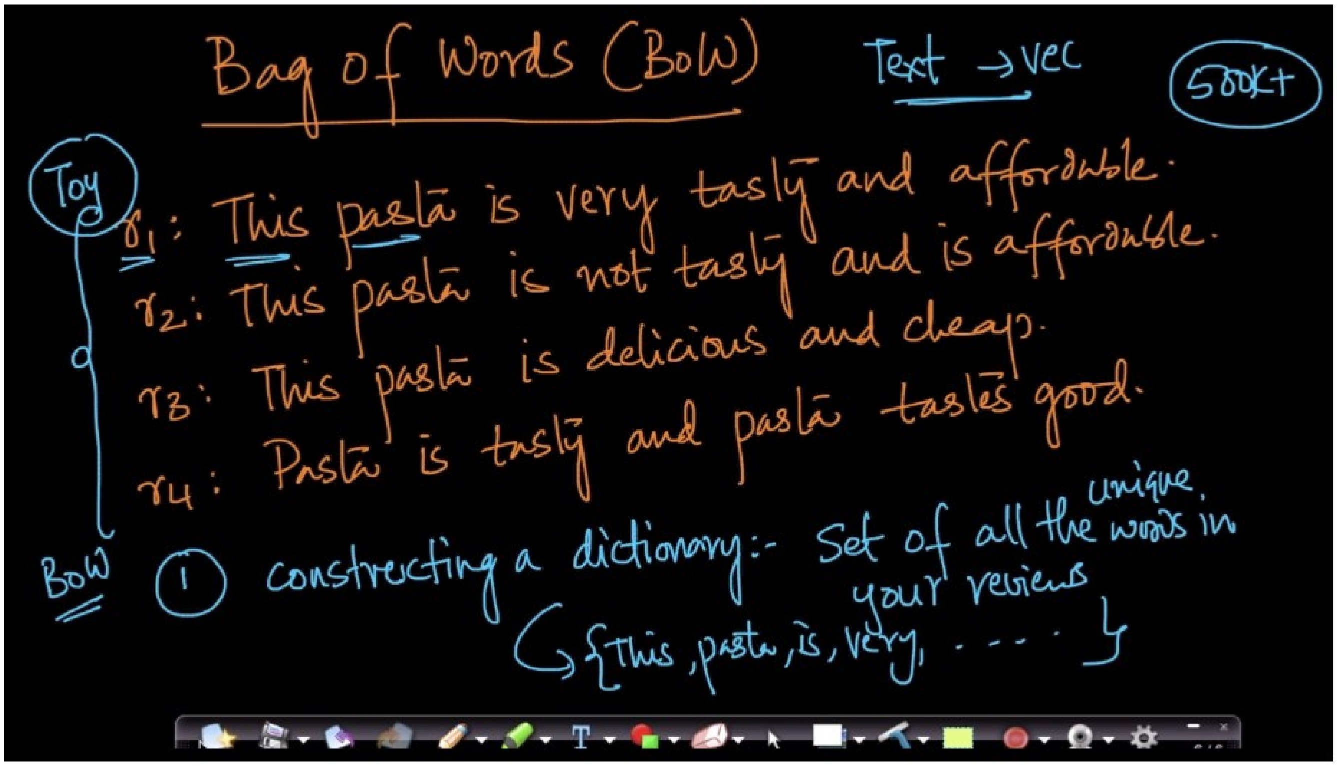
Amazon Fine Food reviews:
Bag of Words:



Bag of Words (BoW) Text → vec 500K+

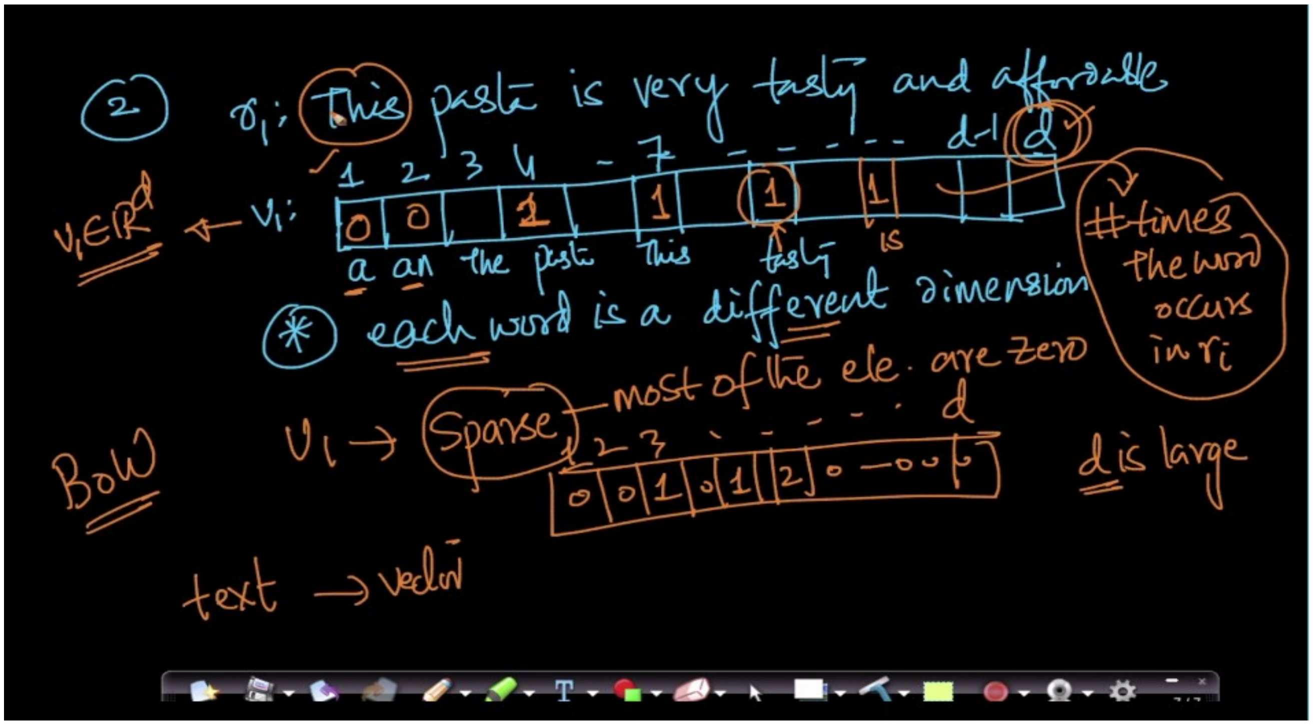
Toy

r1: This pasta is very tasty and affordable.
r2: This pasta is not tasty and is affordable.
r3: This pasta is delicious and cheap.
r4: Pasta is tasty and pasta tastes good.

BoW

① constructing a dictionary:- Set of all the unique words in your reviews
↳ {This, pasta, is, very, . . . }

Each of the review is called as document in NLP.
Constructing a vector:
A vector of length of each words that is ‘d’ unique words if constructed for each review. The frequency of each word in the review is counted. The vector will be highly sparse.



② r1: This pasta is very tasty and affordable

1 2 3 4 - 7 - - - - - d-1 d
0 0 1 1 1 1
a an the pasta This tasty is
$v_1 \in \mathbb{R}^d$ ← v_1:
# times the word occurs in r_i

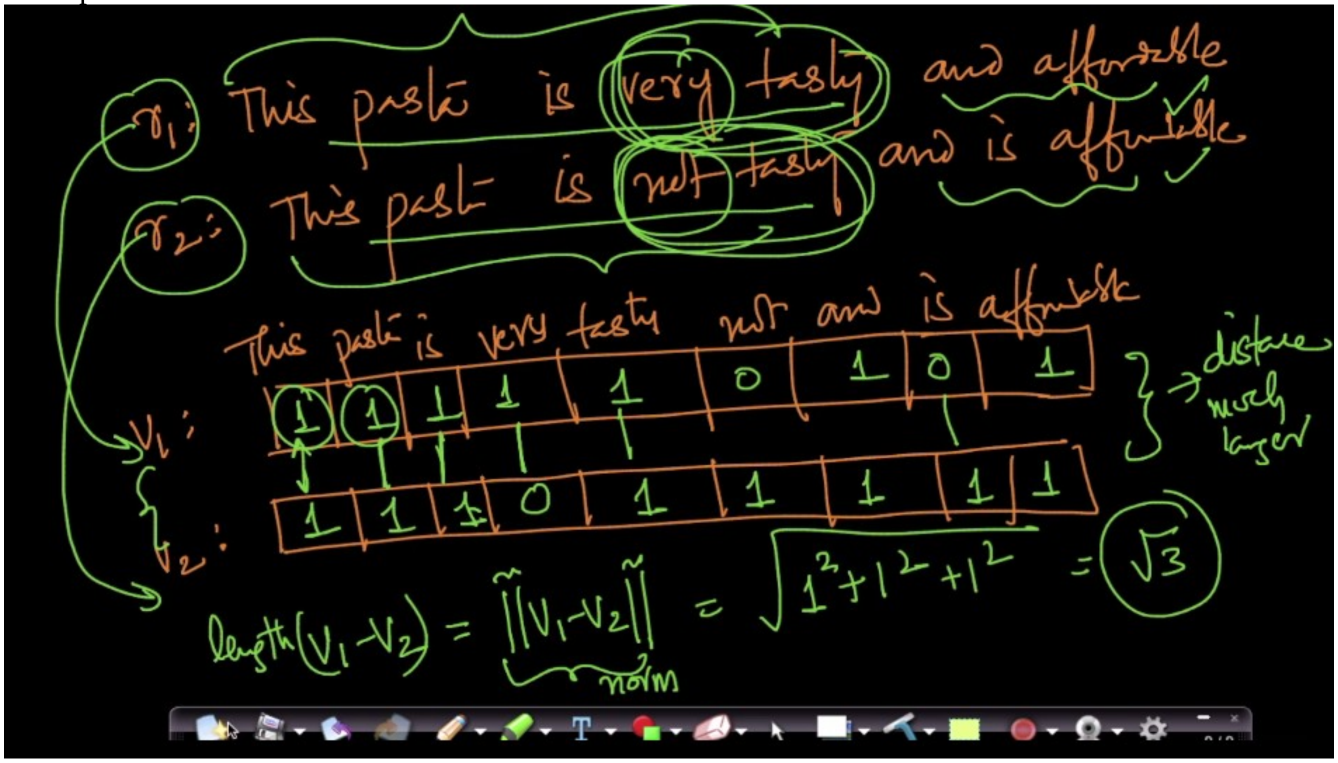
✱ each word is a different dimension

v_1 → Sparse — most of the ele. are zero

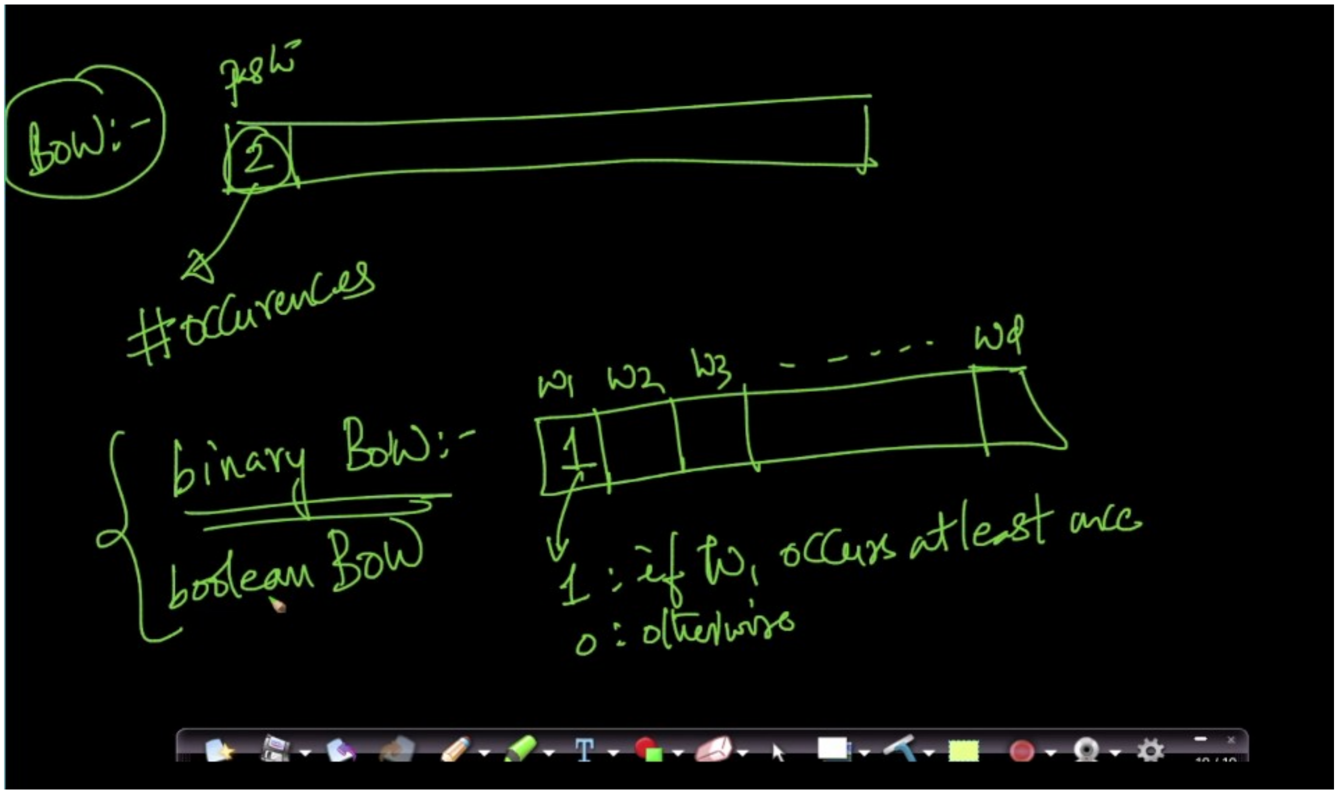
1 2 3 . . . . . d
0 0 1 0 1 2 0 — 0 0 0
d is large
BoW

text → vector

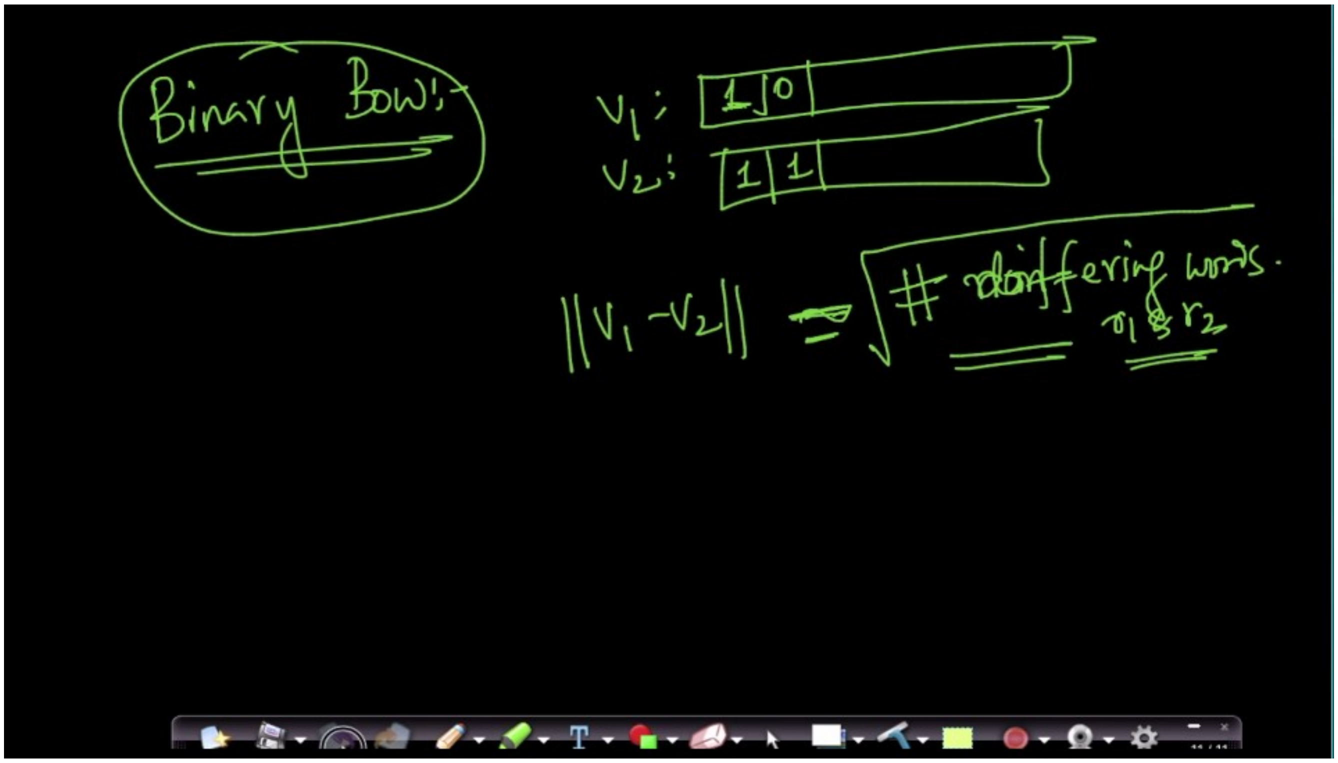
Our objective is similar text must be more similar. In BOW the opposite sentences may be at the closer distance, because it only counts the occurrence of the words in the sentence.
Example:


r1: This pasta is very tasty and affordable
r2: This pasta is not tasty and is affordable
This pasta is very tasty not and is affordable
v1: 1 1 1 1 1 0 1 0 1
v2: 1 1 1 0 1 1 1 1 1
distance much larger
length(v1-v2) = ||v1-v2|| = √(1²+1²+1²) = √3
norm

Binary bag of words: The word occurs or not in the vector.

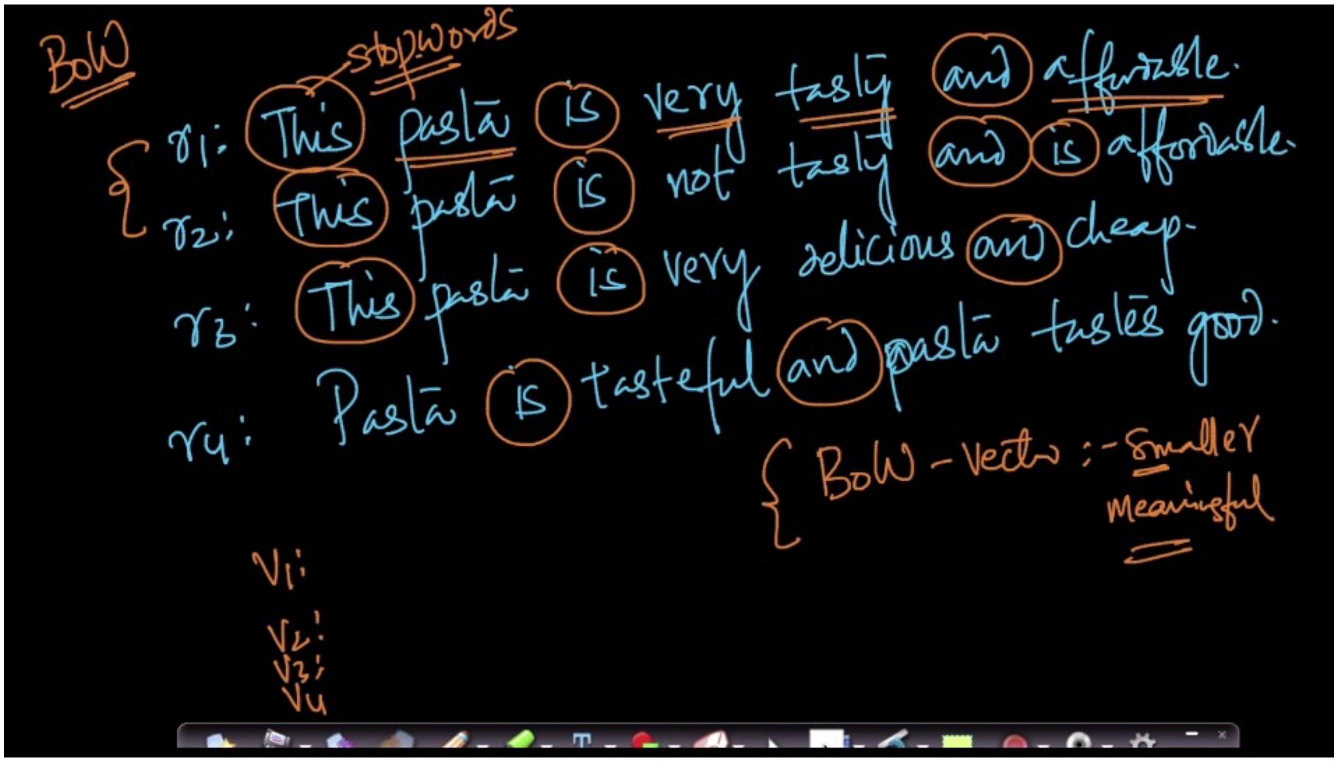


In BOW the distance between the vectors v1 and v2 is almost equal to number of differing words.

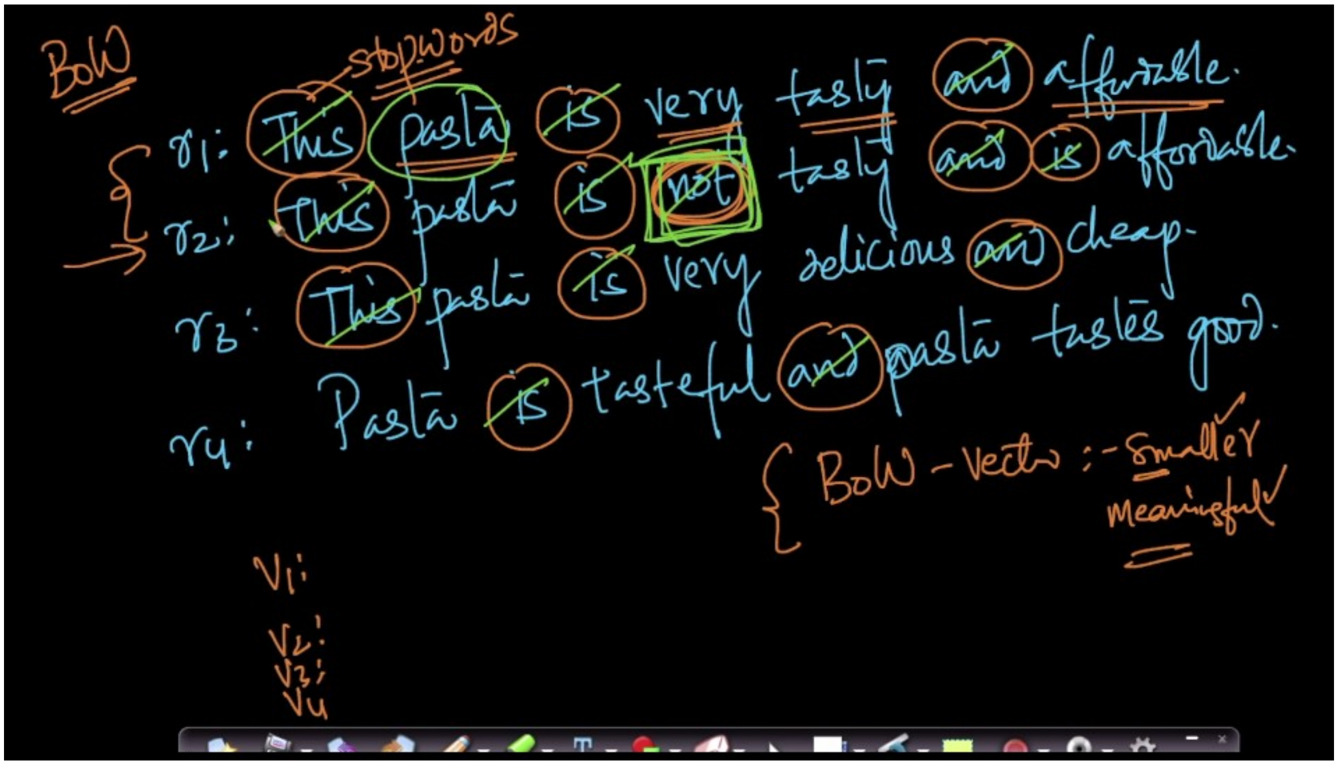

Binary BoW:-
v_1: 1 0
v_2: 1 1
$\|v_1 - v_2\| = \sqrt{\# \text{ differing words } r_1 \& r_2}$

The stop words are removed because they does not give any trivial meaning to the review.

Text Preprocessing- Stemming, Stop-word removal, Tokenization, Lemmatization:

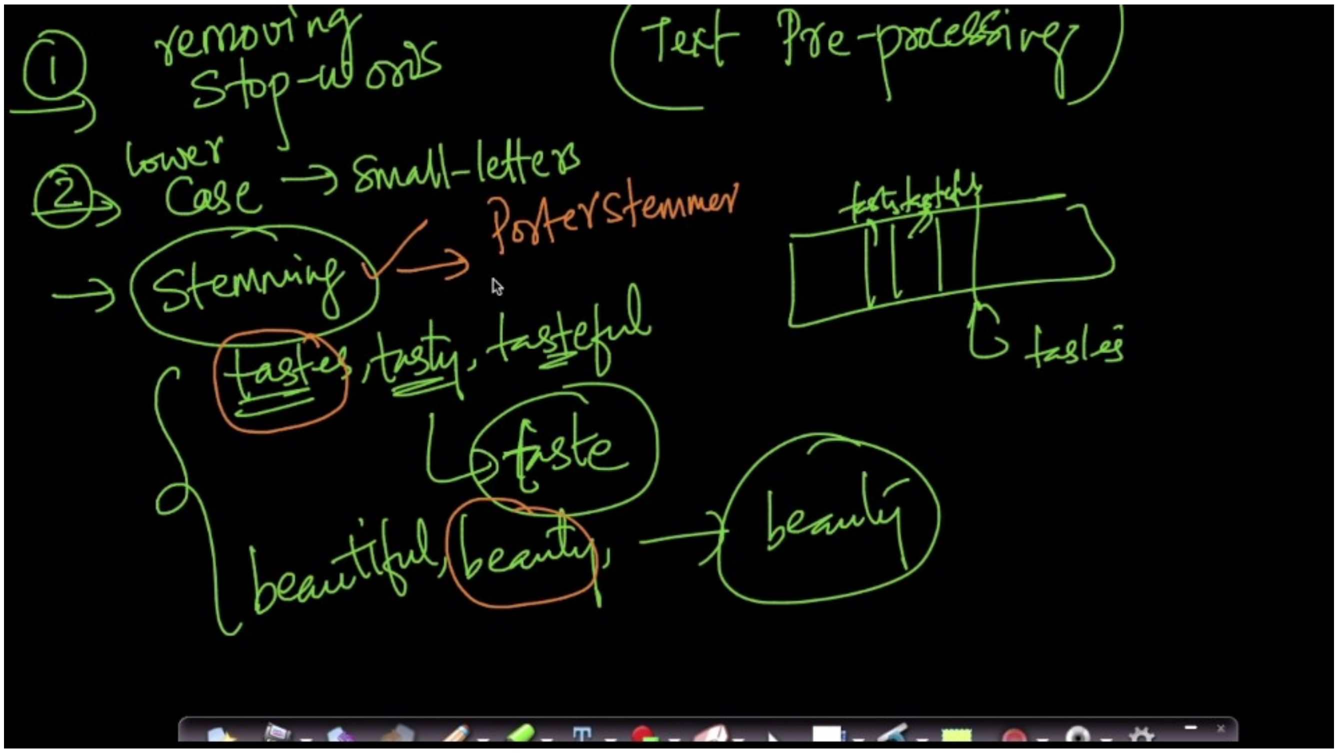

BoW
stopwords
r1: This pasta is very tasty and affordable.
r2: This pasta is not tasty and is affordable.
r3: This pasta is very delicious and cheap.
r4: Pasta is tasteful and pasta tastes good.
BoW - vector :- smaller
meaningful
V1:
V2:
V3:
V4

Removing stop words:


BoW
stopwords
r1: This pasta is very tasty and affordable.
r2: This pasta is not tasty and is affordable.
r3: This pasta is very delicious and cheap.
r4: Pasta is tasteful and pasta tastes good.
BoW - Vector :- Smaller, Meaningful
V1:
V2:
V3:
V4

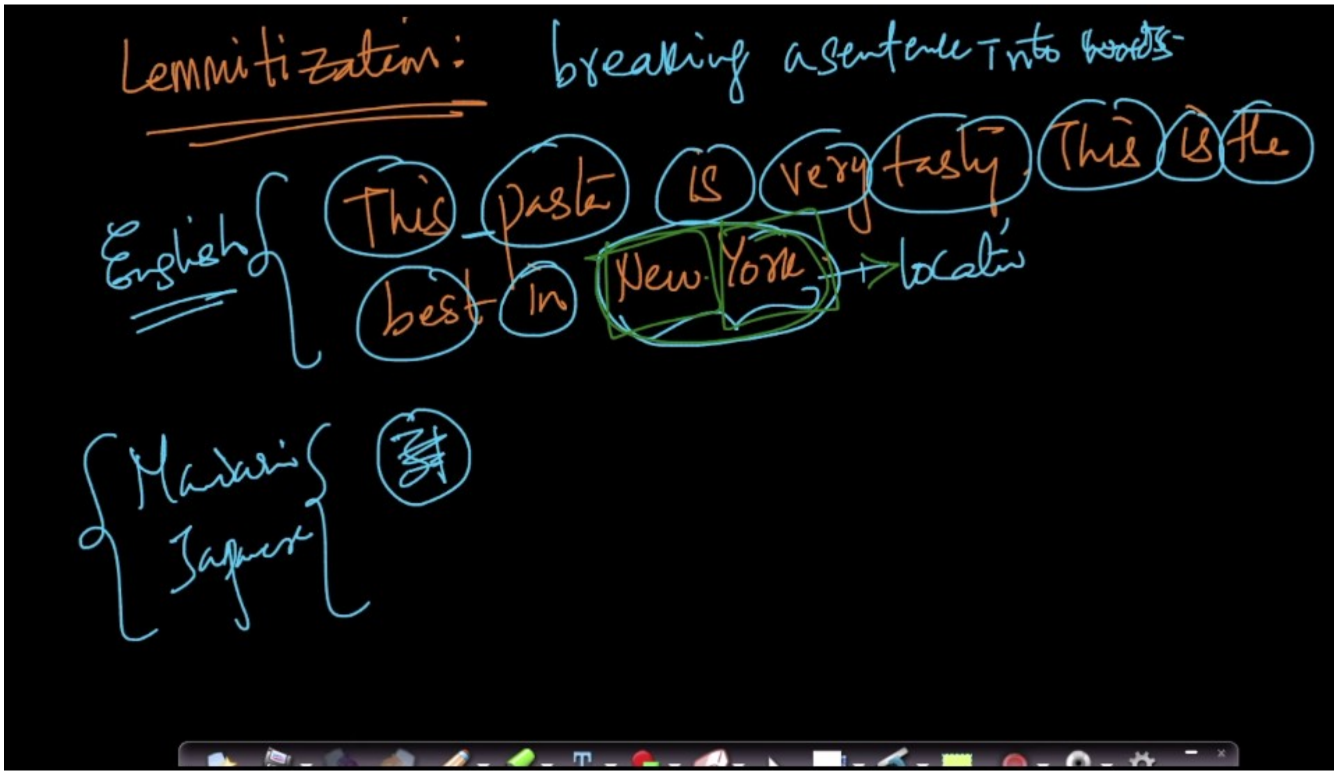
To make everything lower case: Removing the words that are in the uppercase.

Stemming:

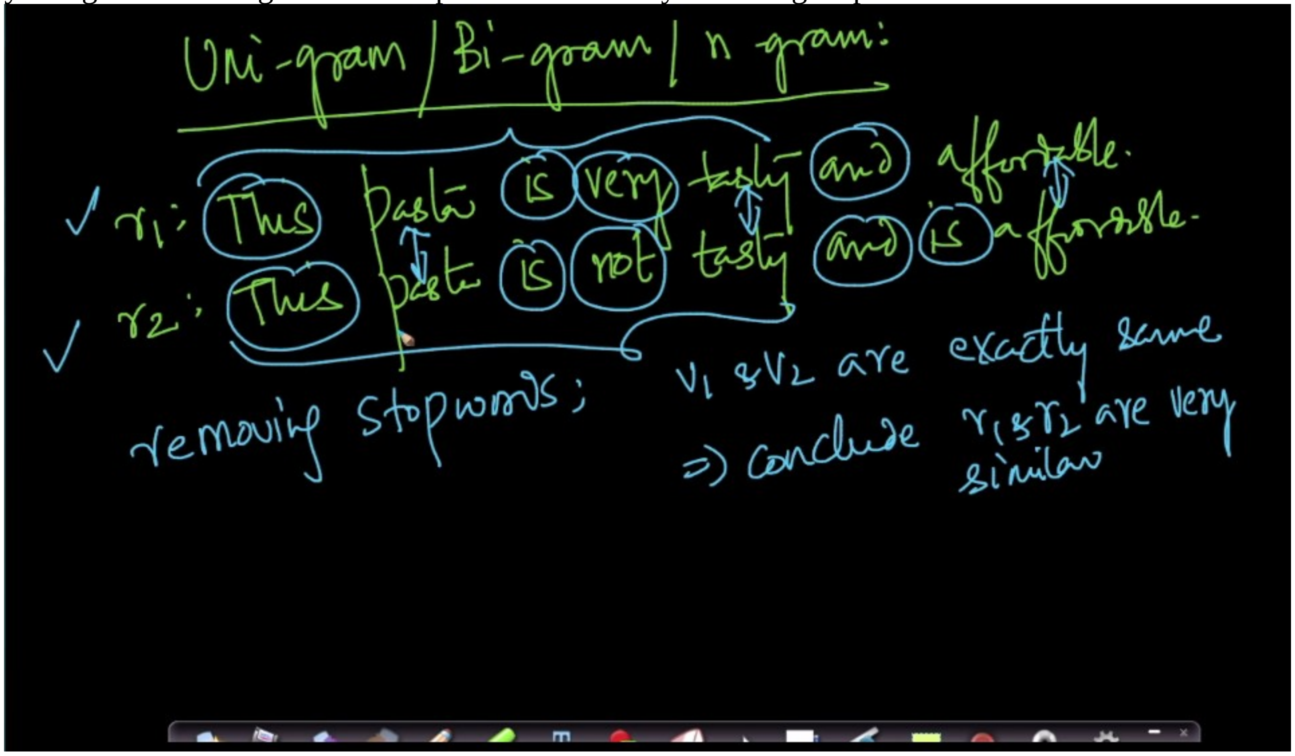

Text Pre-processing
① removing Stop-words
② lower Case → Small-letters
→ Stemming → PorterStemmer
tastes, tasty, tasteful
taste
beautiful, beauty, → beauty
tastes

Lemmatization:

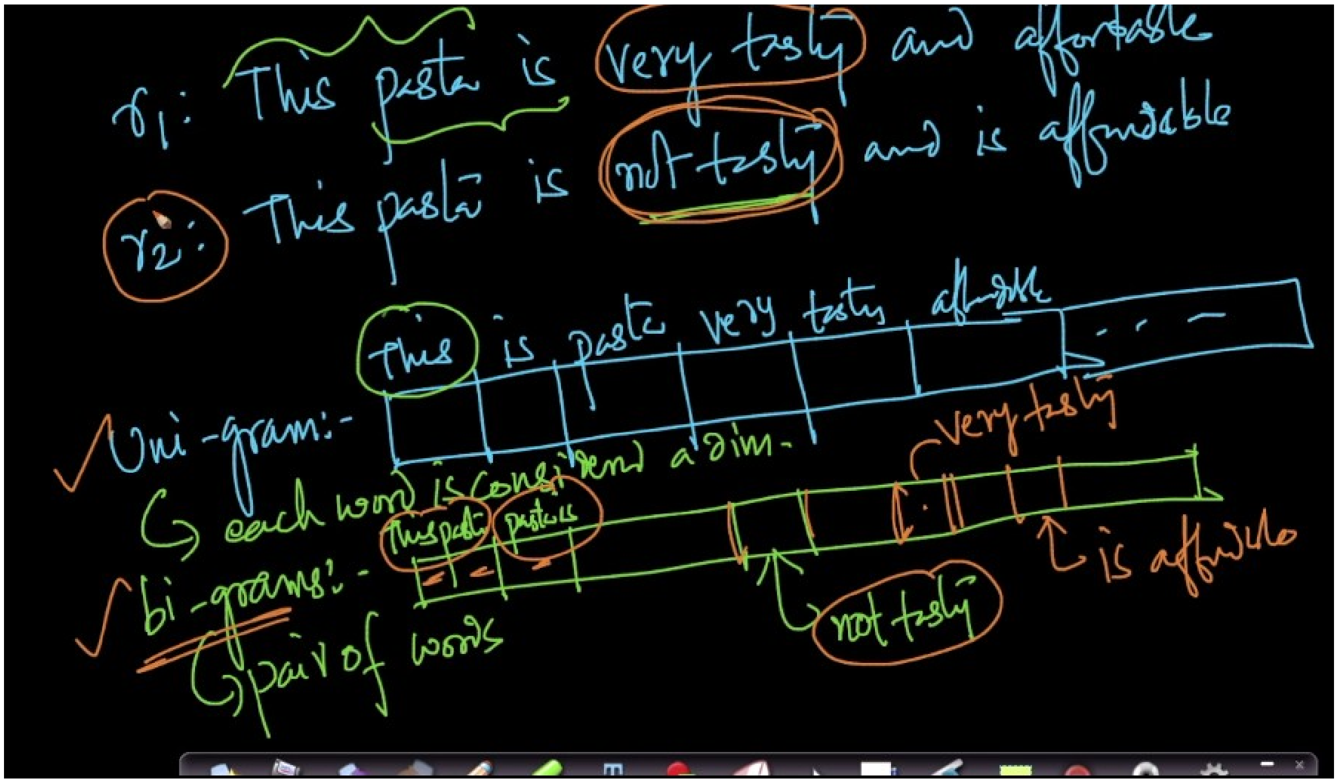
It is all about how to break the words, like New York.


Lemmitization: breaking a sentence into words
English
This pasta is very tasty. This is the best in New York → location
Mandarin
Japanese

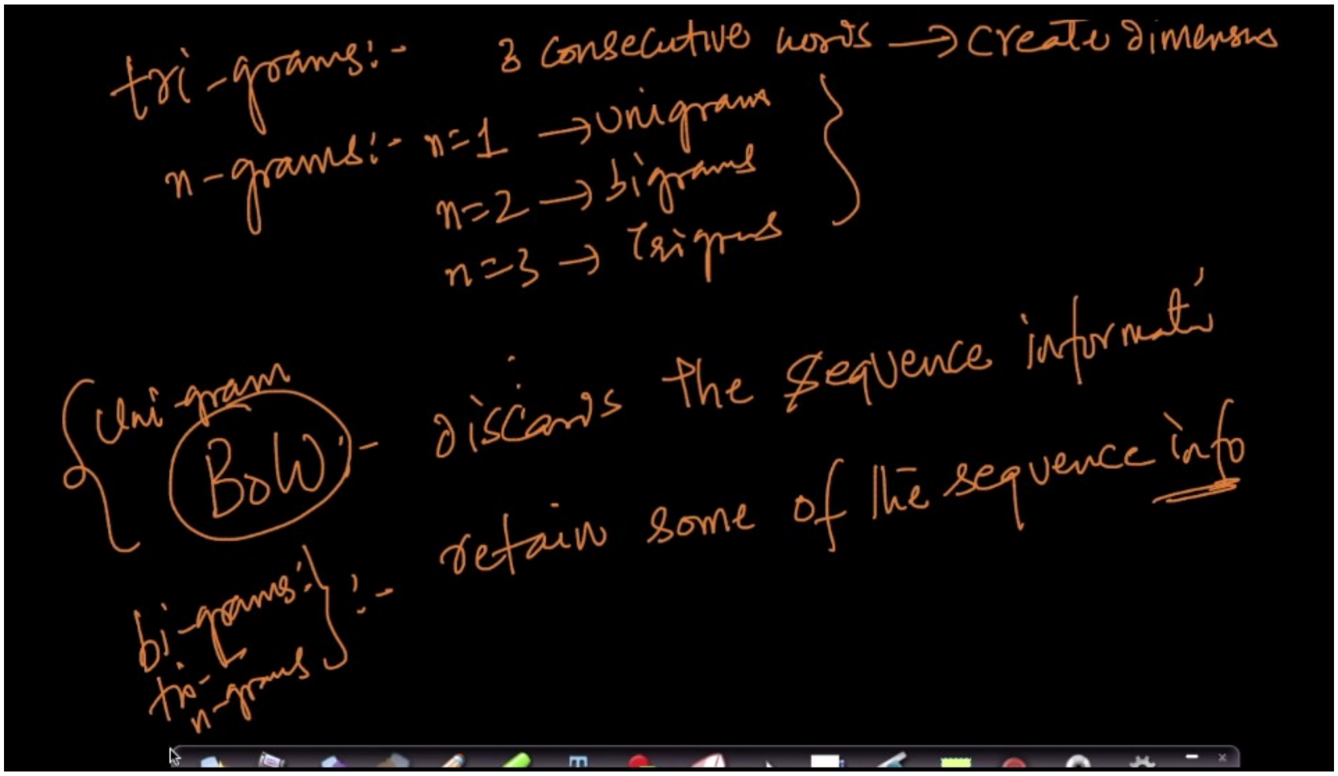
Semantic meaning of words: We will solve this problem by using Word2Vec.
Uni – gram, Bi – gram, N – gram:
By using this we can get out of the problem caused by removing stop words.


Uni-gram / Bi-gram / n gram:
✓ r1: This pasta is very tasty and affordable.
✓ r2: This pasta is not tasty and is affordable.
removing stopwords;
v1 & v2 are exactly same
⇒ conclude r1 & r2 are very similar

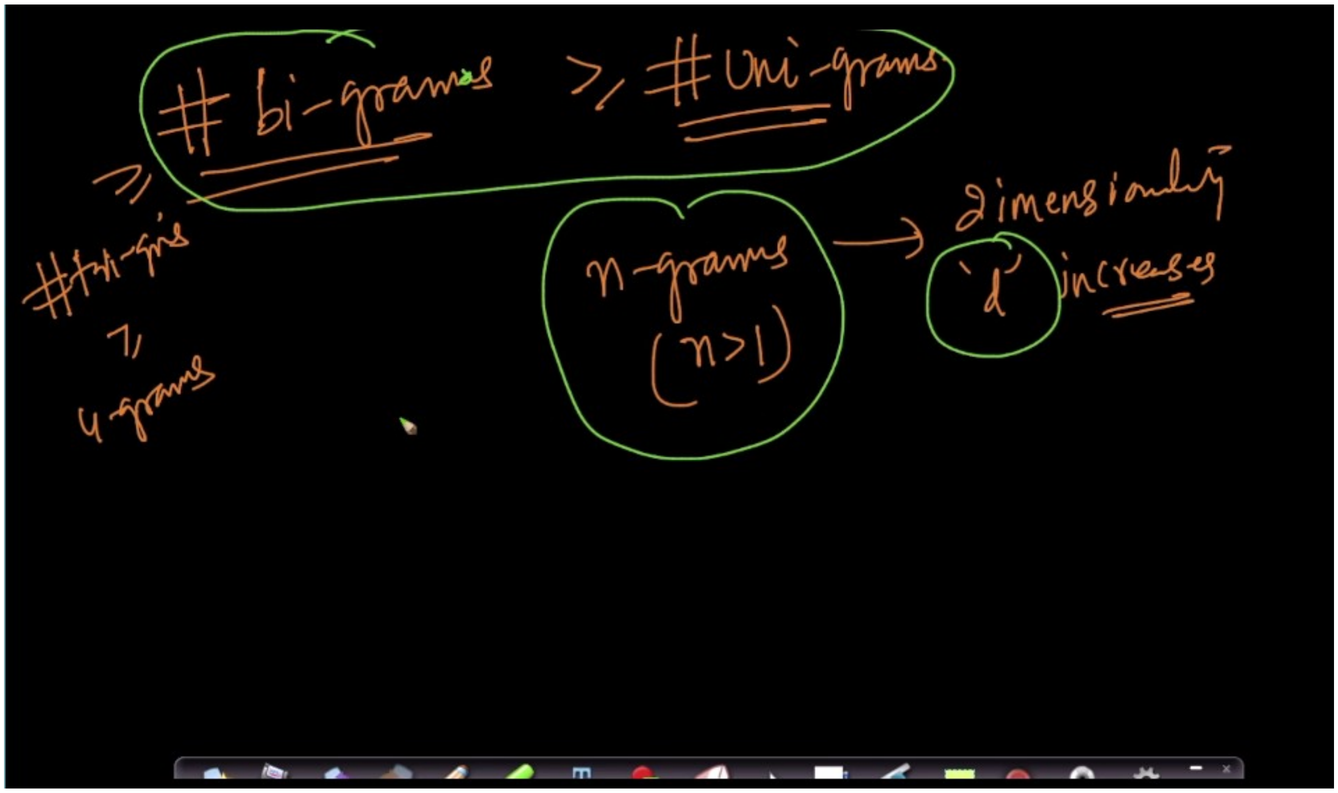
Bi – gram: In this we use pair of words in BOW.


r1: This pasta is very tasty and affordable
r2: This pasta is not tasty and is affordable
this is pasta very tasty affordable
Uni-gram:-
each word is considered a dim.
very tasty
this pasta
pasta is
not tasty
is affordable
bi-grams:-
pair of words

Tri- grams: 3 Consecutive words.

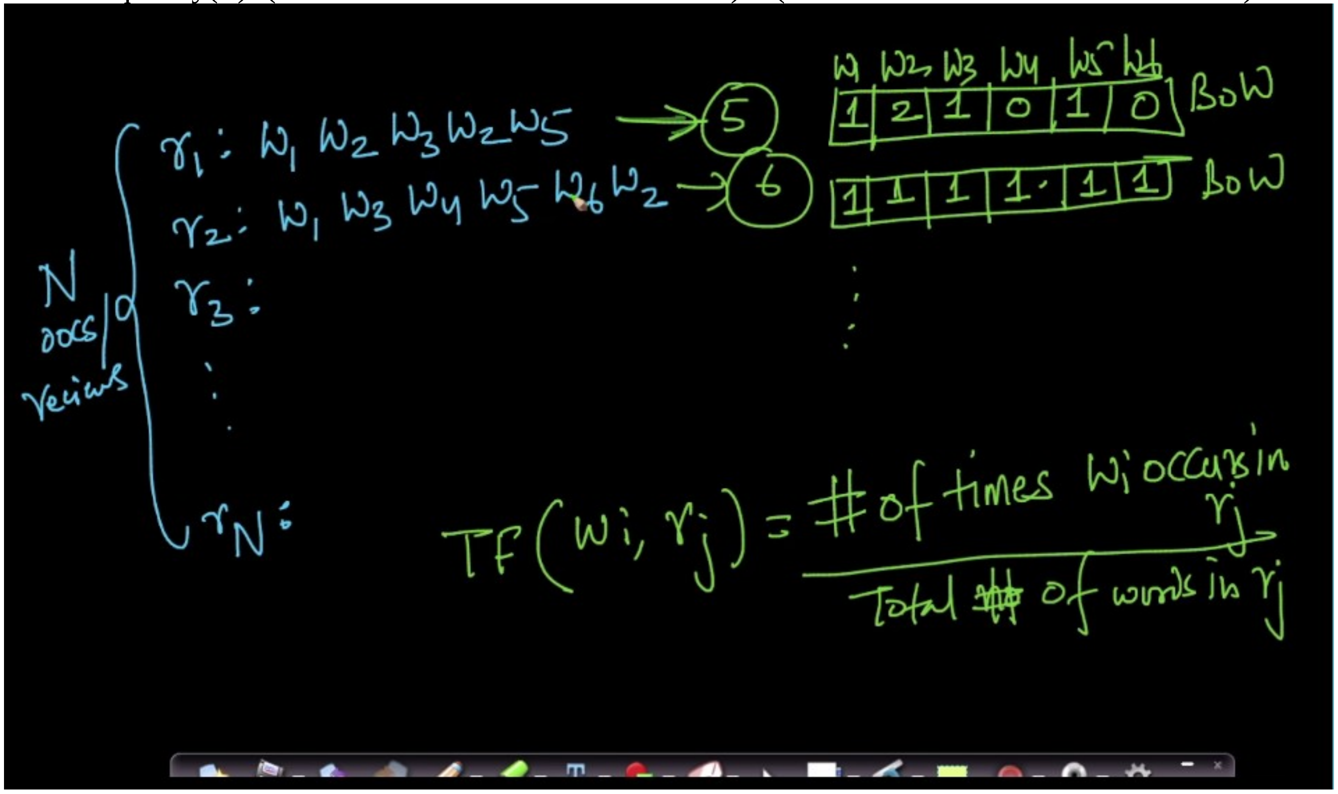

tri-grams:- 3 consecutive words → create dimensions
n-grams:- n=1 → unigrams
n=2 → bigrams
n=3 → trigrams
Uni-gram
BoW :- discards the sequence information
bi-grams:
tri-
n-grams
:- retain some of the sequence info

The number of bi grams are greater than equal to uni grams, the dimensions ‘d’ increases drastically.



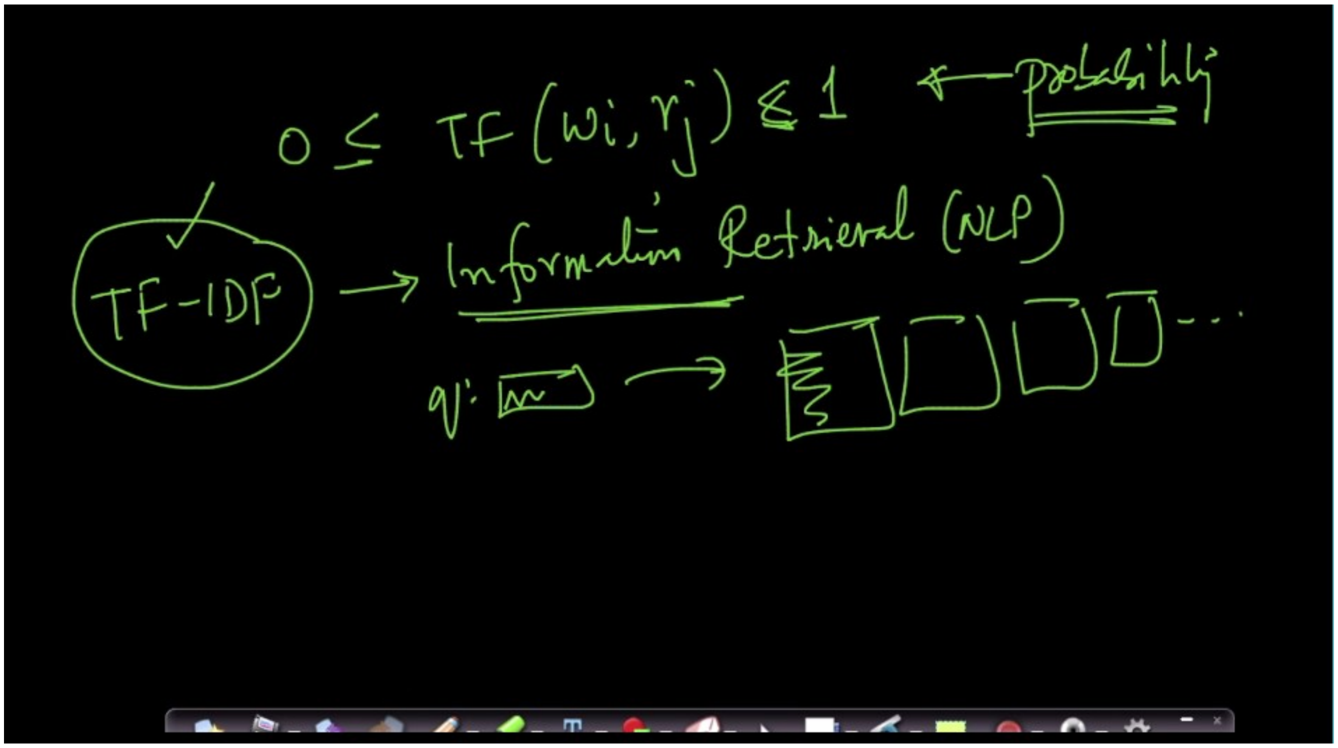
Tf – Idf:

Term frequency(tf): (The number of times a word occurs) / (total number of words in the review)



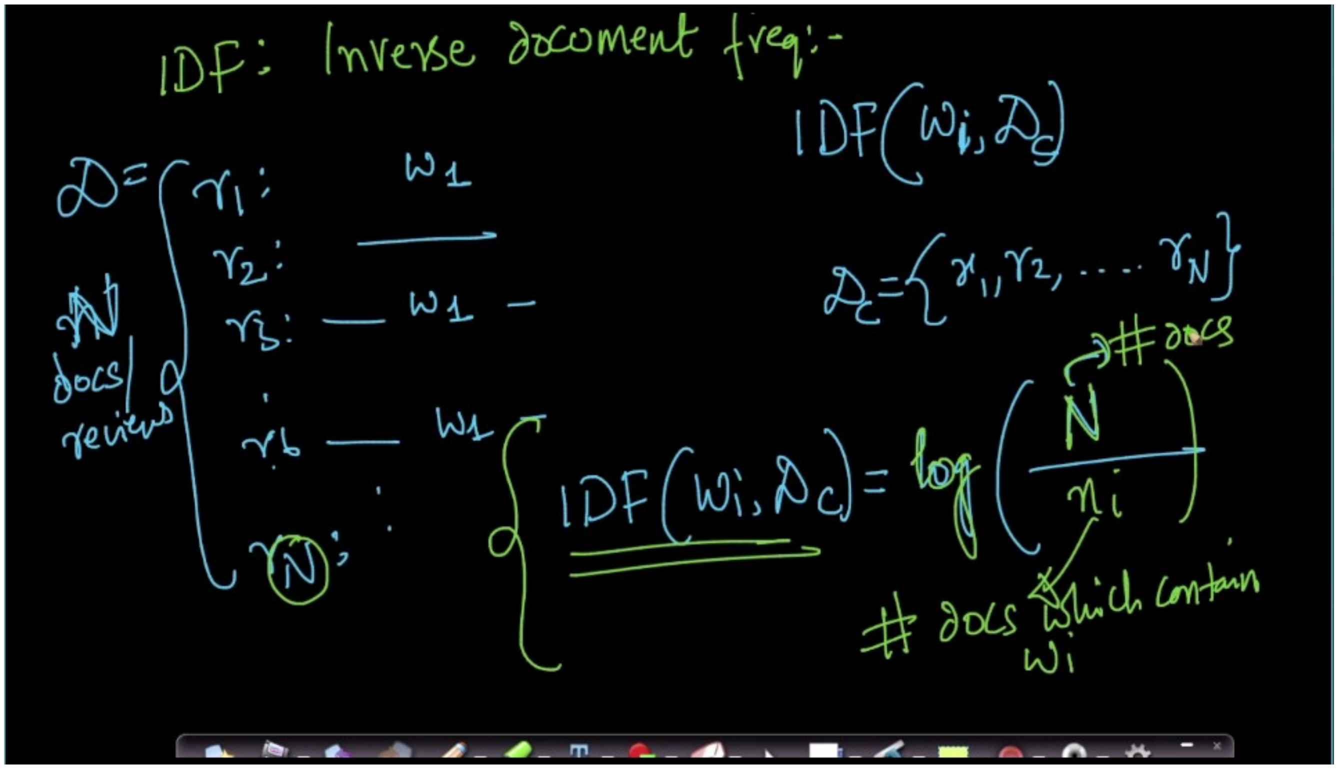
r_1: $w_1\ w_2\ w_3\ w_2\ w_5$ → 5
r_2: $w_1\ w_3\ w_4\ w_5\ w_6\ w_2$ → 6
r_3:
r_N:
N docs/reviews
$w_1\ w_2\ w_3\ w_4\ w_5\ w_6$
1 2 1 0 1 0 BoW
1 1 1 1 1 1 BoW
$TF(w_i, r_j) = \frac{\#\text{ of times } w_i \text{ occurs in } r_j}{\text{Total \# of words in } r_j}$

The term frequency of every word is in between 0 and 1.



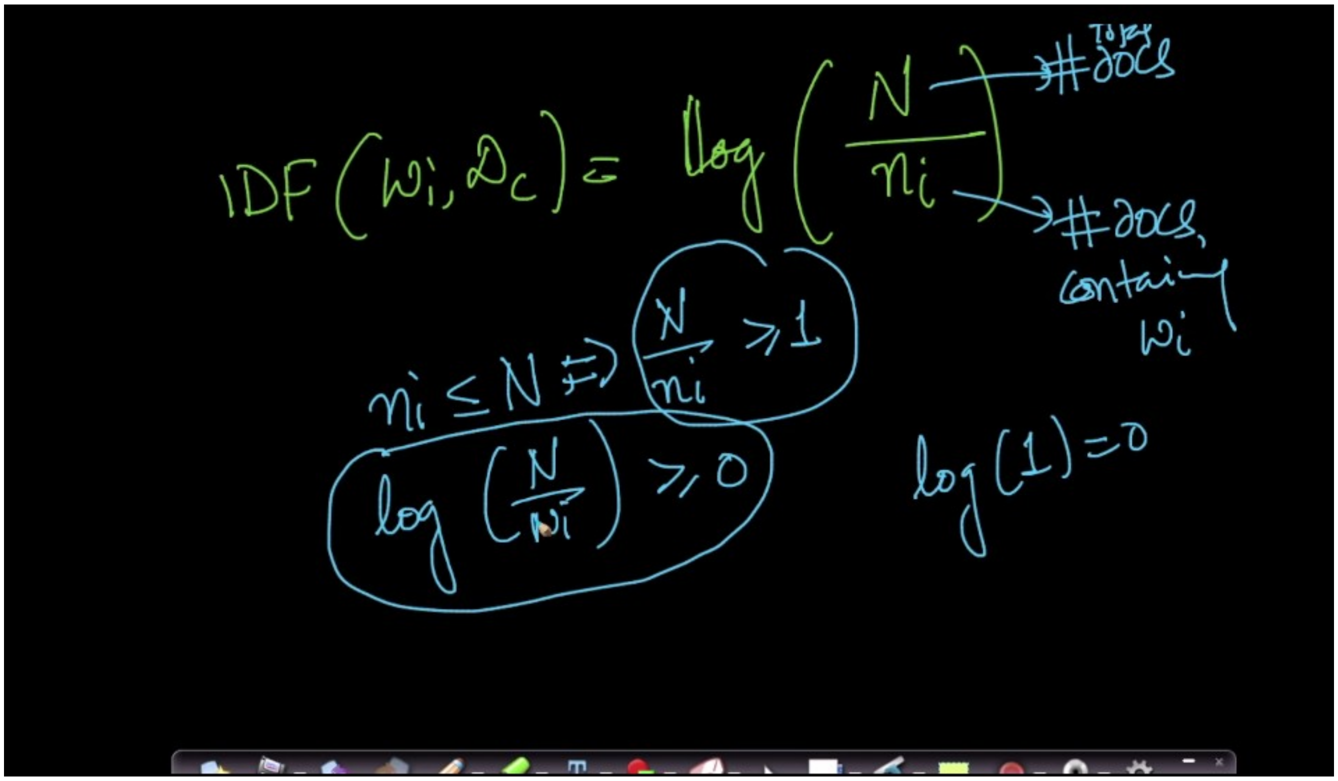
It can be taught of probability of occurring of the word in the review.

IDF(inverse document frequency): The IDF of the word in the document is calculated as the log(#docs / #docs word occurred).



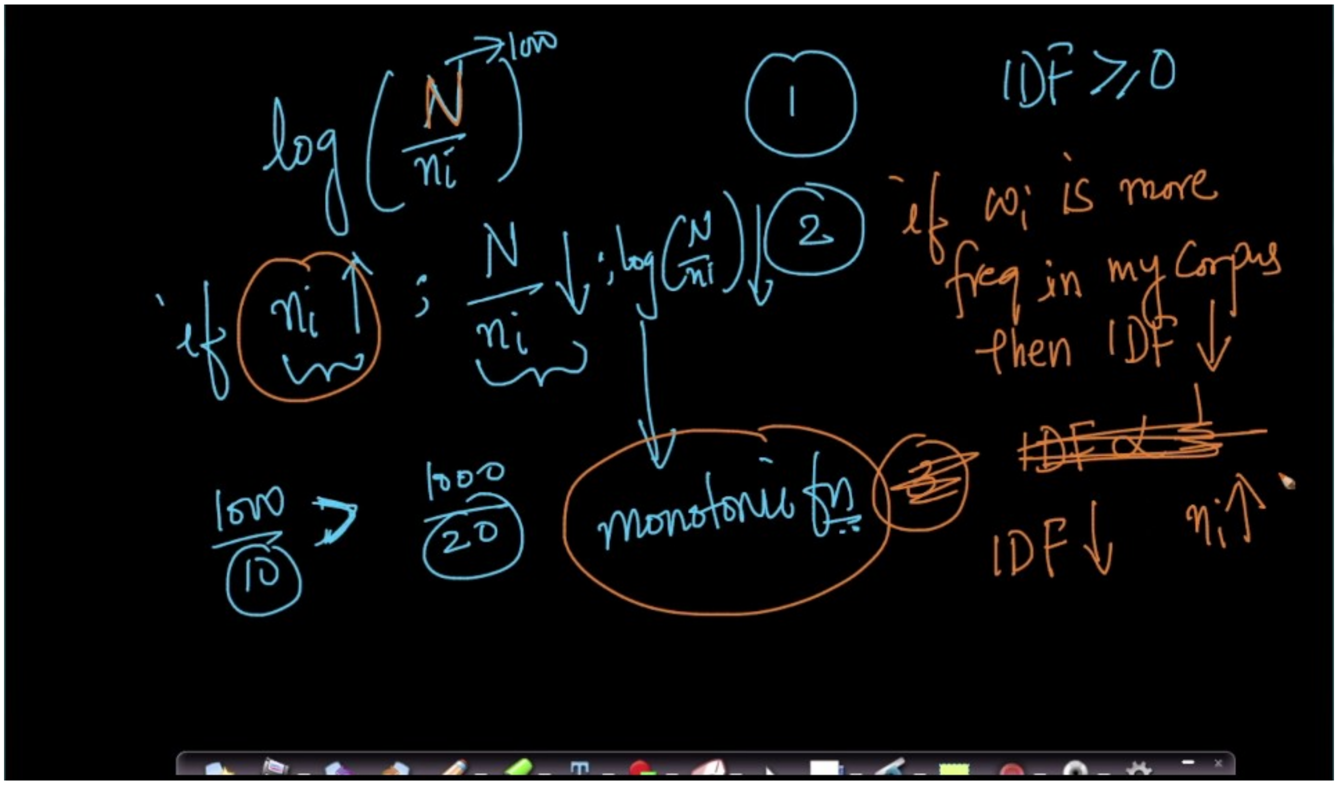
IDF: Inverse document freq:-
$IDF(w_i, D_c)$
$D = \{ r_1: w_1$
$r_2:$
$r_3: — w_1 —$
$r_6 — w_1$
$r_N:$
N docs/reviews
$D_c = \{ r_1, r_2, \dots r_N \}$
$IDF(w_i, D_c) = \log\left(\frac{N}{n_i}\right)$
# docs
# docs which contain w_i

IDF is always greater than equal to 1.



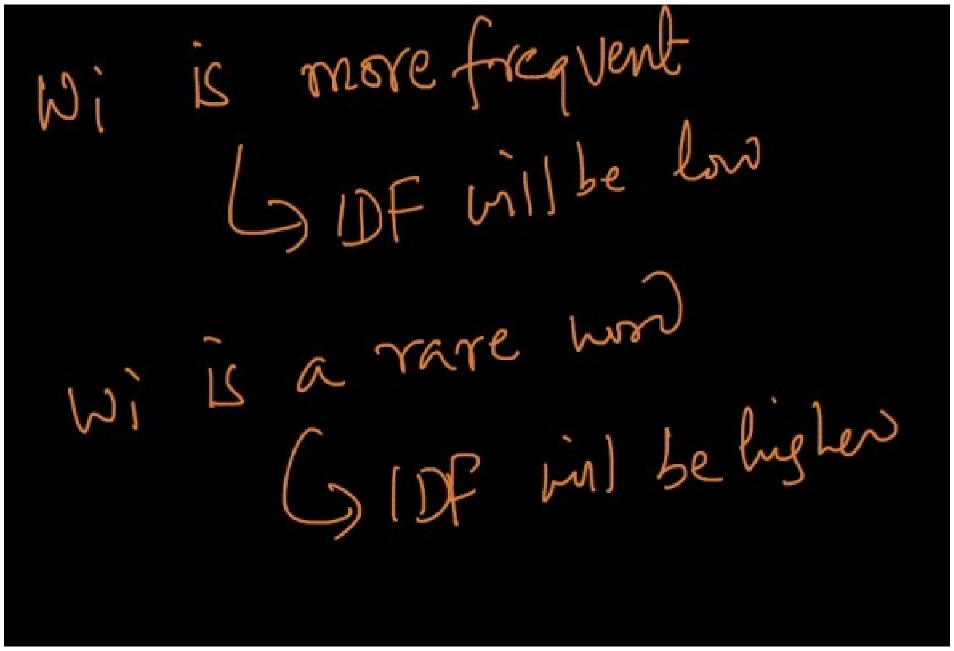
$IDF(w_i, D_c) = \log\left(\frac{N}{n_i}\right)$
$N \rightarrow$ # Total docs
$n_i \rightarrow$ # docs, containing w_i
$n_i \le N \Rightarrow \frac{N}{n_i} \geq 1$
$\log\left(\frac{N}{n_i}\right) \geq 0$
$\log(1) = 0$

Properties of IDF:



$\log\left(\frac{N}{n_i}\right)$, $N \rightarrow 1000$
1
IDF ≥ 0
2
if $n_i \uparrow$; $\frac{N}{n_i} \downarrow$; $\log\left(\frac{N}{n_i}\right) \downarrow$
if w_i is more freq in my corpus then IDF $\downarrow$
$\frac{1000}{10} > \frac{1000}{20}$
monotonic fn
3
IDF $\downarrow$ $n_i \uparrow$

Since, log is a monotonically increasing function.



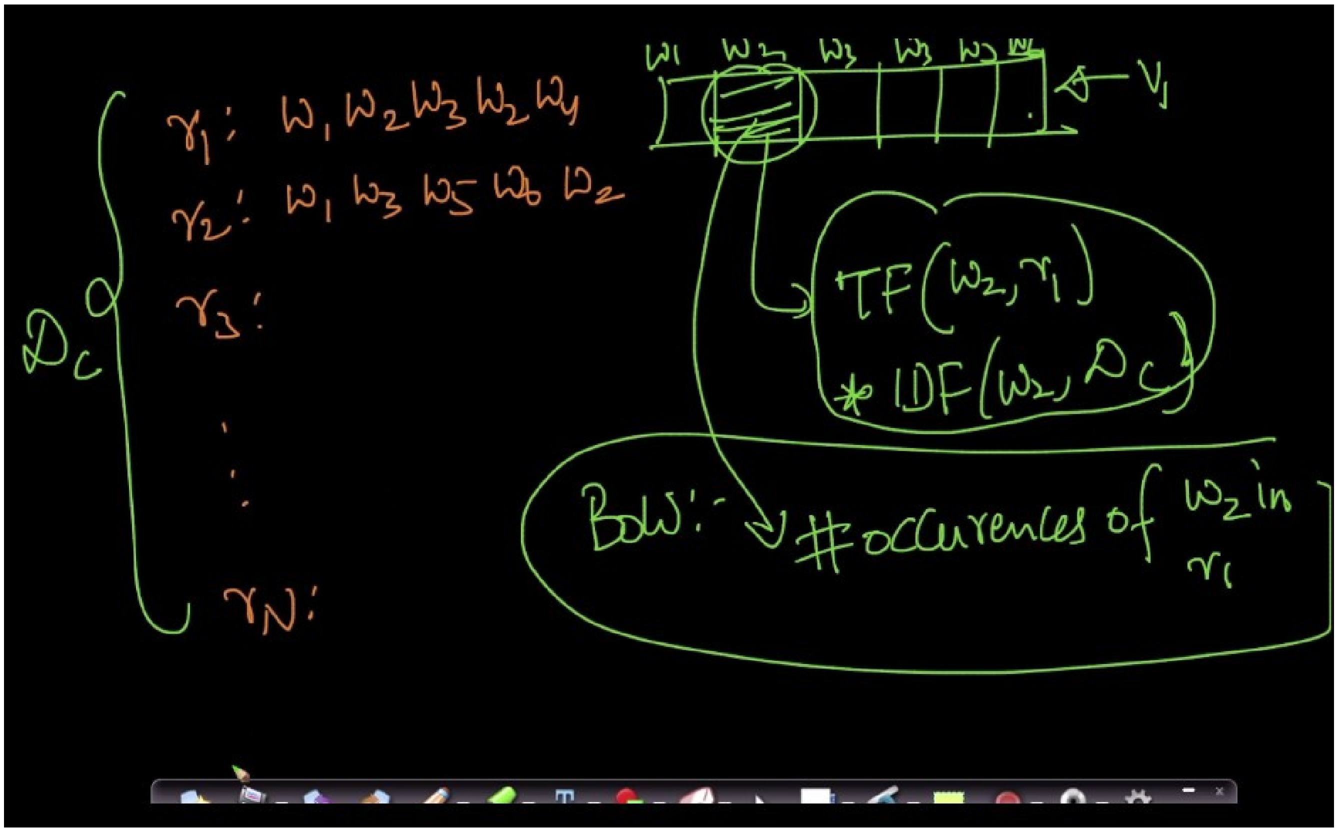
w_i is more frequent

↳ IDF will be low

w_i is a rare word

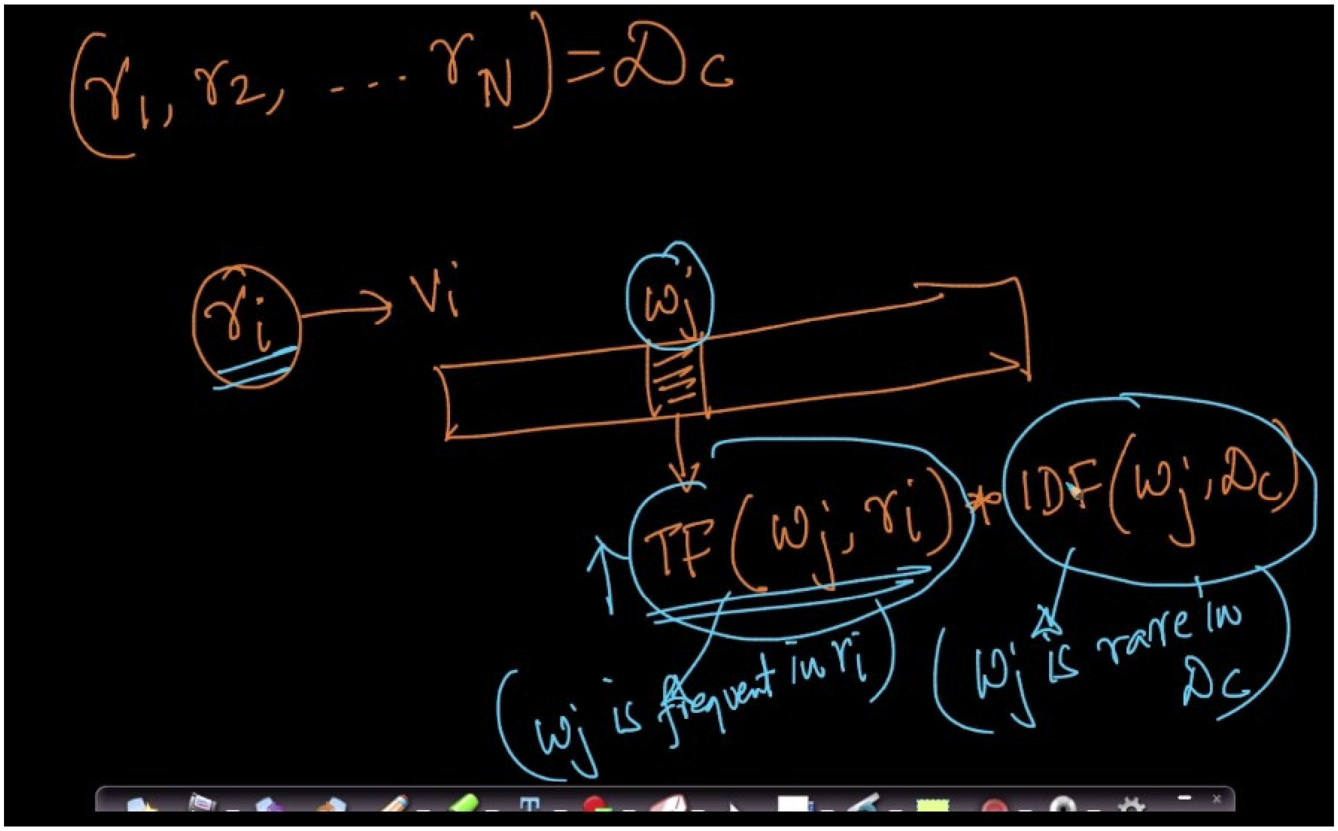
↳ IDF will be higher

Calculation of TF-IDF:

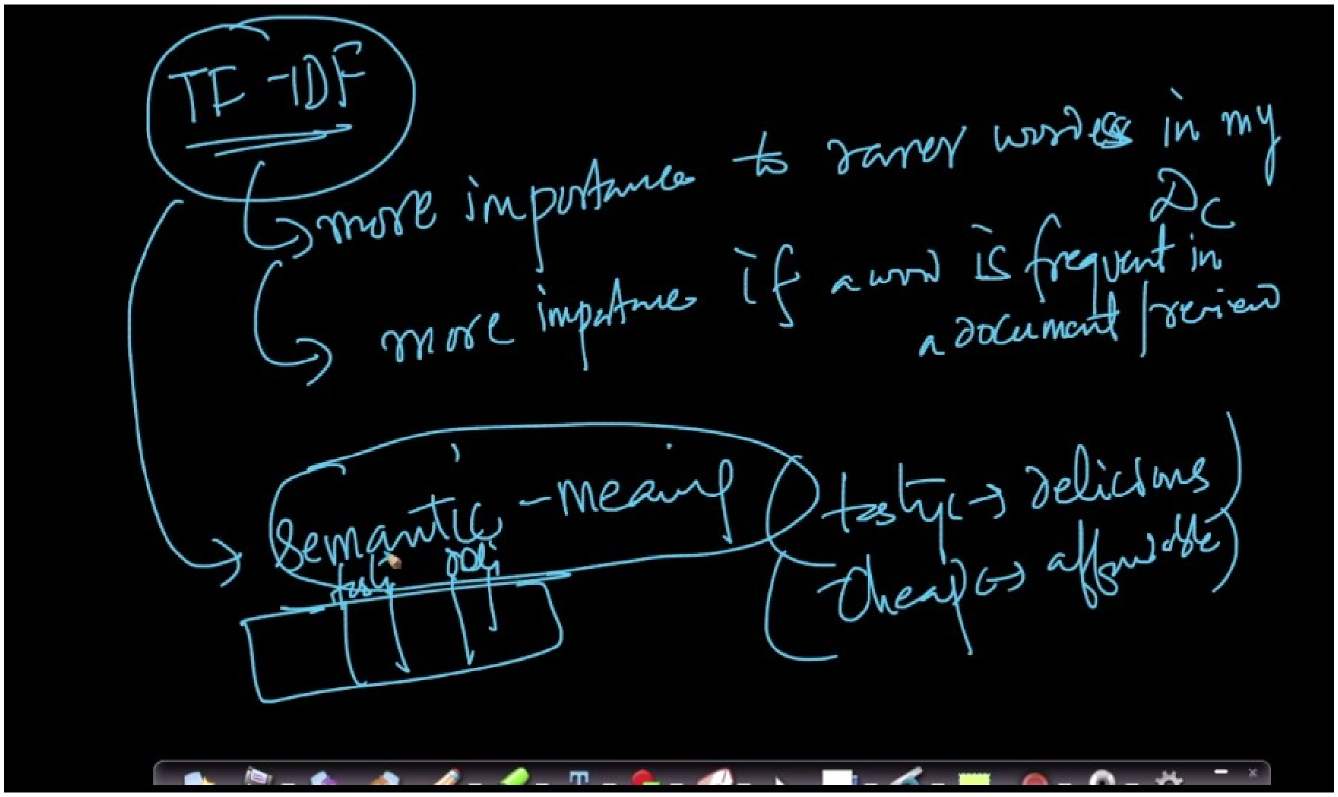


r_1: $w_1 w_2 w_3 w_2 w_4$
r_2: $w_1 w_3 w_5 w_6 w_2$
r_3:
r_N:
D_c
w_1 w_2 w_3 w_3 w_3 w_4
v_1
$TF(w_2, r_1)$
$* IDF(w_2, D_c)$
BoW:-
# occurences of w_2 in r_1

TF says how often the word occurs in the document / review.

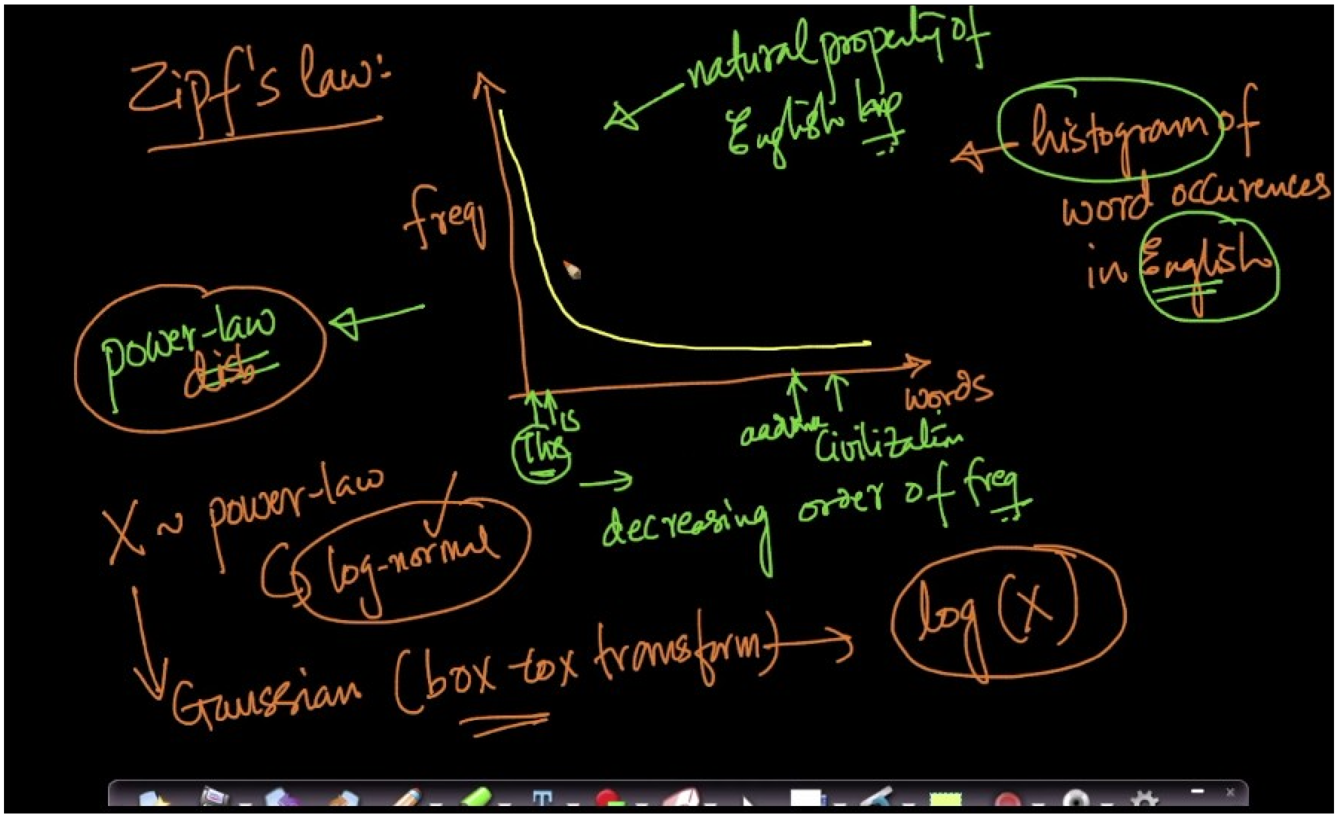

$(r_1, r_2, \dots r_N) = D_c$
$r_i \to v_i$
w_j
$TF(w_j, r_i) * IDF(w_j, D_c)$
(w_j is frequent in r_i)
(w_j is rare in D_c)

TF – IDF:

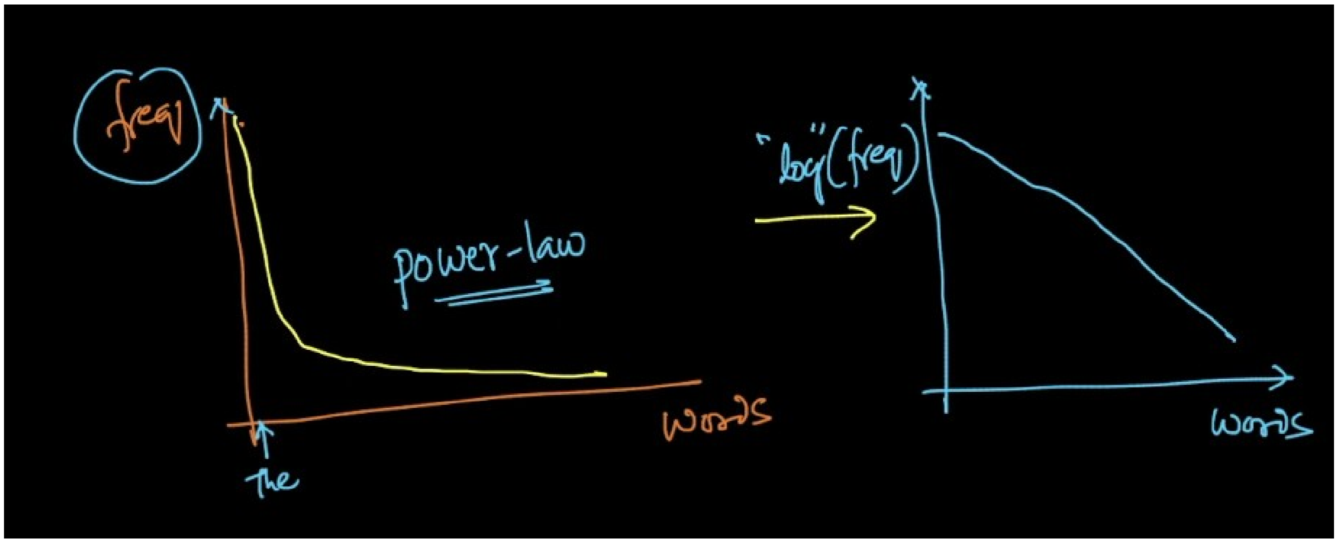

TF-IDF
more importance to rarer words in my D_c
more importance if a word is frequent in a document / review
semantic-meaning
tasty
tasty ↔ delicious
cheap ↔ affordable

Why use log in IDF:

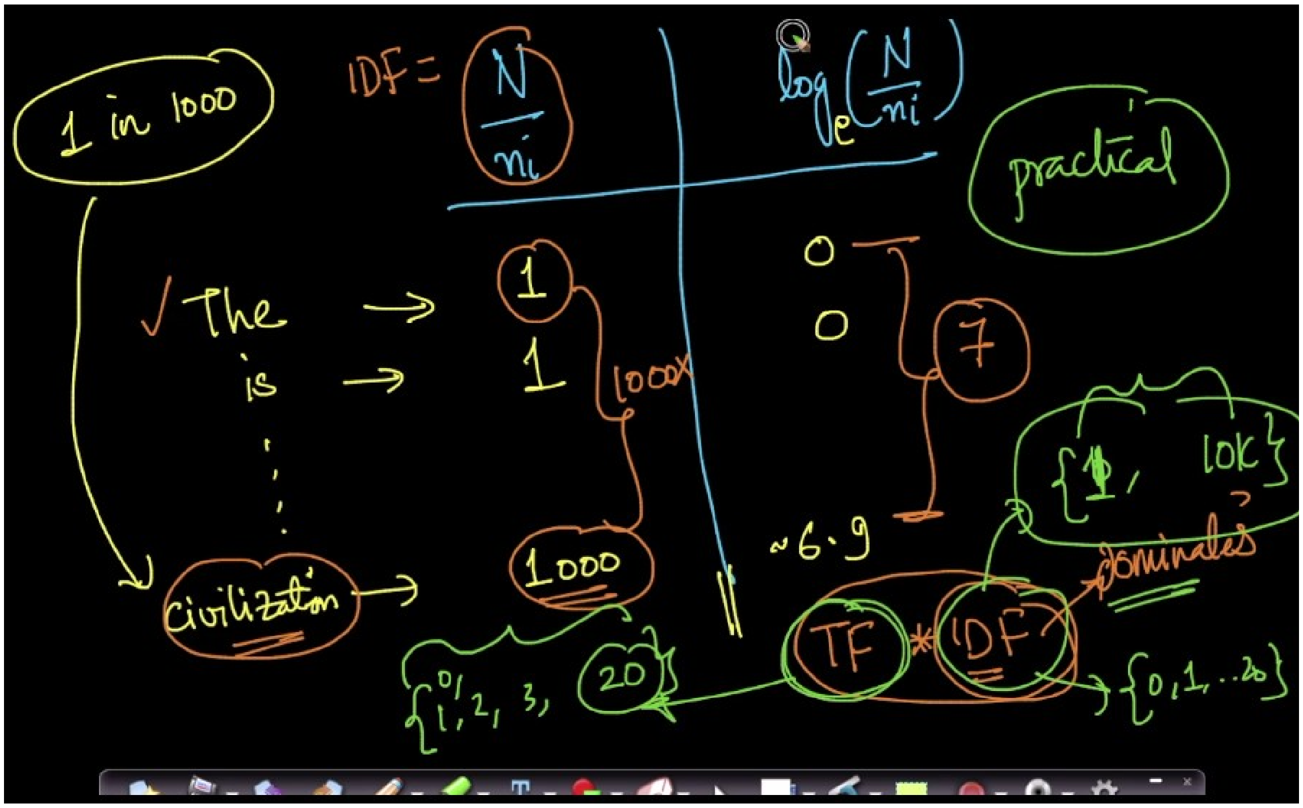
Log is given on the name of Zipf's Law:


Zipf's law:
natural property of English lang
histogram of word occurences in English
freq
power-law ~~dist~~
The
is
aardvark
Civilization
words
→ decreasing order of freq
X ~ power-law
log-normal
Gaussian (box cox transform) → log(X)

Zipf's law says that the words in english follow power law distribution.



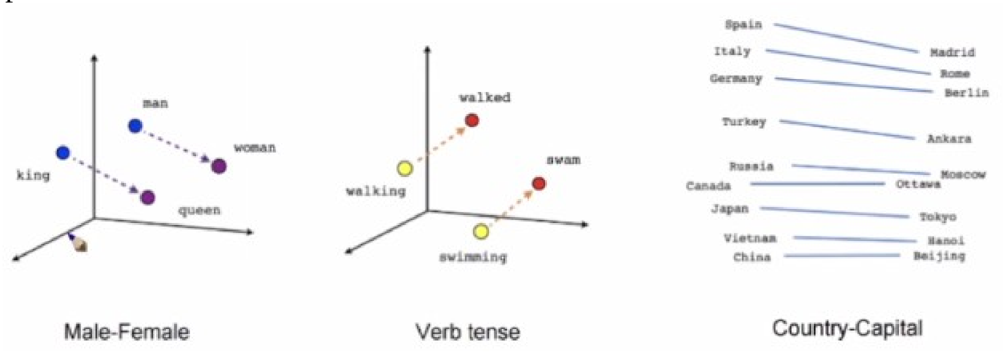
By applying log the range of values differing will decrease.



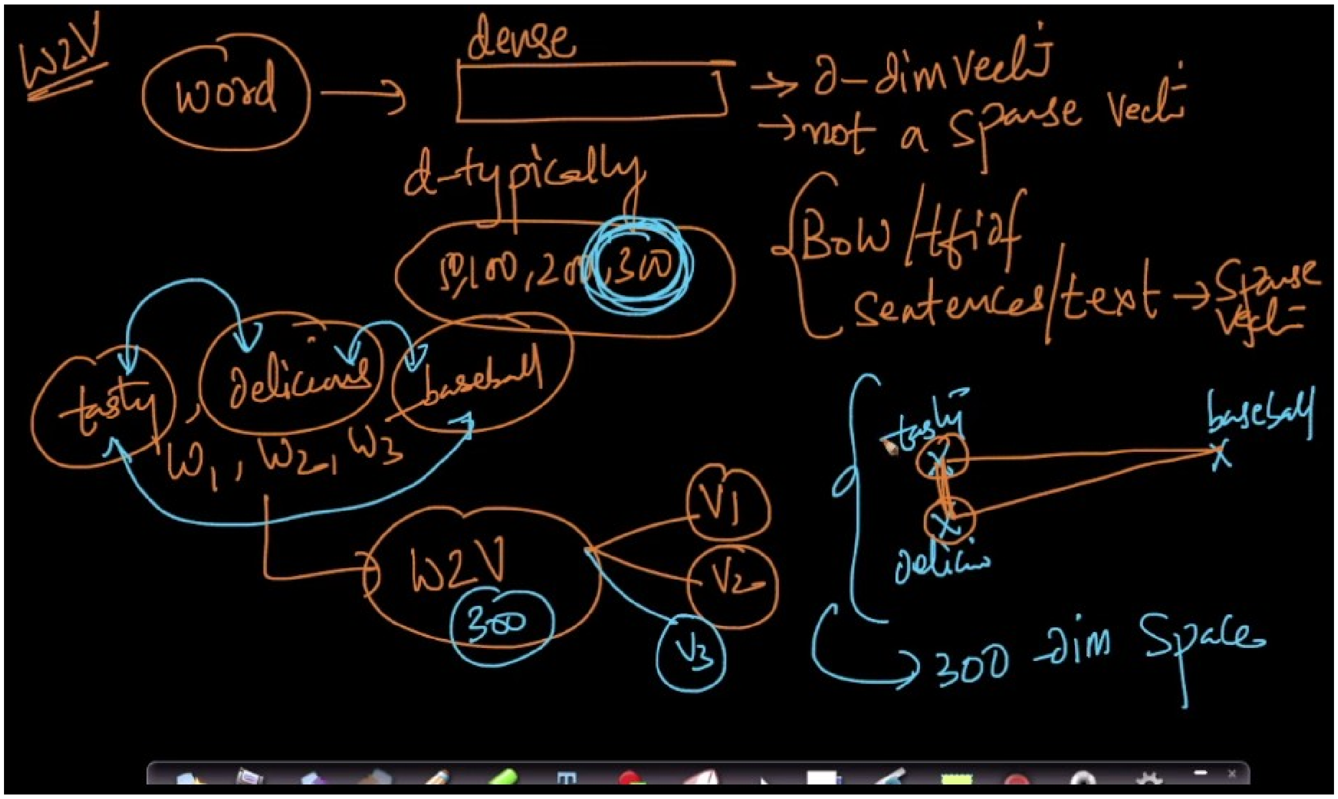
Word2Vec:

It takes the semantic meaning of the words and relationships of the data.

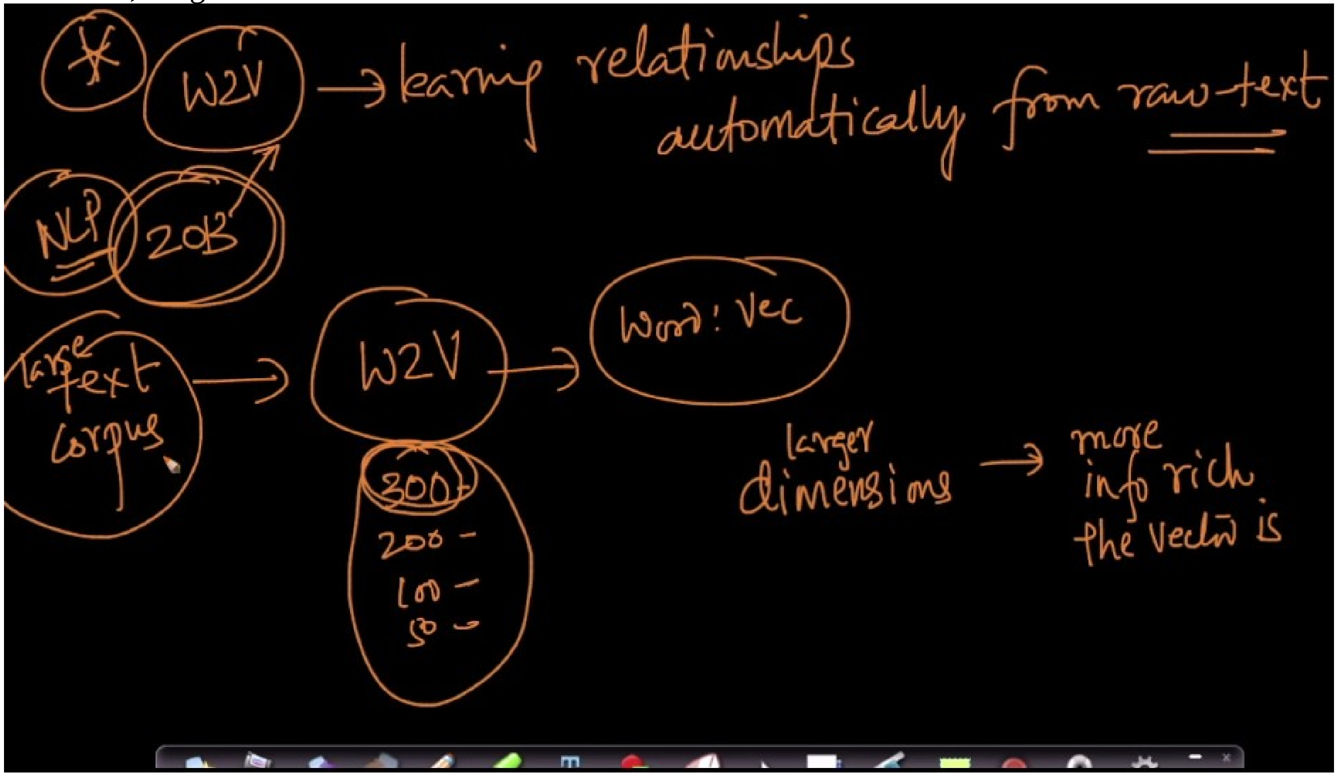
Word2Vec representation:

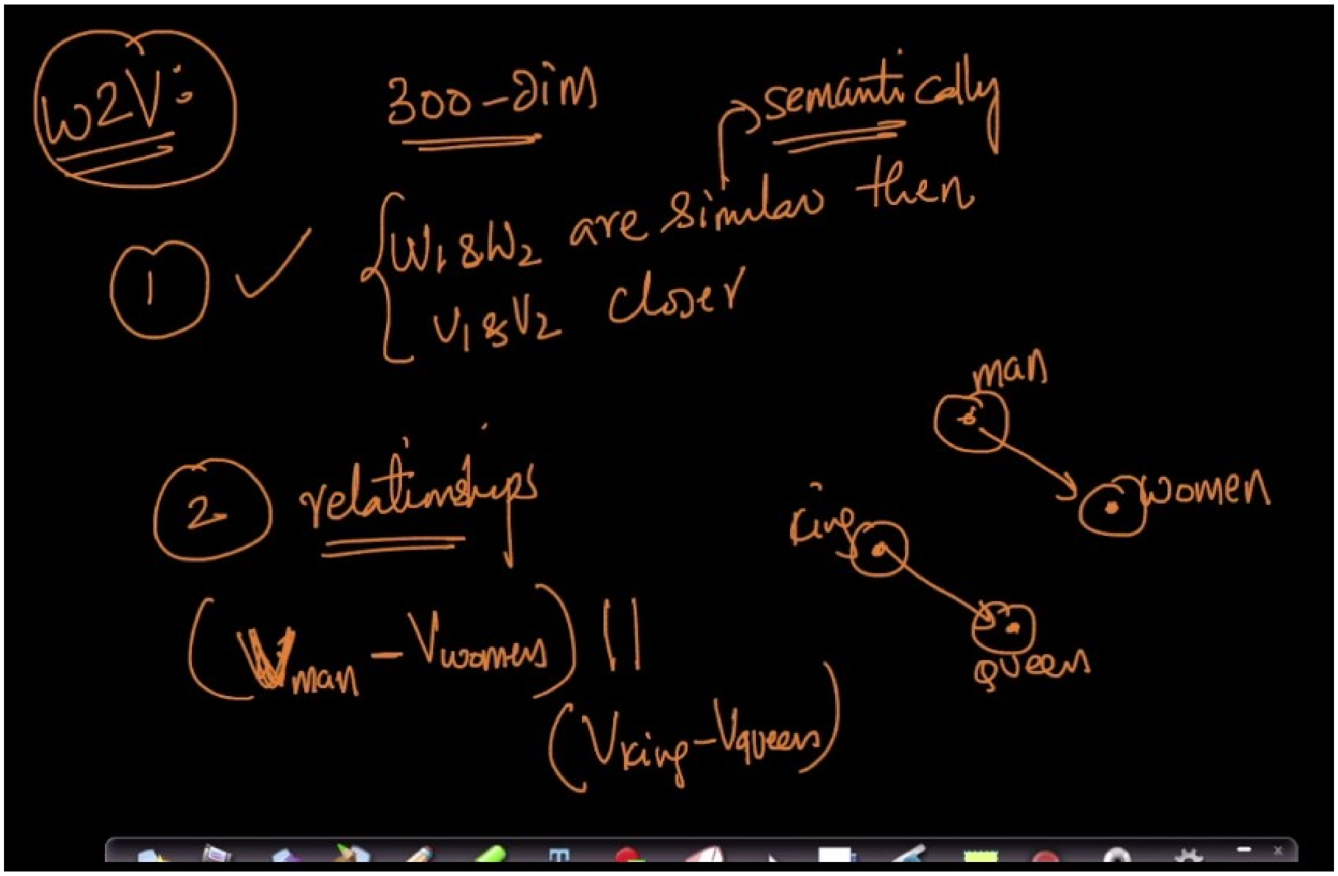


W2V represents the word in the form of dense values:



W2V: It also understands the relationships of the words. It learns all the internal relationship between the words, Larger the dimensions of the vector the more rich is the vector.


W2V → learning relationships automatically from raw-text
NLP
2013
large text corpus
W2V
word : vec
300
200
100
50
larger dimensions → more info rich the vector is



W2V:

300-dim

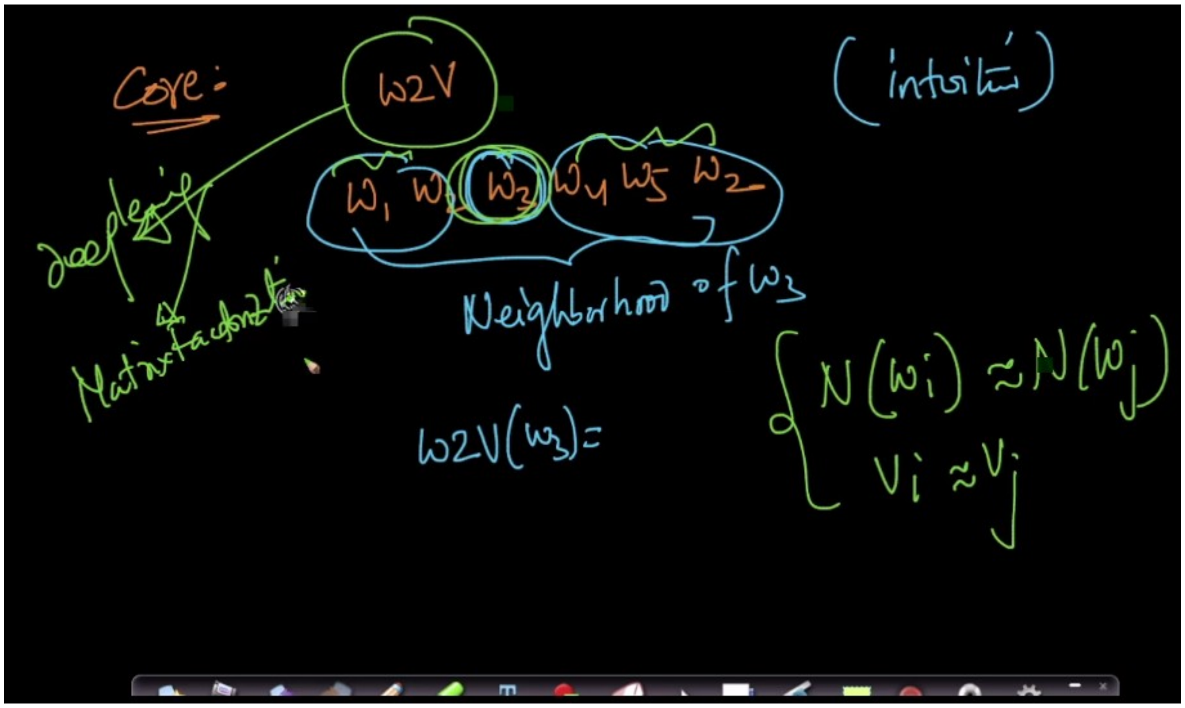
semantically

1 ✓ $\begin{cases} W_1 \,\&\, W_2 \text{ are similar then} \\ V_1 \,\&\, V_2 \text{ closer} \end{cases}$

2 relationships

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

If the data corpus size is large, larger the dimensions of the vector.
Google News W2V has the 300 dimensional vector.
W2V also looks at the neighbors of the word in the sentence.



Core:

W2V

(intuition)

deeplearning

Matrix factorization

$W_1 \; W_2 \; W_3 \; W_4 \; W_5 \; W_2$

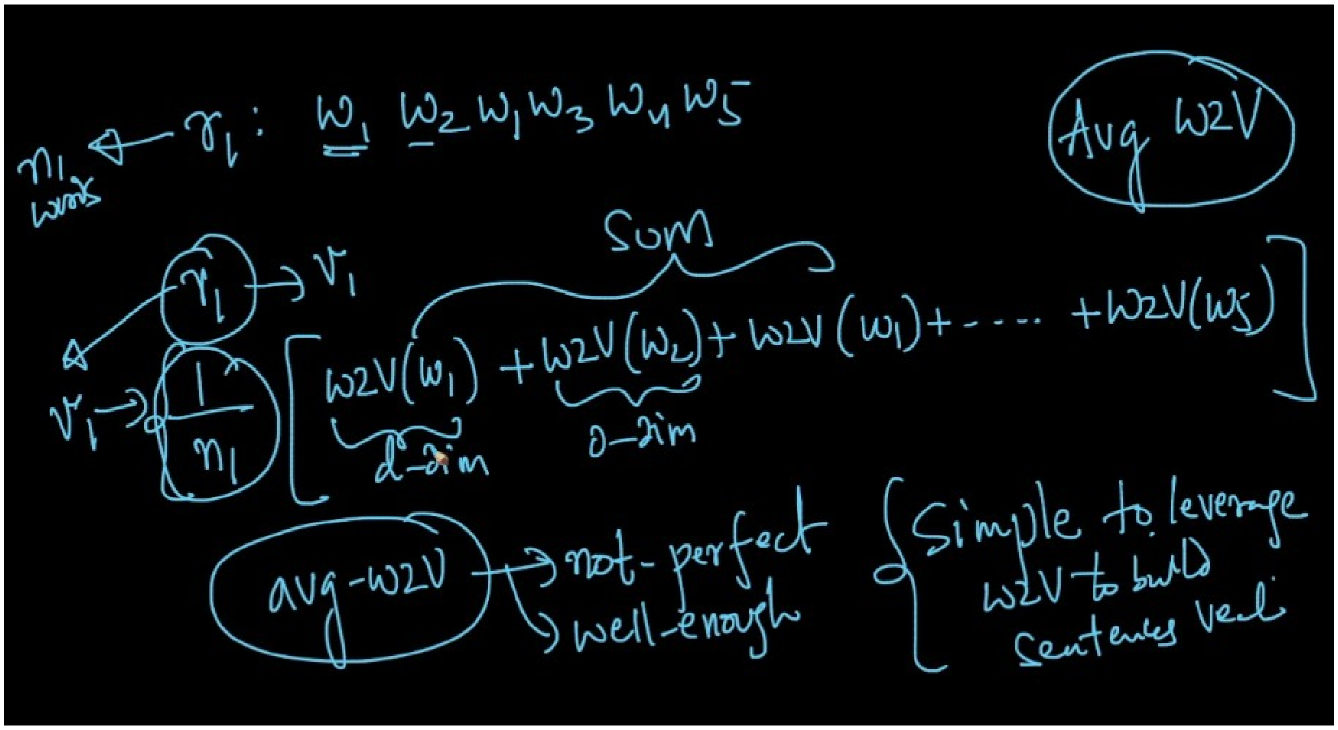
Neighborhood of W_3

$W2V(w_3) =$

$\begin{cases} N(w_i) \approx N(w_j) \\ V_i \approx V_j \end{cases}$

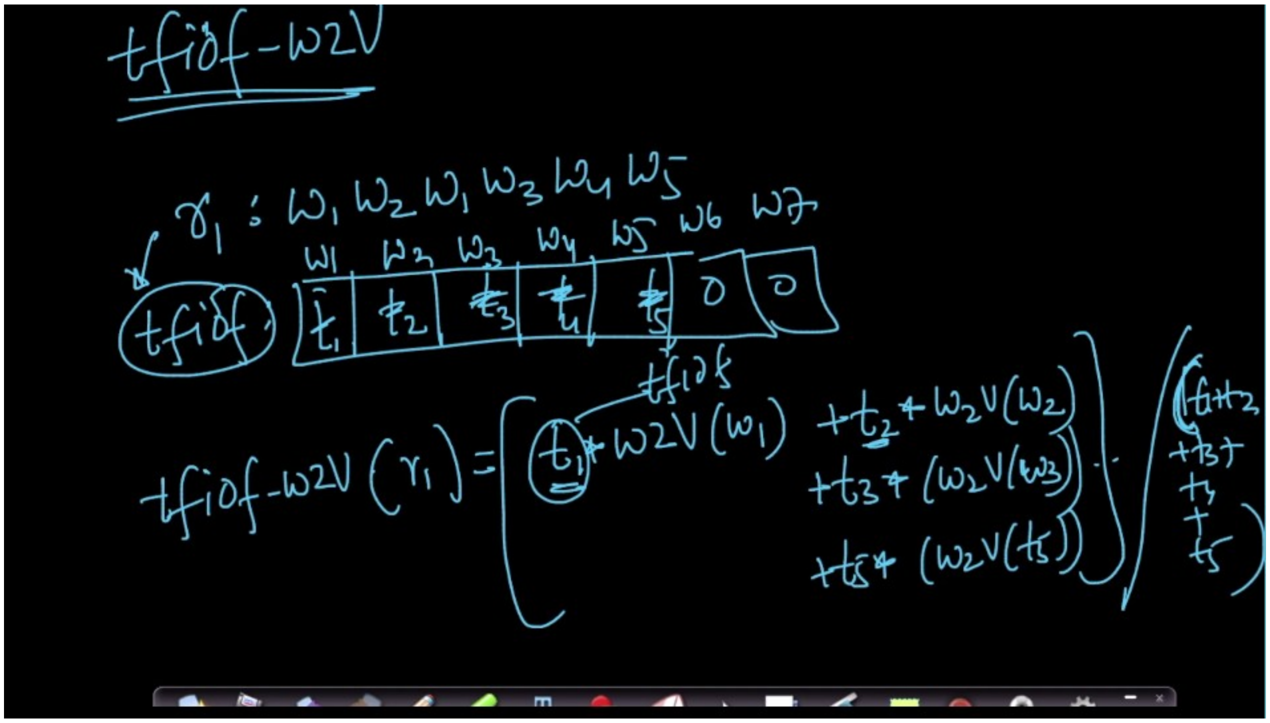
Average – W2V, tf-idf weighted W2V:

Compute the w2v of each of the words and add the vectors of each words of the sentence and divide the vector with the number of words of the sentence.



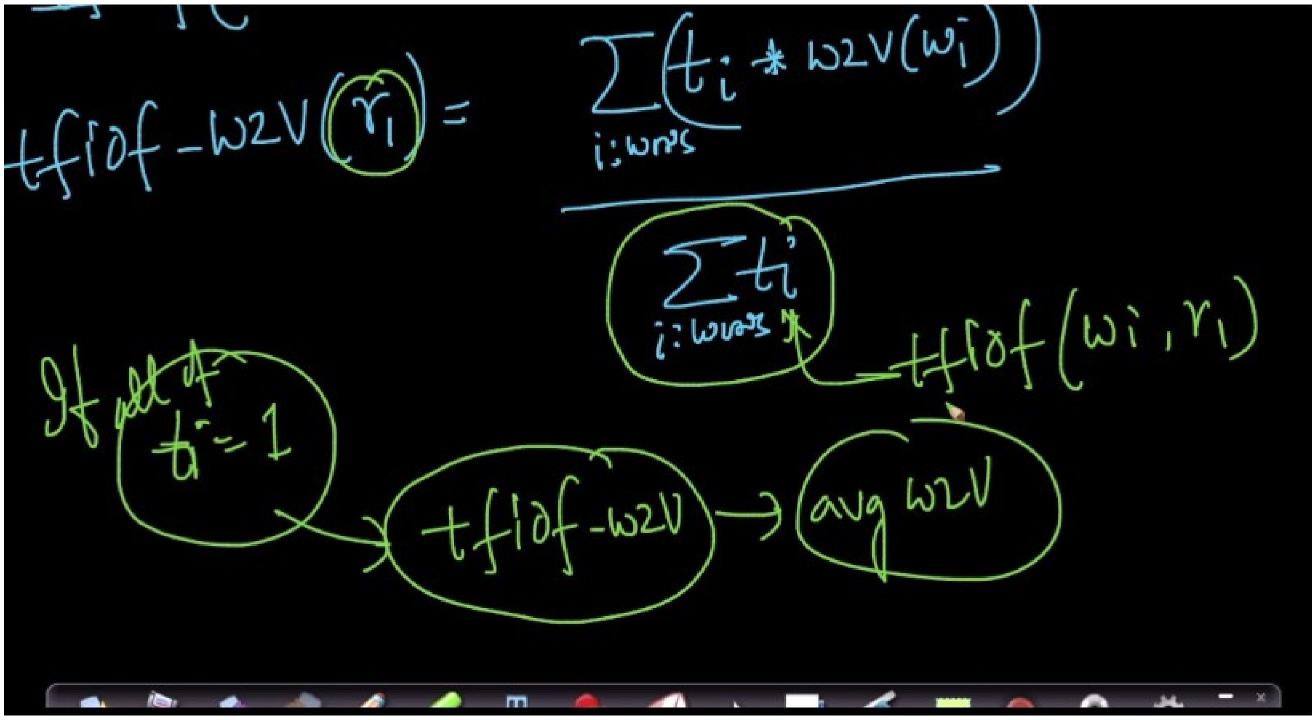
n_1 words ← r_1 : w_1 w_2 w_1 w_3 w_4 w_5
Avg W2V
sum
$r_1 \rightarrow v_1$
$v_1 \rightarrow \frac{1}{n_1}$ [W2V(w_1) + W2V(w_2) + W2V(w_1) + ... + W2V(w_5)]
d-dim
d-dim
avg-W2V → not-perfect
→ well-enough
Simple to leverage W2V to build sentence vec.

TfIdf – w2v:
The W2V value of each of the word is multiplied by the tfidf value and summed up and then divided by the sum of the tfidf values of the sentence.



tfidf-W2V
r_1 : w_1 w_2 w_1 w_3 w_4 w_5
tfidf
w_1 w_2 w_3 w_4 w_5 w_6 w_7
t_1 t_2 t_3 t_4 t_5 0 0
tfidf
tfidf-W2V(r_1) = [t_1 * W2V(w_1) + t_2 * W2V(w_2) + t_3 * (W2V(w_3)) + t_5 * (W2V(t_5))] / (t_1 + t_2 + t_3 + t_5 + t_5)

If all the tfidf values of the words in the review are equal to 1, then it is equal to avg-w2v.


$\text{tfidf-w2v}(r_1) = \dfrac{\sum_{i:\text{words}} (t_i * \text{w2v}(w_i))}{\sum_{i:\text{words}} t_i}$
$t_i = \text{tfidf}(w_i, r_1)$
If all of $t_i = 1$
tfidf-w2v
avg w2v