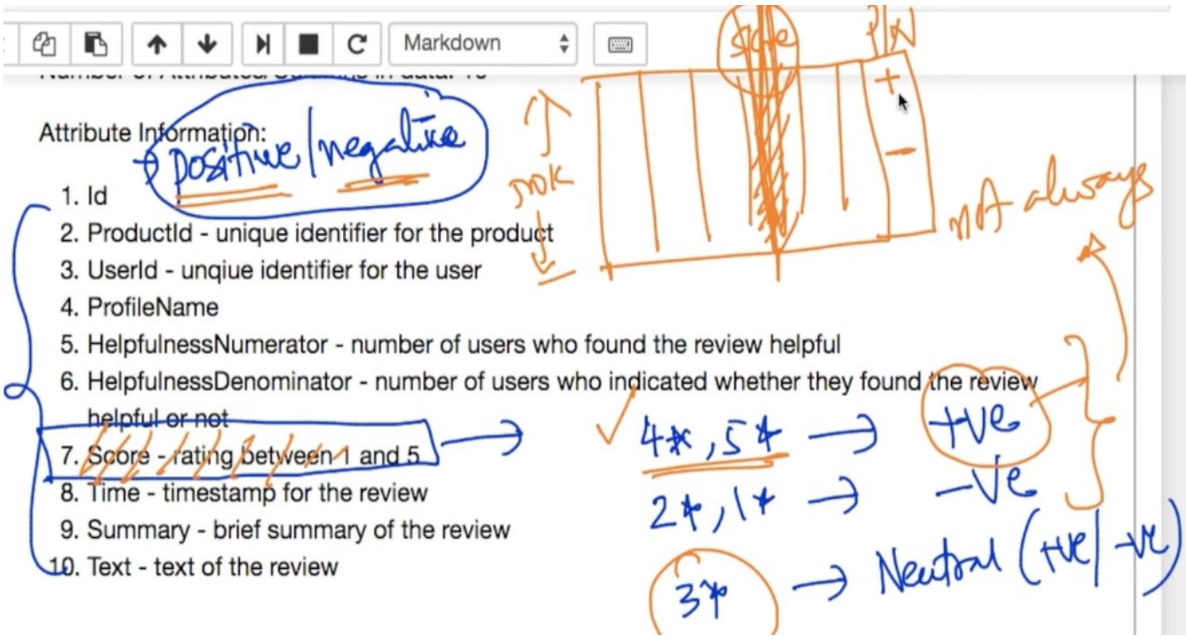
PREDICT RATING PRODUCT REVIEWS ON AMAZON

DATASET OVERVIEW: AMAZON FOOD REVIEWS (EDA)



Dataset understanding :

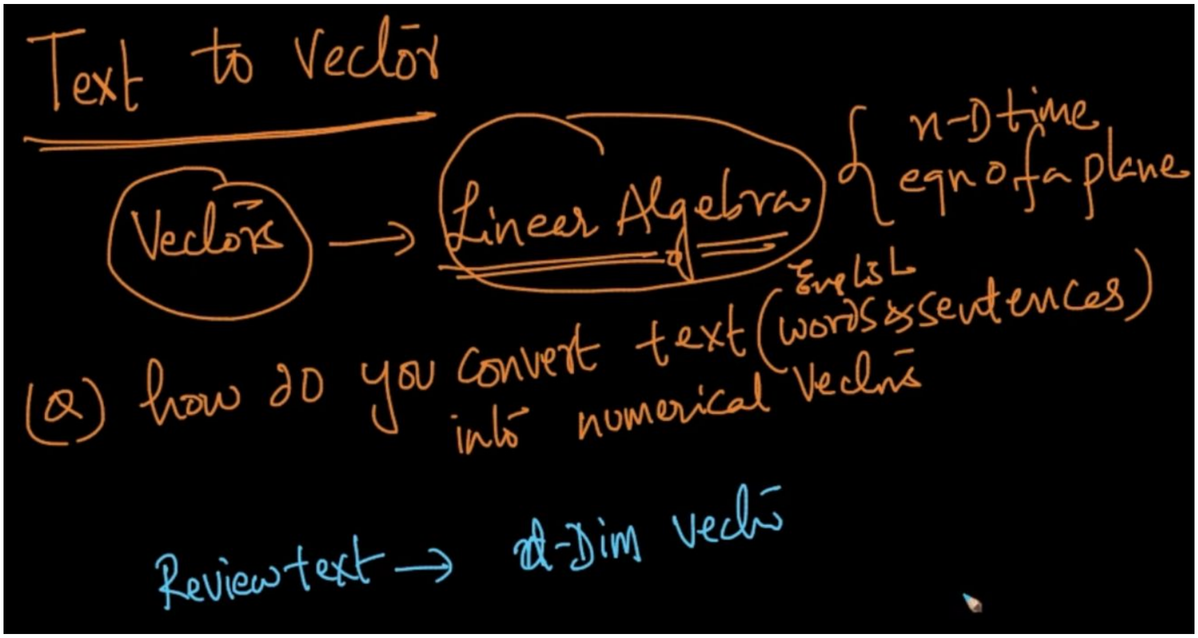
1) HelpfulnessNumerator - Who found review useful useful (Ex - 4500)

2) HelpfulnessDenominator - They found the review useful + not useful (Ex - 4500 + 500(Not use))

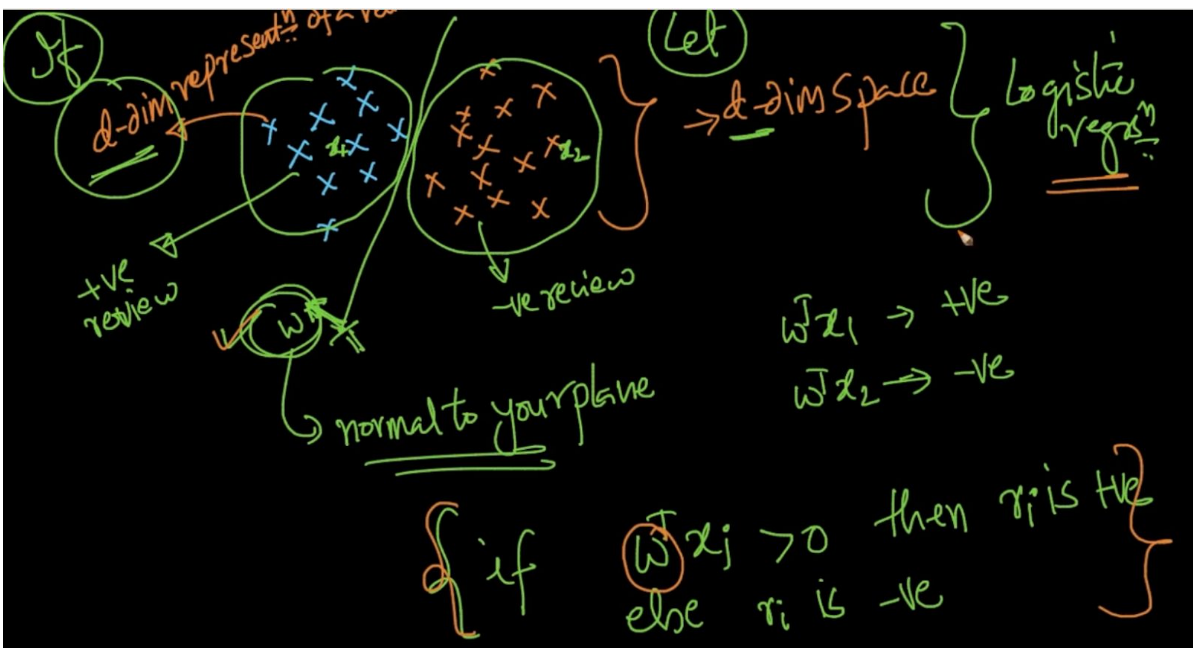
3) Score - It is between 1 to 5 and we want it to be binary so 1*,2* is negative and 4*,5* positive by adding a new label.

4) Text + Summary : Most important in our dataset since it is an **NLP** problem

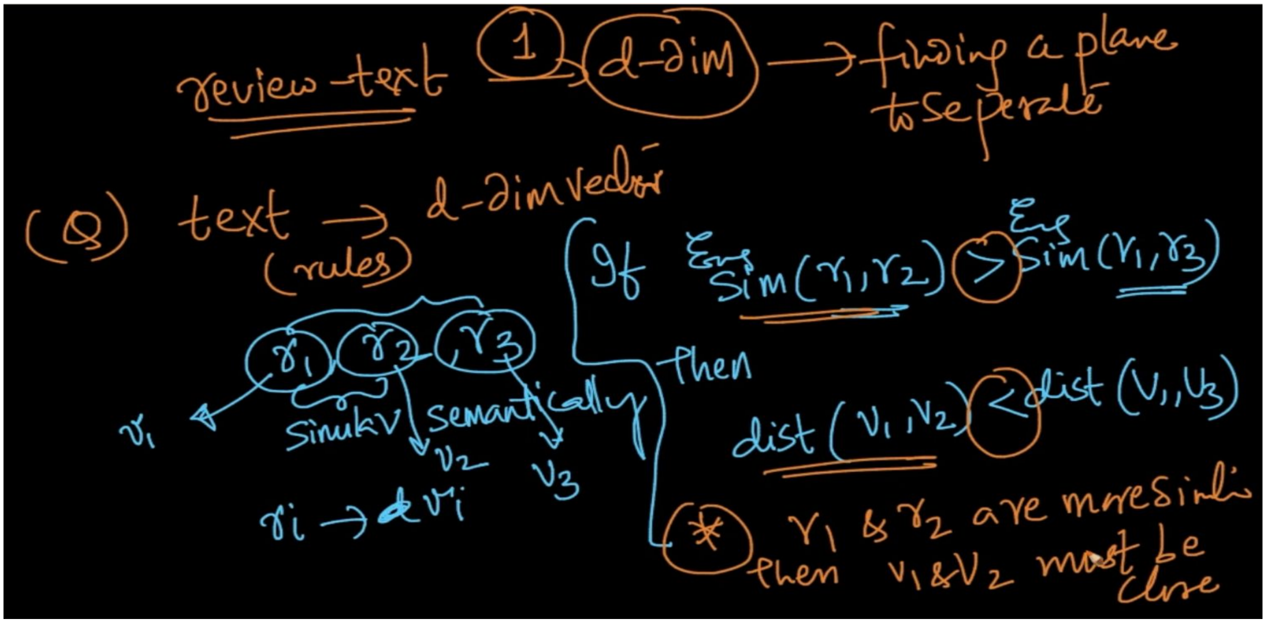
WHY CONVERT TEXT INTO VECTOR ?



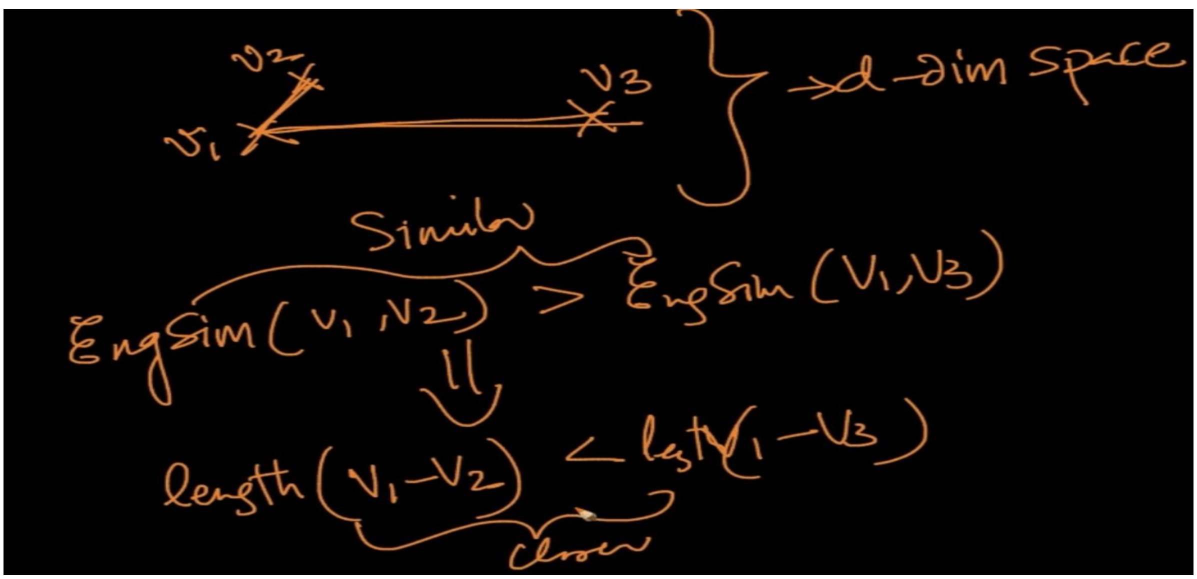
In the Amazon Food review if we could convert our text into vectors we could use all the Linear Algebra things for our sentiment analysis. Let's say we convert our review text to d-dim vector



In a d - dimensional space if the data-points(text reviews) are represented like this then we could separate them using a Linear Algebra by finding a normal to the plane that has separated the dataset



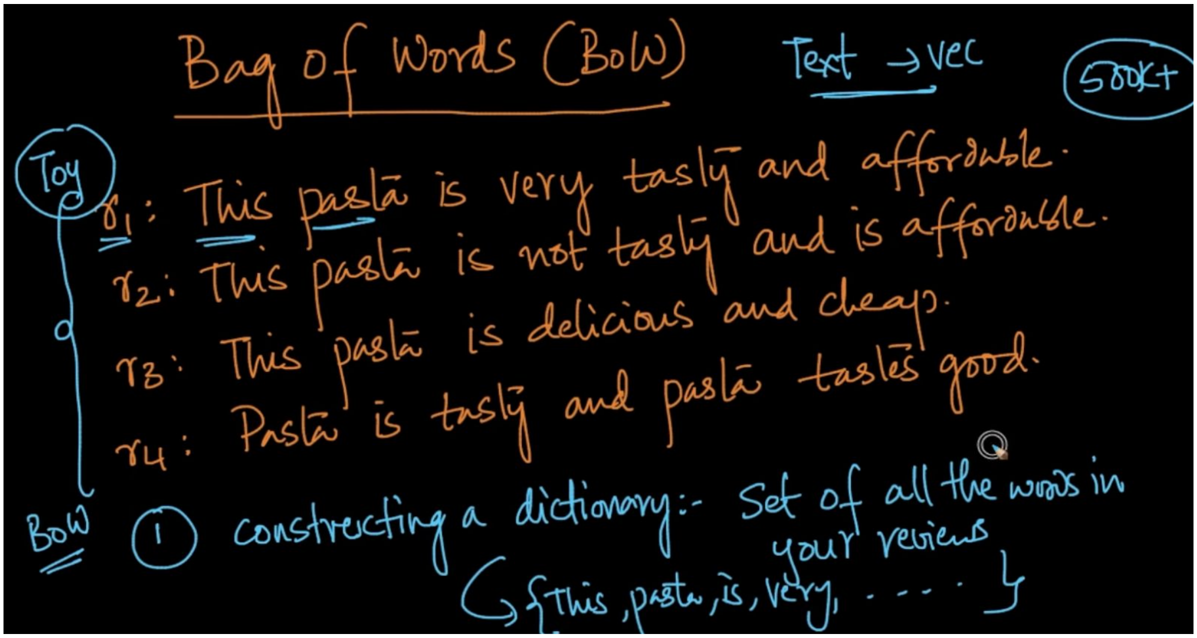
So first our text should be converted into d-dimensional vectors. Then suppose if we've 3 reviews r1,r2 and r3 and r1 and r2 are more similar then distance of r1 and r2 should be less than distance between r1 and r3



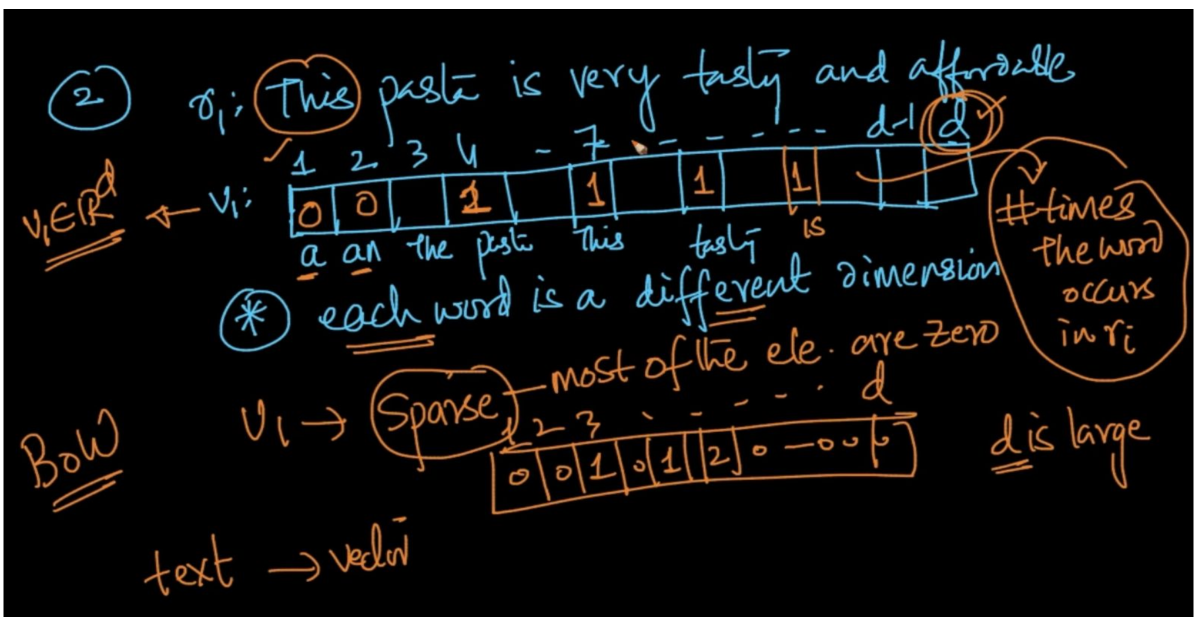
Geometric interpretation. So if the similar texts i.e positive text with positive and negative with negative are aligned or geometrically closer then we can find the plane to separate them.

So converting text to vector s.t they are geometrically closer we'll learn techniques like Bag of words(BOW), word2vec, tf-idf and so on

G OF WORDS (B-O-W)

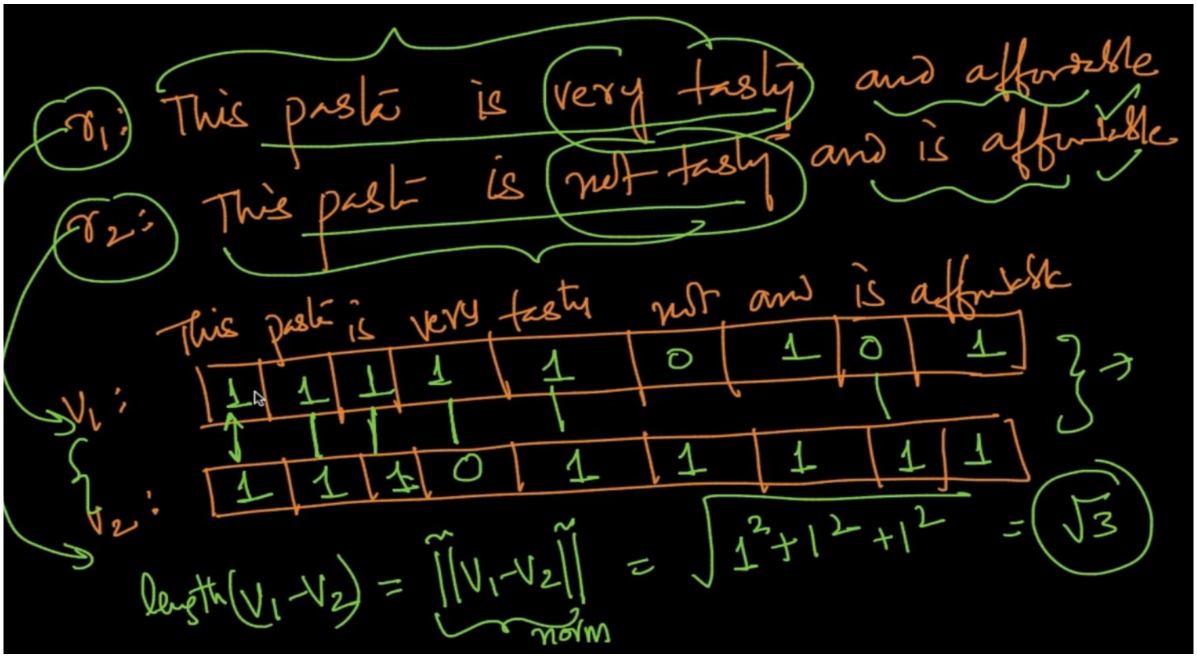


So, these are the 4 reviews. In BoW the 1st step is constructing a dictionary: Set of all the unique words in reviews {This,pasta,is,very …..}



The dict is a d-dim vector containing all the words. Each index will correspond to the word and the value will tell us about the frequency of a word. Since there will be many words in all the reviews (documents) the vector for a single review is very sparse

So in BoW we are trying similar texts to be geometrically closer.



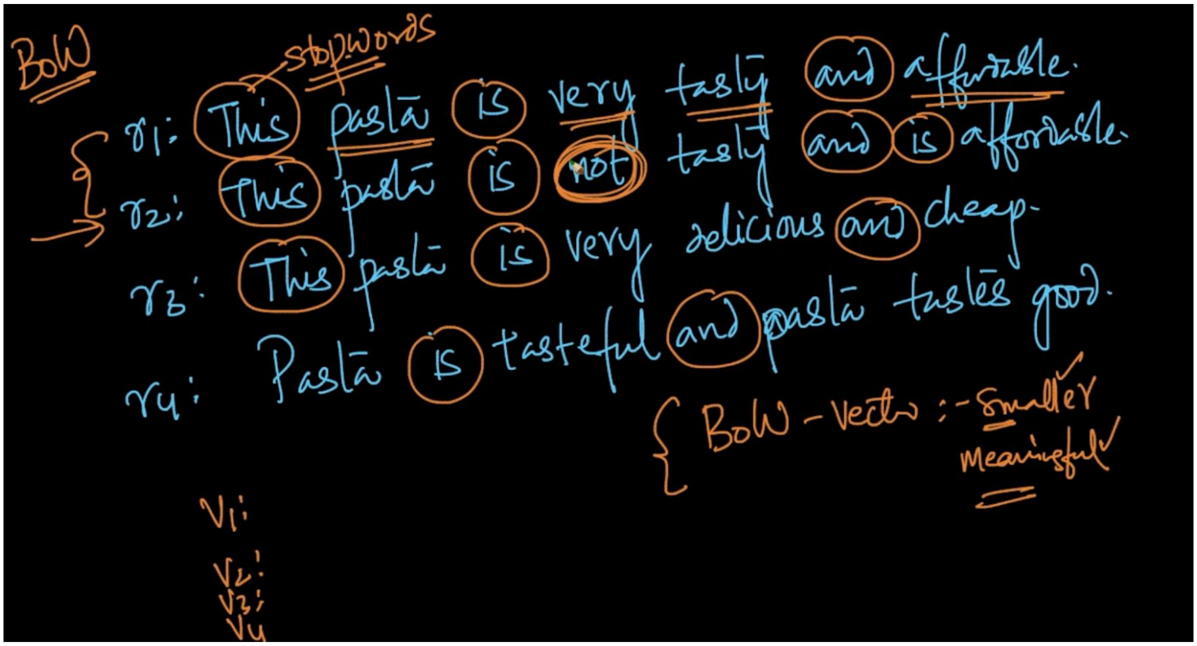
Here, we took 2 reviews and converted them into vectors and then we calculated the differences in vectors and calculated distances as seen but there's a problem . In the 2 reviews, *very tasty* (r1) and *not tasty* (r2) have very different meaning but the distance is less. So it's a problem in BoW

Note:

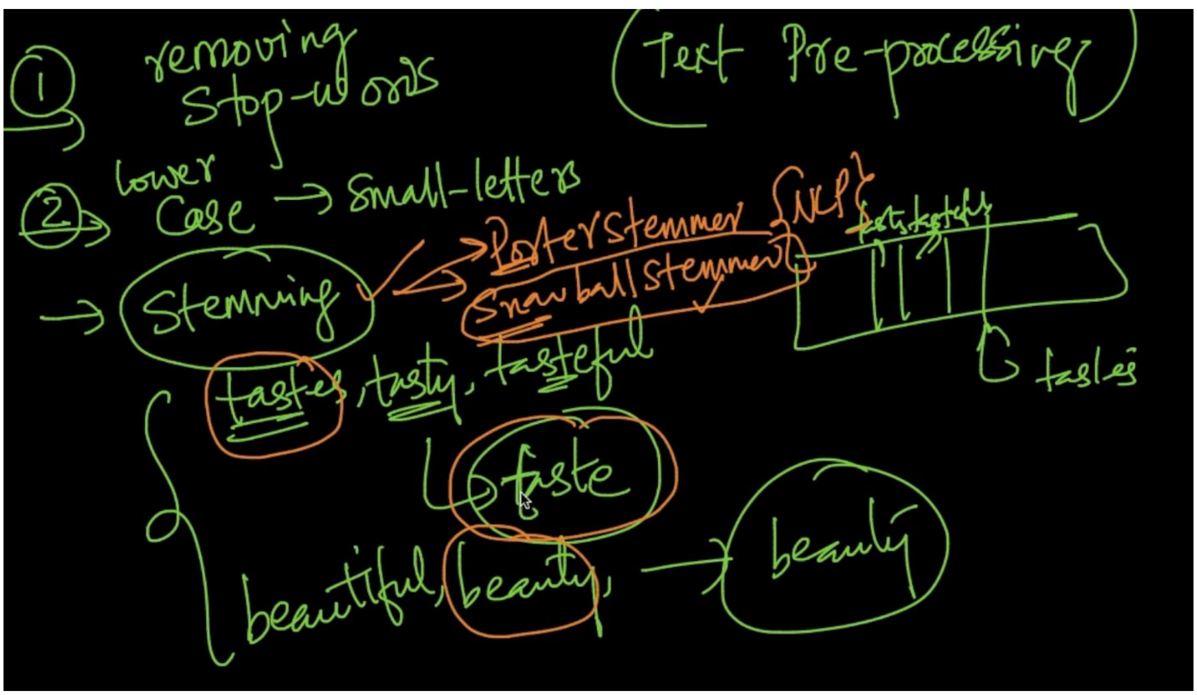
1) Binary BoW: The value in the vector is no. of times the word is occuring so if the word "Pasta" is occuring twice then value will be two. So in binary BoW we if the word is occuring then we'll just give value 1 because we are trying to min the distances of similar texts

2) Words like "The, is, are ,that etc" are not important in text-analysis so we try to remove them

Stop-word removal, Tokenization, Lemmatization (Featurizations)



First step is we remove stopwords like this, is, not etc. It makes the vector smaller and more meaningful.



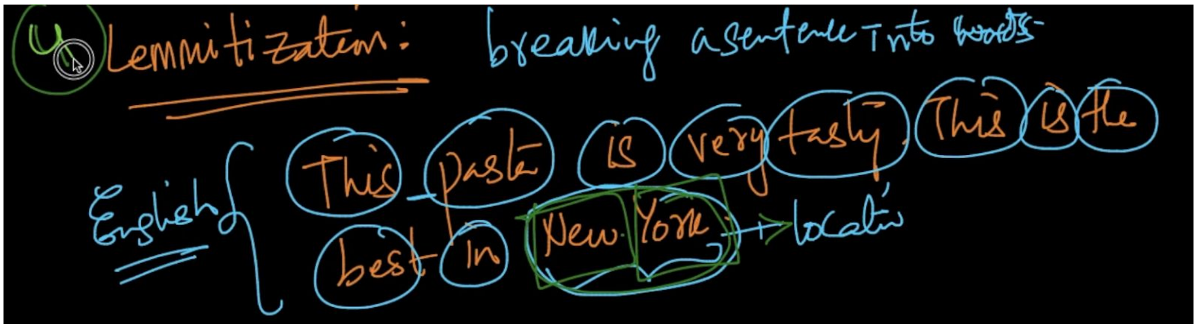
Text Pre-processing steps:

1) Removing stop-words

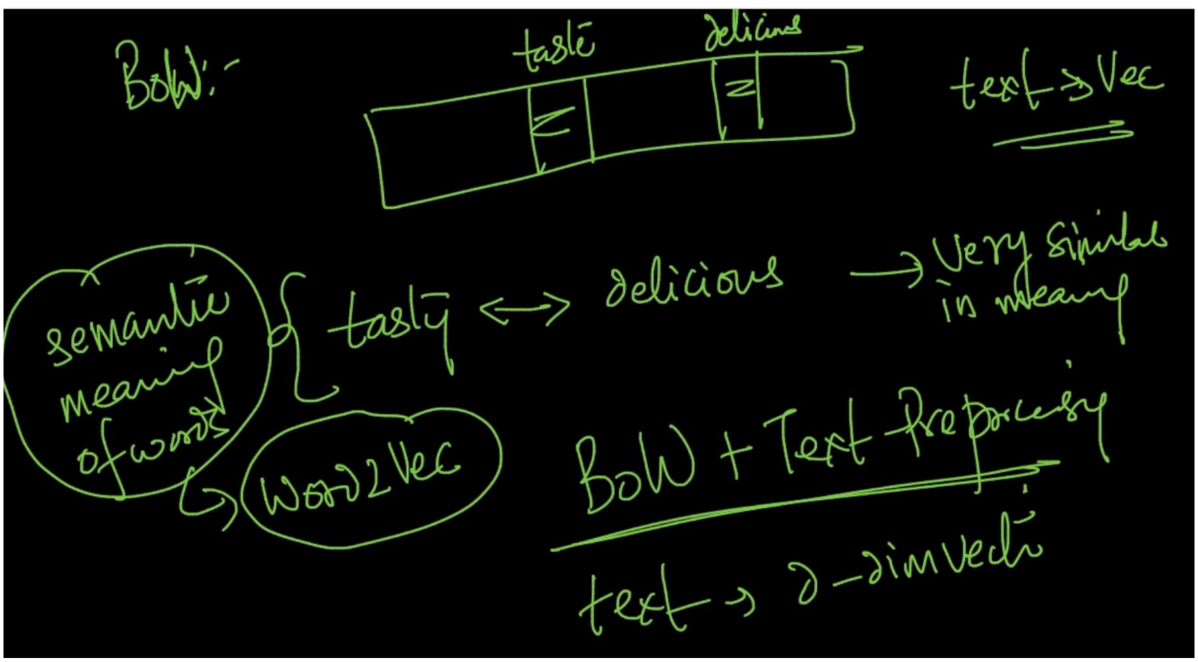
2) Lower case - We convert the letters to lower case . Why? In the above example, *Pasta* and *pasta* are two different words when they are same . So make all of them lowercase

3) Stemming - We get the essence of the word from stemming. For ex: tastes, tasty and tasteful have just one essence taste. So instead of using 3 different vectors for them, we can use a single vector

4) Lemmatization - We break the sentence into words

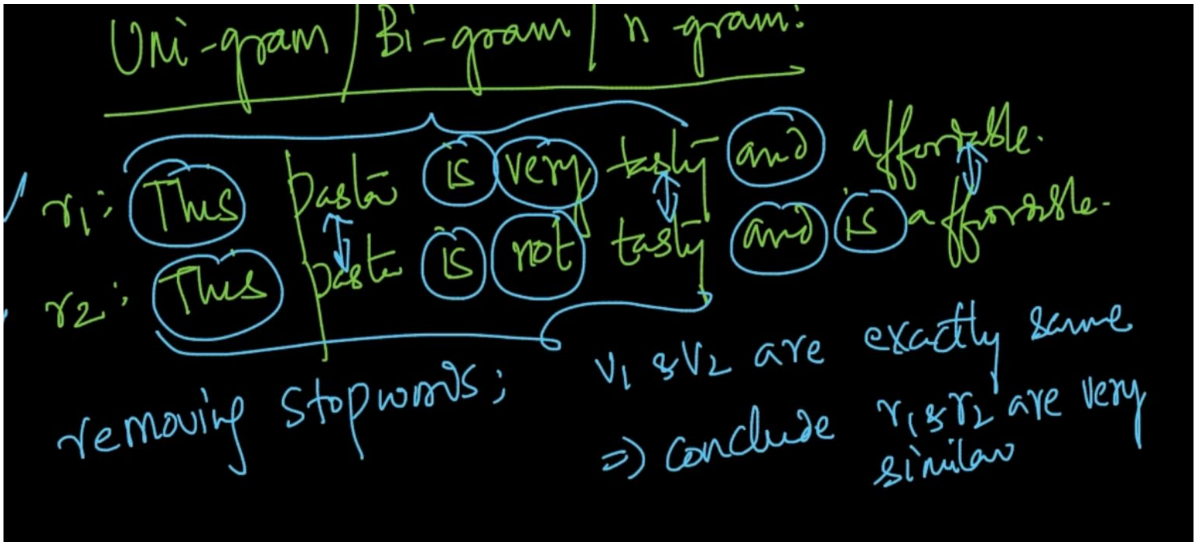


PROBLEM WITH BOW

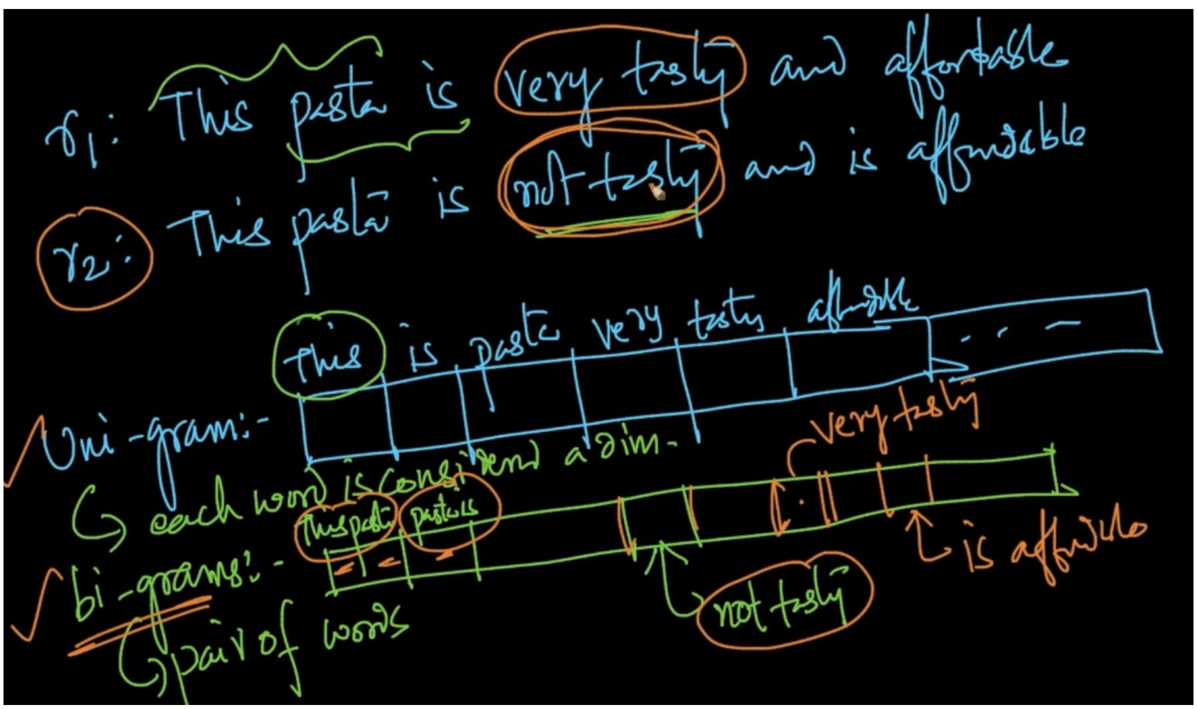


There's a problem with BOW as we can see in r1 and r2 reviews above we can clearly see that they've the same meaning like tasty and delicious are synonyms but BOW will consider as different vectors. So it will not cover the semantic meaning behind the texts. Therefore, with BOW + Text Preprocessing we are converting text into vectors.

Uni-gram/bi-gram/n-gram

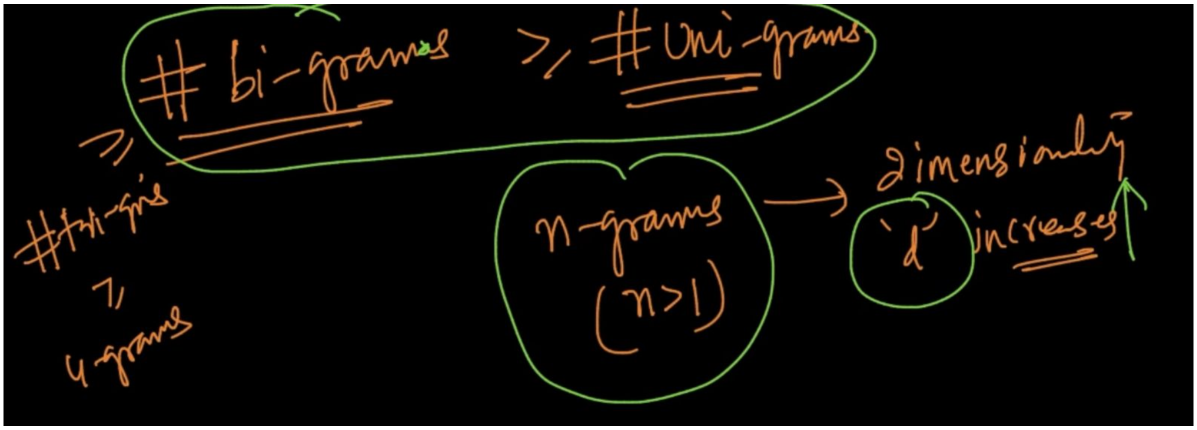


If we are removing stopwords then the above 2 reviews get very similar albeit the fact that they are different. We'll see below how to solve that



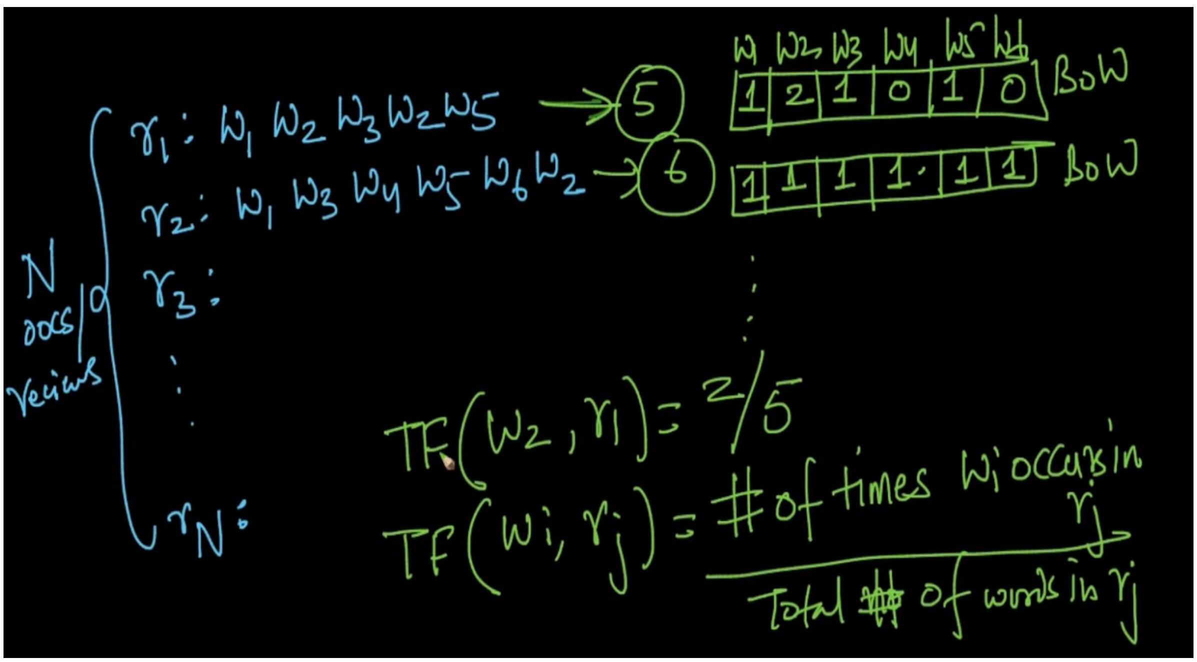
Our traditional approach was uni-gram approach . We use bi-gram in which a vector of **2 consecutive word** is created.

Note - In our unigram all the sequence info was lost as seen but in our n-grams some info is retained



No. of vectors in bi-grams > unigrams. So as n-grams -> dimensionality 'd' increases.

Tf-idf (Term Freqency- Inverse Document Frequency)

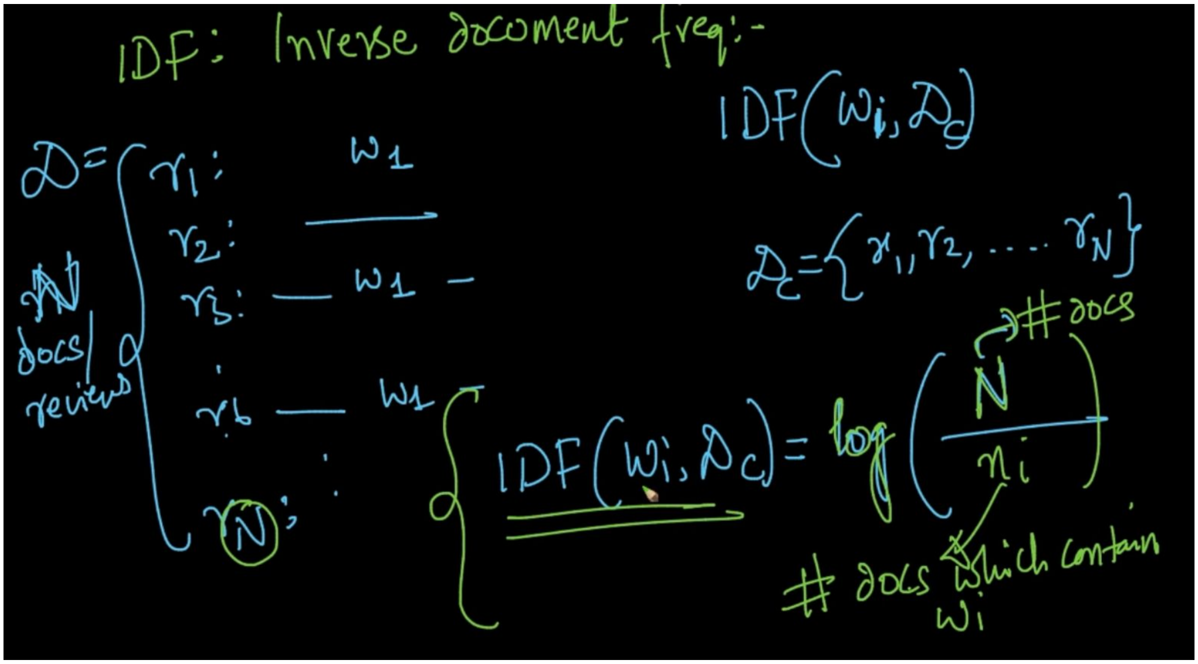


We are making BOW for each review. TF is occurence of a word/ Total no. of words

Ex : TF (W2, r1) = ⅖ as w2 is occuring twice and total no. of words in r1 is 5.

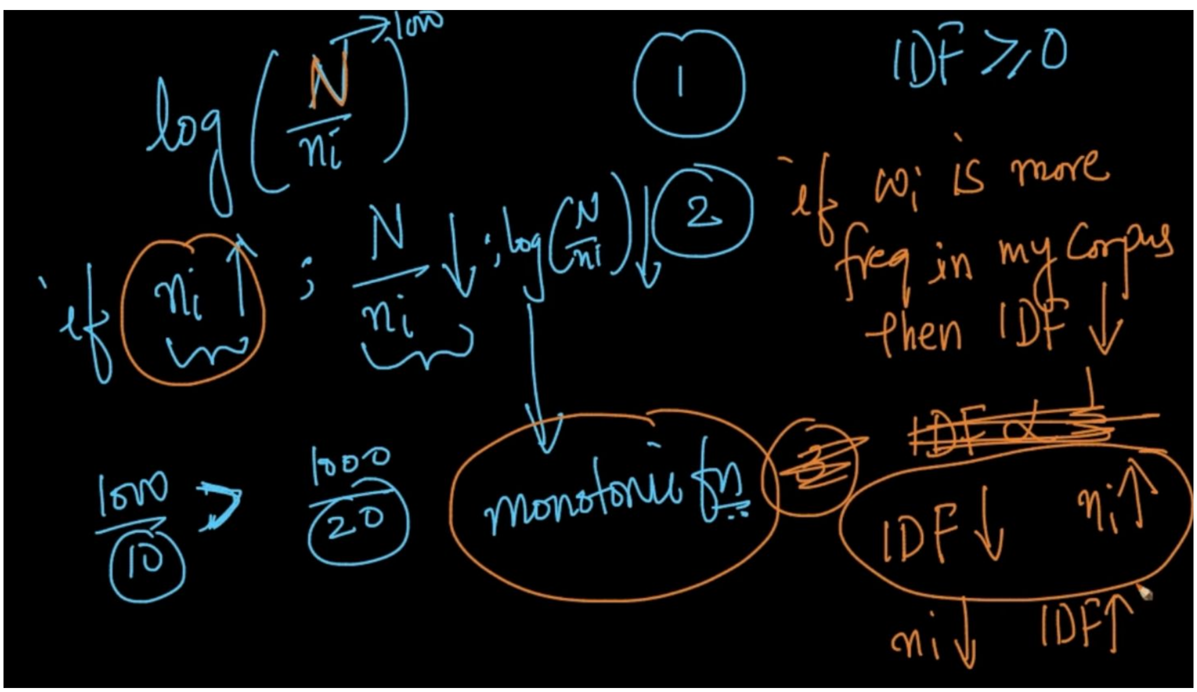
0 < TF(W_i, r_j) < 1 . So, TF is a probabilistic model . Therefore it can be thought as a probability of a word W occurring in text 'r'

IDF



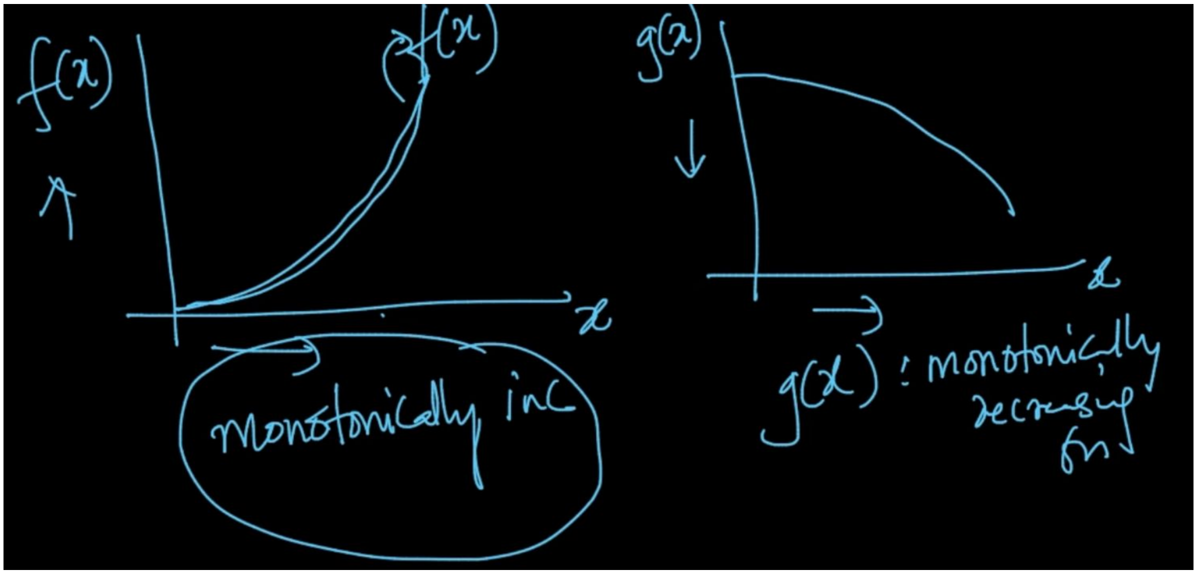
D_c is data corpus i.e collection of all the documents. IDF of a word is log as mentioned above.

As **n** $\leq$ **N** => $\frac{N}{n} \geq 1$ therefore **log(** $\frac{N}{n}$ **)** ≥ 0



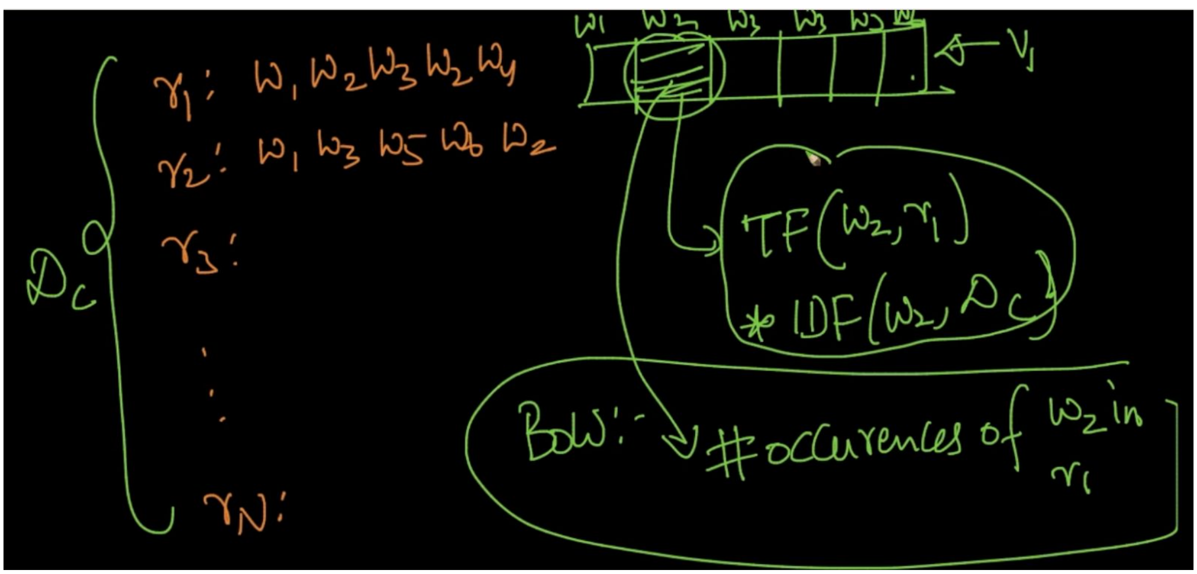
1) As mentioned in the above text IDF ≥ 0

2) ↑ ; $\frac{N}{n}$ ↓; log ($\frac{N}{n}$) ↓ since log is a monotonous function

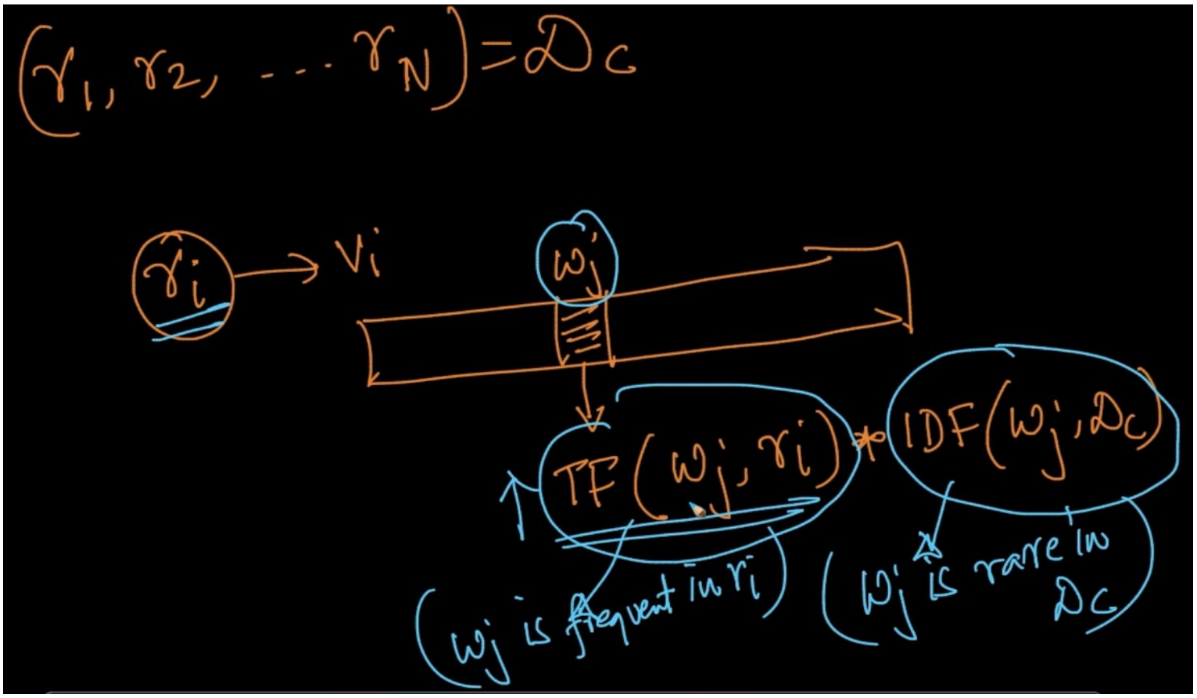


3) IF word W is occuring more than IDF will be low . For ex: the word ‘the’. So ‘the’ has low idf

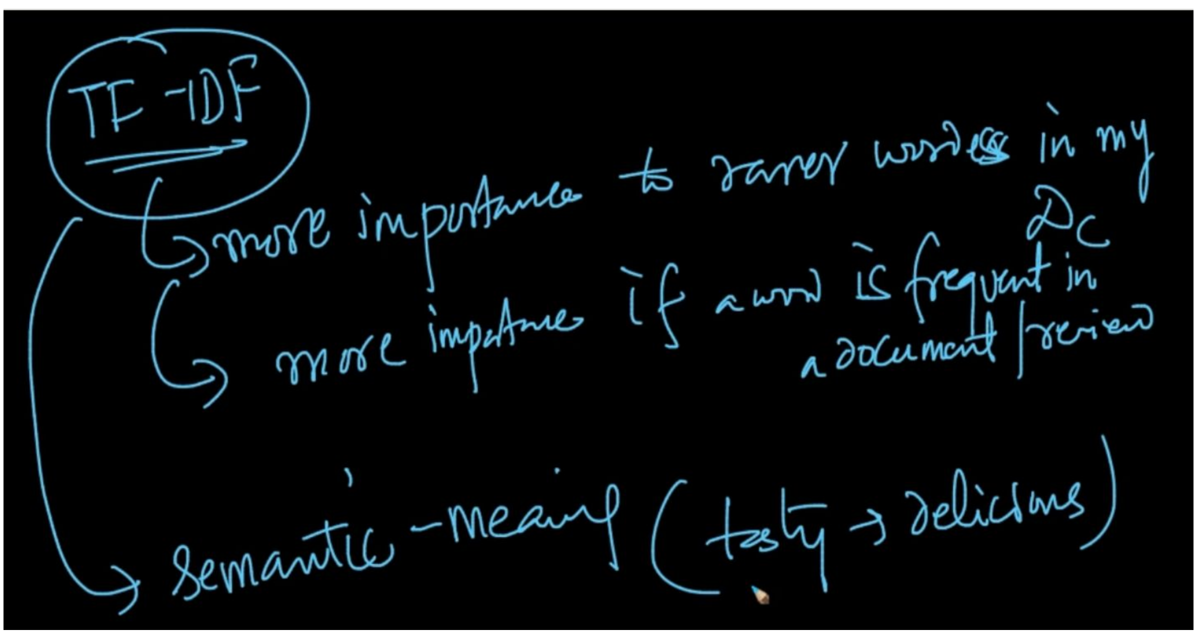
4) IDF ↓ n ↑ . Basically the above 3rd point and n ↓ IDF ↑ i.e if a rare word is there then it’s IDF will be more.



In BOW : We took the occurence of word in the vector but in TF-IDF we multiplty the TF and IDF of word W. Let’s understand the essence of TF-IDF below



So what is happening when TF * IDF . TF says w is more frequent in review r and IDF tells us about the rarity .

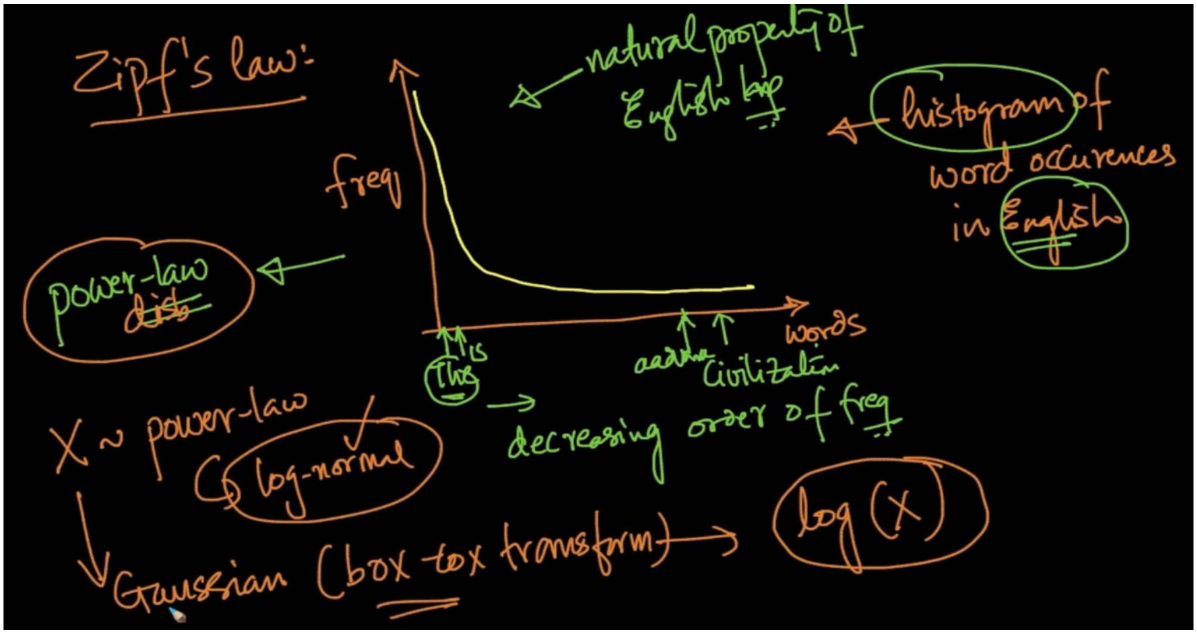


It's importance is mentioned above but semantic meaning is still not covered .

Why use log in IDF?

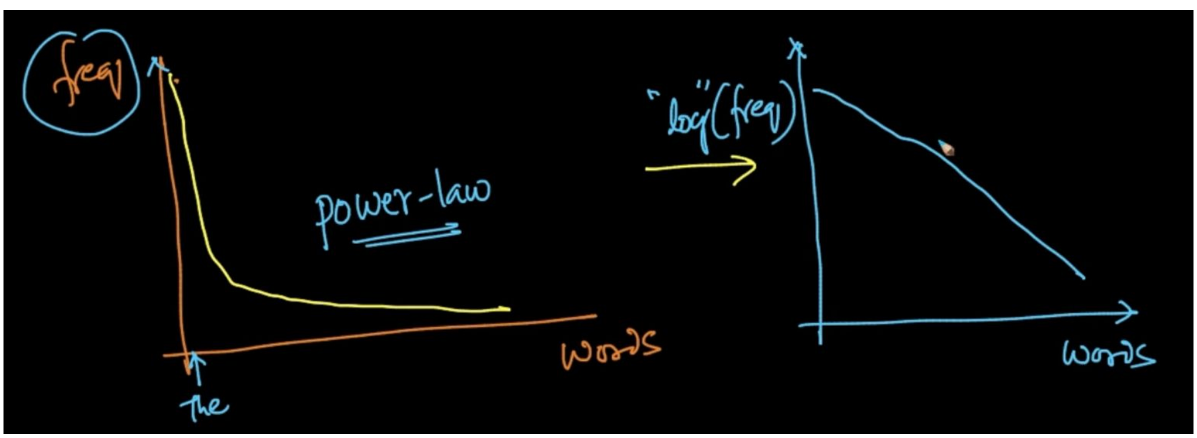
In IDF we know that IDF(w,D) = log(N/n) . Here, N/n made sense because N is no. of total documents and n is no. of docs where the word 'w' occurred but why log.

Let's use Zipf's law to get an understanding of it



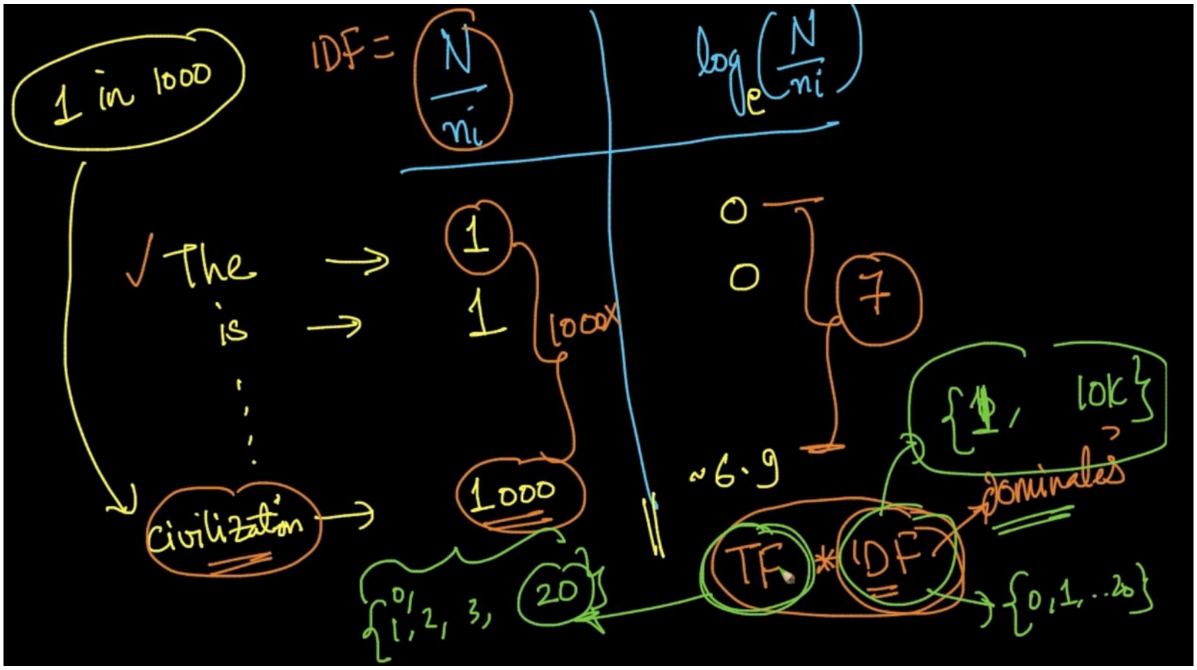
The above is a histogram of words in English let's say Wiki. As seen words like 'the,is, etc' occurs more but words like 'civilization, aardvark etc' has less frequency.

The graph is power-law distribution and we know that if it's power-law then we can convert it to Gaussian by converting the random variable X to log(X).



If we get the log of frequency we get a straight line which is much more manageable.

Let’s get a more practical solution to this

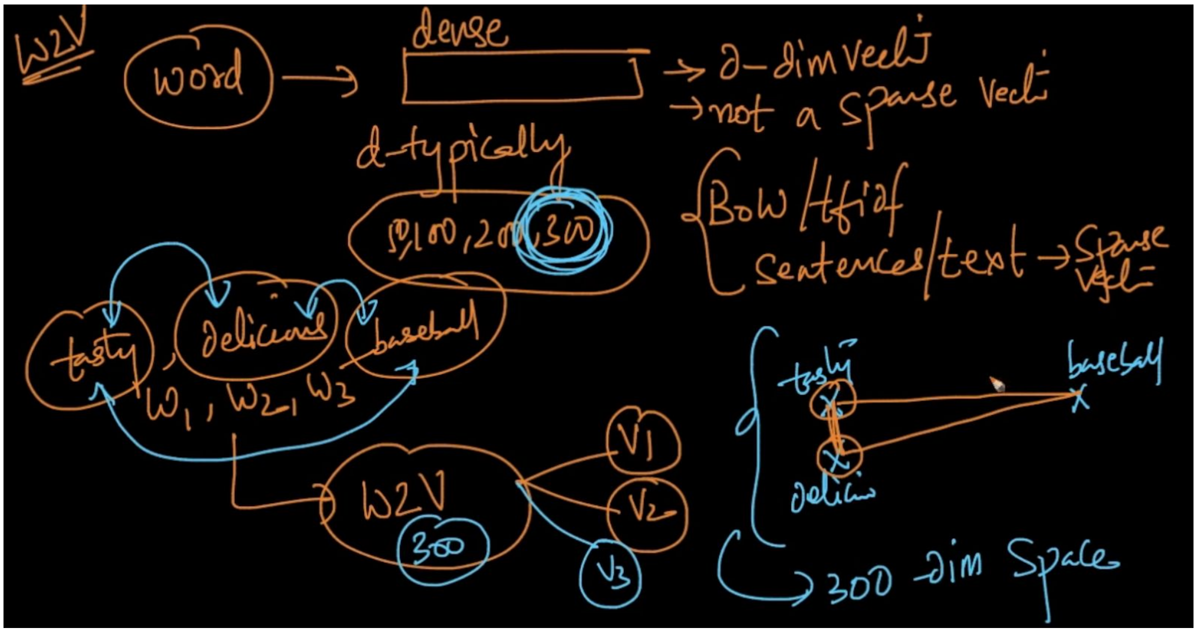


If our IDF was N/n then IDF of *civilization* would be large as seen (1000) and the gap between more common words would be huge but if we are applying log then the gap is reduced to a great extent (from 1000x to 7x) as seen above

In TF*IDF, IDF would’ve dominated because TF is no. of occurences of a word and it wouldn’t be huge for an uncommon word like ‘civilization’ but since IDF of it will be large so TF*IDF wouldn’t have made much sense. So when apply log it gets scaled to a better range and TF *IDF will make more sense.

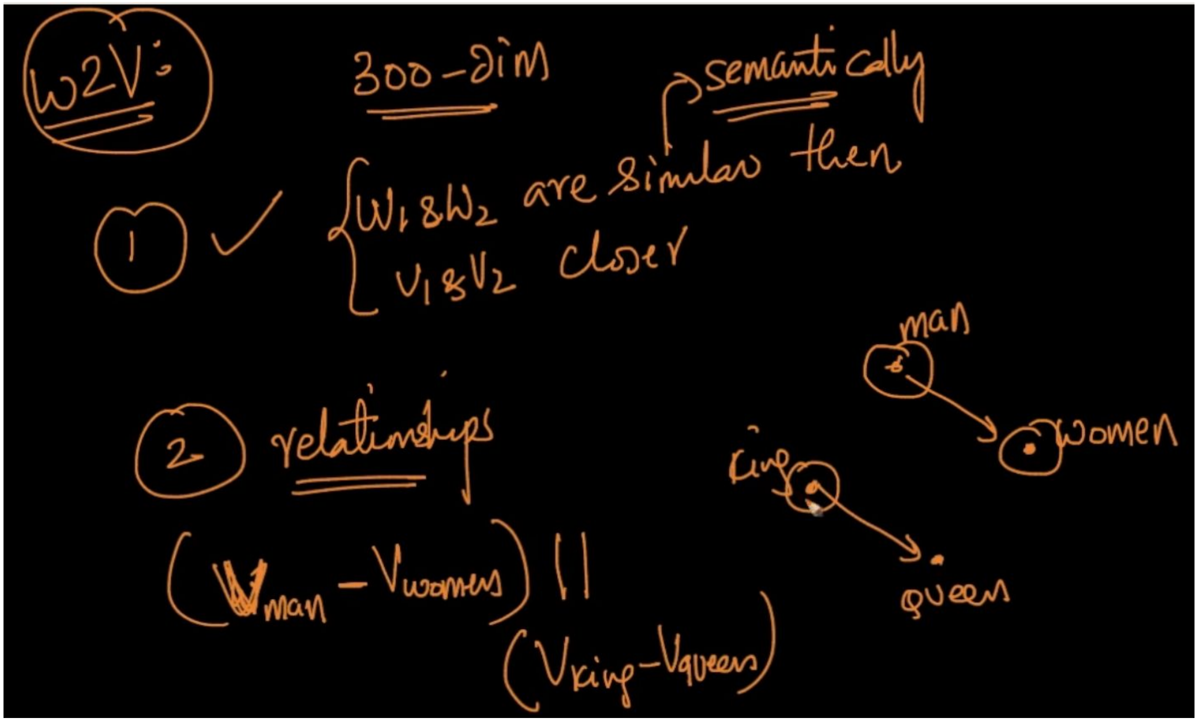
WORD2VEC

In word2vec, we try to get the semantic meaning of the words. As the name suggests it is conversion of a word to a vector.

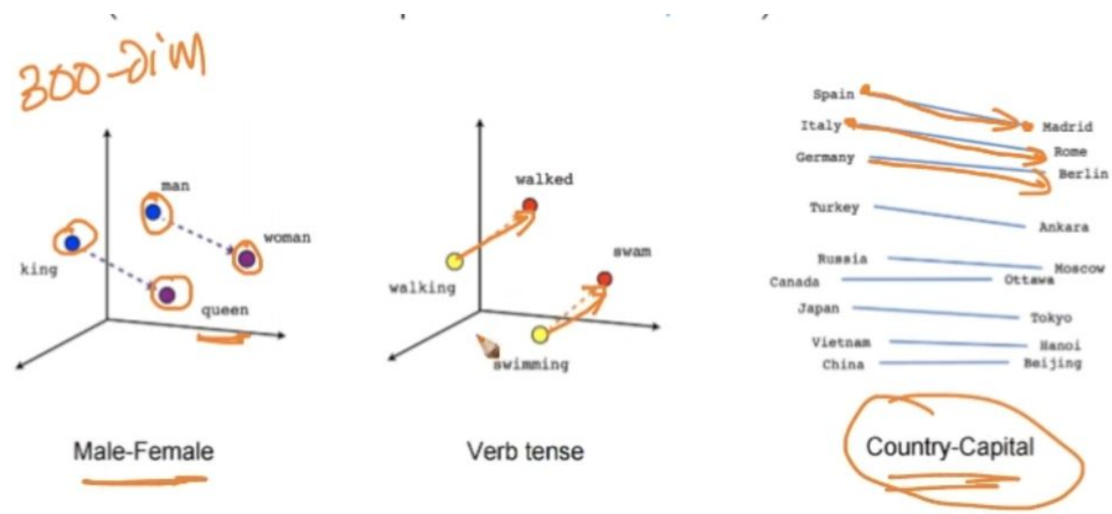


In BOW/tfidf we got a sparse vector of a text but in word2vec it is not sparse. It takes a word and tells us how the dense the word is.

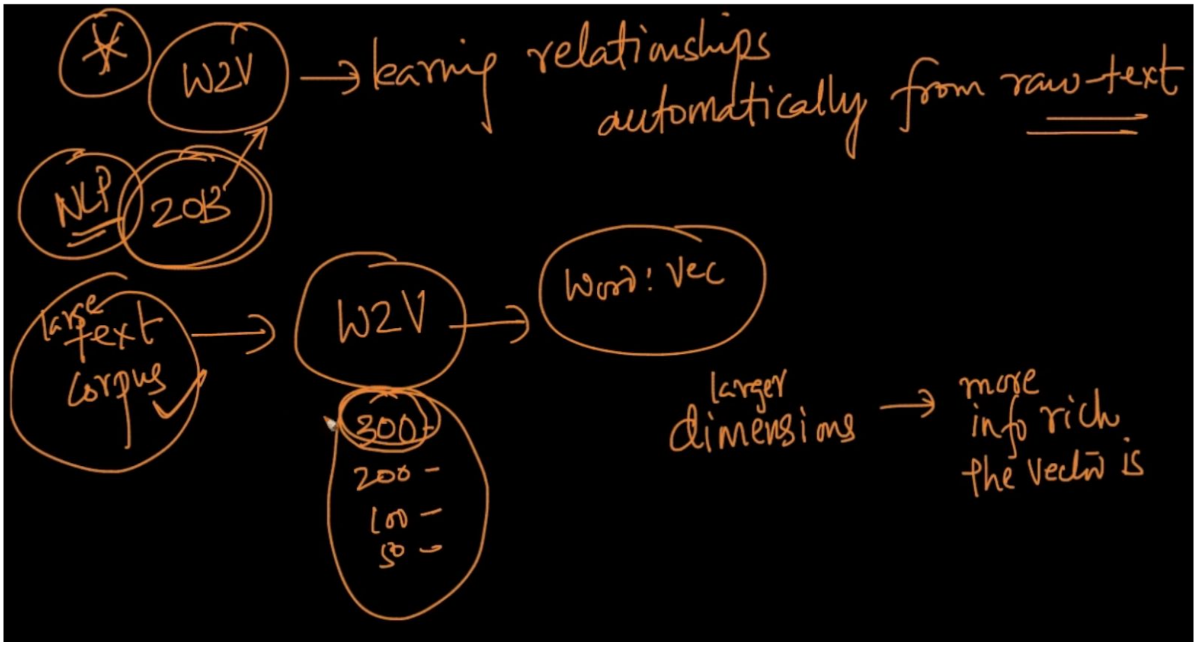
From the above example. We took 3 words *tasty, delicious , baseball* . Suppose word2vec is creating a 300-dimensional vector. So in that vector space *tasty* and *delicious* are close whereas baseball is far away .So word2vec is trying to capture the semantic essence of the words.



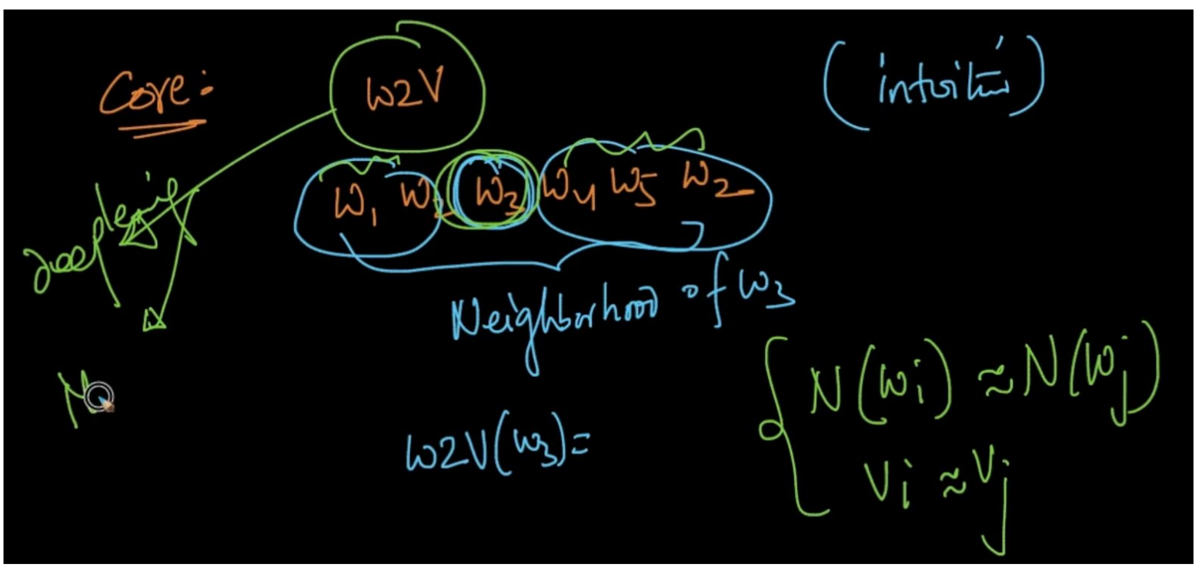
1st point is very clear and in the 2nd point it tells us about the **relationships** of the word. For ex: The words “men” and ‘woman’ and ‘king’ and ‘queen’ the vectors will be parallel as seen above. SO our word2vec is somehow trying to understand the difference between male and female.



It can capture other relationships as well like country-capital . As seen the vectors for country-capital are almost parallel.

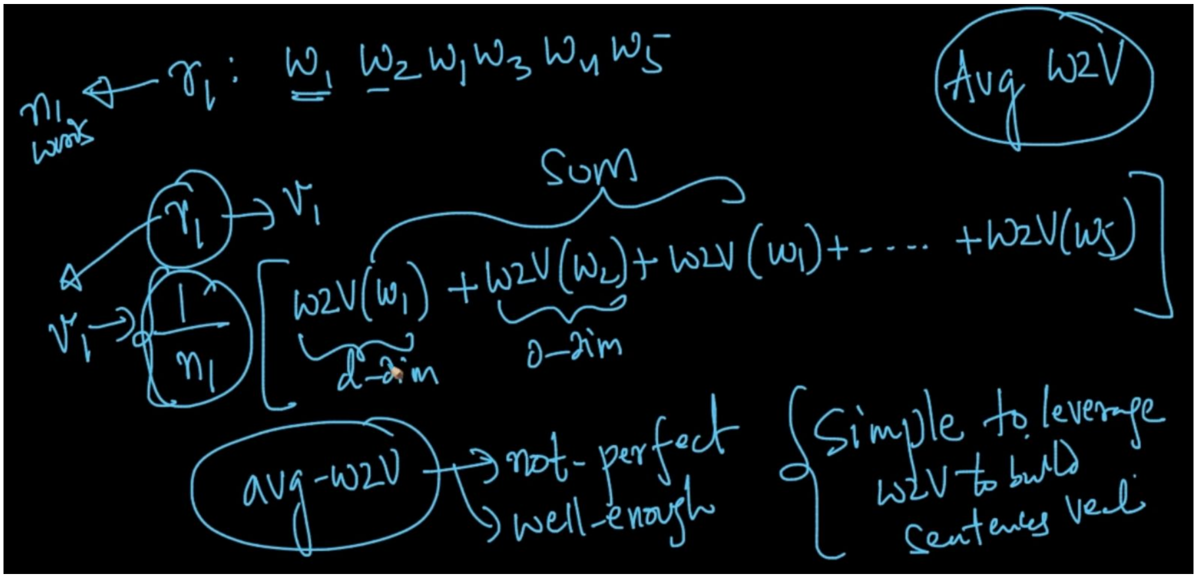


Word2vec automatically learns relationships from text. We give a large text corpus and if large text corpus is given more dimensions will be created and the larger dimensions are the more information vectors we can get .

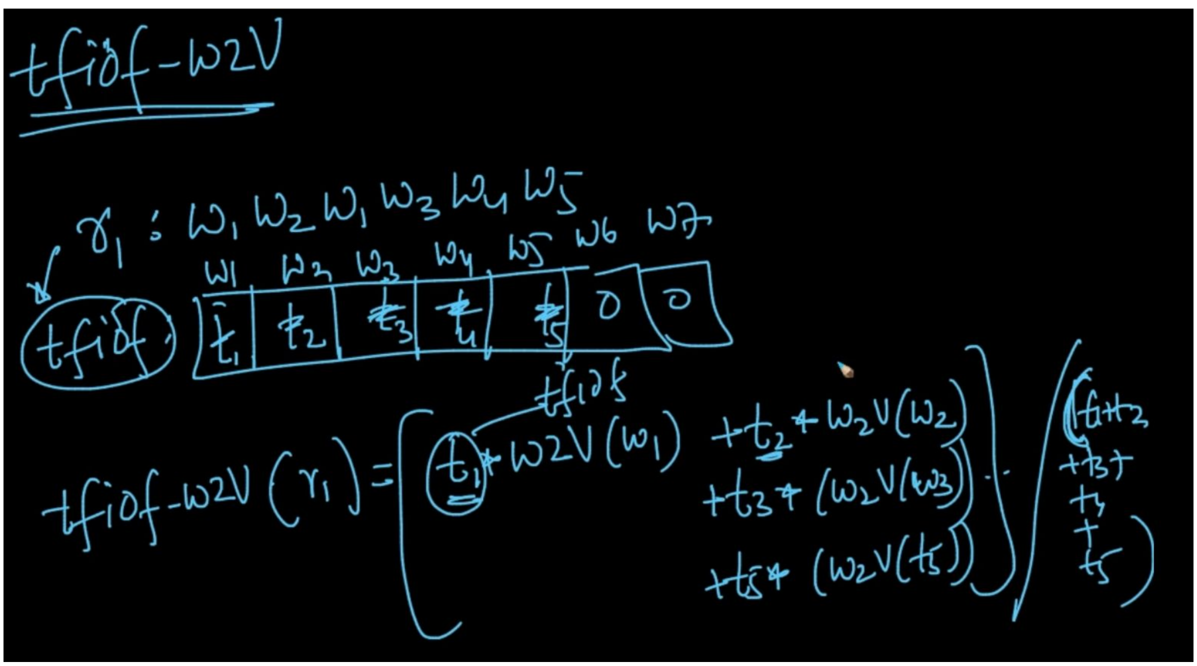


We are taking a word W3 . If we are taking word2vec(W3) it will try to get the surrounding words known as neighborhood. So if the neighborhoods are similar then vectors will be similar as well as seen

Avg-Word2Vec, tf-IDF weighted Word2Vec



Suppose we want a vector of a review we take the w2v(W) of all the words and then take out their averages as seen above



In tfidf-w2v, we get the tfidf of all the words and then w2v . After that we multiply tf-idf * w2v of all the words and take out their average.