NLP Video :06

Introduction of Word Embeddings

हिंदी में

BY:SHASHI KUMAR

INITIATIVE BY: PATHSHALA.AI

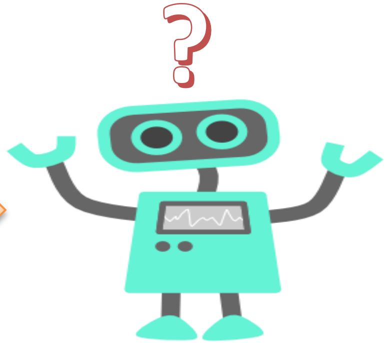
https://www.youtube.com/c/ShashiSAS

Word Embeddings

Machine learning algorithms cannot work with raw text directly; the text must be converted into numbers. Specifically, vectors of numbers.

Word embedding is the collective name for a set of **language modelling** and **feature learning techniques** in natural language processing (NLP) where words or phrases from the vocabulary are mapped to **vectors** of real numbers:-*Wikipedia*

my furry dog



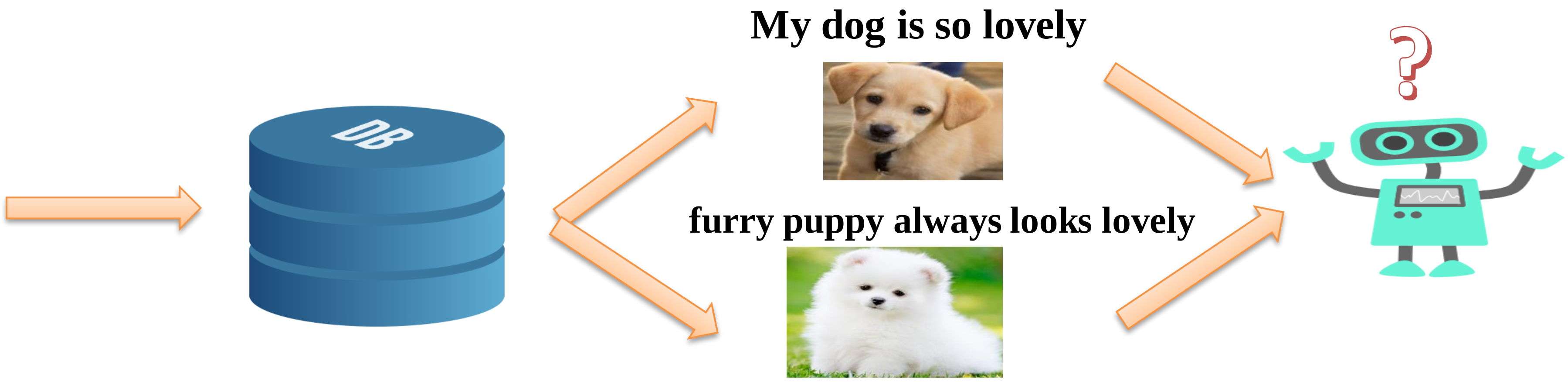
my furry dog

my	furry	dog
1	1	1

More detail @ https://www.youtube.com/c/ShashiSAS

Example 2:-

my furry dog



Count Vectorizer

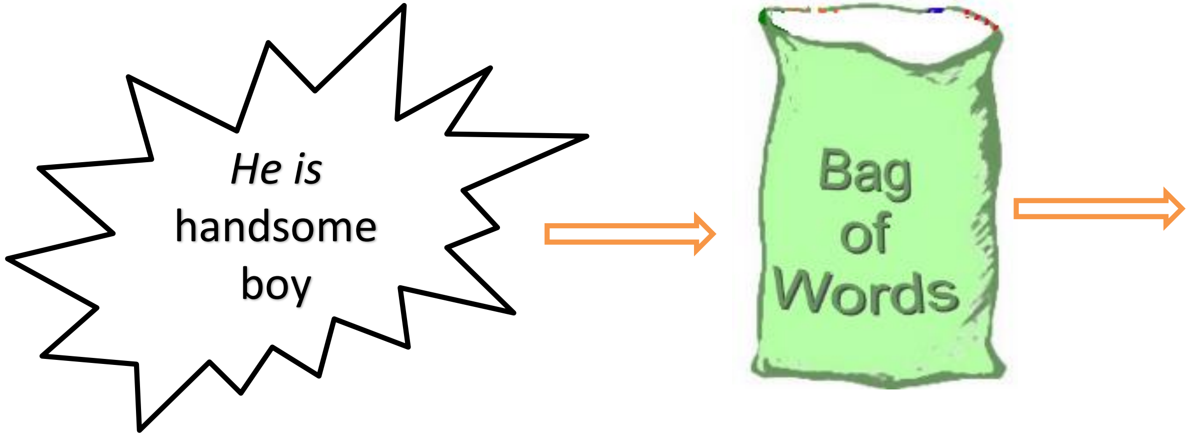
Words Documents	My	dog	is	so	lovely	furry	puppy	always	looks
1	1	1	1	1	1	0	0	0	0
2	0	0	0	0	1	1	1	1	1

I'll explore the following the following word embedding techniques:-

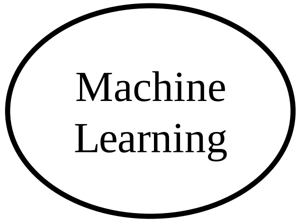
1. Bag of words or Count Vectorizer
2. TF-IDF Vectorizer
3. Word2Vec

Bag of Words Model

1. A bag-of-words model, or **BOW** for short, is a way of extracting features from text for use in modeling, such as with machine learning algorithms.



Words	He	is	handsome	boy
Document 1	1	1	1	1



2. In BOW model a sentence or a document is considered as a **'Bag'** containing words. It will take into account the words and their **frequency** of occurrence in the sentence or the document **disregarding semantic** relationship in the sentences.
3. In ML, while using **text data** we need to represent data in the form which can be processed by ML algorithms and **Bag of Word Algorithm** provide us the way of doing so. It is very easy to understand and implement.

1. “ My dog is so lovely ”

2. “A furry puppy always look lovely”

my	1
dog	1
is	1
so	1
lovely	1

\+

furry	1
puppy	1
always	1
looks	1
lovely	1

→

my	1
dog	1
is	1
so	1
lovely	2
furry	1
puppy	1
always	1
looks	1

Binary Bag of words(BOW) model/Document Term matrix(DTM)

Words / Documents	My	dog	is	so	lovely	furry	puppy	always	looks
1	1	1	1	1	1	0	0	0	0
2	0	0	0	0	1	1	1	1	1

Bag of words problem :-

1. All words have the same importance
2. No semantic information preserved

Example:- He is handsome boy:

Words	He	is	handsome	boy
Document 1	1	1	1	1

BOW model gives same importance to all words but **handsome** is most important word ,how to improve this model to identify the most important words?

Solution:

1. TF-IDF Vectorizer
2. Word2Vec

TF-IDF

Term Frequency (TF) :- the frequency of the word (term) in each document in the corpus and range of value [0,∞].

This	is	an	example
1	1	1	1

However, if there were two documents, **one very long** and **one very short**, it wouldn't be fair to compare them by **word count** alone. A better way to compare them is by a **normalized term frequency**, which is **(term count) / (total terms)** and range of value [0,1].

It is the ratio **of number of times the word appears in a document compared to the total number of words in that document. Each document has its own TF.**

$$\textbf{TF} = \frac{\textit{No.of occurances of a word in a document}}{\textit{No.of words in the document}}$$

This	is	an	example
1/4	1/4	1/4	1/4

Variants of term frequency (tf) weight

weighting scheme	tf weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$
log normalization	$\log(1 + f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K)\frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

More detail @ https://www.youtube.com/c/ShashiSAS

TF-IDF Model

Some semantic information is preserved as un common words are given more importance than common word.

TF = Term Frequency

IDF= Inverse Document Frequency : assigns a higher weight to the rare words in the text corpus.

TF-IDF= TF * IDF

TF-IDF will indicates most important word

Example:-

1. TF = Term Frequency

D1 =“my dog is so lovely”

Words	TF1	TF1
my	1/5	0.20
dog	1/5	0.20
is	1/5	0.20
so	1/5	0.20
lovely	1/5	0.20

+

D2="furry puppy always looks lovely“

Words	TF2	TF2
furry	1/5	0.20
puppy	1/5	0.20
always	1/5	0.20
looks	1/5	0.20
lovely	1/5	0.20

⇒

Words	TF1	TF2
my	0.20	0
dog	0.20	0
is	0.20	0
so	0.20	0
lovely	0.20	0.20
furry	0	0.20
puppy	0	0.20
always	0	0.20
looks	0	0.20

The **document-frequency** tells how often a word will occur in the whole collection of sentences, the information is global and not specific to any sentence.

$$\textbf{Document Frequency (DF)} = \frac{\textbf{number of documents containing a given term (d)}}{\textbf{the numberof documents in corpus (D)}}$$

But since often **<u>D > d</u>** the log of d/D, that is **log(d/D)** gives a **negative value**. To get rid off the negative sign, we simply invert the ratio inside the log expression. Essentially we are **compressing the scale of values** so that very large or very small quantities are smoothly compared. **Now log(D/d) is conveniently called Inverse Document Frequency.**

$$\textbf{IDF} = \boldsymbol{log_e}\left(\frac{\textbf{Number } \boldsymbol{of\ documents}}{\textbf{the number of documents in the collection that contain a } \boldsymbol{word}}\right)$$

Variants of inverse document frequency (idf) weight

weighting scheme	idf weight ($n_t = \lvert\{d \in D : t \in d\}\rvert$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log\left(\frac{N}{1+n_t}\right) + 1$
inverse document frequency max	$\log\left(\frac{\max_{\{t' \in d\}} n_{t'}}{1+n_t}\right)$
probabilistic inverse document frequency	$\log \frac{N - n_t}{n_t}$

Inverse Data Frequency (idf) is used to calculate the **weight of rare words** across all documents in the corpus. The words that occur **rarely** in the corpus have a **high IDF** score.

Importance of IDF:-

Total No of documents= 100,000,000

Term of Interest	Number of Documents Containing that Term	Total Docs / Docs Containing Terms	IDF Values for These Terms
a	100,000,000	1	-
boat	1,000,000	100	2
mobile	100,000	1,000	3
mobilegeddon	1,000	100,000	5

We can see that we are providing the highest score to the term that is the rarest.

Why does IDF use a log ?

The log is used to dampen the effect of the ratio(**compressing the scale of values** so that very large or very small quantities are smoothly compared.).

For example,
If there are 1,00,000 documents and a word appears in 10 of them.
Inverse Document Frequency (without log) = 1,00,000/10 = 10000
Inverse Document Frequency (with log) = log(1,00,000/10) = 4

Why is a log function used for calculating IDF?

The reason for using logs is due to two assumptions frequently made in most IR models; i.e.

1. that scoring functions are additive.
2. that terms are independent.

Let the presence of a term in a document be that event. If terms are independent, it must follows that for any two events, A and B

$$p(AB) = p(A)p(B).$$

Taking logs we can write

$$\log[p(AB)] = \log[p(A)] + \log[p(B)]$$

It is easy to show that for two terms

$$\log(d12/D) = \log(d1/D) + \log(d2/D)$$

validating assumption I; that IDF as a scoring function is **additive**. That is the IDF of a two term query is the sum of individual IDF values. However, this is only valid if terms are **independent** from one another.

IDF= Inverse Document Frequency

$$\text{IDF} = \log_e\left(\frac{No.of\ documents}{\text{the number of documents in the collection that contain a } word}\right)$$

D1 =“my dog is so lovely” **D2="furry puppy always looks lovely“**

Words	**Word frequency in all documents**	**IDF**	**IDF**
my	1	Log(2/2)	0.69
dog	1	Log(2/1)	0.69
is	1	Log(2/1)	0.69
so	1	Log(2/1)	0.69
lovely	2	Log(2/2)	0
furry	1	Log(2/1)	0.69
puppy	1	Log(2/1)	0.69
always	1	Log(2/1)	0.69
looks	1	Log(2/1)	0.69

Term Frequency-Inverse Document Frequency advantages

1. Assigns more weight to rare words and less weight to commonly occurring words.
2. Tells us how frequent a word is in a document relative to its frequency in the entire corpus.
3. Tells us that two documents are similar when they have more rare words in common.

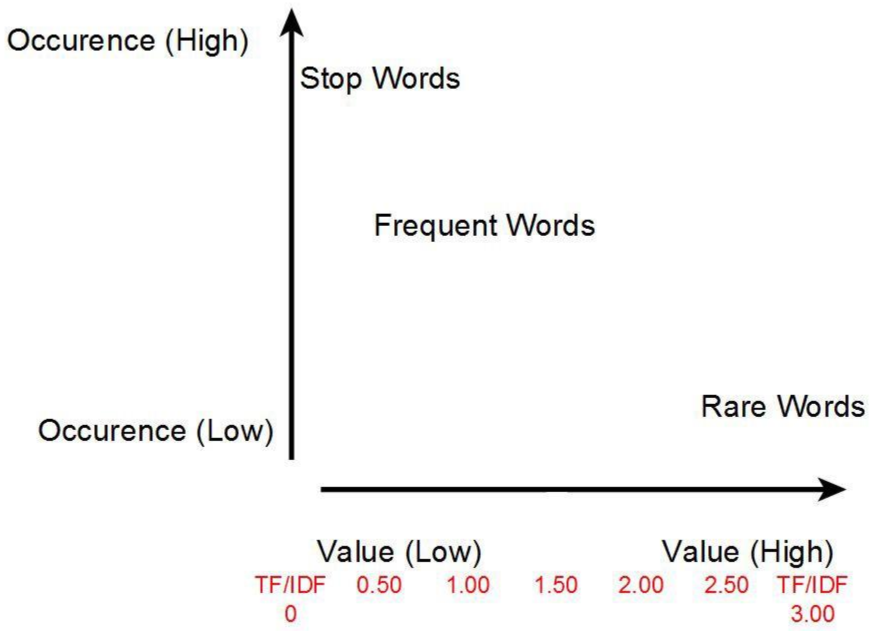
Variation of tf-idf weigh and normalization

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5\times\mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$ (Section 6.4.4)
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/\mathit{CharLength}^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

D1 ="my dog is so lovely"

D2="furry puppy always looks lovely"

Words	IDF	TF1	TF2	TF1*IDF	TF2*IDF
my	0.69	0.20	0	0.138	0
dog	0.69	0.20	0	0.138	0
is	0.69	0.20	0	0.138	0
so	0.69	0.20	0	0.138	0
lovely	0	0.20	0.20	0	0
furry	0.69	0	0.20	0	0.138
puppy	0.69	0	0.20	0	0.138
always	0.69	0	0.20	0	0.138
looks	0.69	0	0.20	0	0.138



Assigns **more** weight to **rare** words and **less** weight to **commonly** occurring words

Ref:-

1. C.D. Manning, P. Raghavan and H. Schütze (2008).Introduction to Information Retrieval. Cambridge University Press, pp. 118-120. https://nlp.stanford.edu/IR-book/pdf/irbookprint.pdf
2. https://en.wikipedia.org/wiki/Tf–idf#Inverse_document_frequency
3. https://github.com/scikit-learn/scikit-learn/blob/b194674c42d54b26137a456c510c5fdba1ba23e0/sklearn/feature_extraction/text.py#L820
4. https://irthoughts.wordpress.com/2009/04/15/why-idf-is-expressed-using-logs/