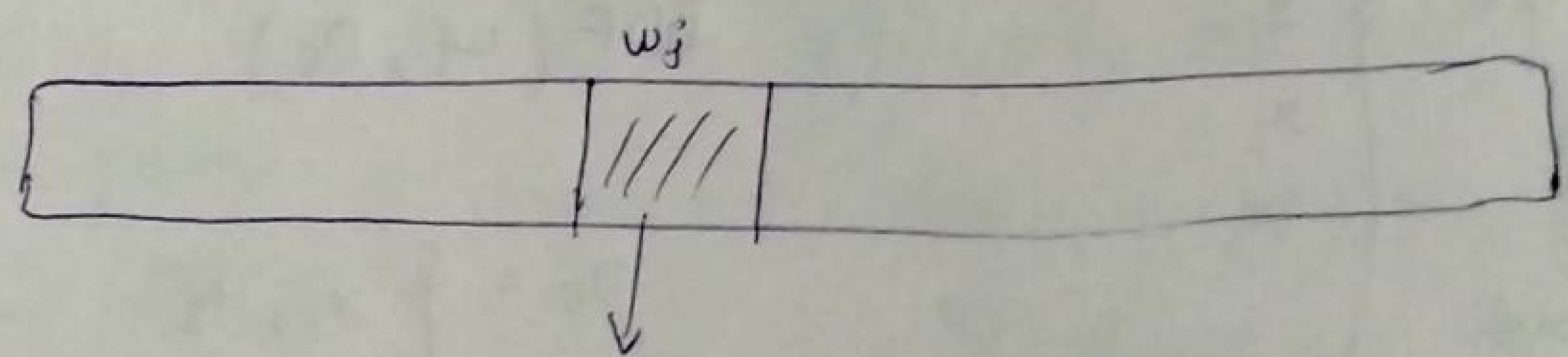
② if $n_i \uparrow$, $\frac{N}{n_i} \downarrow$, $\log\left(\frac{N}{n_i}\right) \downarrow$

as log is a monotonic functions

$f(x)$ behaviour is same as x.

// So, if a word w_i is more frequent in a document corpus then its IDF will be low.

Now, using TF & IDF we will fill TF * IDF in BOW table.

BOW = 

$TF(w_j, r_i) * IDF(w_j, D_c)$

(w_j is frequent in r_i) (w_j is rare in D_c)

TF - IDF :→

- more importance to rarer words in D_c.
- more importance if a word is frequent in a document / review.

Still - Semantic meaning problem is not solved.

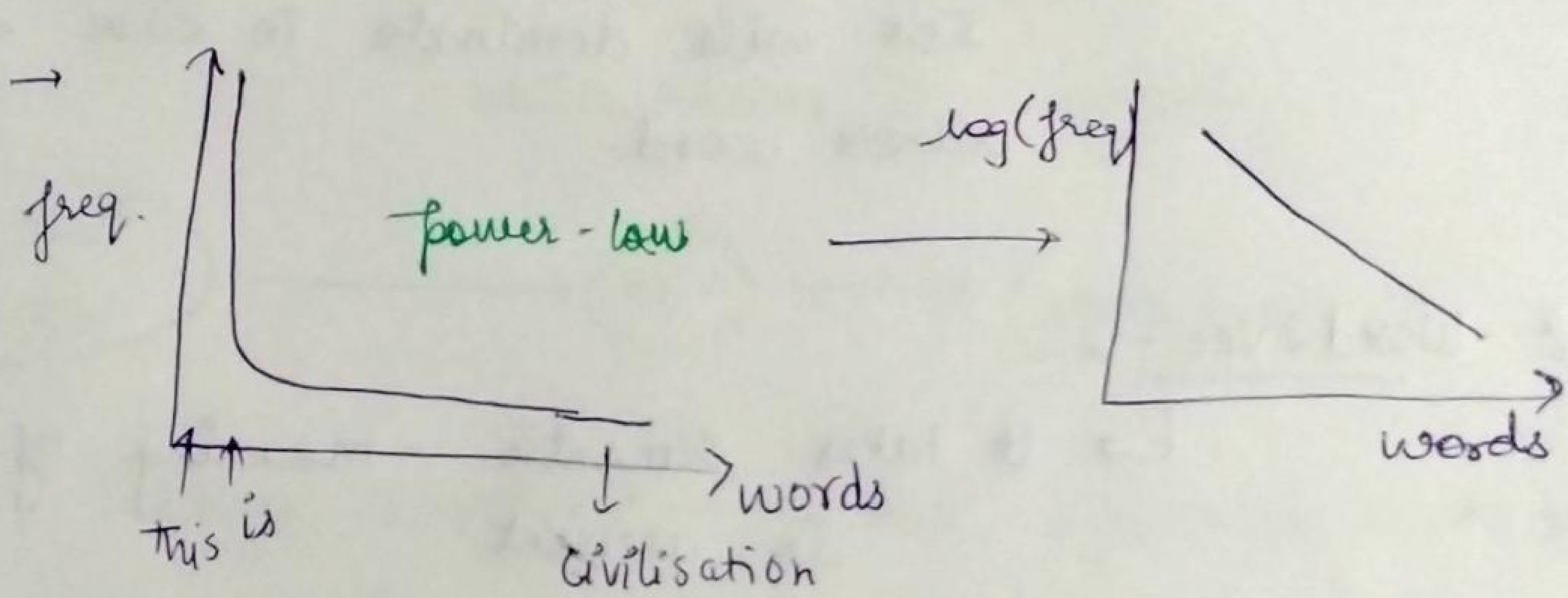
(tasty → delicious)

(cheap → affordable)

These are similar words but in TF_IDF representation also tasty will be 1 dimension & delicious will be other dimension.

Q. Why do we use $\log\left(\frac{N}{n_i}\right)$ for IDF ?

⇒ Zipf's law →



→ decreasing order of frequency

So, Zipf's law say frequency of word & words graph has power law distribution.

and whenever there is a power law distribution, taking log is a good idea.

as managing straight line graph is easier.

⇒ Another practical reason is :→

	$\frac{N}{n_i}$		$\log(\frac{N}{n_i})$	
The →	1	→ diff is 1000 here	0	difference is 7 here
is →	1		0	
⋮			⋮	
civilisation →	1000		~ 6.9	

1 in 1000

So, on calculation TF * IDF, IDF will dominate in case of rarer word.

* Word2Vec :→
  ↳ It takes semantic - meaning of words in account.
  which others were not doing.

For ex :→

w_1 , w_2 , w_3

w_1 → tasty
w_2 → delicious
w_3 → baseball

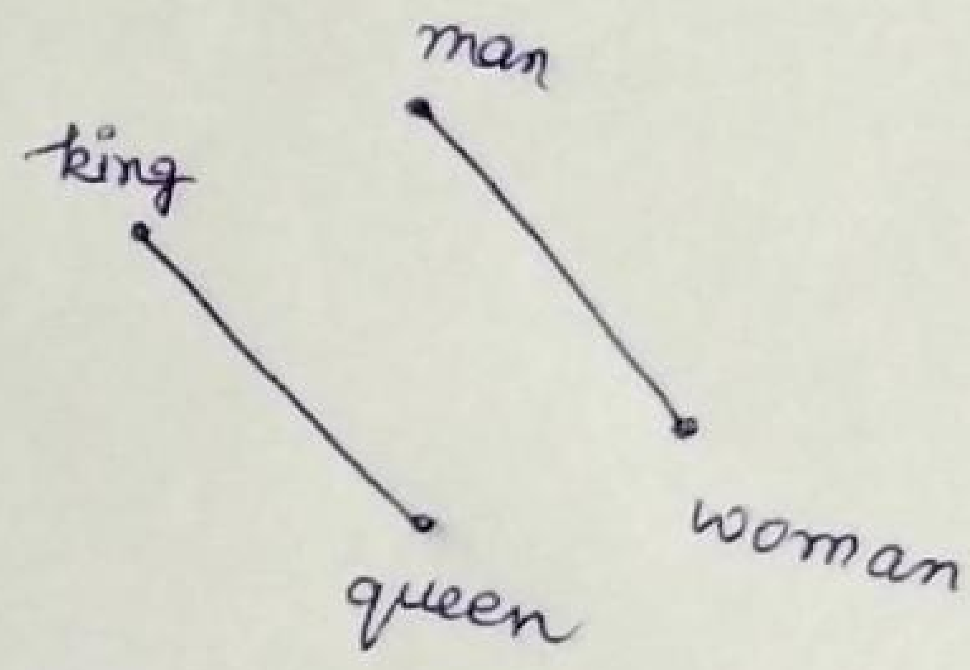
→ W2V → v_1, v_2, v_3

Things W2V tries to achieve :→

① { if w_1 & w_2 are semantically similar then v_1 & v_2 closer }

② It satisfy relationships

$(v_{man} - v_{woman}) \parallel (v_{king} - v_{queen})$



⇒ Imp W2V → learns relationships automatically from raw-text.

↓

which makes it magical.

large text corpus ⟶ W_2V ⟶ (word → vec)

larger dimensions ⟶ more info rich vectors

Core of word2vec :→

$\underbrace{w_1 \quad w_7}$ w_3 $\underbrace{w_4 \quad w_5 \quad w_2}$

neighborhood of w_3

$$W2V(w_3) = \left\{ \begin{array}{l} \text{if } Neigh(w_i) \simeq N(w_j) \\ v_i \simeq v_j \end{array} \right\}$$