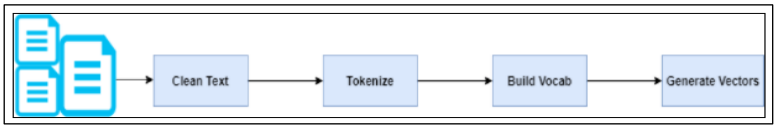
- **Conversion of text to vector**
  - In real world scenarios many times we will be dealing with texts. In order to perform mathematical analysis, we need to convert texts into a vector.



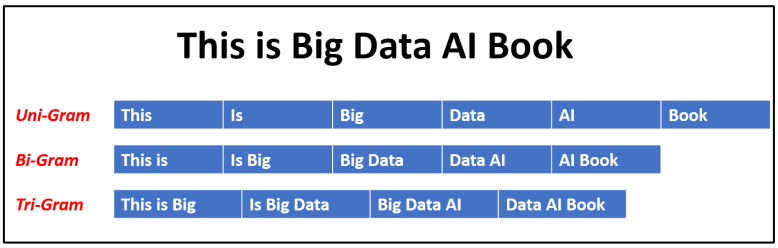
  - Each line which will be set of words having its own vectors of d-dimension. By plotting vectors, we will be able to analyse patterns in the data.
  - Suppose we have 3 reviews r1, r2, r3. They have corresponding vectors v1, v2, v3.

If English sim (r1, r2) > English sim (r1, r3) then dist. (v1, v2) < dist. (v1, v3). i.e., more similar two sentences are, distance would be lesser between them.

  - The techniques used for conversions are bag of words, tf-idf, Word2Vec, Avg-Word2Vec, tf-idf weighted Word2Vec etc.
- **Bag of words**
  - Corpus is collection of all documents. Words are also called as tokens.
  - First, we create a vocabulary of unique words across a corpus.
  - Then the document is encoded with, frequency of each word in document against the unique words
  - Example-
    - Review 1: This movie is very scary and long
    - Review 2: This movie is not scary and is slow
    - Review 3: This movie is spooky and good
    - Vocab (11 words)- [This, movie, is, very, scary, and, long, not, slow, spooky, good]

	1 This	2 movie	3 is	4 very	5 scary	6 and	7 long	8 not	9 slow	10 spooky	11 good	Length of the review(in words)
Review 1	1	1	1	1	1	1	1	0	0	0	0	7
Review 2	1	1	2	0	0	1	1	0	1	0	0	8
Review 3	1	1	1	0	0	0	1	0	0	1	1	6

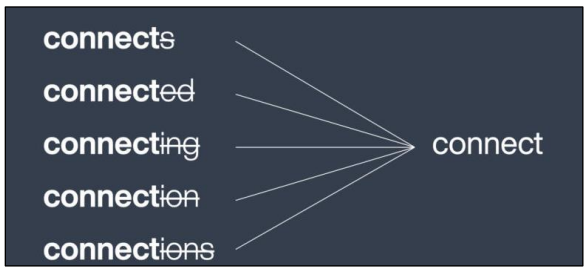
  - We can have a binary bag of words. Where instead of occurrences only word present or not will be checked
  - BOW depend on count of words in each review, this discards the semantic meaning of the documents and documents that are completely opposite in meaning can lie closer to each other when there is very small change in the document (with respect to the words). E.g., R1 & R2 in above example.
- **Bag of n-grams (unigram bigram trigram)**



  - A 1-gram (or unigram) is a one-word sequence. For the sentence" I love reading blogs about data science on Analytics Vidhya",
  - Unigrams would be: "I", "love", "reading", "blogs", "about", "data", "science", "on", "Analytics", "Vidhya".
  - A bigram is a 2-word sequence of words, like "I love", "love reading", or "Analytics Vidhya".
  - A trigram is a 3-word sequence of words like "I love reading", "about data science" or "on Analytics Vidhya".
  - We can have n-grams similarly but Dimensionality of the vector increases when including n grams.
  - Uni grams BOW discards sequence of information, with n grams we can retain some of the sequence info
  - If there is no repetition of words,
    number of unigrams > number of bigrams > number of trigrams.
    Example 1: Rome is not built in a day.
    Here there are no repeated words. So
    unigrams --> not, in, day, is, a, built, Rome
    bigrams --> Rome is, is not, not built, built in, in a, a day.
    trigrams --> Rome is not, is not built, not built in, built in a, in a day.

- If there is repetition of words in the given sentence,
  number of unigrams < number of bigrams < number of trigrams.
  Example 2:horse is a horse, of course, of course. accept it
  Here there are repeated words
  unigrams --> horse, of, course, a, is, it, accept
  bigrams --> horse is, is a , a horse, horse of, of course, course of, course accept, accept it
  trigrams --> horse is a, is a horse, a horse of, horse of course, of course of, course of course, of course accept, course accept it

- **Text Pre-processing: Stemming, Stop-word removal, Tokenization, Lemmatization**
- **Stemming-**



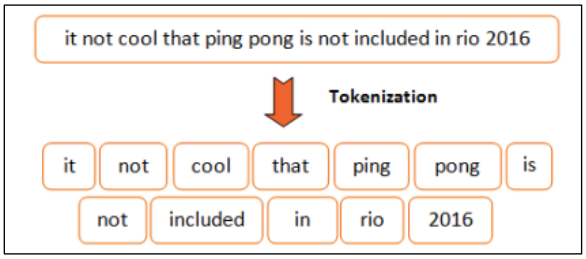
  - In English, a verb can appear in many different forms. E.g., the word "connect" may appear as: connects, connected, connecting, connection, and connections. Without any processing, machines will consider these words as five different words and calculate separately. Therefore, we will need to reduce these into root form "connect".
  - Stemming reduce redundancy for words with the same meaning, so we can obtain an accurate calculation and increase the accuracy of subsequent tasks.
  - Two of the algorithms are: Porter Stemmer and Snowball Stemmer; Snowball stemmer is more powerful

- **Stop Word Removal-**

```
> stopwords("english")
  [1] "i"          "me"         "my"         "myself"     "we"
  [6] "our"        "ours"       "ourselves"  "you"        "your"
 [11] "yours"      "yourself"   "yourselves" "he"         "him"
 [16] "his"        "himself"    "she"        "her"        "hers"
 [21] "herself"    "it"         "its"        "itself"     "they"
 [26] "them"       "their"      "theirs"     "themselves" "what"
 [31] "which"      "who"        "whom"       "this"       "that"
 [36] "these"      "those"      "am"         "is"         "are"
 [41] "was"        "were"       "be"         "been"       "being"
 [46] "have"       "has"        "had"        "having"     "do"
```

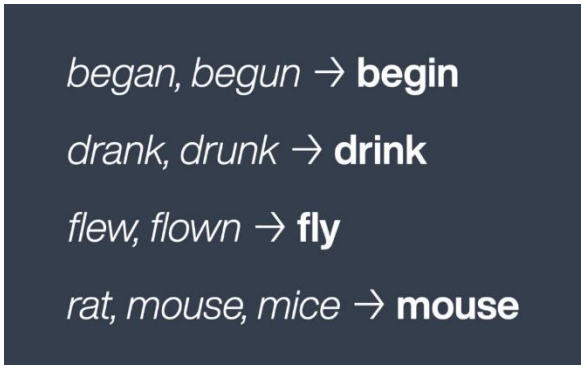
  - Stop words refer to terms that appear frequently in a natural language (e.g. is, am, a, in) but do not aid in understanding the semantic meaning.
  - Removing stop words prevents high occurrence words from taking up space in database and taking up the processing time.
  - We should convert everything in lowercase first and then we can remove stop words.
  - NLTK, Spacy are python libraries that can be used for removing stop words

- **Tokenization**



  - It is difficult for machines to understand the semantics and context of a document, a paragraph, or even a sentence as a whole. Tokenization is process to break them to down their smallest semantic units call token or words.
  - For example, given a sentence "Dr. Sinha is a data scientist ", we can break them down into six tokens using space as the separator. We also remove punctuations and symbols.
  - This approach is suitable for any language that uses blank space as a separator. For languages that do not, such as Chinese, Japanese, and Thai, we will need to use a dictionary or apply external knowledge to split words into tokens.

- **Lemmatization**



  - In some cases, stemmer is unable to reduce word variations to its root form by simply chopping away the suffixes e.g., "begin, began", "drank, drunk", "eat, ate", or words like "rats, mouse, mice".
  - We will need to utilize a dictionary or external knowledge to group inflected forms of a word together, and this process is called lemmatization.

- **Part-of-speech tagging**

  - we perform part-of-speech tagging to identify the role of a word in a sentence.
  - This is based on the relationship and context with the surrounding words.
  - Part of speech includes nouns, verbs, adverbs, adjectives, pronouns, prepositions, and other sub-categories.

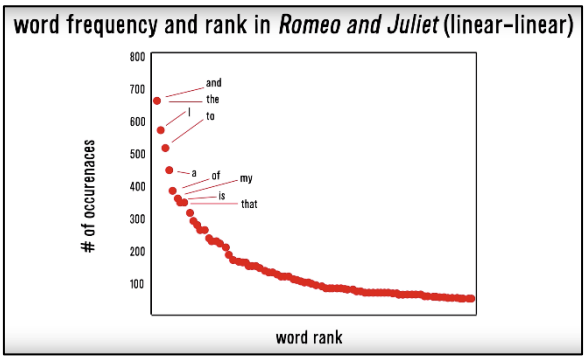
- **TF-IDF(Term Frequency- Inverse Document Frequency)**
- **Term Frequency (tf):**
  - It gives us the frequency of the word in each document in the corpus. It is the ratio of number of times the word appears in a document compared to the total number of words in that document.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}} \qquad \begin{aligned} n(i,j) &= no\ of\ times\ i\ coming\ in\ j \\ \sum_k n_{i,j} &= total\ no\ of\ words\ in\ document \end{aligned}$$

- **Inverse Data Frequency (idf):**
  - It 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. It is given by the equation below.

$$idf_{i,j} = \log \frac{N_j}{df_i} \qquad \begin{aligned} N_j &= Total\ number\ of\ j\ documents \\ df_i &= Number\ of\ documents\ with\ term\ i\ in\ it. \end{aligned}$$

  - Usage of log can be understood from Zipf's law. The law postulates that the top 18% of most frequently used words account for 80% of word occurrences.
  - Top 20 Most Used Words- The, Of, And, To, A, In, Is, I, That, It, For, You, Was, With, On, As, Have, but, Be, They



  - some words such as 'the' and 'in' are frequently found in usage, while such as geometry and civilization occur less in text documents. When a log – log plot is applied we can find a straight line.
  - IDF without a log will have large numbers that will dominate in the ML model;
  - Taking log will bring down dominance of IDF in TFIDF values;

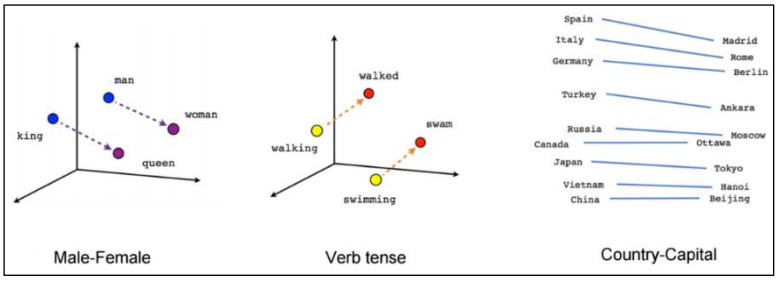
Word	TF A	TF B	IDF	TF*IDF A	TF*IDF B
The	1/7	1/7	log(2/2) = 0	0	0
Car	1/7	0	log(2/1) = 0.3	0.043	0
Truck	0	1/7	log(2/1) = 0.3	0	0.043
Is	1/7	1/7	log(2/2) = 0	0	0
Driven	1/7	1/7	log(2/2) = 0	0	0
On	1/7	1/7	log(2/2) = 0	0	0
The	1/7	1/7	log(2/2) = 0	0	0
Road	1/7	0	log(2/1) = 0.3	0.043	0
Highway	0	1/7	log(2/1) = 0.3	0	0.043

  - Combining these two we come up with the TF-IDF score (w) for a word in a document in the corpus. It is the product of tf and idf:

$$w_{i,j} = tf_{i,j} * idf_{i,j}$$

  - Example:
    - Sentence A: The car is driven on the road.
    - Sentence B: The truck is driven on the highway.

- **Word2Vec**



  - The idea behind Word2Vec is pretty simple. We're making an assumption that the meaning of a word can be inferred by the company it keeps. This is analogous to the saying, "show me your friends, and I'll tell who you are".
  - If we have two words that have very similar neighbours (meaning: the context in which it's used is about the same), then these words are probably quite similar in meaning or are at least related. For example, the words shocked, appalled and astonished are usually used in a similar context.
  - If two words are semantically similar, then vectors of these words are geometrically closer

- It takes a text corpus (large size): for every word it builds a vector: dimension of the vectors is given as an input; this uses words in the neighbourhood: if the neighbourhood is similar then the words have similar vectors;
- Google took data corpus from Google News to train its Word2Vec method

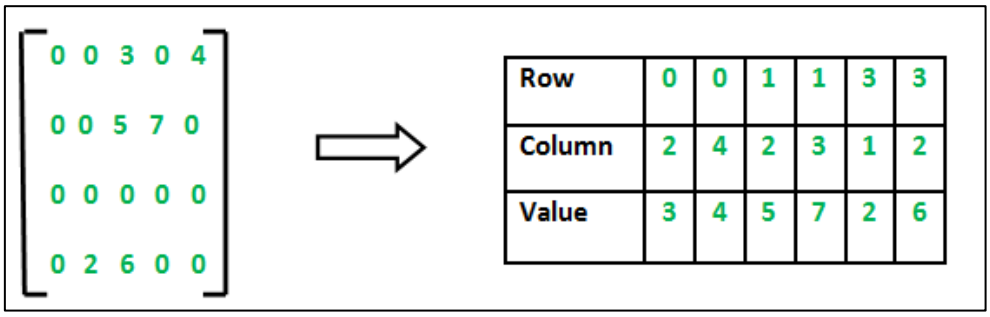
- **Avg-Word2Vec, tf-idf weighted Word2Vec**
  - Let's say we have a sentence with n words w1 w2 w1 w3 w4 w5. Now each of these words will have their own d-dimensional w2v. taking average of all these vectors is called as average w2v.

$$\begin{bmatrix} W_{11} \\ W_{12} \\ \\ W_{1n} \end{bmatrix}_{W_1} + \begin{bmatrix} W_{21} \\ W_{22} \\ \\ W_{2n} \end{bmatrix}_{W_2} + \ldots + \begin{bmatrix} W_{n1} \\ W_{n2} \\ \\ W_{nn} \end{bmatrix}_{W_n} = \begin{bmatrix} \frac{W_{11}+W_{21}+\ldots+W_{n1}}{n} \\ \\ \frac{W_{1n}+W_{2n}+\ldots+W_{nn}}{n} \end{bmatrix}_{D}$$

  - Similarly, for every word in a sentence we will have a tf-idf value. Let's say t1,t2,t3,t4,t5. To compute tf idf weighted Word2Vec.

$$\text{tf idf weighted Word2Vec} = \frac{\sum_i^n t_i * w2v(w_i)}{\sum t}$$

- **Sparse Vectors**
  - A matrix is a two-dimensional data object made of m rows and n columns, therefore having total m x n values. If most of the elements of the matrix have 0 value, then it is called a sparse matrix. By contrast, if most of the elements are nonzero, then it is a dense matrix.
  - Why to use Sparse Matrix instead of simple matrix ?
    - Storage: There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.
    - Computing time: Computing time can be saved by logically designing a data structure traversing only non-zero elements.



  - $sparsity = \frac{\#(zero\ elements)}{total\ elements}$ $\quad Density = 1 - sparsity$
    - [[1, 0, 0, 1, 0, 0]
    - [0, 0, 2, 0, 0, 1]
    - [0, 0, 0, 2, 0, 0]]
    - Sparsity = 13/18 = 0.722
    - Density = 1 -0.722 = 0.278
  - Dense matrices store every entry in the matrix. Sparse matrices only store the nonzero entries. Sparse matrices don't have a lot of extra features, and some algorithms may not work for them.
  - We use them when we need to work with matrices that would be too big for the computer to handle them, but they are mostly zero, so they compress easily. E.g. Unigram BOW representation.
  - It is wasteful to use general methods of linear algebra on such problems, because most of the $O(N^3)$ arithmetic operations devoted to solving the set of equations or inverting the matrix that involves zero operands. So, we convert it into sparse matrix.