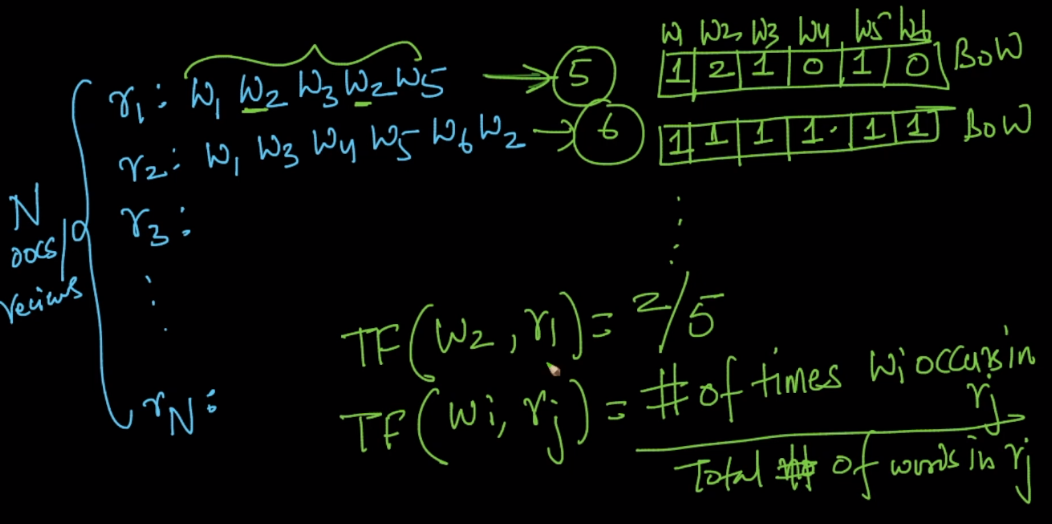


TF is calculated as:

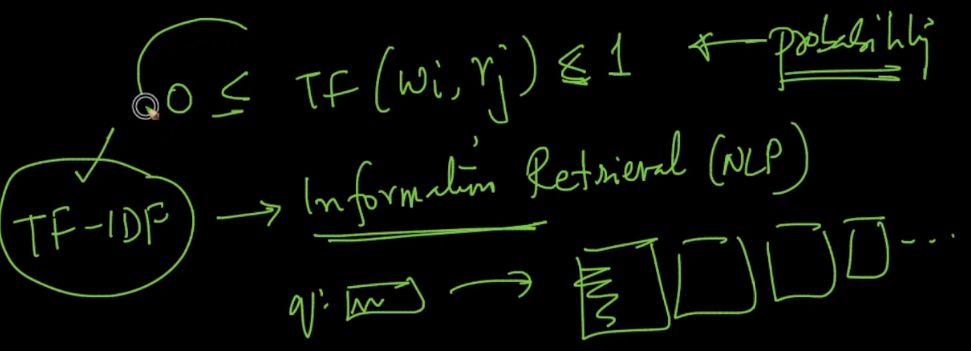
For any review/document let’s say r1, the TF of any word in r1 let’s say w2 will be the:

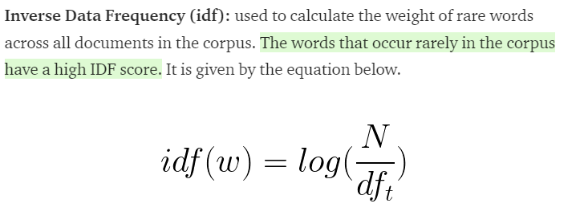
**Number of times w2 occurs in r1 / No of words in r1.**

Therefore TF(w2, r1) = 2/5



The value of TF will always come in between 0 and 1, hence TF can be seen as probability of occuring any word in a document.



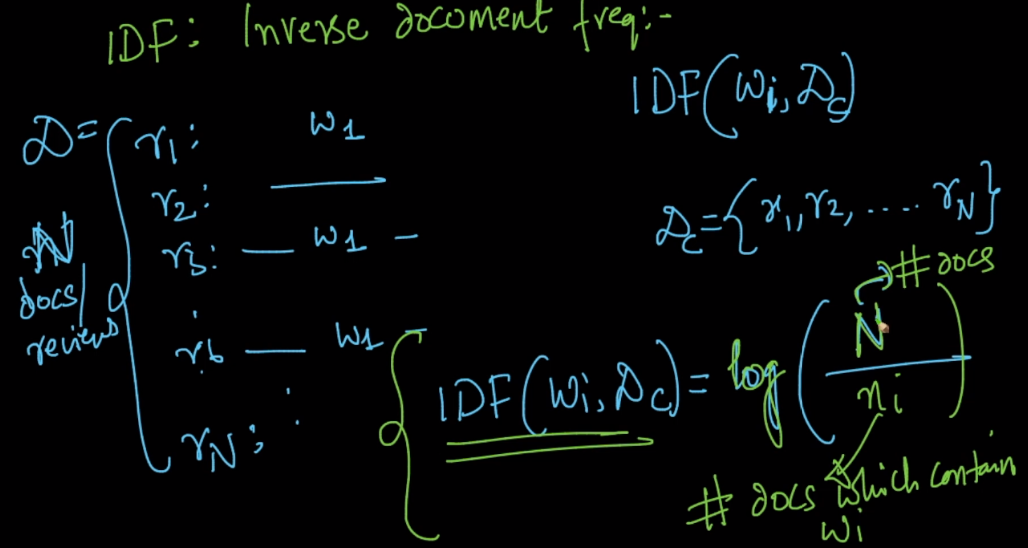


**How IDF is calculated:**

IDF of any word wi in document corpus(all the reviews/documents) will be log( N / ni).

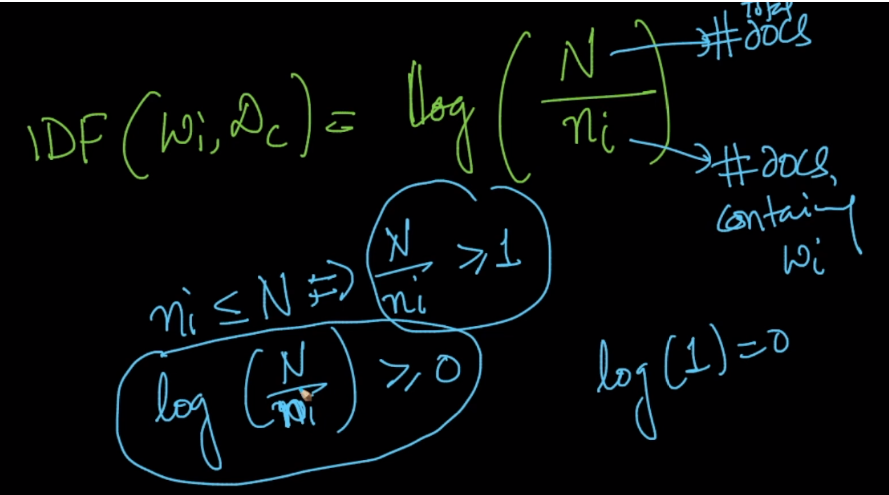
Where N is number of documents

ni is the number of documents which contains wi.



ni is always less than or equal to N, therefore there log will be always >= 0,

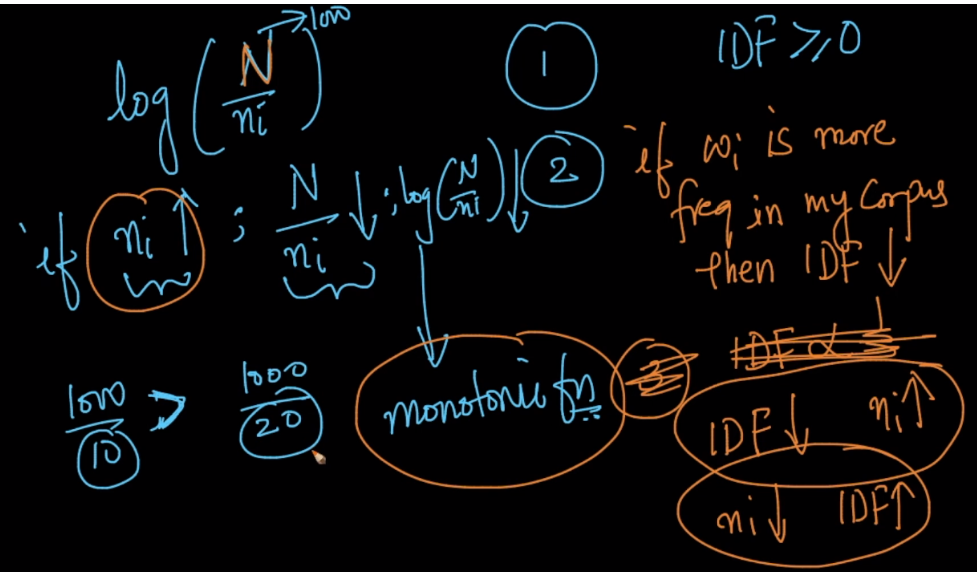
because the extreme case where ni become equal to N and now it’s log(1) will be 0.

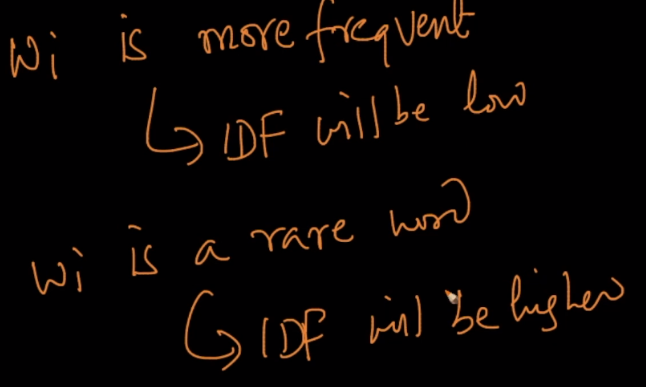


Here thing to note is that when ni increases the N/ ni decreases, and hence IDF will decrease.

Means we can conclude if wi is more frequent in my corpus then IDF will be less,

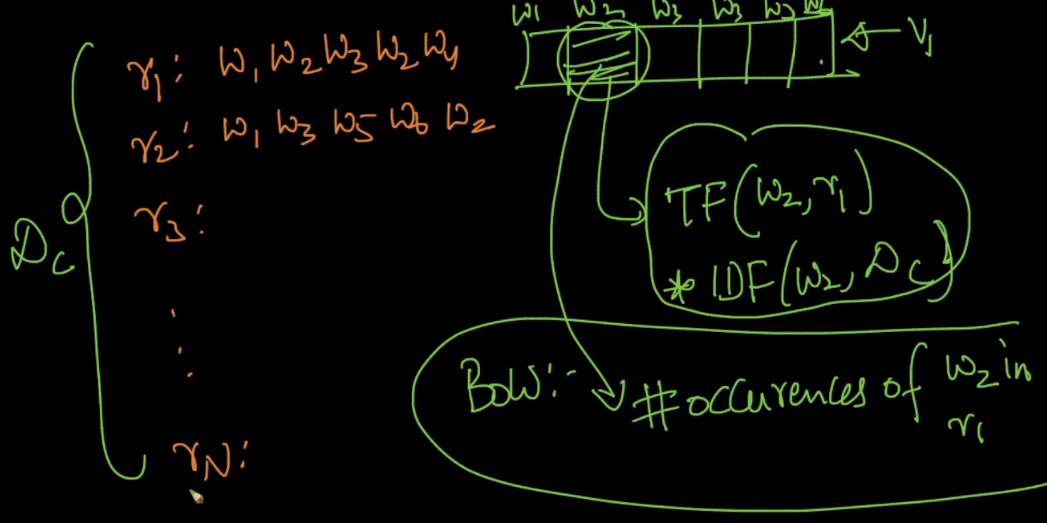
And if wi is less frequent in corpus then IDF will be high.

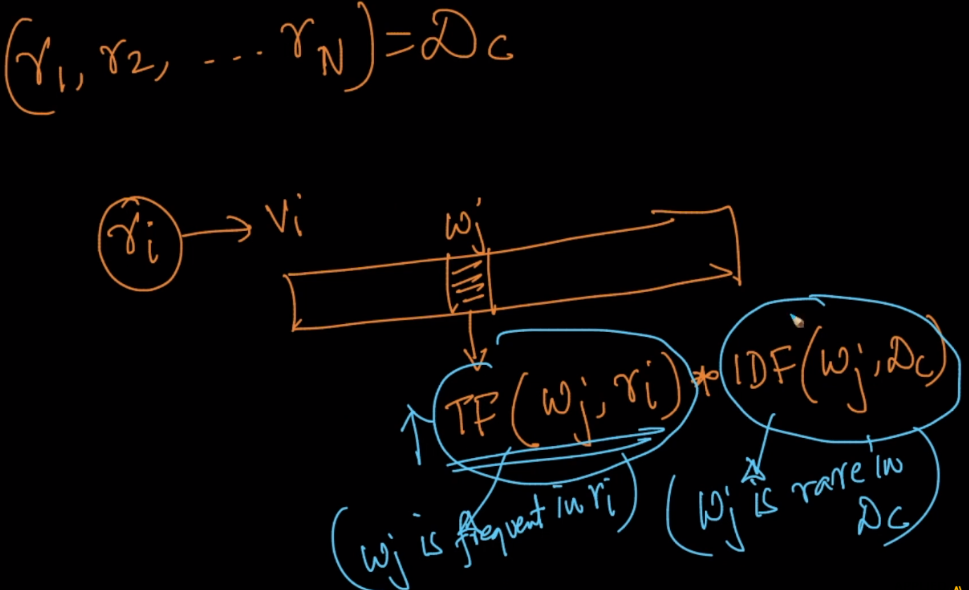




Now For IDF, vector will be generated as, each dimension/word will be:

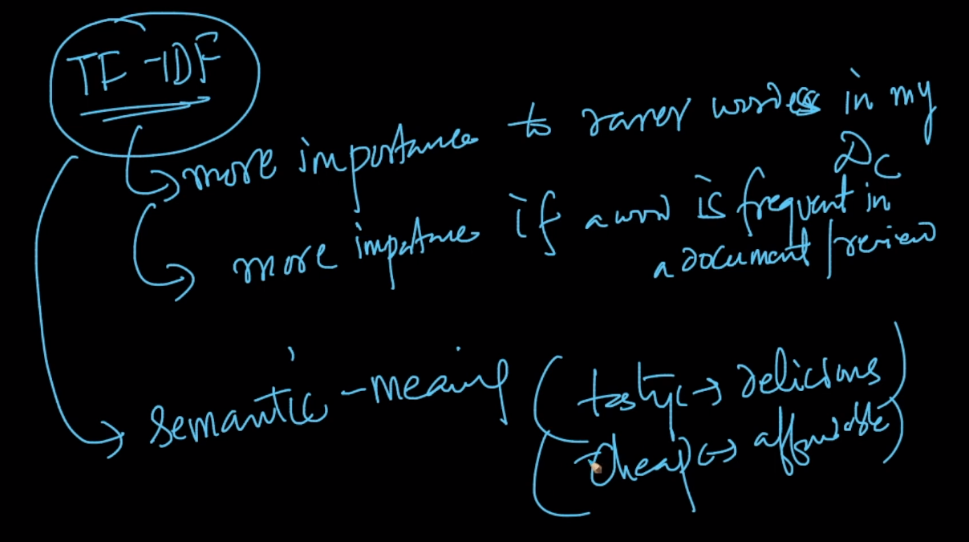
TF(w2, r1) \* IDF(w2, Dc)





Therefore Tf-Idf gives more importance to rarer words and more frequent words.

But still TF-IDF are not able to keep semantic meaning.



Links:

<https://medium.freecodecamp.org/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3>

<https://www.quora.com/What-are-the-best-ways-to-use-TF-IDF-as-a-feature-to-represent-a-string-with-a-lot-of-words>

