Ch1 Text data cleaning, preprocessing and vectorization

Video 2: Data cleaning: Deduplication

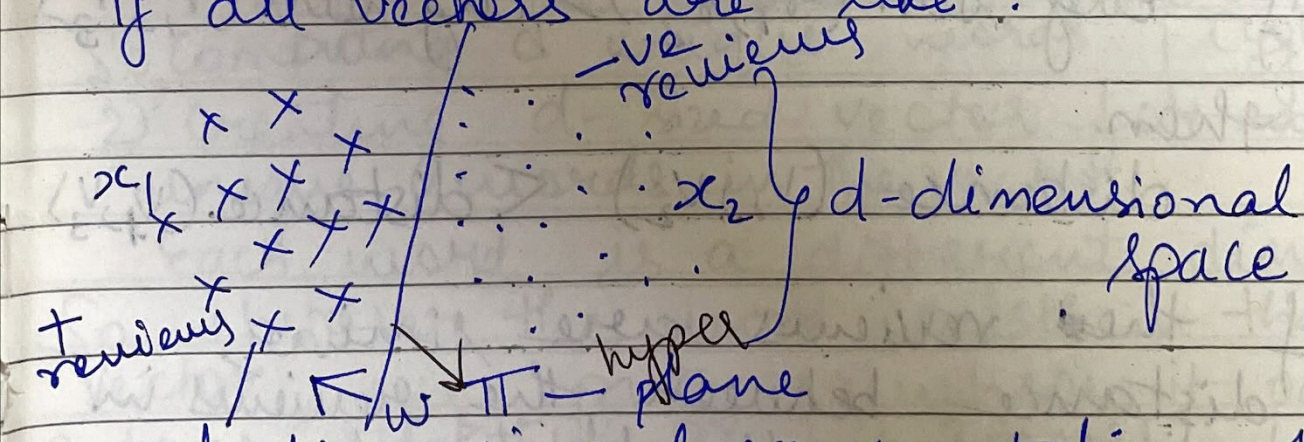
Video 3: Why Convert text to a Vector

We had studied vectors in linear algebra n-dimensional vectors time.

How do you convert text (English words and sentences) into numerical vectors.

Review text in the Amazon Fine Food Reviews is an n-Dim vector.

If all vectors are like :-



d-dimensional representation of vectors.

W — normal to the plane

$W^T x_1 \rightarrow$ +ve

$W^T x_2 \rightarrow$ -ve

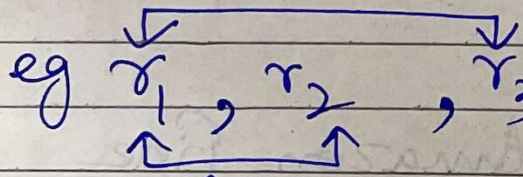
If I somehow could find my w then if $w^T x_i > 0$ then x_i is +ve else x_i is −ve.

x_i = review.

Steps:

1) Converting text to d-dim — is the first step then we find the plane to separate.

2) Converting text to d-dim has some rules.



Similar semantically

English language

v_1 v_2 v_3 are vectors in d-dim space

If Eng Sim (r_1, r_2) > Eng Sim (r_1, r_3)

then

distance (v_1, v_2) < distance (v_1, v_3)

If the reviews are similar, distance between the reviews in d-dim space should be less, ie, they should be close.

length of $v_1 - v_2$ < length of $v_1 - v_3$

minus

It will be easier to find plane when distance b/w the text/reviews is closer.

Conclusion:-

So, find text in d-dim vector such that similar text must be closer geometrically (computing geometric distance)

Video 4: Bag of Words (BoW)

Converts text to numerical vector

Steps in Bag of Words:-

1) Constructing a dictionary (not a python dict)

dictionary → create a set → Set of all the unique words across the ~~corpus. Document~~ corpus (collection of docs)

So, computing d unique words.

2) creating d-sized vector for each unique words in the vector.

Each word is a different dimension

So wherever the word exists in the vector, we put 1 or where it occurs 2 times, we put 2 or 3 if 3 times.

$V_i \in \mathbb{R}^d$

V_i is a sparse vector.

Note

Semantically, in English.

Similar text must result in closer vectors.

length of $(V_1 - V_2) = ||V_1 - V_2||$

Toy example :-

	This	pasta	is	very	tasty	not	and
V_1 :	1	1	1	1	1	0	1
		Subtract	↕	↕		↕	
V_2 :	1	1	2	0	1	1	1
	=	=	=	=	=	=	=
	0	0	1	1	0	1	0

r_1 : This pasta is very tasty and affordable.

r_2 : This pasta is not tasty and is affordable

So length $(V_1 - V_2) = \underbrace{||V_1 - V_2||}_{\text{norm}}$

$= \sqrt{1^2 + 1^2 + 1^2}$

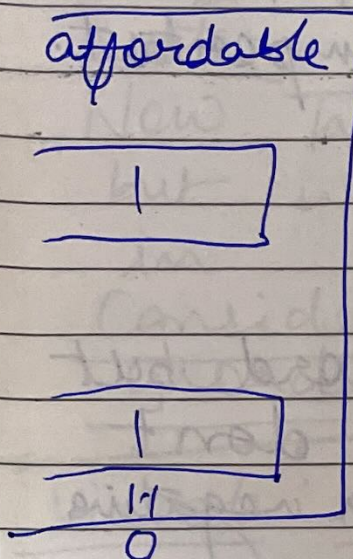
$= \sqrt{3}$

length of $V_1 = 8$

Bag of words doesn't work well with small difference in words.

~~like~~ Variations of Bag of words:-

1) Binary Bag of words / Boolean BoW



Instead of count put 1 if it occurs (0 otherwise) [atleast once]

$\|V_1 - V_2\| \sim$ # of differing words.

$\sqrt{(1)^2 + (1)^2} = \sqrt{2}$ distance b/w V_1 & V_2 in Binary BoW

Words like This is and are not very useful. Also called as Stop Words. But words which are trivial matter more.

Doubt

Video 5 : Text preprocessing : Stemming, Stop Word removal, Tokenization, Lemmatization

Stop words → and, is, This, etc.
words which are not important.

Very is a stop word.

In english not is a stop word but for some sentences we don't get to know that it is a negative sentence, so be careful.

1) Removing Stopwords carefully is called Text preprocessing. not is a stopword in case which might change the meaning of the sentence after removal

2) Make all your words lowercase.

3) Stemming :- Eg- Tasty and Tasteful has base word as Tast
So instead of separating these words of same base lets make them one.

Multiple algorithms for Stemming
a) Porter Stemmer (very old)
b) Snowball Stemmer. (more powerful)

5) Lemmatization: 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 neighbouring sentences or even an entire document.

4) Tokenization:

Tokenization is all about breaking a sentence into words and groups them so they form a meaning (eg- New York)
It is done by linguists

Note:-
Now Tasty and Delicious are synonyms but we create 2 different dimensions in Bag of words. But in W2V Considers semantic meaning of words unlike Bag of words.

Video 6: unigram, bigram and n-grams

This comes under lemmatization.

Unigram - Each word is considered a dimension.

Bigrams - Pair of consecutive words are considered as dimension.

Tri-grams - 3 consecutive words make a dimension.

n-grams. n=1 Uni grams
n=2 Bi grams
n=3 Tri "

Unigram BoW - discards the sequence info.

But if you use bigrams, trigrams or n-grams maintain the sequence information in BOW, it will be good.

~~#bigrams > #unigrams.~~

~~add~~

#trigrams > #bigrams > #unigrams.

So, n-grams ($n>1$) dimensionality increases drastically.

Used in Information Retrieval (BoW, W2V, TF/IDF

↓

Video 7 tf-idf (term frequency - inverse document frequency)

word, review/sentence

$$TF(w_i, r_j) = \frac{\#\text{ of times } w_i \text{ occurs in doc } j}{\text{Total \# of words in doc } j}$$

word, document

Defn

↓ So $0 \leq TF(w_i, r_j) \leq 1$

Term Frequency is the probability of finding a word w_i in a document doc j.

Try: D (data corpus) = Collection of documents.

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

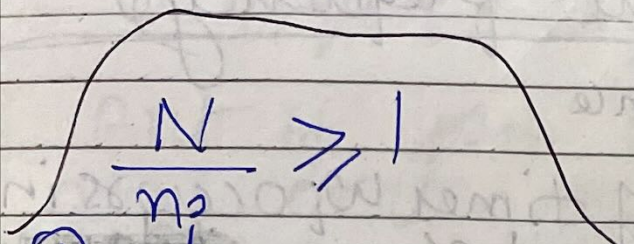
Data Corpus

where N = # of documents in Corpus

n_i = # of docs in corpus which contain w_i

TFIDF → Doesn't take the semantic meaning of words like Tasty & Delicious.

Note $n_i \leq N$



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

① So $\log\left(\frac{N}{n_i}\right) \geq 0$ $\because \log(1) = 0$.

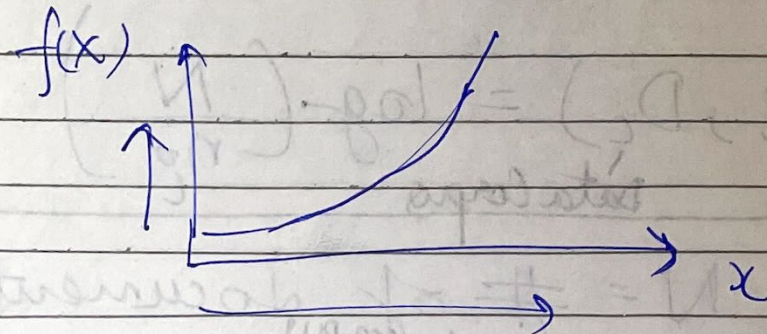
So IDF ≥ 0.

②

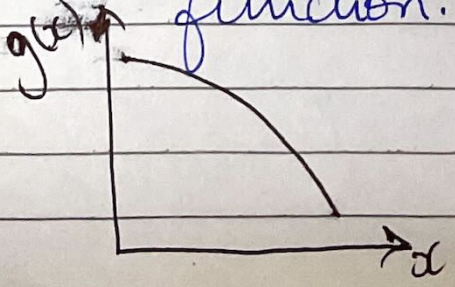
If $n_i \uparrow$, $\frac{N}{n_i} \downarrow$

Then $\log\left(\frac{N}{n_i}\right)$ also $\downarrow$

log is a monotonic function.



~~Axiom:~~ Monotonically increasing function.



As $x \uparrow$ $g(x) \uparrow$, it is monotonically decreasing fun.

$\log(x)$ is valid only if x is (+ve).

→ If w_i is more frequent in my corpus then its IDF will be lower.

→ As w_i is a rare word then its IDF will be higher.

If we combine TF & IDF

$$TF\text{-}IDF = TF(w_i, r_j) * IDF(w_i, D)$$

r_j → doc, D → corpus

→ TF is ↑ when word occurs freq. in document
frequent → TF ↑

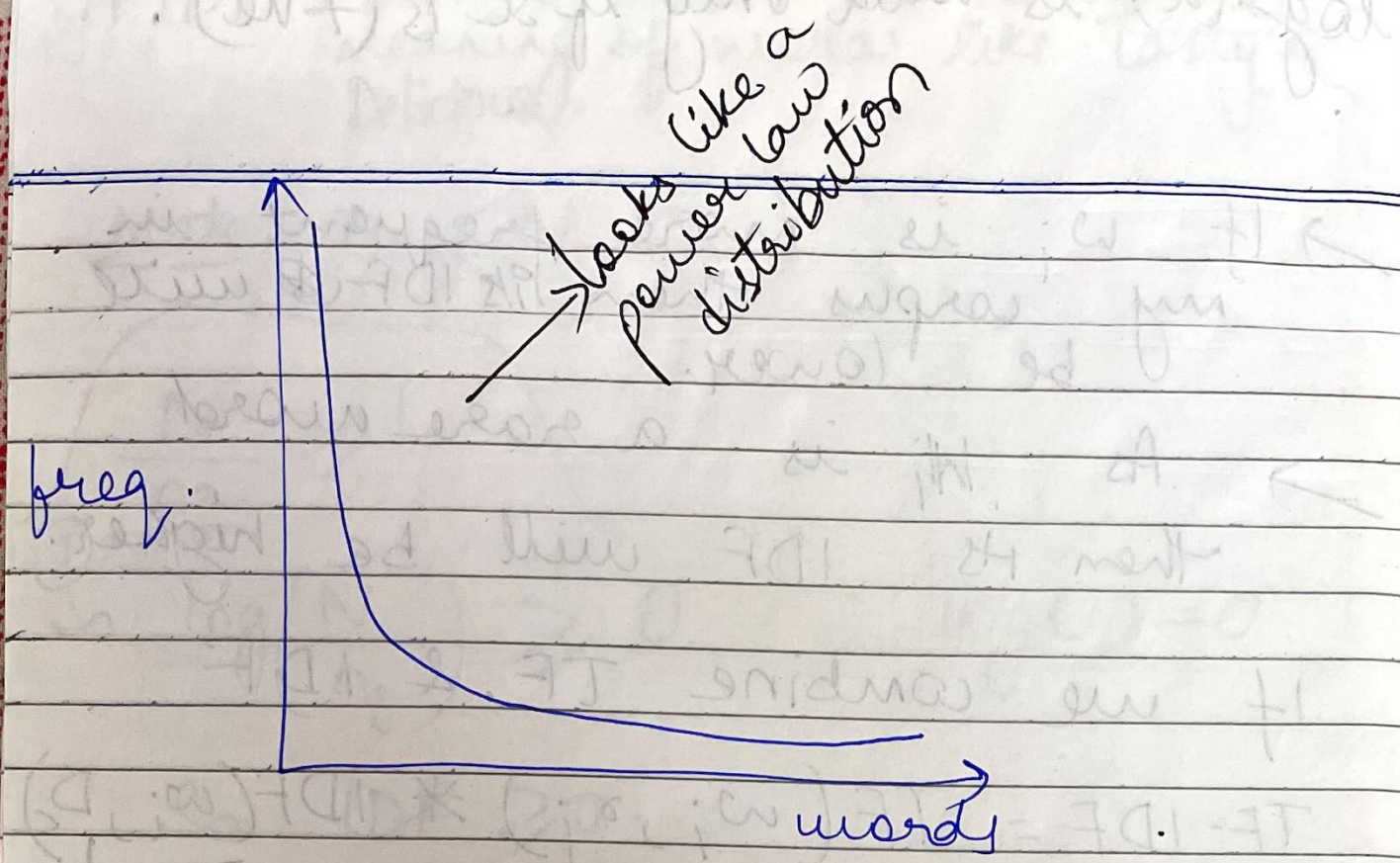
→ IDF is ↑ when word is rare in corpus
rare → RIDF ↑

Video 8: Why use log in IDF?

log was just used as a heuristic or a hack.

It's explanation given in Zipf's law. In Zipf's law if frequency of words is in y axis and words occurs on x-axis. The (histogram) graph falls very sharply

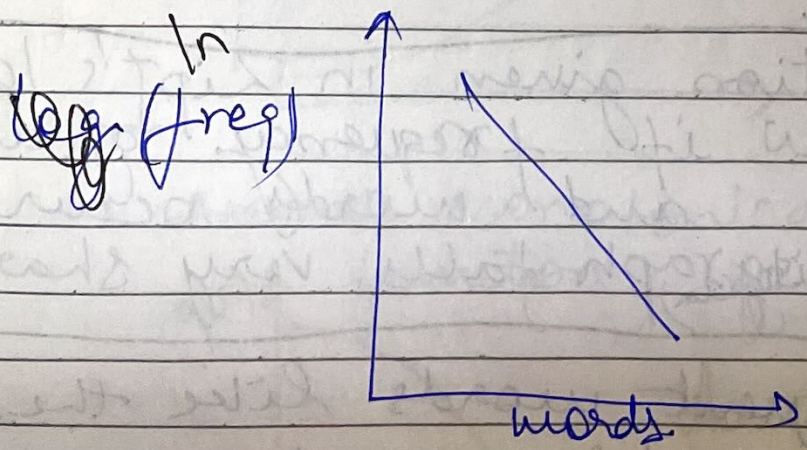
More frequent words like the, is, of etc are more higher and many more words which are trivial occur less frequently



If we plot log of freq. it becomes a straight line in graph.

It is just like we convert power law to Gaussian distribution by taking log of it (Box Cox transform)

We are taking ln (natural log).



$\log_e(x) = \ln(x)$ same
natural log

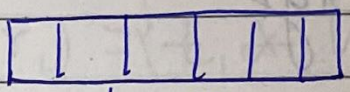
In this way IDF in TF * IDF has less dominance

So TF-IDF is in range 0-20 not in 1000 (as it may happen if we don't take log$_e$).

When this TF and IDF both are in same range then their multiplication makes sense and IDF over TF is equal in almost not too dominant and TF not too less dominant.

Video 9: Word2Vec

It takes semantic meaning of the words/sentences unlike BoW and TFIDF

word → 

d-dim vector but not a sparse vector and as a dense d-dim vector.

BOW & TFIDF → Sparse
W2V → Dense

Word2Vec takes only 1 word into consideration unlike the first 2 which take entire sentence into consideration

① Similar

If W_1 and W_2 are semantically closer then these points will be closer.

② It also satisfies relationships between words, like Male-Female gender of certain words.

and it can also consider Country capital relationships.

It learns all these relationships almost on its own.

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

Mikolov published this for W2V and Google uses it essentially in almost every NLP.

You need to give a very large text corpus eventually the dimension will also become large.

Essentially W2V looks at the neighbours of each word.

W2V from Google News is a 300 dim Vector

Neighbour $N(w_i) \approx N(w_j)$

vector $- V_i \approx V_j$

$$\frac{tfidf(W_1) * W2V(W_1) + \cdots}{tfidf(W_1 \text{ in doc}) + tfidf(W_2 \text{ in doc}) + \cdots}$$

The below strategies work for converting sentences to vectors.

Video 10: Avg Word2Vec, tf-idf weighted Word2Vec

Avg W2V

doc r_1: $w_1\ w_2\ w_1\ w_3\ w_4\ w_5$ — n_1 words

$$V_1 = \frac{1}{n_1}\left[W2V(w_1) + W2V(w_2) + W2V(w_1) + W2V(w_3) + W2V(w_4) + W2V(w_5)\right]$$

(W2V(w_i): vector representation of word w_i)

each $W2V(w_i)$ is a d-dim vector so we take avg. of its sum.

Works fairly well, not always.

TFIDF weighted W2V

doc r_1: $w_1\ w_2\ w_1\ w_3\ w_4\ w_5$

w_1	w_2	w_3	w_4	w_5	w_6	w_7
t_1	t_2	t_3	t_4	t_5	t_6	t_7

tfidf vector for r_1 doc

$$tfidf\ W2V(r_1) = \{t_1 * W2V(w_1) + t_2 * W2V(w_2) + t_1 * W2V(w_1) + t_3 * W2V(w_3) + t_4 * W2V(w_4) + t_5 * W2V(w_5)\} \div \{t_1 + t_2 + t_3 + t_4 + t_5 + t_6 + t_7\}$$

$$tfidf\text{-}W2V(r_1) = \frac{\sum_{i=words}(t_i * W2V(w_i))}{\sum_{i=words} t_i}$$