1. Corpus - Contains text and speech data that can be used to train AI and machine learning systems.

* A corpus can be defined as a collection of machine-readable authentic texts (including transcripts of spoken data) that is sampled to be representative of a particular natural language
* Data corpus refers to all data collected for a particular research project, while data set refers to all the data from the corpus that is being used for a particular analysis.

1. TF-IDF - Term Frequency - Inverse Document Frequency (TF-IDF) is a widely used statistical method in natural language processing. It measures how important a term is within a document relative to a collection of documents. Words within a text document are transformed into importance numbers by a text vectorization process

* **Term Frequency:** TF of a term or word is the number of times the term appears in a document compared to the total number of words in the document.
* **Inverse Document Frequency**: IDF of a term reflects the proportion of documents in the corpus that contain the term. Words unique to a small percentage of documents (e.g., technical jargon terms) receive higher importance values than words common across all documents (e.g., a, the, and).
* TF-IDF of a term is calculated by multiplying TF and IDF scores.
* Commonality within a document measured by TF is balanced by rarity between documents measured by IDF. The resulting TF-IDF score reflects the importance of a term for a document in the corpus.
* [Machine learning](https://monkeylearn.com/blog/gentle-guide-to-machine-learning/) with natural language is faced with one major hurdle – its algorithms usually deal with numbers, and natural language is, well, text. So, we need to transform that text into numbers, otherwise known as [text vectorization](https://monkeylearn.com/blog/beginners-guide-text-vectorization/).

1. Word Embeddings - Word embeddings are a type of word representation that allows words with similar meanings to have a similar representation.

* Word embeddings are in fact a class of techniques where individual words are represented as real-valued vectors in a predefined vector space.
* **Word Embeddings or Word vectorization is a methodology in NLP to map words or phrases from vocabulary to a corresponding vector of real numbers which used to find word predictions, word similarities/semantics.**

1. Document Embeddings – Document embedding is usually computed from the word embeddings in two steps. First, each word in the document is embedded with the word embedding then word embeddings are aggregated. The most common type of aggregation is the average over each dimension.

* Differs from [word embeddings](https://github.com/flairNLP/flair/blob/master/resources/docs/TUTORIAL_3_WORD_EMBEDDING.md) in that they give you one embedding for an entire text, whereas word embeddings give you embeddings for individual words.

1. Contextual Embeddings - **contextual embedding methods** are used to learn **sequence-level semantics** by considering the sequence of all words in the documents. Thus, such techniques learn **different representations** for **polysemous words**, e.g. "left" in example below, based on their context.

Eg:  "I **left** my phone on the **left** side of the table."

* Capture word semantics in context such that it can represent differently under different even though it is same word.

2.)

**TfidfVectorizer -** It converts a collection of raw documents to a matrix of **TF-IDF** features. As tf–idf is very often used for text features, the class **TfidfVectorizer** combines all the options of **CountVectorizer** and **TfidfTransformer** into a single model. The TfidfVectorizer uses an in-memory vocabulary (a python dict) to map the most frequent words to feature indices and hence compute a word occurrence frequency (sparse) matrix.

**Gensim Word2Vec -** Word2Vec is an algorithm designed by Google that uses neural networks to create word embeddings such that embeddings with similar word meanings tend to point in a similar direction. **Word2Vec**consists of models for generating word embedding. These models are shallow two-layer neural networks having one input layer, one hidden layer, and one output layer. Word2Vec utilizes two architectures.

* **CBOW (Continuous Bag of Words):** CBOW model predicts the current word given context words within a specific window. The input layer contains the context words and the output layer contains the current word. The hidden layer contains the number of dimensions in which we want to represent the current word present at the output layer.
* **Skip Gram:**Skip gram predicts the surrounding context words

within specific window given current word. The input layer contains the current word and the output layer contains the context words.

**Gensim Doc2vec** - Modified version of [word2vec](https://thinkinfi.com/gensim-word2vec-python-implementation/). The main objective of doc2vec is to convert sentence or paragraph to vector (numeric) form. In [**Natural Language Processing**](https://thinkinfi.com/complete-guide-for-natural-language-processing-in-python/) Doc2Vec is used to find related sentences for a given sentence (instead of word in [**Word2Vec**](https://thinkinfi.com/word2vec-skip-gram-explained/)).

BERT (Bidirectional Encoder Representations from Transformers) – Main goal is to predict the masked word.

* **Masked Language Model:**  One of the drawbacks of traditional word embedding used in earlier language models is the lack of contextual meaning.
* **Next Sentence Prediction (NSP):** Create a long-term relationship between sentences**.**