☆ Module : 3

→ How classification works :-

→ Amazon Fine Food Reviews Analysis:

No. of review : 568,454
No. of users : 256,059
No. of products: 74,258
Time span : Oct 1999 - Oct 2012
No. of Attributes / Columns in data : 10

Attribute Information:

1. Id
2. product Id
3. User Id
4. profile Name
5. HelfulnessNumerator
6. Helfulness Denominator
7. Score (rating)
8. Time
9. Text :

Objective : Given a review, determine whether the review is positive (4,5 rating) or negative (1 or 2)

If we have rating we can predict it using simple If-else but we need to predict the review without score(rating)

☆ Data cleaning : Deduplication

It is observed that the reviews data had many duplicate entries. Hence It was necessary to remove duplicates in order to get unbiased results for the analysis

#code :- sorted_data = filtered_data.sort_values('ProductId', axis=0);

final = sorted_data.drop_duplicates(subset={"UserId","ProfileName","Text","Time"}, keep="first", inplace=False);

* Text to vector

the problem we had is given a feature (without rating) we need to determine polarity of review. The most useful feature is text feature and summary. Both of them are basically text. If we can somehow convert text to vector we can use the power of linear algebra.

Q) How do you convert ~~sin~~ text into vector.

→ Bag of words:-

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 looks good.

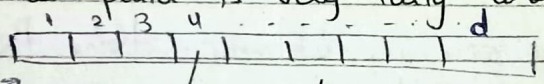
In BOW:

step ① : Constructing a dictionary :- set of all the words in your review:

let we have d-unique words. ← { This, pasta, is, very, tasty, and, affordable, not, delicious, cheap, go

②

x_1 : This pasta is very tasty and affordable.

V_1 : 

Vector of size d

↳ each index corresponds to a word in new dictionary.

each entry keeps the freq of that word occured in review.

V_1 will be a sparse approach because there may be huge words in a dictionary, but a sentence can have 10, 15, 80, 100 words.

Each word in BOW is a different dimension.

We can improve BOW using some technique.

look at the R_1, we can see that there are various useless words, these words are called stop word.

like in R_1 { is, and, this } these are stop words.

If we remove them from dictionary then BOW dictionary will be smaller and we have only important words in vector.

• Note: Some time removing stop words can cause problems. Hence stop word removal is not a golden rule.

• → Another thing is, we should make all the words in lower case letter. because we can have 2 diff words in dictionary i.e Pasta, pasta are same but they can be counted 2.

→ stemming:- there are some words which signifies similar meaning like tasty, tastefull (both are talking about the parent word taste) so, we can use only one representation.
There are various stemmer: (i) Porterstemmer
(ii) Snowballstemmer **

→ Lemmitization:- Lemmitization is all about breaking up the sentences into words.

Limi of BOW:- we are not considering semantic ~~relation~~ meaning of words. like tasty and delicious are treated as 2 diff words and hence diff dimension in BOW but tasty and delicious are very much similar.

→ Uni-gram | Bi-gram | n-gram:-
When we are leaving stop words:- we saw various stop words.

r_1 : This pasta is very tasty and affordable
r_2 : This pasta is not tasty and is affor

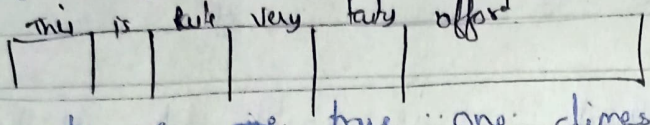
↓ removing stop words

r_1 : pasta tasty afforda
r_2 : pasta tasty afford

After removal of stop words $r_1 == r_2$ but in original Review they are opposite. How to tackle this problem.

we use uni-gram / bigram to handle this problem.

Uni-gram:

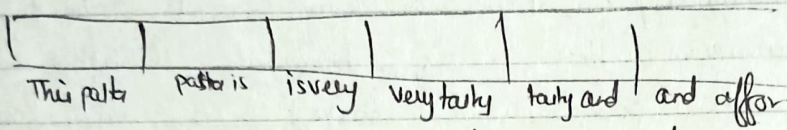


In uni-gram we have one dimension for each word.

bi-gram: we have dimension for 2 consecutive words in sentence

ex: This pasta is very tasty and affords

bigram:



Similarly we can have n-gram (n - consecutive words for each dimension)

So, Bigram or higher gram retains the sequential information

But the no. of bigrams > no. of unigrams. hence it increases the vocab vector size.

Hence n-gram → dimensionality 'd' increases.
(n>1)

☆ TF-IDF :- It is a variation of BOW

TF-IDF :- Term freq - Inverse document freq.

N documents / sentences

r_1 : w_1 w_2 w_3 w_2 w_5 → 5 words

r_2 : w_1 w_3 w_4 w_6 : w_2-w_5 → 6 words.

r_3

r_N:

$$TF(w_i, r_j) = \frac{\#\text{ of time } w_i \text{ occurs in } r_j}{\text{Total \# of words in } r_j}$$

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

$0 \le TF(w_i, r_j) \le 1$

Basically TF says what is the probability of finding w_i in document r_j

IDF => inverse document frequency.

So we have N documents / reviews

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

D: {r1, r2, r3 ... rN} D is collection of all documents / reviews.

N → Total No. of documents

n_i → No. of documents which contain word i.

we know that N is always greater than n_i hence

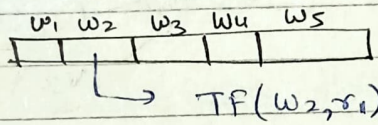
$N/n_i > 1$

$N/n_i = 1$ { if word w_i occurs in every document }

the IDF of that word = 0.

So, if a word w_i is more frequent its IDF will be lower.

r_1: w_1 w_2 w_3 w_2 w_4

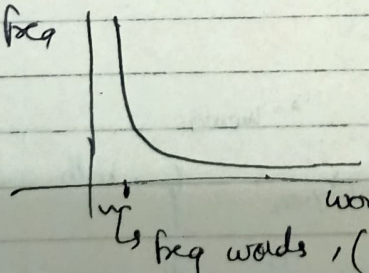


each word is represented as its TF * IDF value.

TF-IDF → give more imp to word rare in whole Docum but frequent in current document.

The reason behind the ~~log~~ useage of log in IDF is zipfs law

zipfs law



zipfs law state that most part of sentence is covered by small set of words :-

we can see that above plot is following Power-law

In power-law we saw that to convert a power-law to gaussian distribution we took log of X.

Also applying log make the range of IDF smaller.

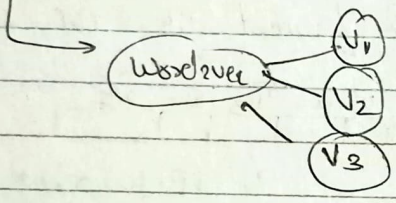
If IDf is very large then IDf will dominate ove TF. to balance then we apply log to IDf to make it small.

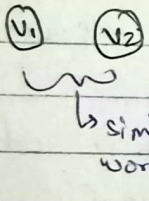
* Word2Vec :- This take semantic meaning into consideration. We will see full mathematics of word2vec in dapp learn. But now we only see it as black box.

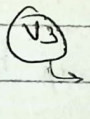
word → [ ] → d-dim. vector
→ not a sparse vector

Note of BOW, TF-IDF, takes sentence at a time, word2vec give vector for each word.

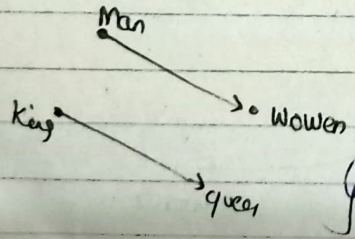
tasty delicious
w_1, w_2, w_3 → baseball



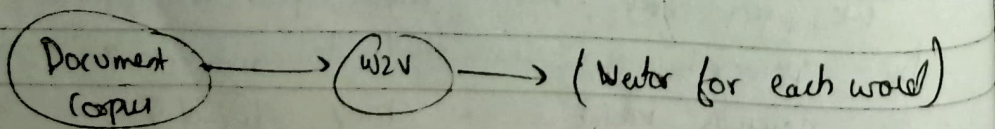




① * W2V : * Take semantic meaning into consider.
② It take retionship into consideration. (Ex. Man, woven King, Queen.



~~dem~~ Similarly it capture various other type of relationship.
like country-capital.



→ Avg w2v , tf-idf w2v :-

w2v gives vector for each word, but our sentences are collection of words. so how do I convert a sentence into vector using w2v.

let r_1 : w_1 w_2 w_1 w_3 w_4 w_5

$$\frac{w2v(w_1) + w2v(w_2) + w_2v(w_1) + w2v(w_3) + w2v(w_4) + w_2v(w_5)}{\#\text{ of words.}}$$

Avg-w2v → works well, but not perfect.

→ tf-idf w2v :-

r_1 : w_1 w_2 w_1 w_3 w_4 w_5

↓ calculate tf-idf vector for r_1

Tf-idf	t_1	t_2	t_3	t_4	t_5	t_6

~~Tf-idf-w2v(r_1) = [t_1 w2v(w_1) | t_2 w2v(w_2) ...]~~

$$Tf\text{-}idf\text{-}w2v(r_1) = \left(\frac{t_1 * w_2v(w_1) + t_2 * w_2v(w_2) + t_3 * w_2v(w_3) + t_4 * w_2v(w_1)}{(t_1 + t_2 + t_3 + t_4 + t_5 + t_6)} \right)$$

$$Tf\text{-}idf\text{-}w2v(r_1) = \frac{\sum_{i=w_1}^{w_n} (Tf(w_i) * IDf(w_i) * w_2v(w_i))}{\sum_{i=w_1}^{w_n} Tf(w_i) * IDf(w_i)}$$