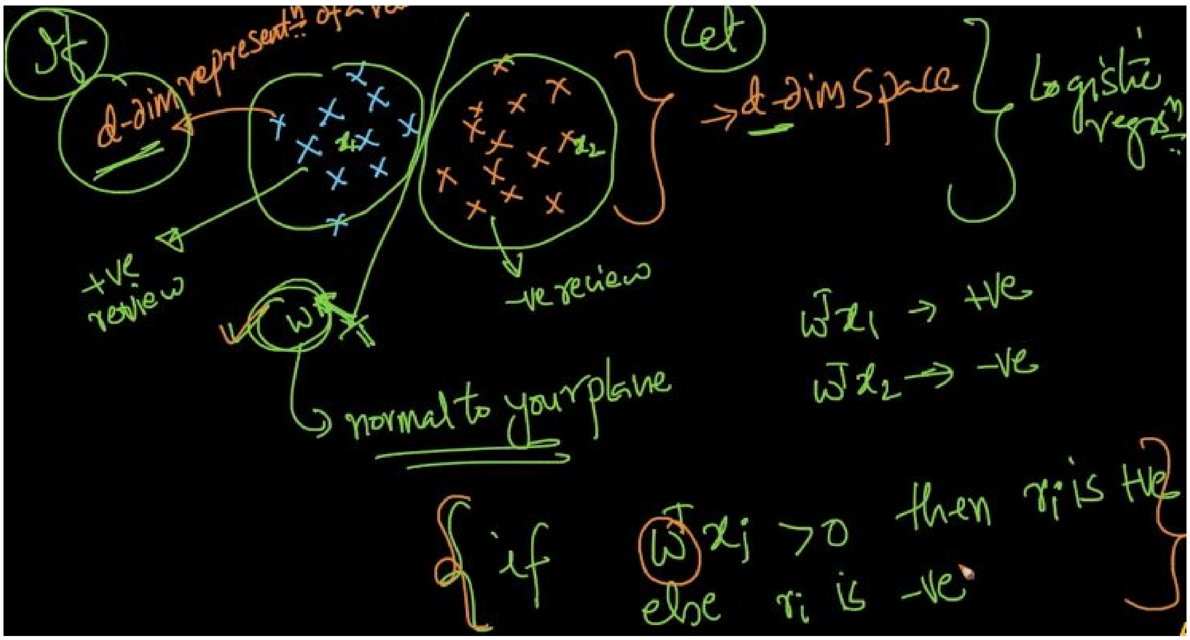
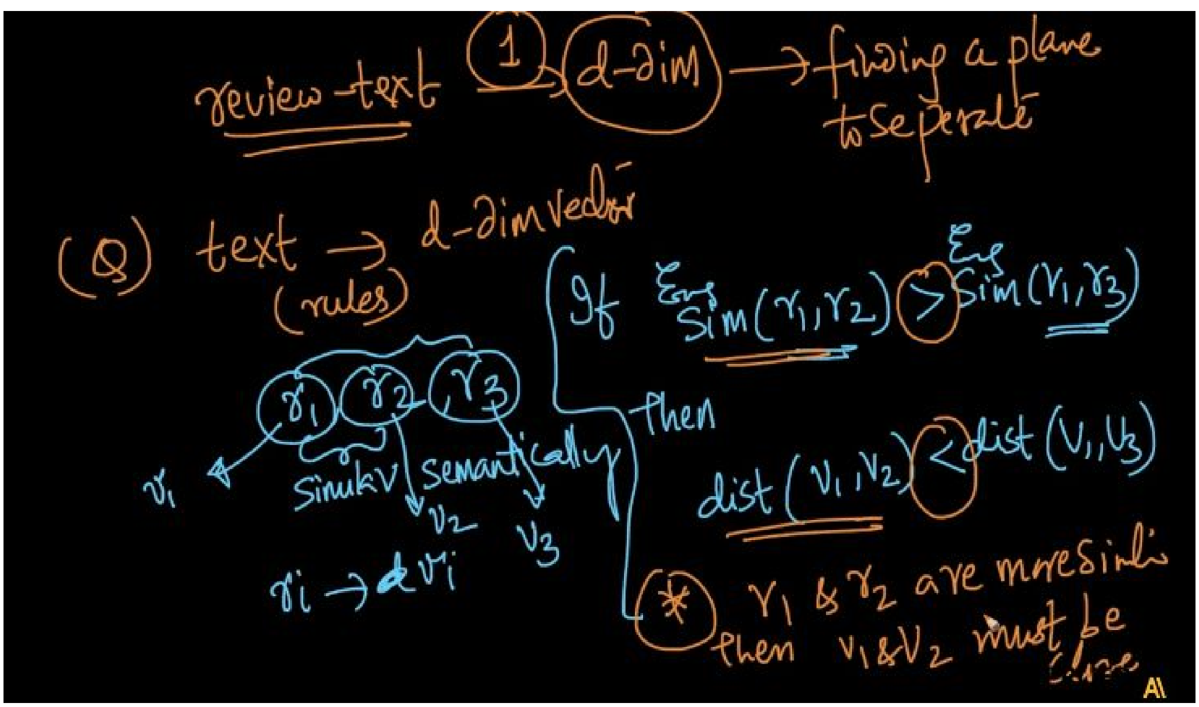
Converting Text (Words and Sentences) into numerical Vector ?

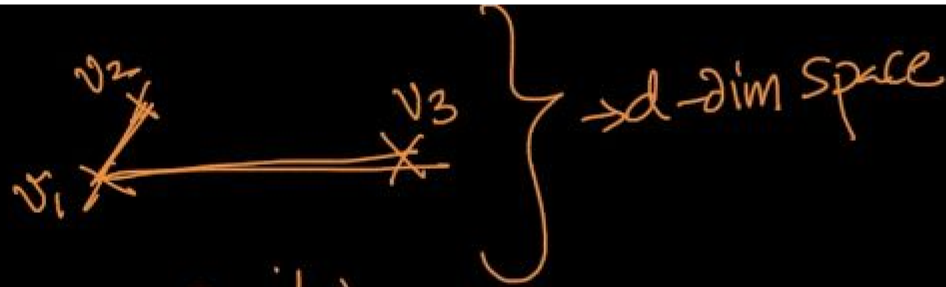
If we can convert any text to a vector, we can leverage the power of linear algebra.

For our case study of Amazon fine food reviews,

If we convert Review text into d dimensional vector







Similar

$$\text{EngSim}(v_1, v_2) > \text{EngSim}(v_1, v_3)$$

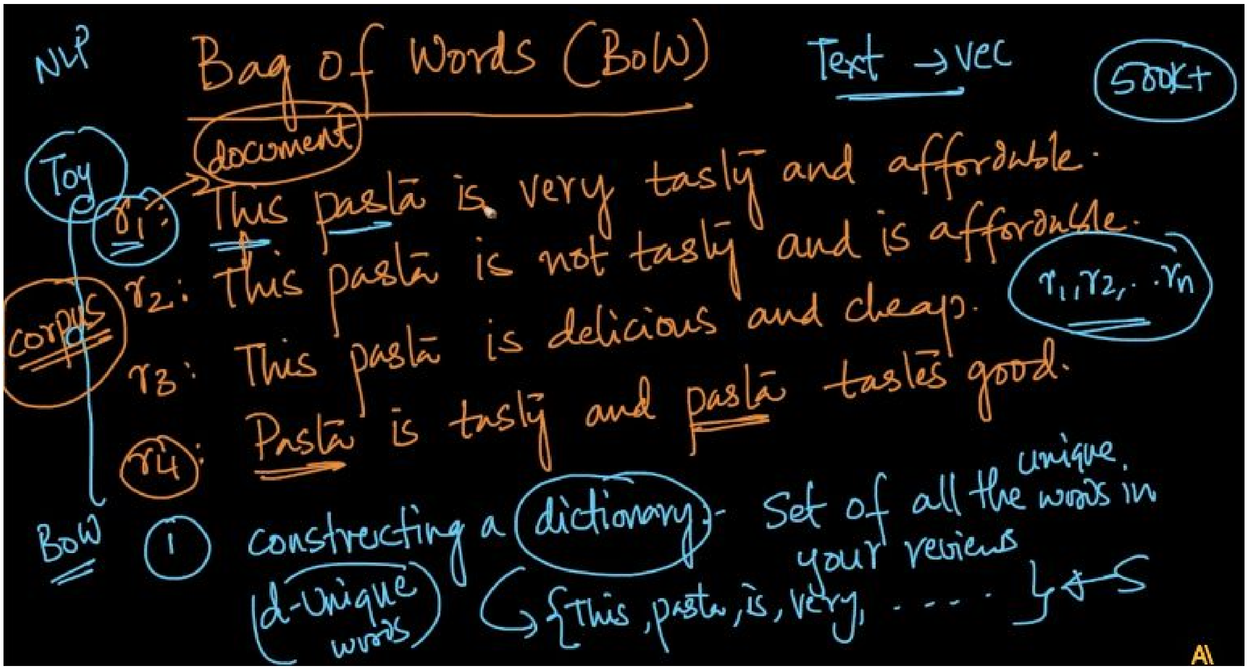
$$\Downarrow$$

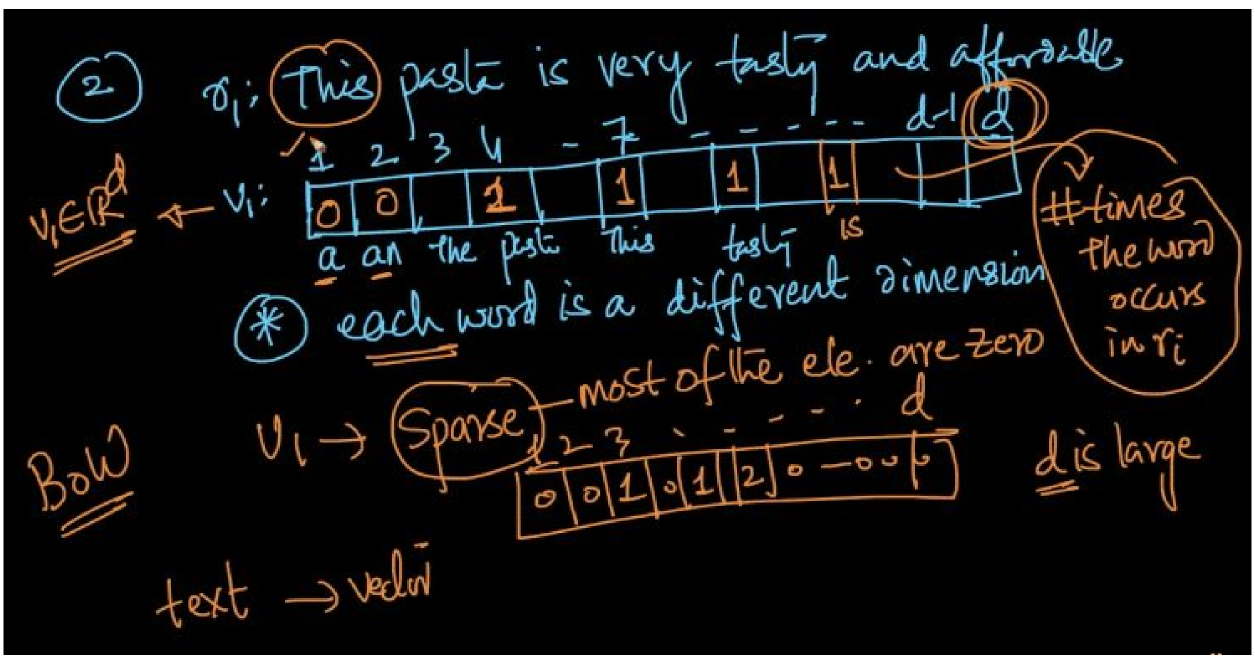
$$\underbrace{\text{length}(v_1 - v_2) < \text{length}(v_1 - v_3)}_{\text{closer}}$$

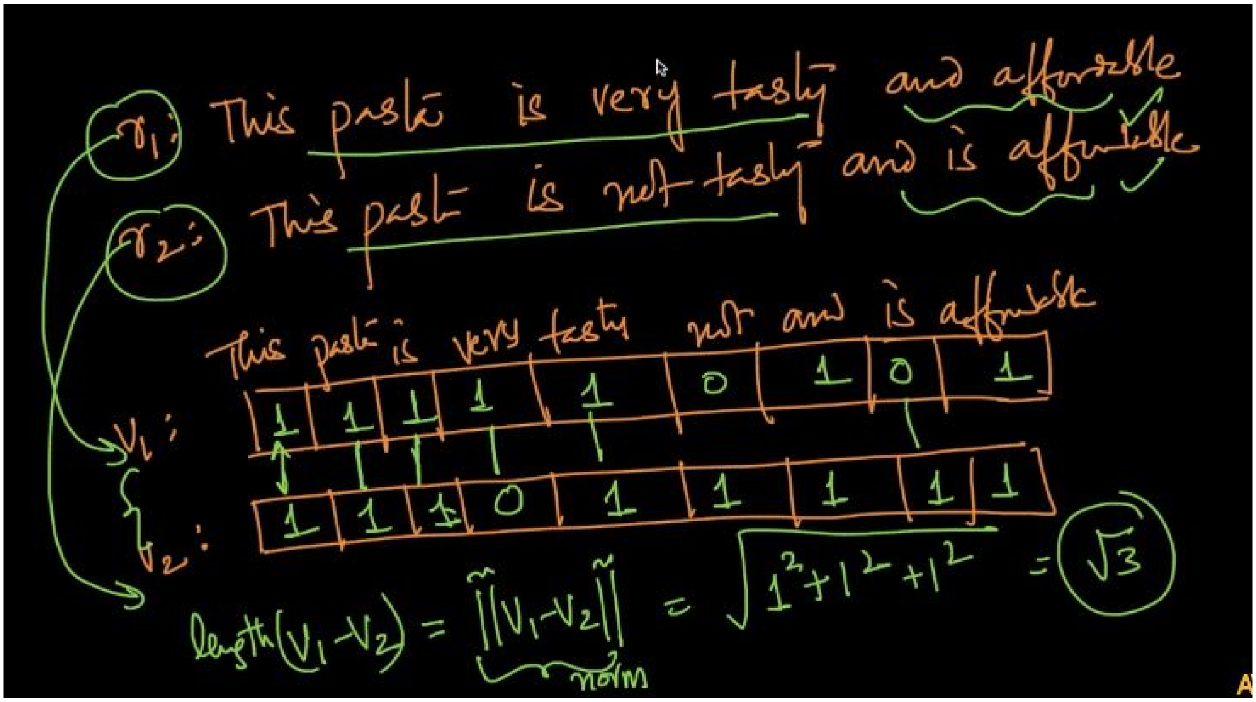
find {text → d-dim vector}
s.t. similar text must be closer geometrically

BoW
tf-idf
w2v
avg w2v
tfidf w2v

Bag of Words (BoW)





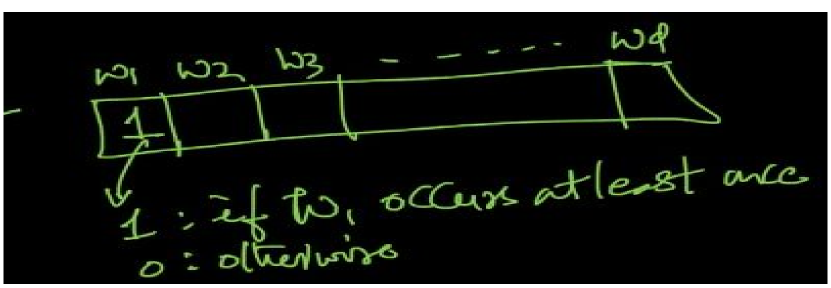


Bag of words can be thought of counting the common words when all the values exist once.

BoW doesn't work well when there are small changes. BOW depends on the count of words in each document corpus.This discards the semantic meaning of the documents such that documents that are completely opposite in meaning can lie closer to each other like above example.

Binary BoW or boolean BoW

It doesn't reflect the no of occurrence or count but summarizes if the words occurs or not.



||v1-v2|| = sqrt(no of differing words) or sqrt(unique words between 2 documents)

Improving BoW (Text Preprocessing)

D1: **This** pasta **is** very tasty **and** affordable.
D2: **This** pasta **is** not tasty **and** **is** affordable.
D3: **This** pasta **is** delicious **and** cheap.
D4: Pasta **is** tasty **and** pasta tastes good;

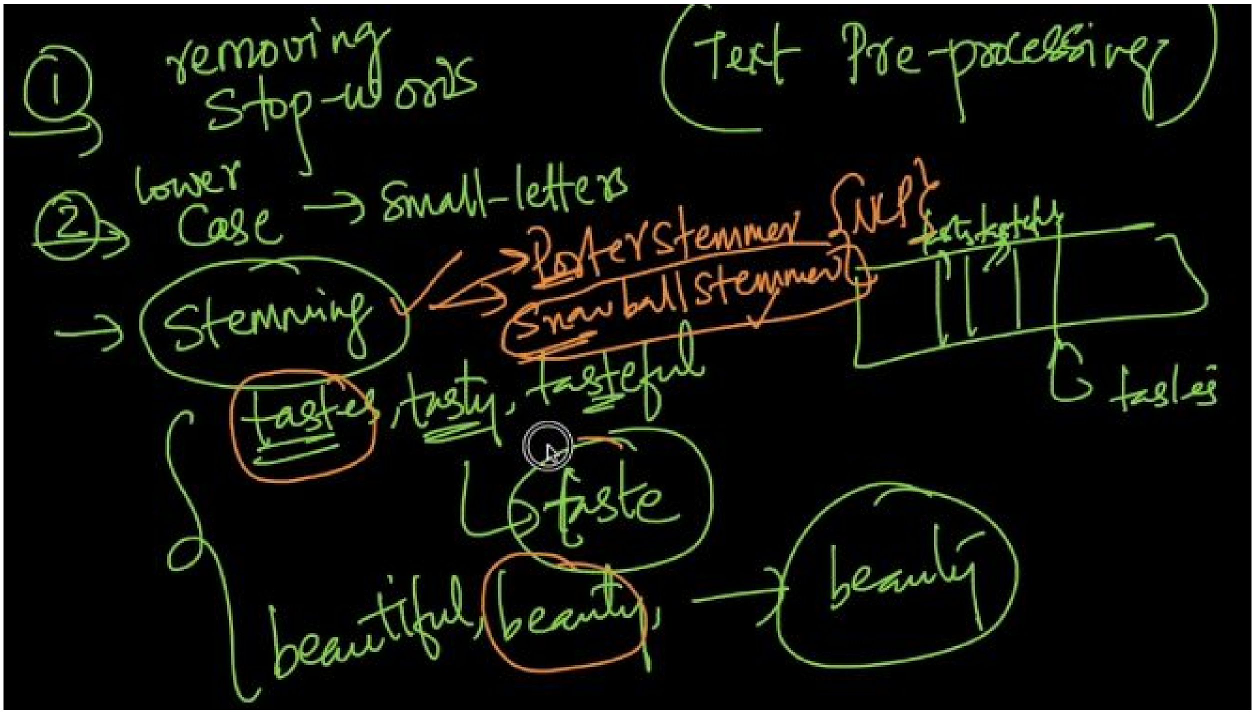
The **Bold** *and* **Underlined** *words are* ***stopwords***.

Stopwords *are words which are filtered out before or after processing of natural language data (text).Though "stop words" usually refers to the most common words in a language, there is no single universal list of stop words used by all natural language processing tools, and indeed not all tools even use such a list.*

They also depend on domain knowledge but are usually meaningless and have no value to the semantic meaning of the document.

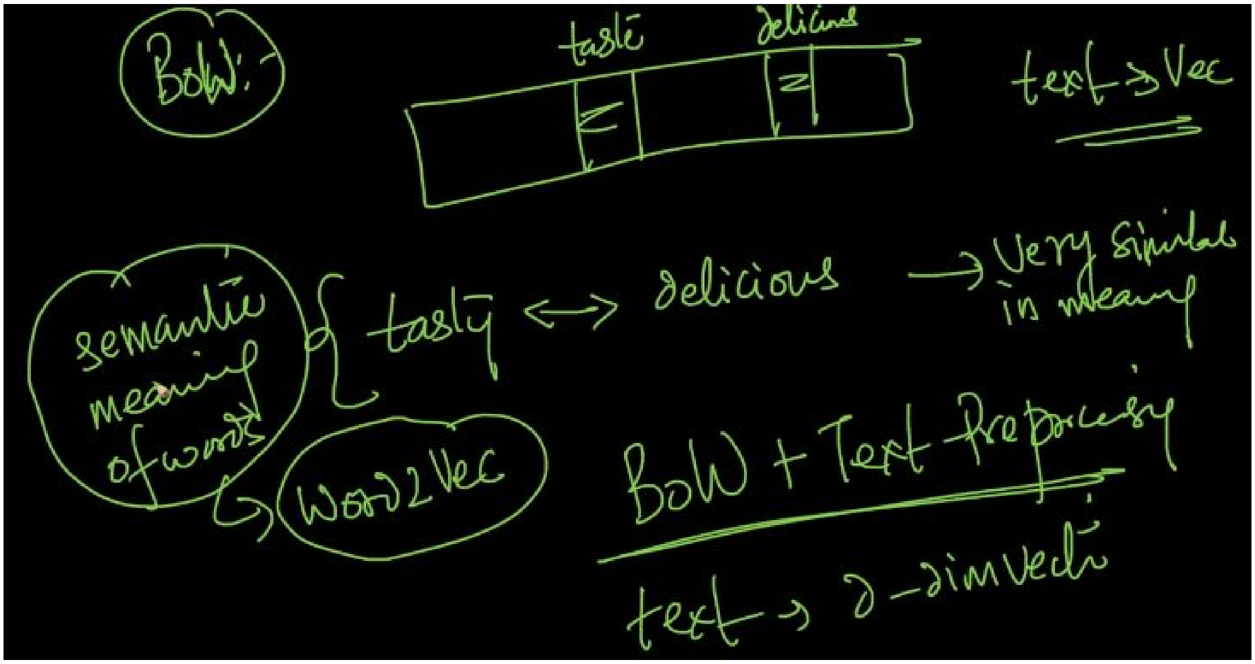
```
{'their', 'isn', 'such', 'where', 'this', 'they', 'while', 'about', 'ther
e', 'myself', 'from', 'mightn', 'was', 'between', 'who', 'are', 'only',
 'our', 'those', 'through', 'any', 'is', 'a', 'nor', 'mustn', 'shouldn',
 'yourself', 'no', 'itself', 'that', 'himself', 'out', 'what', 'my', 'aga
inst', 'below', 's', 'for', 'be', 'into', 'few', 'needn', 'you', 'aren',
 'when', 'all', 'him', 'but', 've', 'yours', 'being', 'why', 'own', 'up',
'whom', 're', 'and', 'she', 'me', 'of', 'than', 'doesn', 'both', 'same',
 'too', 'am', 'how', 'not', 'her', 'd', 'until', 'o', 'your', 'yourselve
s', 'by', 'other', 'once', 'an', 'just', 'to', 'these', 'don', 'its', 'ha
ven', 'having', 'some', 'shan', 'theirs', 'under', 'we', 'ain', 'it', 'a
t', 'in', 'y', 'the', 'off', 'herself', 'down', 'because', 'i', 'now', 't
hemselves', 'each', 'or', 'were', 'if', 'can', 'did', 'm', 'which', 'coul
dn', 'ourselves', 'hadn', 'has', 'wasn', 'with', 'here', 'further', 'the
m', 'hasn', 'should', 'ma', 'then', 'he', 'very', 'above', 'been', 'did
n', 'during', 'most', 'hers', 'will', 'have', 'doing', 'again', 'had', 'd
o', 'before', 'as', 'wouldn', 'his', 'after', 'ours', 'does', 'so', 'on',
'more', 't', 'won', 'weren', 'over', 'll'}
```

Removing the stop word is not always the best choice. For example :
D2: **This** pasta **is** not tasty **and** **is** affordable.
If we remove not from D2, it completely changes the meaning of the document.
But, for most of context, if we remove stop words we could have a smaller and meaningful.



Lemmatization 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.

Tokenization : Breaking a sentence into words. Language and content dependent.



BoW doesn’t take semantic meaning in consideration so we use Word2vec.

Stemming just removes or stems the last few characters of a word, often leading to incorrect meanings and spelling. Lemmatization considers the context and converts the word to its meaningful base form, which is called Lemma. Sometimes, the same word can have multiple different Lemmas. We should identify the Part of Speech (POS) tag for the word in that specific context. Here are the examples to illustrate all the differences and use cases:

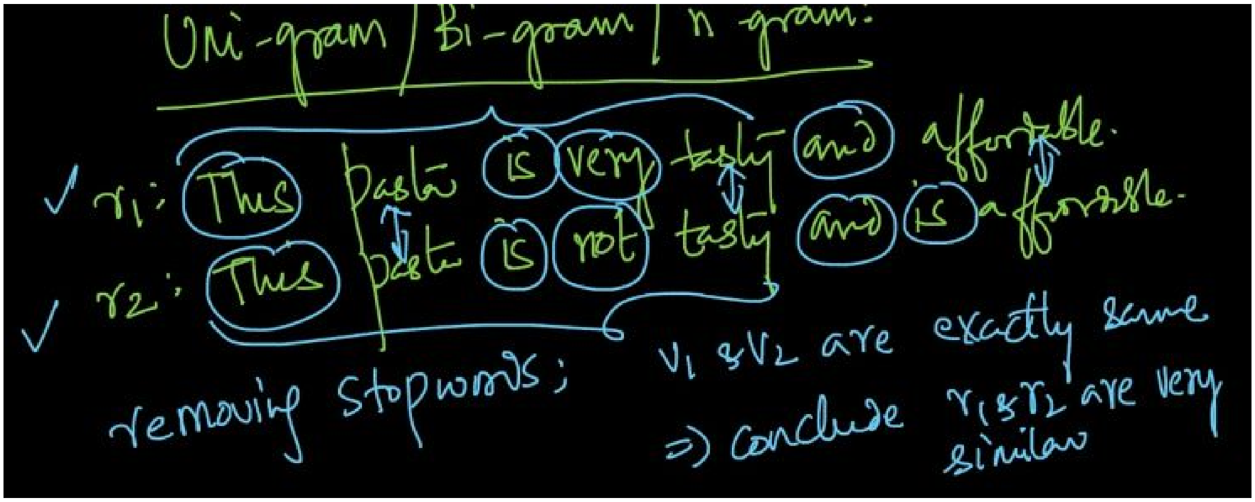
If you lemmatize the word 'Caring', it would return 'Care'. If you stem, it would return 'Car' and this is erroneous.

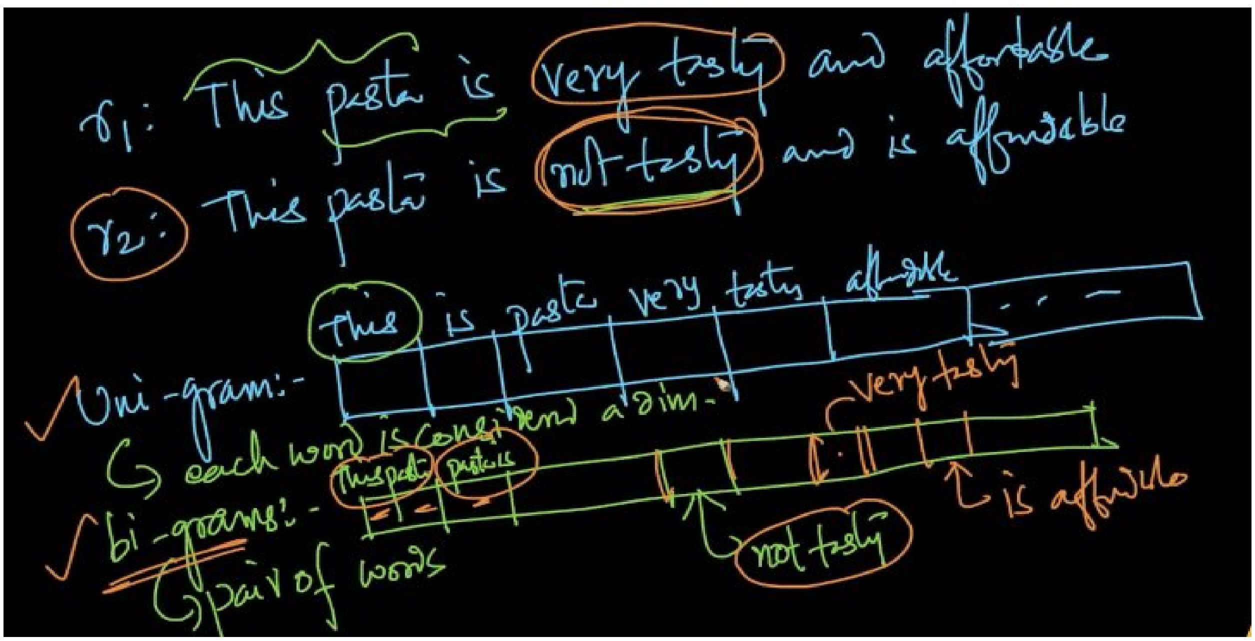
If you lemmatize the word 'Stripes' in verb context, it would return 'Strip'. If you lemmatize it in noun context, it would return 'Stripe'. If you just stem it, it would just return 'Strip'.

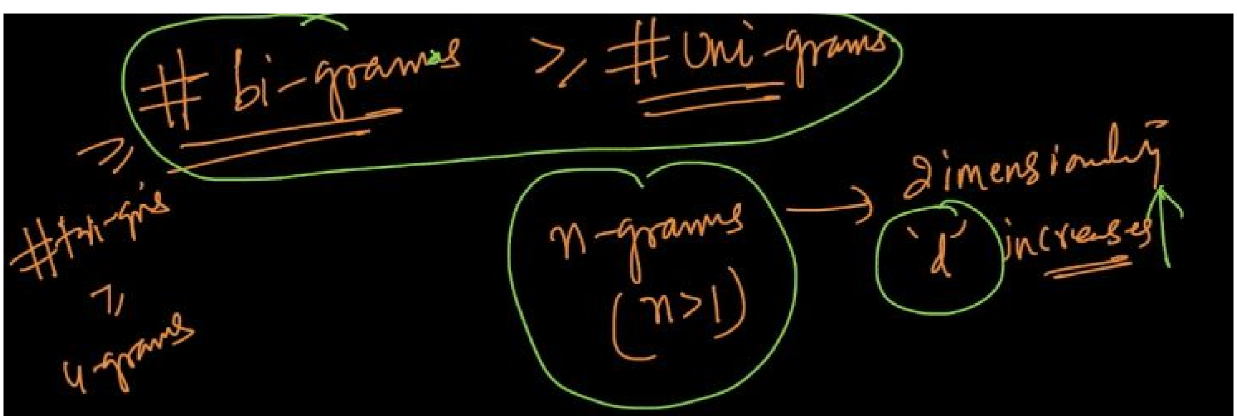
You would get the same results whether you lemmatize or stem words such as walking, running, swimming... to walk, run, swim etc.

Lemmatization is computationally expensive since it involves look-up tables and what not. If you have a large dataset and performance is an issue, go with Stemming. Remember you can also add your own rules to Stemming. If accuracy is paramount and the dataset isn't humongous, go with Lemmatization.

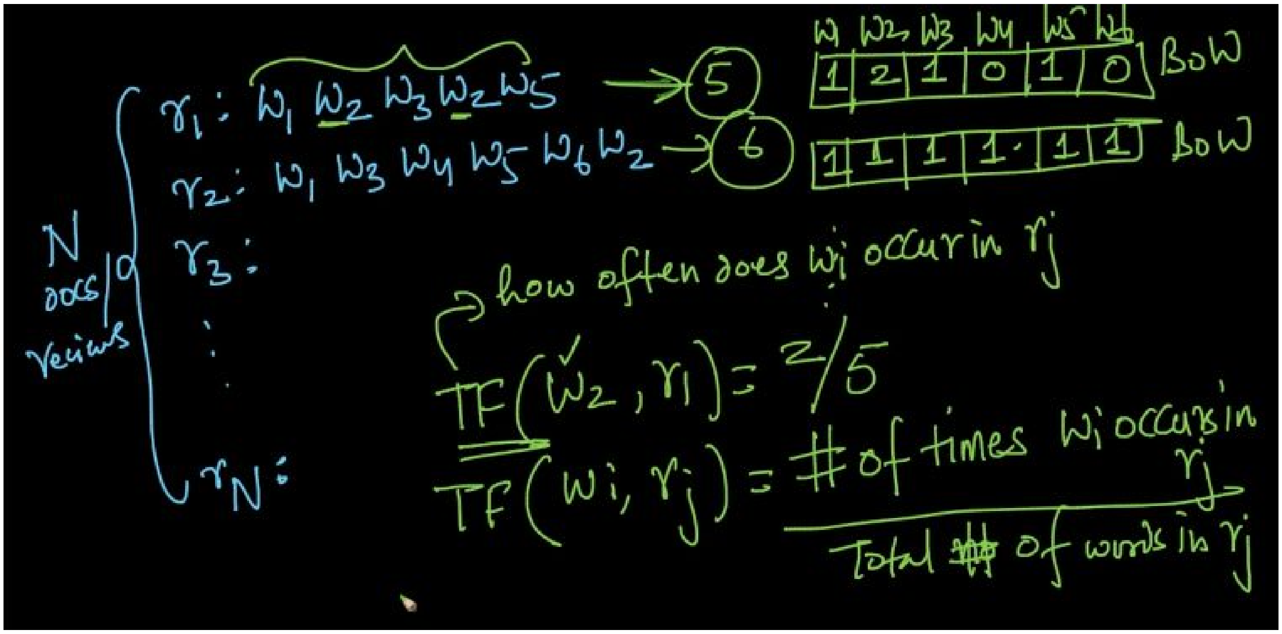
uni-gram, bi-gram, n-grams.

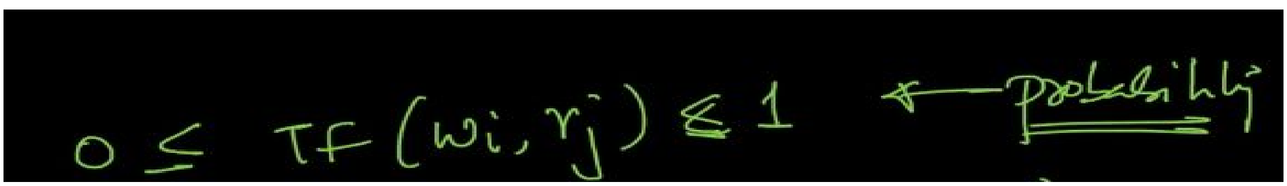


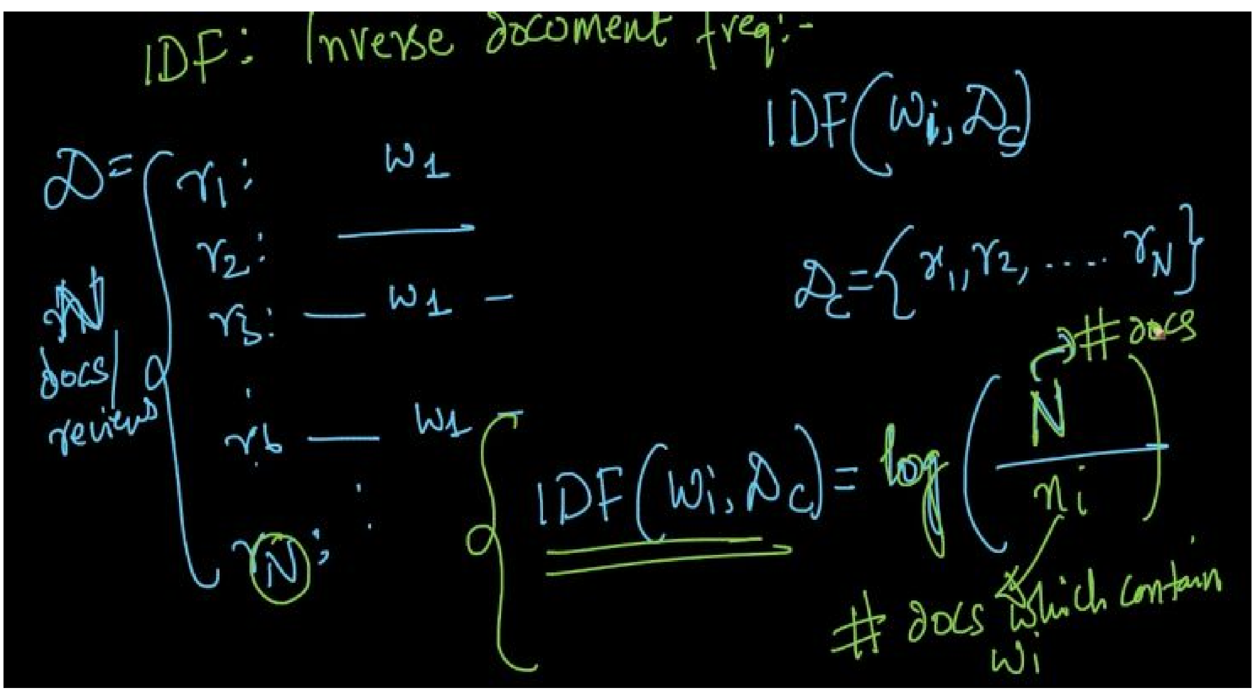




tf-idf (term frequency- inverse document frequency)







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

$N \rightarrow$ total # docs

$n_i \rightarrow$ # docs containing w_i

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

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

$\log(1) = 0$

$\log\left(\frac{N}{n_i}\right)$, $N \rightarrow 1000$

(1) $IDF \ge 0$

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

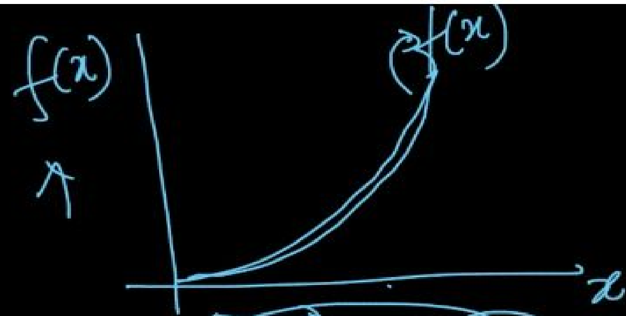
if w_i is more freq in my corpus then IDF $\downarrow$

$\frac{1000}{10} > \frac{1000}{20}$

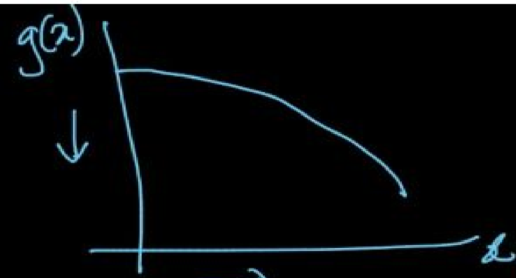
monotonic fn

IDF $\downarrow$ $n_i \uparrow$

$n_i \downarrow$ IDF $\uparrow$

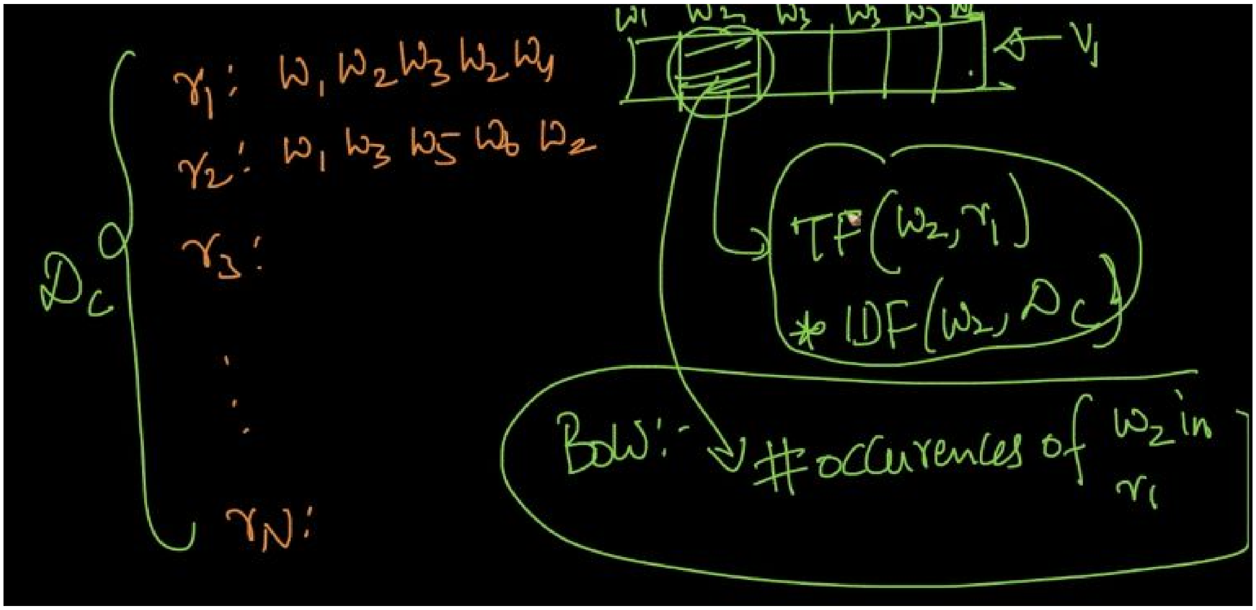


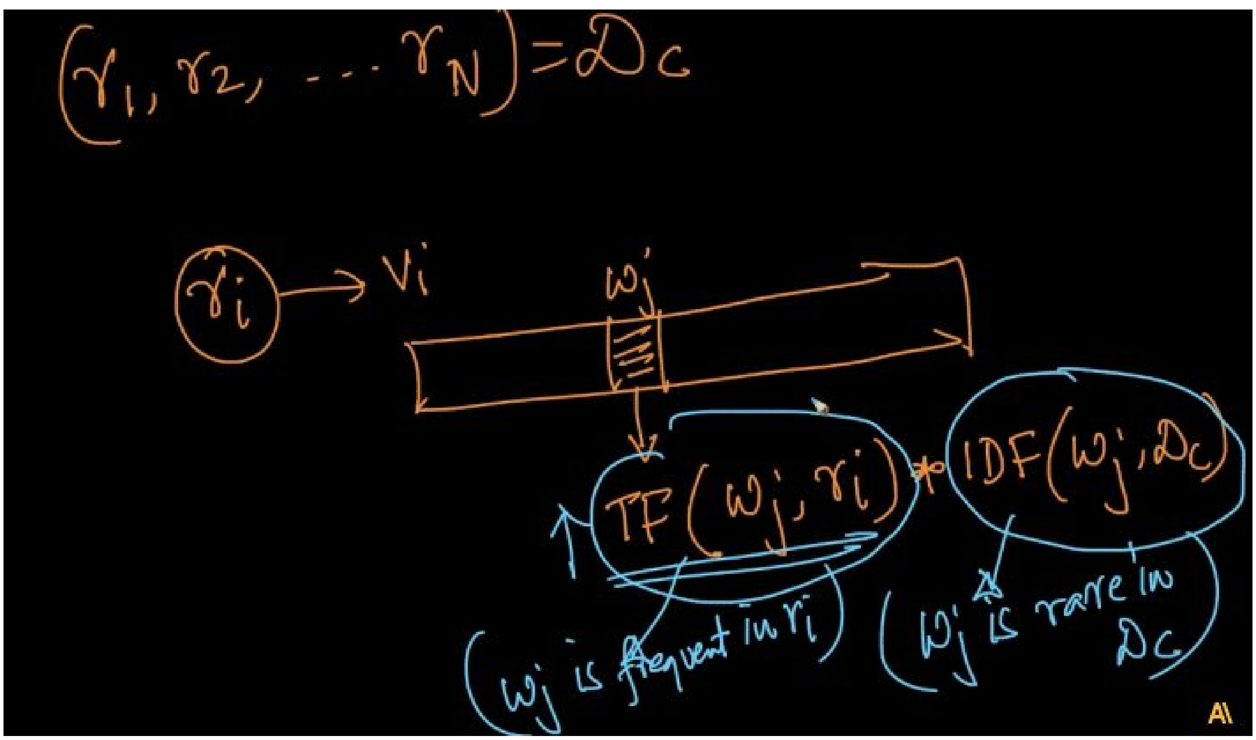
monotonically inc

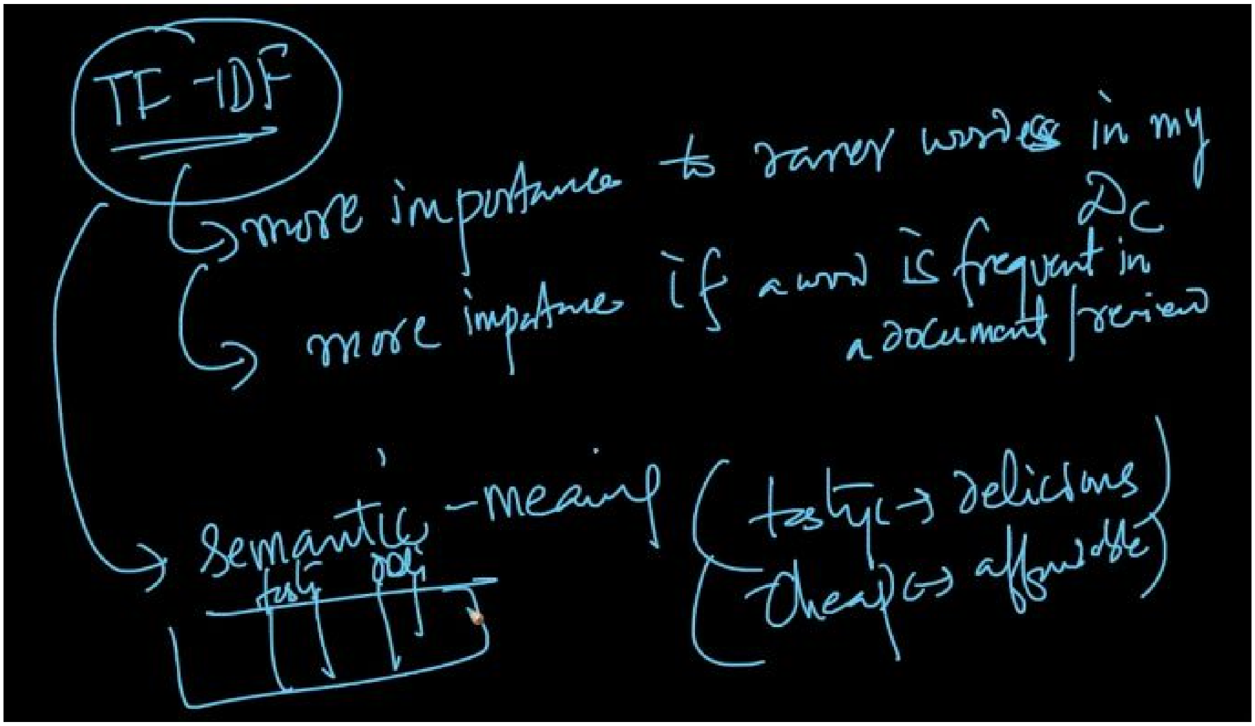


g(x) : monotonically decreasing fn

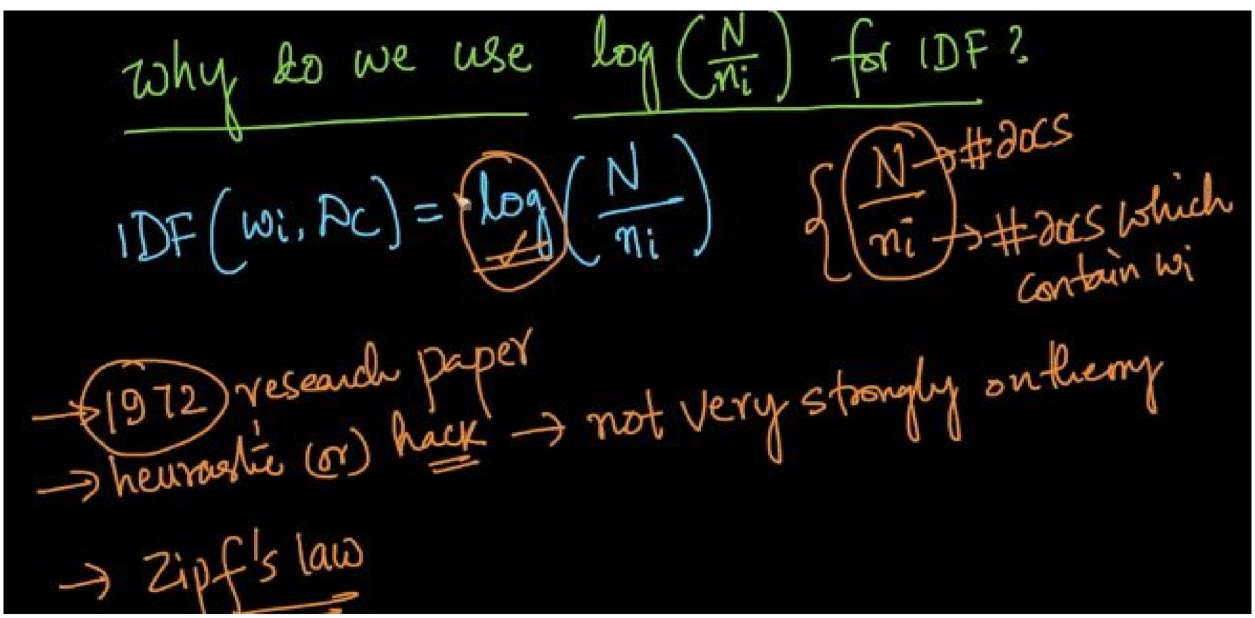
When W_i is more frequent, IDF will be low
When W_i is a rare word, IDF will be higher.

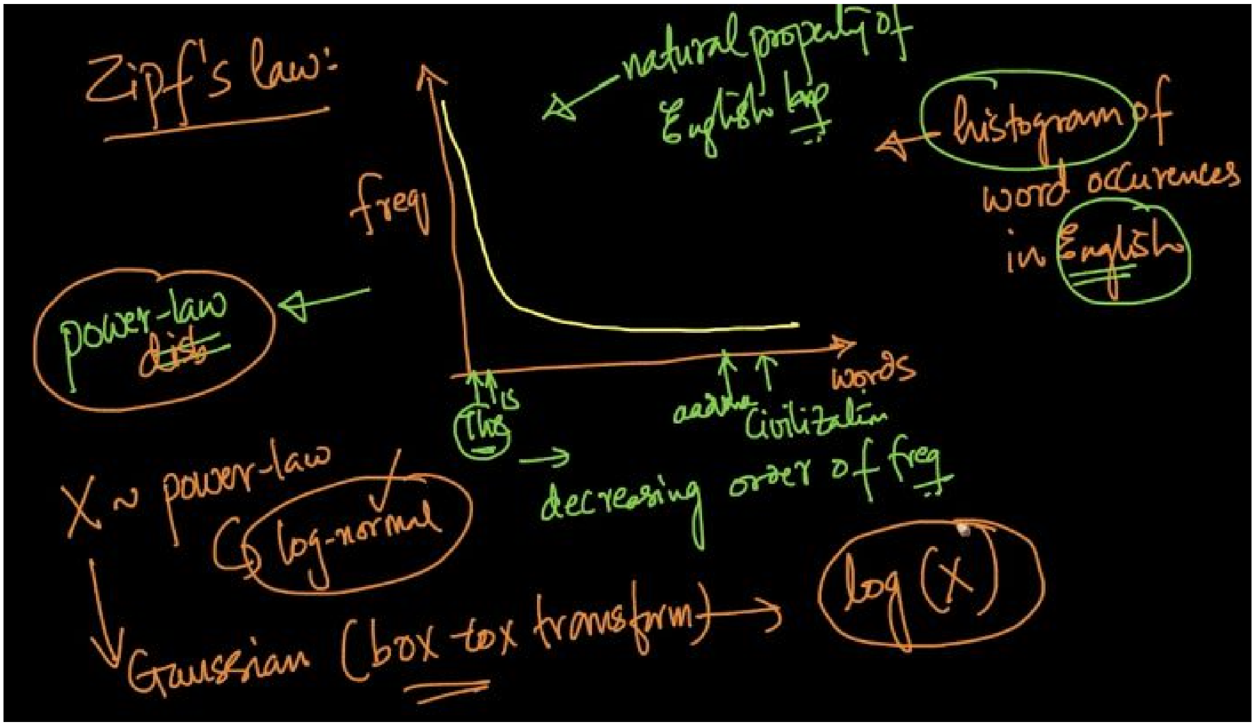


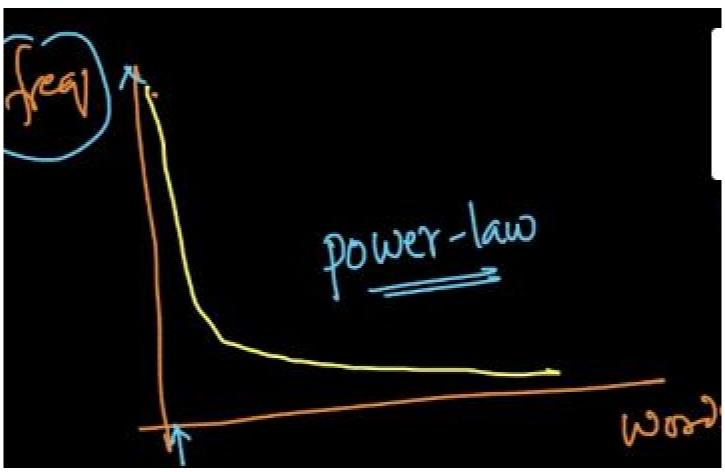


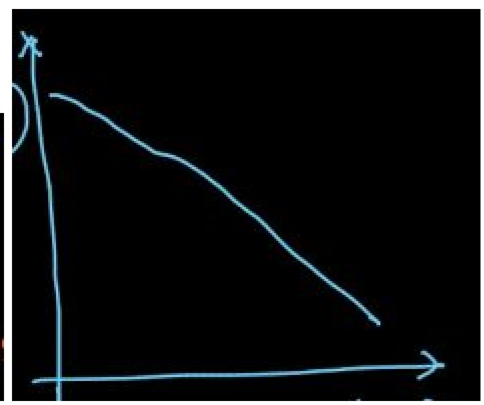


Why use log in the IDF?





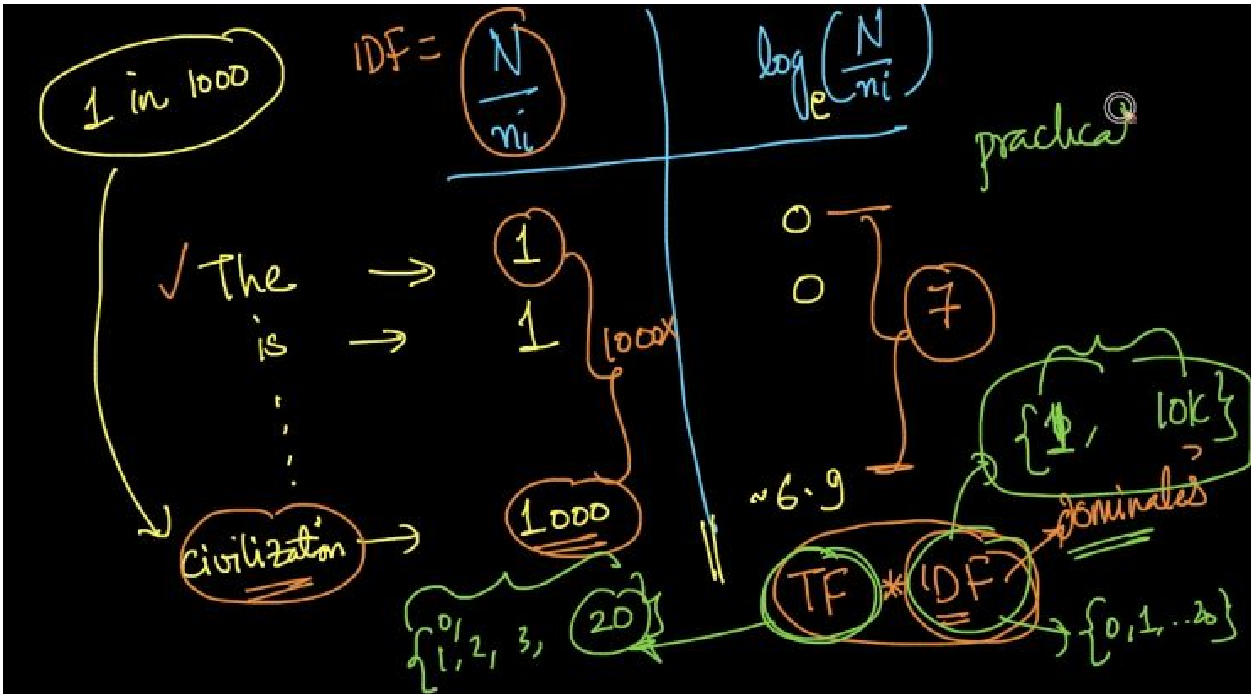


[1]

log(freq) vs log(word) gives straight line

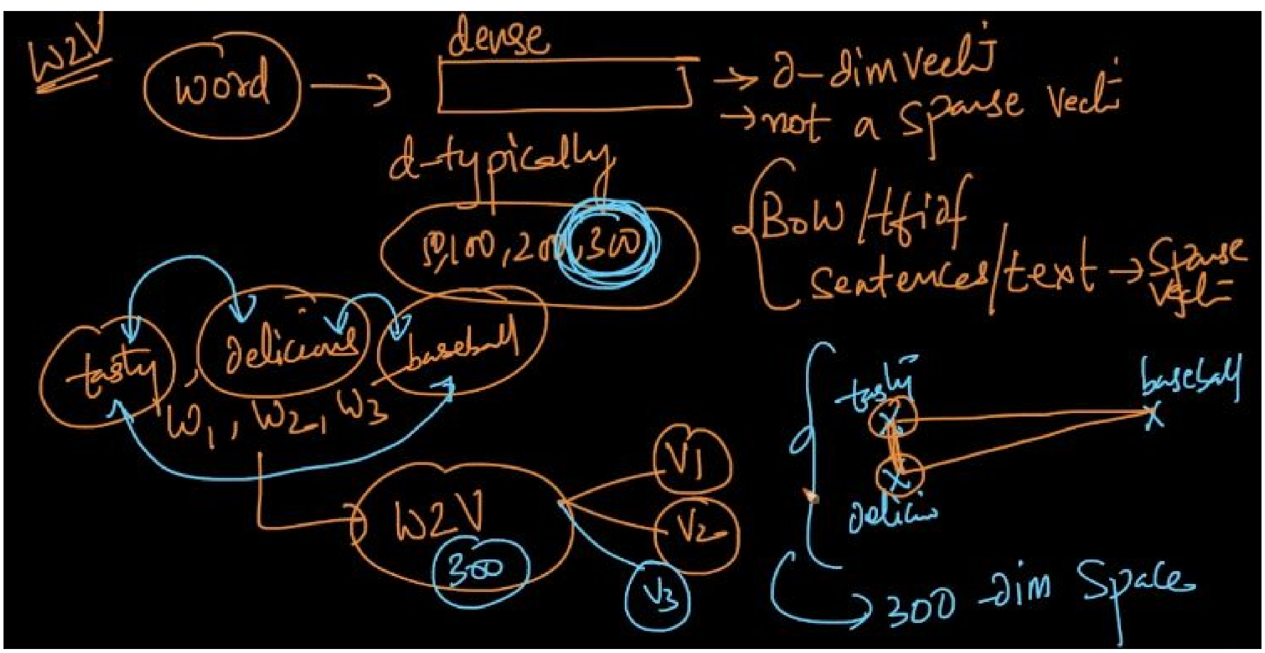
Note:
IDF without a log in the formula will have large numbers that will dominate in the ML model
Taking log will bring down dominance of IDF in TF- IDF values;

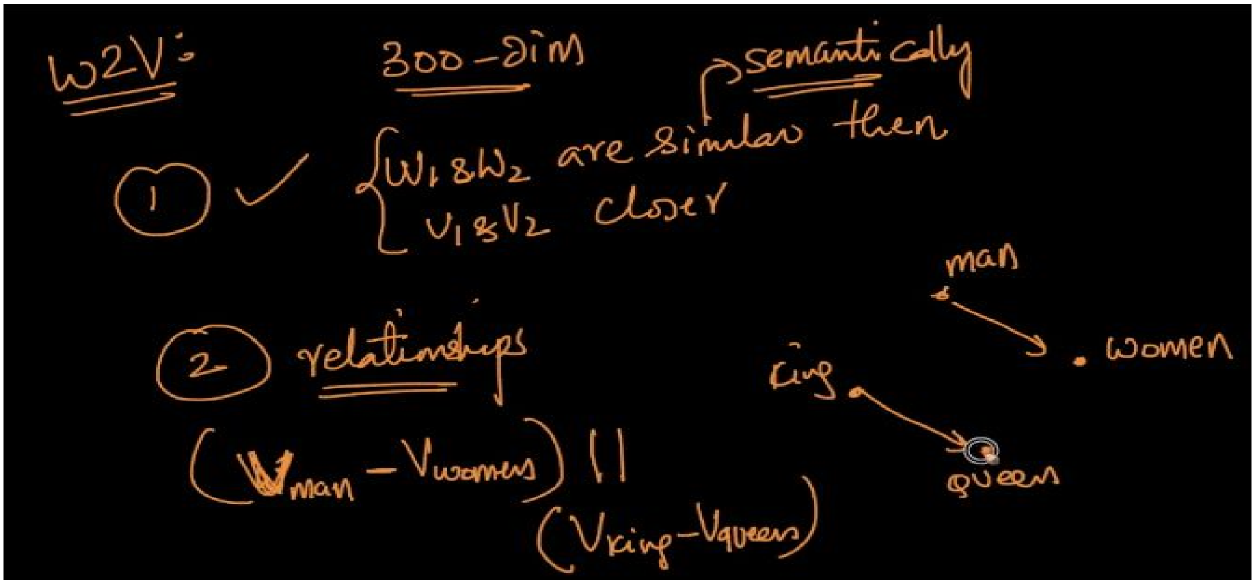
1

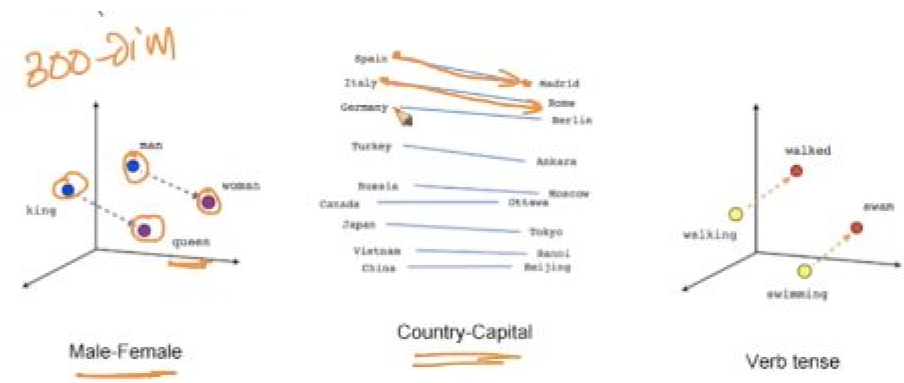


Word2Vec

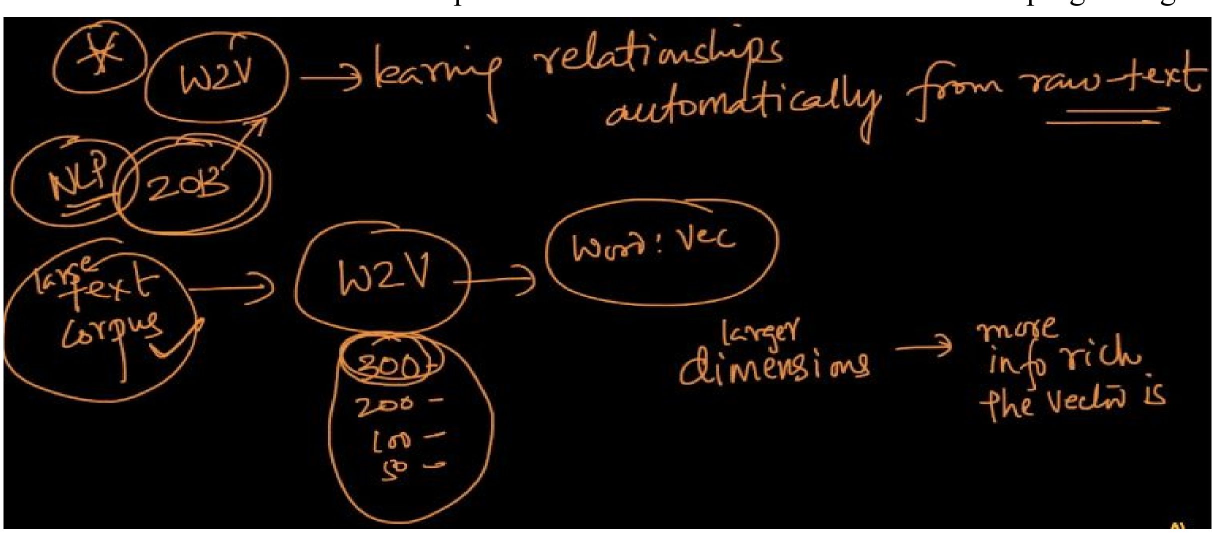
State of art methodology to convert word to vec. Based on semantic meaning .

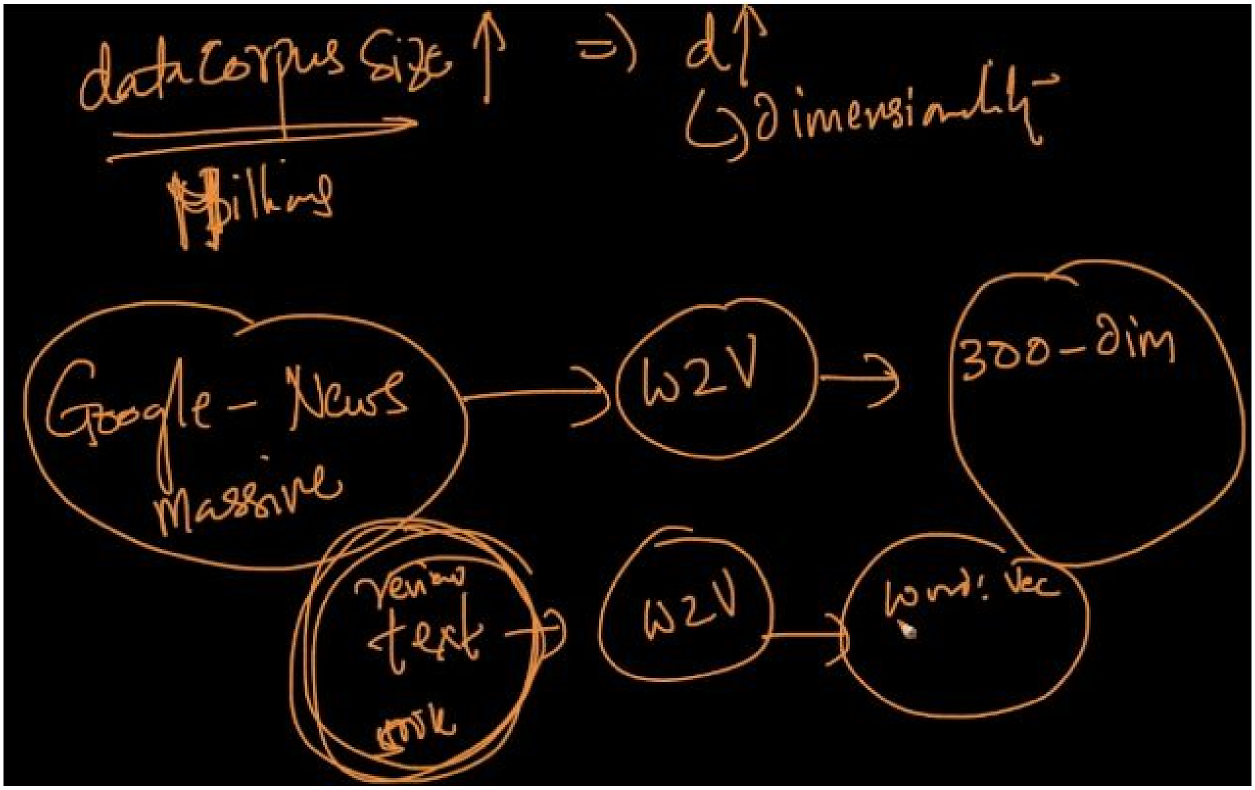




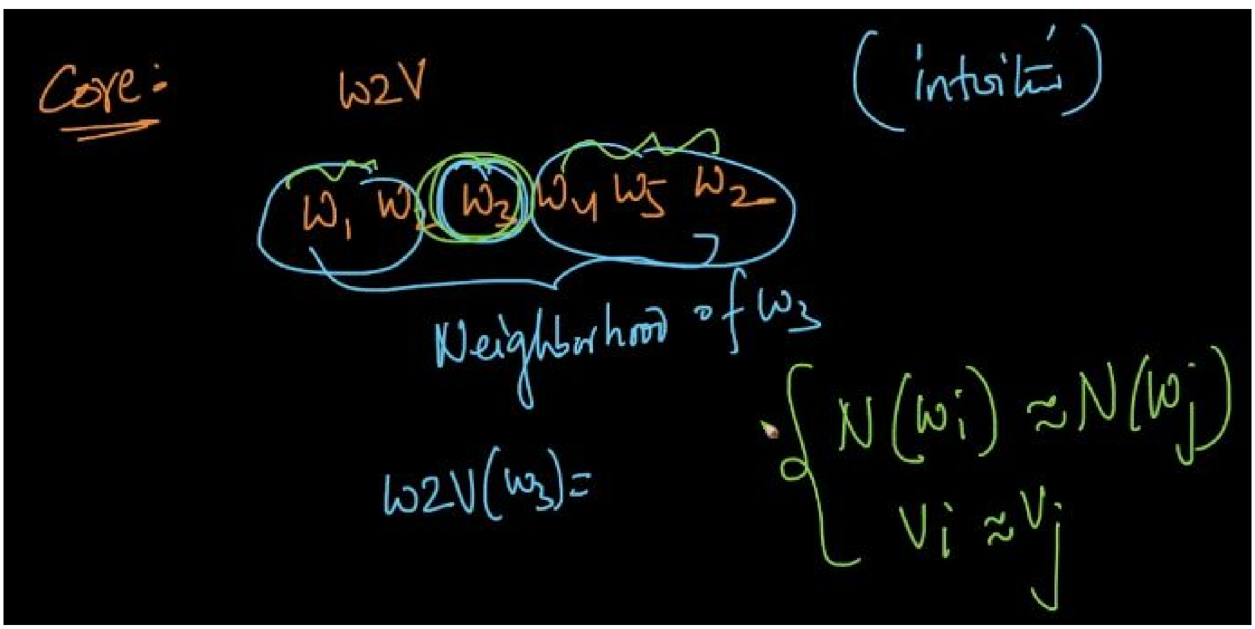


Word2Vec learns all the relationships from raw-test without the involvement of programing.



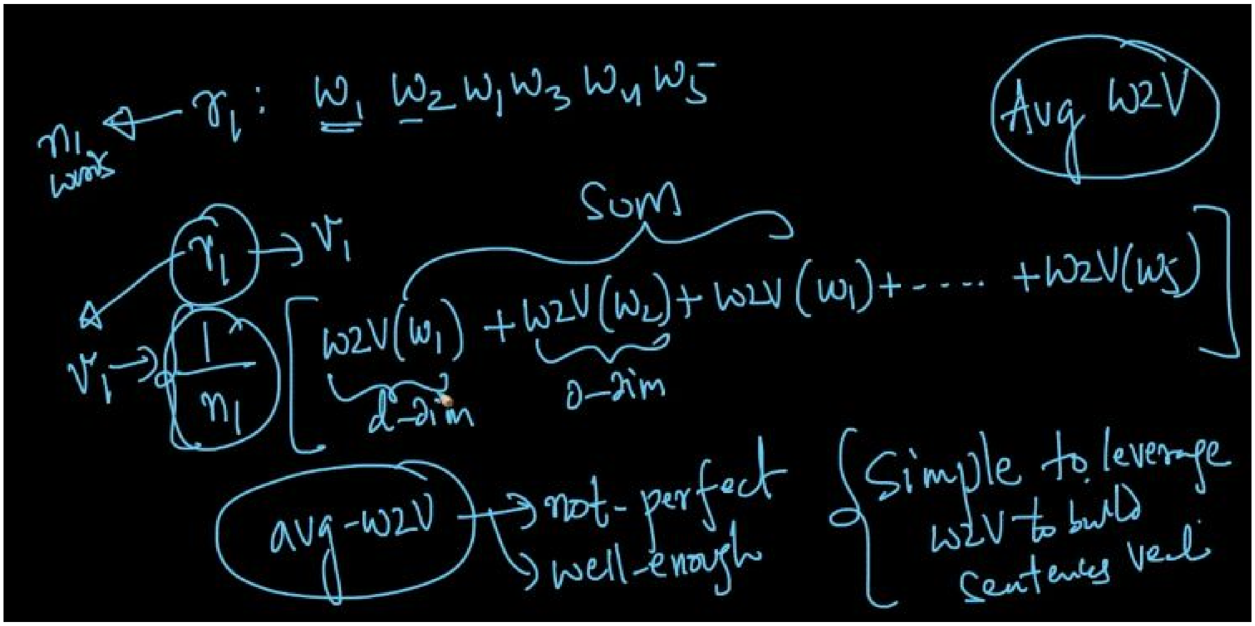


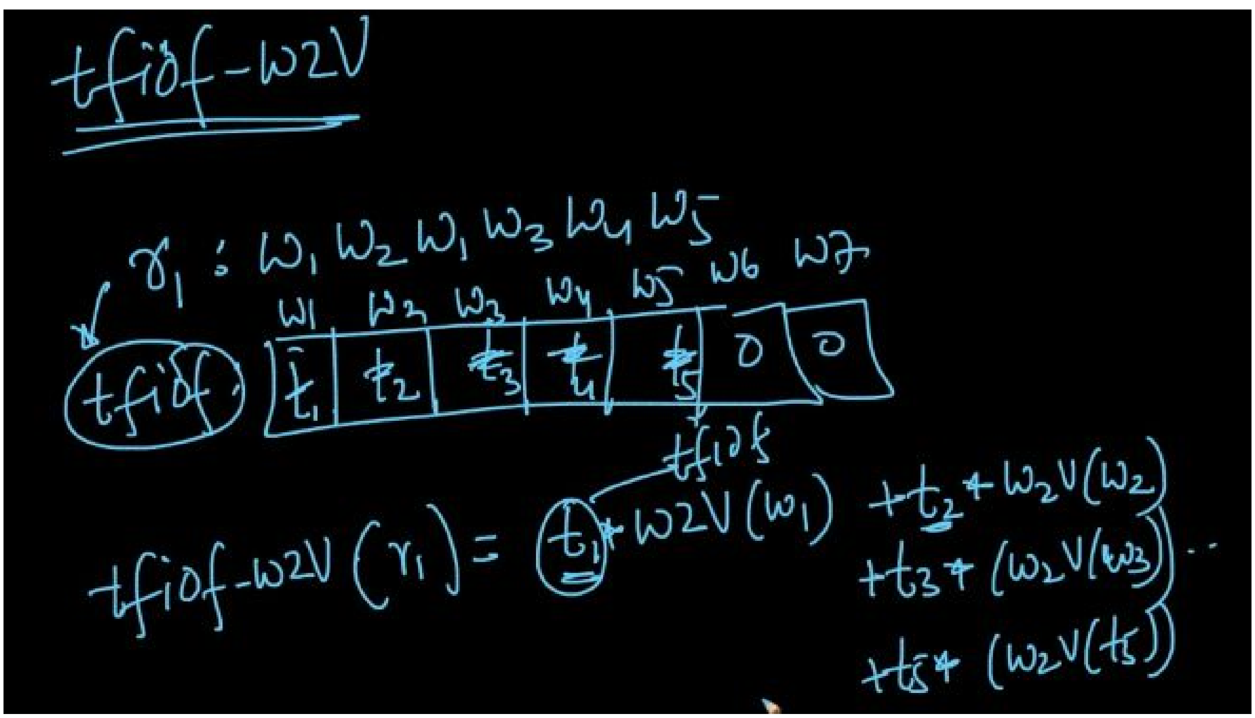
data corpus size ↑ ⇒ d ↑
dimensionality
Google-News
Massive
W2V
300-dim
review
text
W2V
word: vec

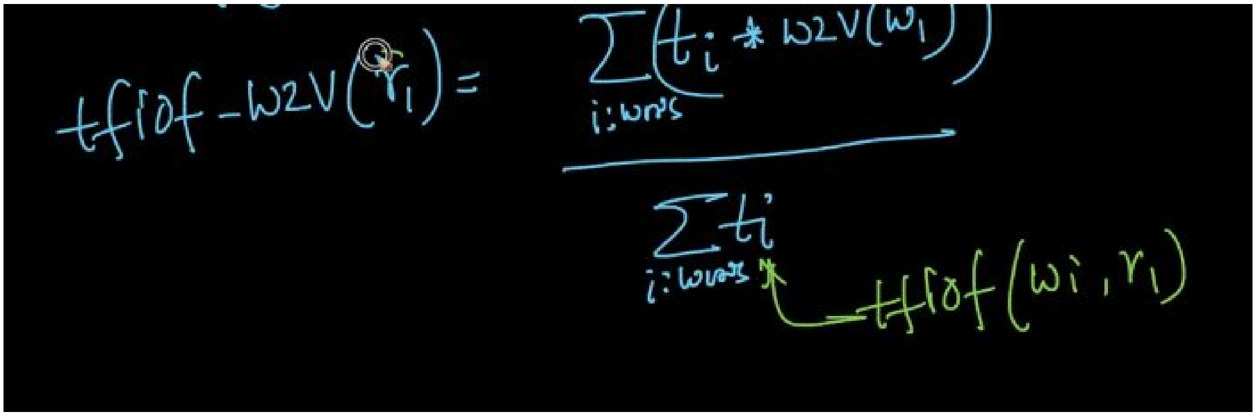


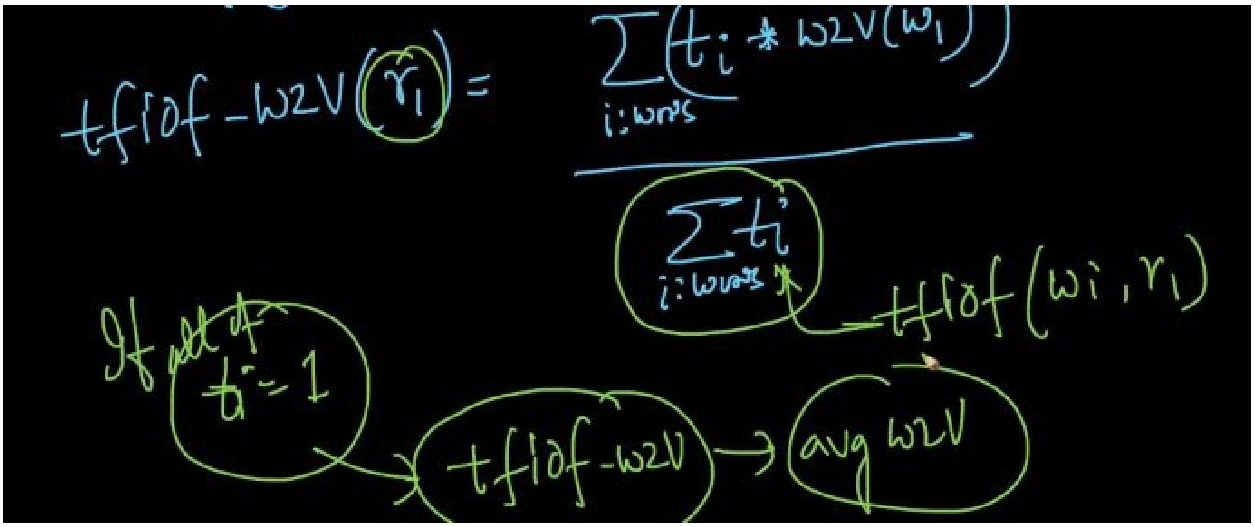
Core:
W2V
(intuition)
W1 W2 W3 W4 W5 W2
Neighborhood of W3
W2V(W3)=
N(Wi) ≈ N(Wj)
Vi ≈ Vj

Avg-Word2Vec, tf-idf weighted Word2Vec



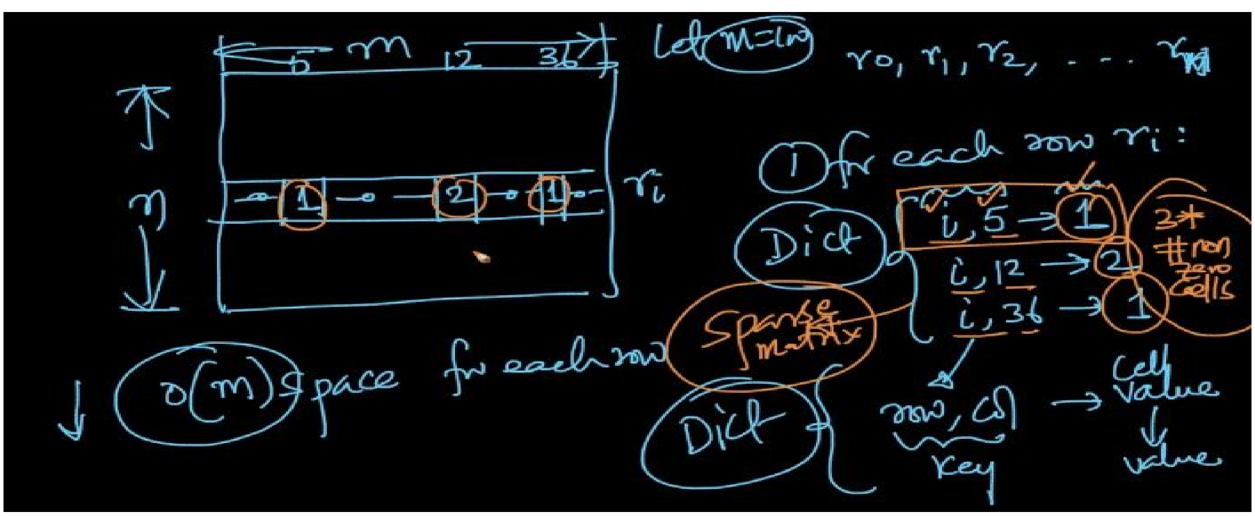






sparsity of the matrix = no.of zero elements / total elements

We can store sparse matrices with a less order of space by efficient method using by storing the row and column indices with corresponding values;



CountVectorizer() does this automatically . It has an inbuilt system to remove the super sparsity of the dataset.

CountVectorizer()

Documentation:

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html