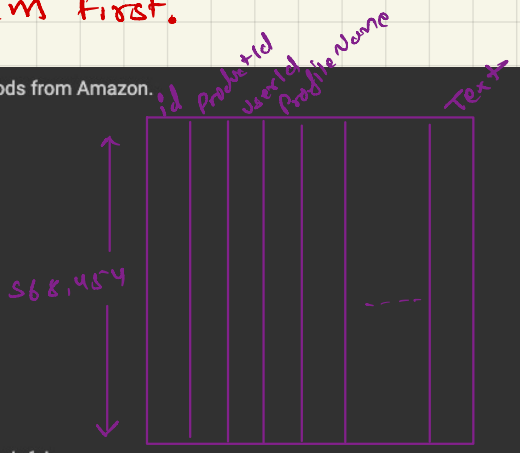
Dataset overview Amazon fine food review (EDA)

→ we must understand the problem first.

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454
Number of users: 256,059 ← No. of distinct users
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10



Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Task :- using this 8 attribute we want to predict our review is '+ve' or Not.

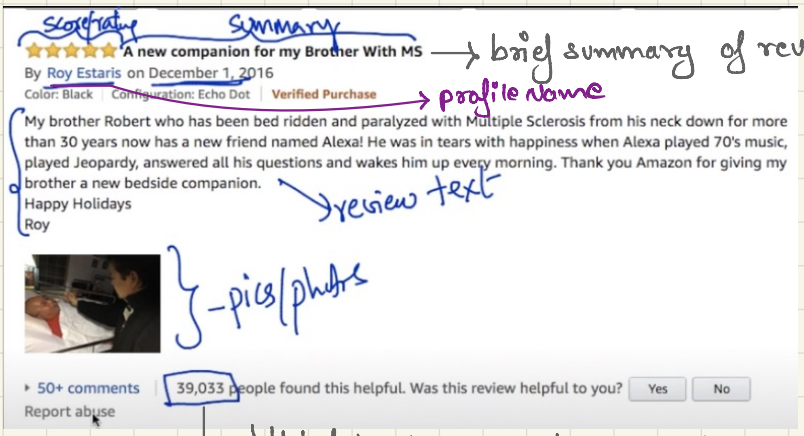
(most useful) → NLP (Natural language processing)

Objective:

Given a review, determine whether the review is positive (Rating of 4 or 5) or negative (rating of 1 or 2). [Not directly given review is '+ve' or Not]

[Q] How to determine if a review is positive or negative?

[Ans] We could use the Score/Rating. A rating of 4 or 5 could be cosnidered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is nuetral and ignored. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.



→ brief summary of review.

assumption :-

If somebody has rated 4 or 5 :→ (high chance) '+ve review'

" " " " 1 or 2 :→ (high chance) '-ve review'

a score of 3 :→ Neutral (don't know '+ve' or '-ve')

→ Helpfulness numerator (Yes)

HelpfulnessDenomenator (Yes + No)

→ "database.sqlite" → storing a database in a lightweight sql format.

Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it easier to query the data and visualise the data efficiently.

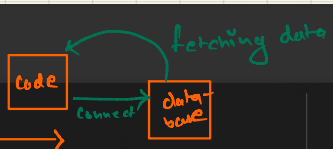
Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score id above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
```

Natural language tool kit

sklearn → ML - libraries in python

[1]. Reading Data




```
# using the SQLite Table to read data.
con = sqlite3.connect('database.sqlite')
#filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 5000""", con)

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rating.
def partition(x):
    if x < 3:
        return 'negative'
    return 'positive'

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

use this connection & run this command.

replaced all my code to positive & negative

```
Number of data points in our data (5000, 10)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	positive	1303862400	Good Quality P
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	negative	1346976000	Not as Advert
				Natalia Corres "Natalia					

Unix time stamp

Data cleaning: Deduplication ⟶ v.v imp.

→ In real world → 20 to 30% time goes to data cleaning & preprocessing.

→ In real world, you will have to look for problem present in data. It takes time.

[2] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

```
[ ] display= pd.read_sql_query("""
    SELECT *
    FROM Reviews
    WHERE Score != 3 AND UserId="AR5J8UI46CURR"
    ORDER BY ProductID
    """, con)
    display.head()
```

→ sort them by product ID

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	S
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUA VANILLA W
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUA VANILLA W
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUA VANILLA W
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUA VANILLA W
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUA VANILLA W

→ Redundancy of data. we want to eliminate it.

→ Removing duplication.

As can be seen above the same user has multiple reviews of the with the same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

+ Code + Text

```
[ ] #Sorting data according to ProductId in ascending order
    sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
```

```
[ ] #Deduplication of entries
    final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
    final.shape
```

→ search for this func

```
(4986, 10)
```

(364173, 10)

```
[ ] #Checking to see how much % of data still remains
    (final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
69.72
```

* Cleaning data is mostly on "gut feeling". → practice.

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

```
[ ] display= pd.read_sql_query("""
    SELECT *
    FROM Reviews
    WHERE Score != 3 AND Id=44737 OR Id=64422
    ORDER BY ProductID
    """, con)

    display.head()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	> 1	5	1224892800	Bought This for M
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	> 2	4	1212883200	Pure cocoa t

→ error

```
[ ] final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```
→ only keep Helpfulness Num. < = Helpful den.

```
[ ] #Before starting the next phase of preprocessing lets see the number of entries left
    print(final.shape)

    #How many positive and negative reviews are present in our dataset?
    final['Score'].value_counts()
```

```
(4986, 10)
1    4178
0     808
Name: Score, dtype: int64
```

positive 307061 → more 'tve'
negative 57110 → less '-ve'

Why Convert text to a vector

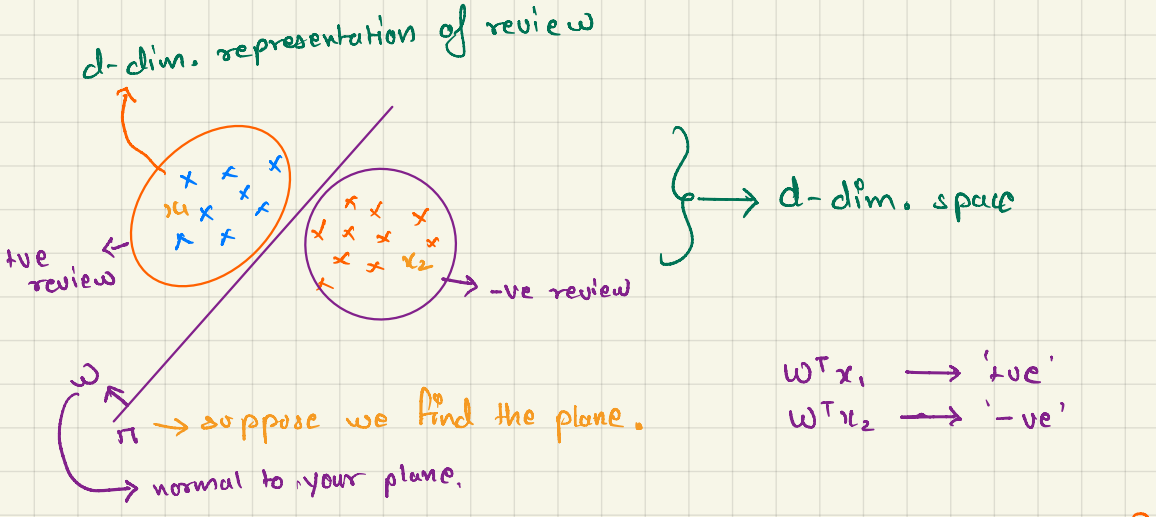
→ from a feature we will have to predict → sentiment polarity (positive/Negative)

→ In above data two of the imp. parameters are → Summary
→ Text.

Q) How can we understand these text using computer?

→ using vector we can use all mathematical power.

Q) How do you convert text (words & sentences) into numerical vector?



if $w^T x_i > 0$ then 'x_i' is 'tve'
else 'x_i' is '-ve'

Task :- review - text ⟶ d - dim. vector ⟶ finding a plane to separate

Q) text ⟶ d-dim vector

→ rules :-

r_1 , r_2 , r_3

→ if r_1 & r_2 are more similar than r_1 & r_3. [Here I mean similarity in English.]

review 1.

→ if English sim (r_1, r_2) > English sim (r_1, r_3)

then

dist (v_1, v_2) < dist (v_1, v_3)

v_1 :- vector 1 for review 1.

∴ if r_1 & r_2 are more similar than v_1 & v_2 must be close.

v_3 } → d-dim. space.

if Eng sim (v_1, v_2) > Eng sim (v_1, v_3)

then

length $(v_1 - v_2)$ < length $(v_1 - v_3)$ } → if that the case the it will much more easier for me to find a plane that separates '+ve' & '-ve' revi.

Q) How to go from "text" ⟶ "d-dim vector", such that similar text must be close geometrically ?

Techniques to do that →

- BOW
- tf - idf
- w2v
- avg w2v
- tf-idf - w2v

Bag of words (Text to vec)

r_1: This pasta is very tasty and affordable
r_2: This pasta is not tasty and is affordable
r_3: This pasta is delicious and cheap.
r_4: Pasta is tasty and pasta tastes good.

→ set of all reviews or documents called as "corpus".

BOW:-

① Constructing a dictionary :- set of all unique words in your reviews.

↳ { This, pasta, is, very . . . . . } (d- unique words)

→ here, I have assume 'n' reviews (or 'n' documents). and in this d- unique words are present.

⇒ r_1: This pasta is very tasty and affordable.

vector

	1	2	3	4	5	. . .						d-1	d
V_1 :	0	0	0	1				1				1	
	a	an	the	pasta				This				tasty	

$V_1 \in R^d$

* each word is a different dimension.

d → is large

→ each cell here, telling # of word occur in 'r_1'.

V_1 → 'sparse vector' → most of the ele. in a vector is zero.

→ The above approach is BOW. (text → vector)

→ similar text/review/document must result in vectors which are closeby.

⇒

r_1: This pasta is very tasty and affordable

r_2: This pasta is not tasty and is affordable

	this	pasta	is	very	tasty	not	and	affordable
V_1 :	1	1	1	1	1	0	1	1
V_2 :	1	1	2	0	1	1	1	1
	0	0	1	1				

→ # of occurences.

length $(V_1 - V_2) = |V_1 - V_2| = \sqrt{1^2 + 1^2 + 1^2} = \sqrt{3}$ → it is very low.

↳ the 'r_1' & 'r_2' are completly different in similarity. But, the distance is very less.

Note:- Bow does not work very well if there is small changes in words.

Binary Bow or Boolean Bow :- does word occur → 1, if not → 0 } → not count no. of occurences

But, :- $||v_1 - v_2|| = ||v_1 - v_2|| = \sqrt{\text{no. of different words}}$

↓ # occurence BOW ; ↳ Binary BOW

Q) How to make 'd' small ? (no. of words in dictionary)

Text - preprocessing

STOP-word removal, Tokenization, Lemmatization (text → vectors)

→ Improving 'BOW'.

r_1 : This pasta is very tasty and affordable (stop words (does not matter much).)

r_2 : This pasta is not tasty and is affordable

r_3 : This pasta is delicious and cheap.

r_4 : Pasta is tasty and pasta tastes good.

→ If we remove 'stop words' from documents our BOW will be smaller and still meaningful.

↳ from nltk.corpus import stopwords

These are all stop words in english. →

```
{'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'}
```

But, 'Not' is also considered as a stop-word in english.

If so, then our, v_1 & v_2 or (r_1 & r_2) will totally similar.

But, in reality they are quiet opposite. This is one of the problem in removing stop-words. And sometimes, removing stop-words is not good like in above example.

Text-preprocessing :- i) remove stop-words

ii) convert everything in lower case. [Pasta ⟷ pasta]

iii) stemming

iv) Lemmitization

Stemming :-

taste ⟷ tasty ⟷ tastful :- all are coming from word 'taste'.

- It would be great if I can convert all three of them into their common word. This is called stemming.

  → There are multiple algorithm to do stemming.

    → porter stemmer
    → snow ball stemmer

iv) Lemmitization :- breaking a sentence into words.

→ This | pasta | is | very | tasty | . | This | is | the | best | in | New York | .

We don't want to break it. Because, it will change the whole meaning. So, we use Lemmitizers which will group it into one-word. (They have dictionary of grouped words)

Drawbacks

v) One problem in BOW :- Here, we are not taking semantic meaning of words.

Bow :-

tasty ⟷ delicious ⟶ are very similar in meaning.

Uni-gram / Bi-gram / n-grams

r_1 : This pasta is very tasty and affordable

r_2 : This pasta is not tasty and is affordable

→ after removing stopwords ; V_1 & V_2 are exactly the same. So, r_1 & r_2 are very similar.

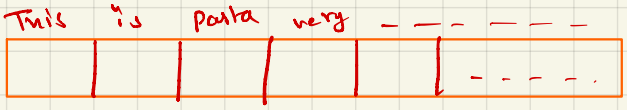
⟶ How to solve that problem?

r_1 : This pasta is very tasty and affordable

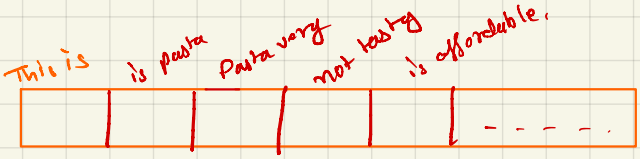
r_2 : This pasta is not tasty and is affordable

] → these one will be different in bi-gram models.

Unigram :- each word is considered a dim.



Bi-gram :- two - words considered a dim.



Trigram :- 3 consecutive words.

→ Uni-gram BOW :- discards the sequence information.

→ Bi-gram BOW :- retain some of the sequence information.

But, there is also problem

⇒ # tri-grams ≥ # bi-grams ≥ # uni-grams

here 'd' value is always ≥ 'd' value of unigrams.
(Think of dictionary dimension)
(here, the uniqueness of ele. will be less).

https://medium.com/mti-technology/n-gram-language-model-b7c2fc322799

read this blog. A very good blog to understand the uni-gram, bi-gram & n-gram.

TF - IDF (Term frequency - inverse document frequency)

N docs/ review

r_1 : w_1 w_2 w_3 w_2 w_5
r_2 : w_1 w_3 w_4 w_5 w_6 w_2
r_3 :
r_4 :
⋮
r_N :

BOW → v_1 :

w_1	w_2	w_3	w_4	w_5	w_6
1	2	1	0	1	0

BOW → v_2 :

w_1	w_2	w_3	w_4	w_5	w_6
1	1	1	1	1	1

→ how often does 'w_i' occur in 'r_j'

$$TF(w_i, r_j) = \frac{\#\text{ of times '}w_i\text{' occur in '}r_j\text{'}}{\text{Total \# of words in '}r_j\text{'}}$$

$$TF(w_2, r_1) = \frac{2}{5}$$

$0 \leq TF \leq 1$ → probability

TF-IDF → Information Retrieval (NLP) ⇝ finding word in search engine

IDF :-

D_c = N docs/ review ← document corpus

r_1 : w_1
r_2 : —
r_3 : w_1
r_4 : —
⋮
r_N : w_1 —

$IDF(w_i, D_c)$

$D_c = \{r_1, r_2, r_3 \dots r_N\}$

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

N → # of documents

n_i → # of documents that contain 'w_i'.

→ here, $n_i \leq N$, so, $\frac{N}{n_i} \geq 1$ ∴ $\log\left(\frac{N}{n_i}\right) \geq 0$

→ if $n_i \uparrow$ then $\frac{N}{n_i} \downarrow$ then $\log\left(\frac{N}{n_i}\right) \downarrow$ so, IDF ↓

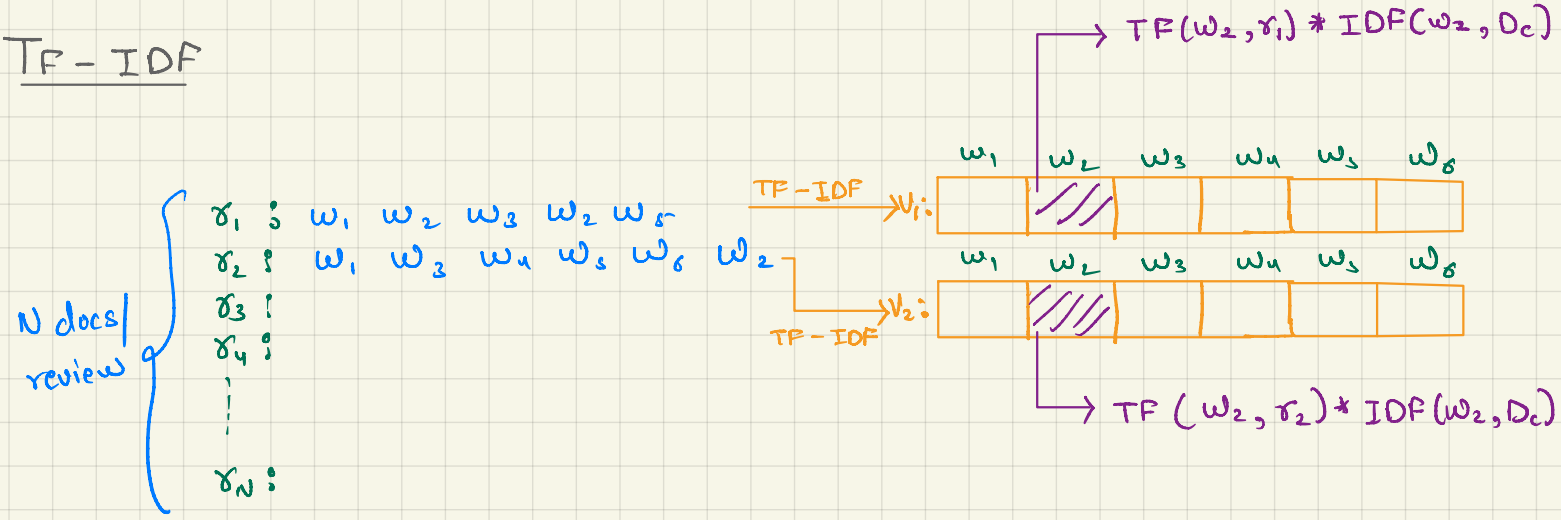
coz log is monotonic inc^n fun^n.

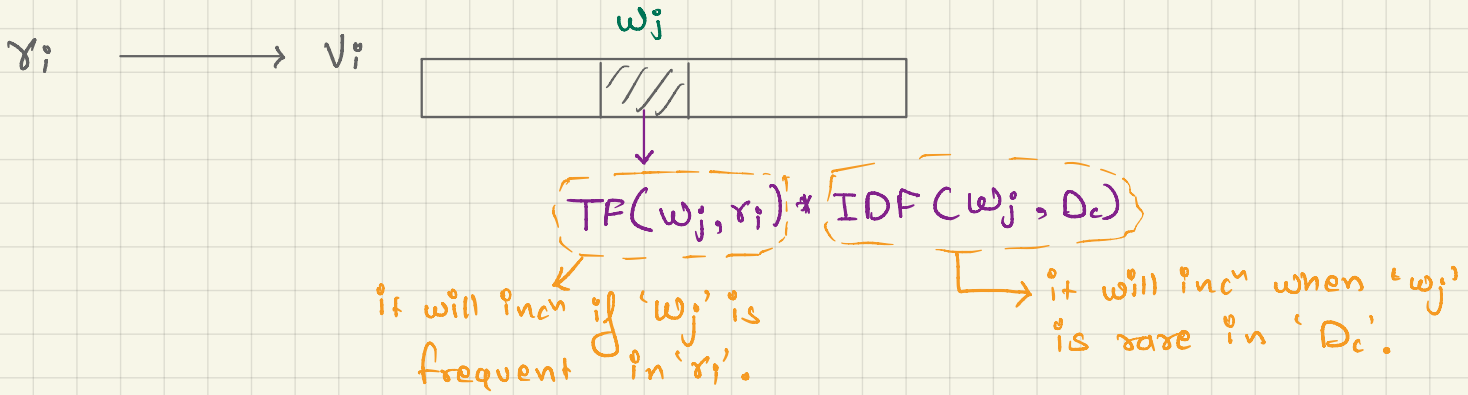
If 'w_i' is more frequent in my corpus then IDF will be lower.

so, when $n_i \uparrow$ then IDF $\downarrow$ (frequent word) [eg:- 'The']

when $n_i \downarrow$ then IDF $\uparrow$ (rare word)

TF - IDF





TF - IDF

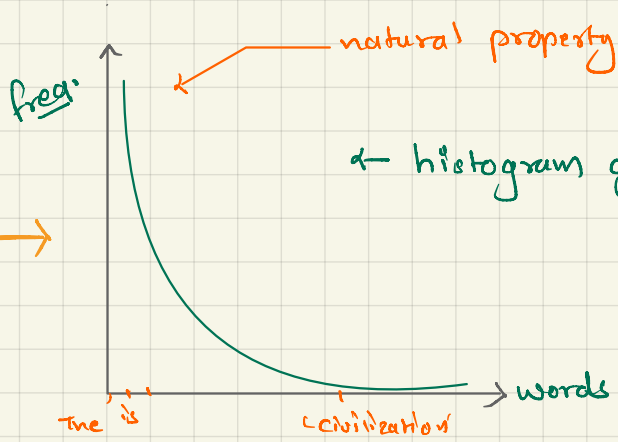
- → It gives more importance to 'rarer' words in my 'D_c'.
- → It gives more importance if the word is frequent in a review or document.
- limitation → It also does not deal with semantic-meaning words. (same like BOW)
  eg:- Tasty ↔ delicious.

Why do we use $\log\left(\frac{N}{n_i}\right)$ for IDF?

→ 1972 research paper

→ heurastic (or) hack → not very strongly on theory :- Ziff's law

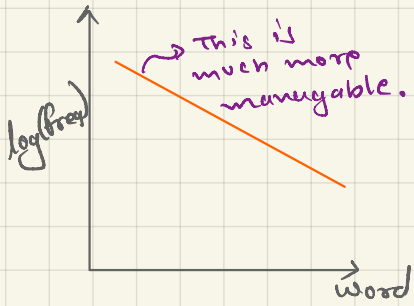
Zipf's law :-



natural property of English lang.

← histogram of word occurences in English.

Good example of 'Power law' →



This is much more manageable.

$X \sim$ PowerLaw

↳ log-normal

↳ Gaussian (box-cox transform) → $\log(x)$

2nd explanation :-

	$\frac{N}{n_i}$	$\log\left(\frac{N}{n_i}\right)$
The →	1	0
is → (1000x)	1	0
⋮		
civilization →	1000	6.9

7x (times)

→ TF * IDF

If we do not consider 'log' then IDF dominates.