Natural Language Processing

Natural Language Processing

- NLP is a branch of data science that consists of systematic processes for analyzing, understanding, and deriving information from the text data in a smart and efficient manner.
- By utilizing NLP and its components, one can organize the massive chunks of text data, perform numerous automated tasks and solve a wide range of problems such as –
  - automatic summarization, machine translation, named entity recognition, relationship extraction, sentiment analysis, speech recognition, and topic segmentation etc.

NLP : Terminology

- **Tokenization** – process of converting a text into tokens (words or entities)
- **Tokens** – words or entities present in the text
- **Text object** – a sentence or a phrase or a word or an article

NLP : Terminology

- **White-space Tokenization**

Also known as **unigram tokenization**. In this process, the entire text is split into words by splitting them from white spaces.For example, in a sentence- “I went to New- York to play football.”

This will be splitted into following tokens: “I”, “went”, “to”, “New-York”, “to”, “play”, “football.”

NLP : Terminology

- **Regular Expression Tokenization**

A regular expression pattern is used to get the tokens. For example, consider the following string containing multiple delimiters such as comma, semi-colon, and white space.

- Sentence= “Football, Cricket; Golf Tennis“
- Tokens= “Football”, ”Cricket”, “Golf”, “Tennis”
- Using Regular expression, we can split the text by passing a splitting pattern

NLP : Terminology

- **Stemming**
- Stemming is an elementary rule-based process for removing inflectional forms from a token and the outputs are the **stem** of the world.
- For example, “laughing”, “laughed“, “laughs”, “laugh” will all become “laugh”, which is their stem, because their inflection form will be removed.
- For example, consider a sentence: “His teams are not winning”
- After stemming the tokens that we will get are- “hi”, “team”, “are”, “not”, “winn”

What is Corpus?

- A **Corpus** is defined as a collection of text documents
- For example a data set containing news is a corpus or the tweets containing Twitter data is a corpus
- Corpus consists of documents, documents comprise paragraphs, paragraphs comprise sentences and sentences comprise further smaller units which are called **Tokens**.

What is Corpus?

- Tokens can be words, phrases, or Engrams, and **Engrams** are defined as the group of n words together.
- For example, consider this given sentence-

  “I love my phone.”
- In this sentence, the uni-grams(n=1) are: I, love, my, phone
- Di-grams(n=2) are: I love, love my, my phone
- And tri-grams(n=3) are: I love my, love my phone

Corpus : Example

```
corpus = ["This is a brown house. This house is big. The street number
is 1.",
          "This is a small house. This house has 1 bedroom. The street
number is 12.",
          "This dog is brown. This dog likes to play.",
          "The dog is in the bedroom."]
```

Text Vectorization

- ***Text Vectorization*** is the process of converting text into numerical representation. Some popular methods to accomplish text vectorization:

1. **Binary Term Frequency**
2. **Bag of Words (BoW) Term Frequency**
3. **(L1) Normalized Term Frequency**
4. **(L2) Normalized TF-IDF**
5. **Word2Vec**

Text Vectorization : Binary Term Frequency

- Binary Term Frequency captures presence (1) or absence (0) of term in document.

	12	bedroom	big	brown	dog	house	likes	number	play	small	street
0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0
1	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	1.0
2	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0
3	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0

Text Vectorization : Bag of Words (BoW) Term Frequency

- Bag of Words (BoW) Term Frequency captures frequency of term in document

	12	bedroom	big	brown	dog	house	likes	number	play	small	street
0	0.0	0.0	1.0	1.0	0.0	2.0	0.0	1.0	0.0	0.0	1.0
1	1.0	1.0	0.0	0.0	0.0	2.0	0.0	1.0	0.0	1.0	1.0
2	0.0	0.0	0.0	1.0	2.0	0.0	1.0	0.0	1.0	0.0	0.0
3	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0

Text Vectorization : (L1) Normalized Term Frequency

- (L1) Normalized Term Frequency captures normalized BoW term frequency in document.

	12	bedroom	big	brown	dog	house	likes	number	play	small	street
0	0.000000	0.000000	0.166667	0.166667	0.0	0.333333	0.0	0.166667	0.0	0.000000	0.166667
1	0.142857	0.142857	0.000000	0.000000	0.0	0.285714	0.0	0.142857	0.0	0.142857	0.142857
2	0.000000	0.000000	0.000000	0.200000	0.4	0.000000	0.2	0.000000	0.2	0.000000	0.000000
3	0.000000	0.500000	0.000000	0.000000	0.5	0.000000	0.0	0.000000	0.0	0.000000	0.000000

(L2) Normalized TFIDF (Term Frequency–Inverse Document Frequency)

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.

$$TF = \frac{\text{number of times the term appears in the document}}{\text{total number of terms in the document}}$$

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).

$$IDF = log(\frac{\text{number of the documents in the corpus}}{\text{number of documents in the corpus contain the term}})$$

The TF-IDF of a term is calculated by multiplying TF and IDF scores.

$$TF\text{-}IDF = TF * IDF$$

(L2) Normalized TFIDF (Term Frequency–Inverse Document Frequency)

Imagine the term t appears 20 times in a document that contains a total of 100 words. Term Frequency (TF) of t can be calculated as follow:

$$TF = \frac{20}{100} = 0.2$$

Assume a collection of related documents contains 10,000 documents. If 100 documents out of 10,000 documents contain the term t, Inverse Document Frequency (IDF) of t can be calculated as follows

$$IDF = log\frac{10000}{100} = 2$$

Using these two quantities, we can calculate TF-IDF score of the term t for the document.

$$TF\text{-}IDF = 0.2 * 2 = 0.4$$

Word2vec

- **Word embedding** is a technique where individual words are transformed into a numerical representation of the word (a vector). Where each word is mapped to one vector, this vector is then learned in a way which resembles a neural network.
- Simplest embedding is **one hot encoding**

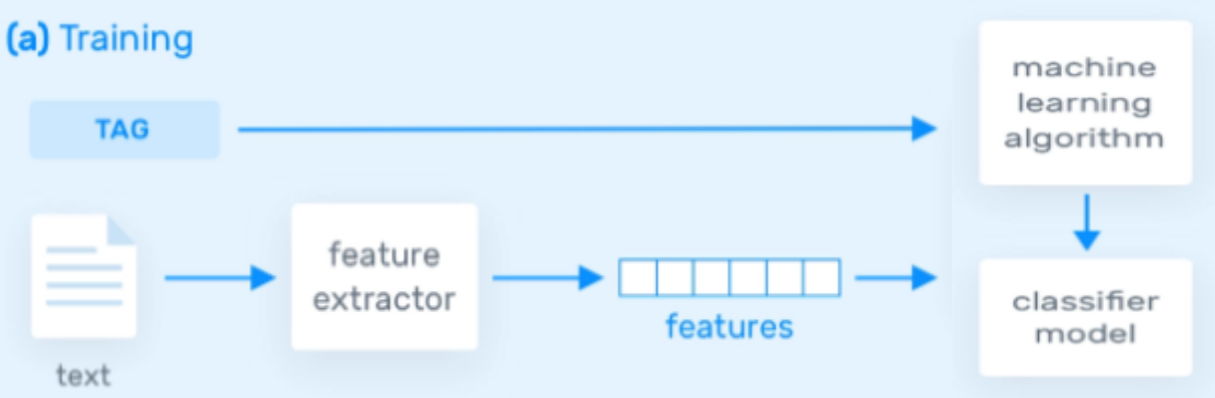
```
have = [1, 0, 0, 0, 0, 0, ... 0]
a    = [0, 1, 0, 0, 0, 0, ... 0]
good = [0, 0, 1, 0, 0, 0, ... 0]
day  = [0, 0, 0, 1, 0, 0, ... 0] ...
```

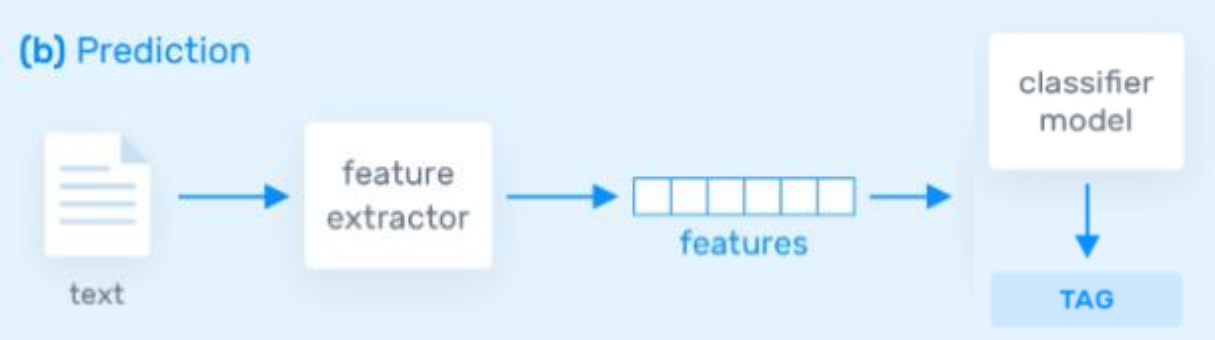
- Disadvantage: They can be large depending upon size of corpus
- Word2Vec, a common method of generating word embeddings. The effectiveness of **Word2Vec** comes from its ability to group together vectors of similar words. 2 ways to perform word2vec:
  - **CBOW (Continuous Bag of Words):**This model architecture essentially tries to predict a target word from a list of context words.
  - **Continuous Skip-Gram Model**: The skip-gram model is a simple neural network with one hidden layer trained in order to predict the probability of a given word being present when an input word is present.

Text Classification

- Text classification is one of the classical problem of NLP. Examples include – **Email Spam Identification, topic classification of news, sentiment classification and organization of web pages by search engines**.
- **Text classification, in common words is defined** as a technique to systematically classify a text object (document or sentence) in one of the fixed category.
- It is really helpful when the amount of data is too large, especially for organizing, information filtering, and storage purposes.
- A typical natural language classifier consists of two parts: **(a) Training (b) Prediction.**
- Firstly the **text input is processedd** and **features are created**. The machine learning models then learn these features and is used for predicting against the new text.

Text Classification





Text Classification

- Machine learning text classification learns to make classifications based on past observations using **pre-labeled examples** as training data. A “tag” is the pre-determined classification or category that any given text could fall into.
- The first step towards training a machine learning NLP classifier is feature extraction: a method is used to transform each text into a numerical representation in the form of a vector.
- For example, if we have defined our dictionary to have the following words {*This, is, the, not, awesome, bad, basketball*}, and we wanted to vectorize the text *“This is awesome,”* we would have the following vector representation of that text: (1, 1, 0, 0, 1, 0, 0).

Text Classification

- Then, the machine learning algorithm is fed with training data that consists of pairs of feature sets (vectors for each text example) and tags (e.g. *sports*, *politics*) to produce a classification model.
- Once it's trained with enough training samples, the machine learning model can begin to make accurate predictions. The same feature extractor is used to transform unseen text to feature sets, which can be fed into the classification model to get predictions on tags (e.g., *sports*, *politics*):

Text Classification

```
training_corpus = [
                  ('I am exhausted of this work.', 'Class_B'),
                  ("I can't cooperate with this", 'Class_B'),
                  ('He is my badest enemy!', 'Class_B'),
                  ('My management is poor.', 'Class_B'),
                  ('I love this burger.', 'Class_A'),
                  ('This is an brilliant place!', 'Class_A'),
                  ('I feel very good about these dates.', 'Class_A'),
                  ('This is my best work.', 'Class_A'),
                  ("What an awesome view", 'Class_A'),
                  ('I do not like this dish', 'Class_B')]
test_corpus = [
                ("I am not feeling well today.", 'Class_B'),
                ("I feel brilliant!", 'Class_A'),
                ('Gary is a friend of mine.', 'Class_A'),
                ("I can't believe I'm doing this.", 'Class_B'),
```

Text Classification

```
model = NBC(training_corpus)
print(model.classify("Their codes are amazing."))
>>> "Class_A"
print(model.classify("I don't like their computer."))
>>> "Class_B"
print(model.accuracy(test_corpus))
>>> 0.83
```

Text Matching / Similarity : Clustering

- One of the important areas of NLP is the matching of text objects to find similarities.
- Important applications of text matching includes automatic spelling correction, data de-duplication and genome analysis etc.
- A number of text matching techniques are available depending upon the requirement.

Text Matching / Similarity : Clustering

- **Levenshtein Distance** – The Levenshtein distance between two strings is defined as the minimum number of edits needed to transform one string into the other, with the allowable edit operations being insertion, deletion, or substitution of a single character. Following is the implementation for efficient memory computations.
- levenshtein("analyze","analyse")

Text Matching / Similarity : Clustering

- **Phonetic Matching** – A Phonetic matching algorithm takes a keyword as input (person's name, location name etc) and produces a character string that identifies a set of words that are (roughly) phonetically similar.
- It is very useful for searching large text corpuses, correcting spelling errors and matching relevant names.
- **Soundex and Metaphone** are two main phonetic algorithms used for this purpose. **Python's module Fuzzy** is used to compute soundex strings for different words,

Text Matching / Similarity : Clustering

- **Phonetic Matching (Soundex)–**

Step 1- Save the first letter. Remove all occurrences of a, e, i, o, u, y, h, w.
Step 2- Replace all consonants (include the first letter) with digits.
b, f, p, v → 1
c, g, j, k, q, s, x, z → 2
d, t → 3
l → 4
m, n → 5
r → 6
Step 3- Replace all adjacent same digits with one digit.
Step 4- If the saved letter's digit is the same the resulting first digit, remove the digit (keep the letter).
Step 5- Append 3 zeros if result contains less than 3 digits. Remove all except first letter and 3 digits after it.

```
import fuzzy
soundex = fuzzy.Soundex(4)
print soundex('ankit')
>>> "A523"
print soundex('aunkit')
>>> "A523"
```

"Robert" and "Rupert" yield "R163"

Text Matching / Similarity : Clustering

- **Cosine Similarity** – When the text is represented as vector notation, a general cosine similarity can also be applied in order to measure vectorized similarity. Following code converts a text to vectors (using term frequency) and applies cosine similarity to provide closeness among two text

Text Matching / Similarity: Clustering

- **Cosine Similarity –**

	the	best	data	science	course	is	popular
D1	1	1	1	1	1	0	0
D2	0	0	1	1	0	1	1

- $D1 = [1, 1, 1, 1, 1, 0, 0]$
- $D2 = [0, 0, 1, 1, 0, 1, 1]$

Using these two vectors we can calculate cosine similarity. First, we calculate the dot product of the vectors:

$$D1 \cdot D2 = 1 \times 0 + 1 \times 0 + 1 \times 1 + 1 \times 1 + 1 \times 0 + 0 \times 1 + 0 \times 1 = 2$$

Second, we calculate the magnitude of the vectors:

$$\|D1\| = \sqrt{1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 0^2 + 0^2} = \sqrt{5}$$

$$\|D2\| = \sqrt{0^2 + 0^2 + 1^2 + 1^2 + 0^2 + 1^2 + 1^2} = \sqrt{4}$$

Finally, cosine similarity can be calculated by dividing the dot product by the magnitude

$$similarity(D1, D2) = \frac{D1 \cdot D2}{\|D1\|\|D2\|} = \frac{2}{\sqrt{5}\sqrt{4}} = \frac{2}{\sqrt{20}} = 0.44721$$

Important Libraries for NLP (python)

- **Scikit-learn**: Machine learning in Python
- **Natural Language Toolkit (NLTK):** The complete toolkit for all NLP techniques.
- **Pattern** – A web mining module for the with tools for NLP and machine learning.
- **TextBlob** – Easy to use nlp tools API, built on top of NLTK and Pattern.
- **spaCy** – Industrial strength NLP with Python and Cython.
- **Gensim** – Topic Modelling for Humans
- **Stanford Core NLP** – NLP services and packages by Stanford NLP Group.

Text Visualization

- Text visualization is the technique of using graphs, charts, or word clouds to showcase written data in a visual manner. This provides quick insight into the most relevant keywords in a text, summarizes content, and reveals trends and patterns across documents.

Text Visualization.. Use

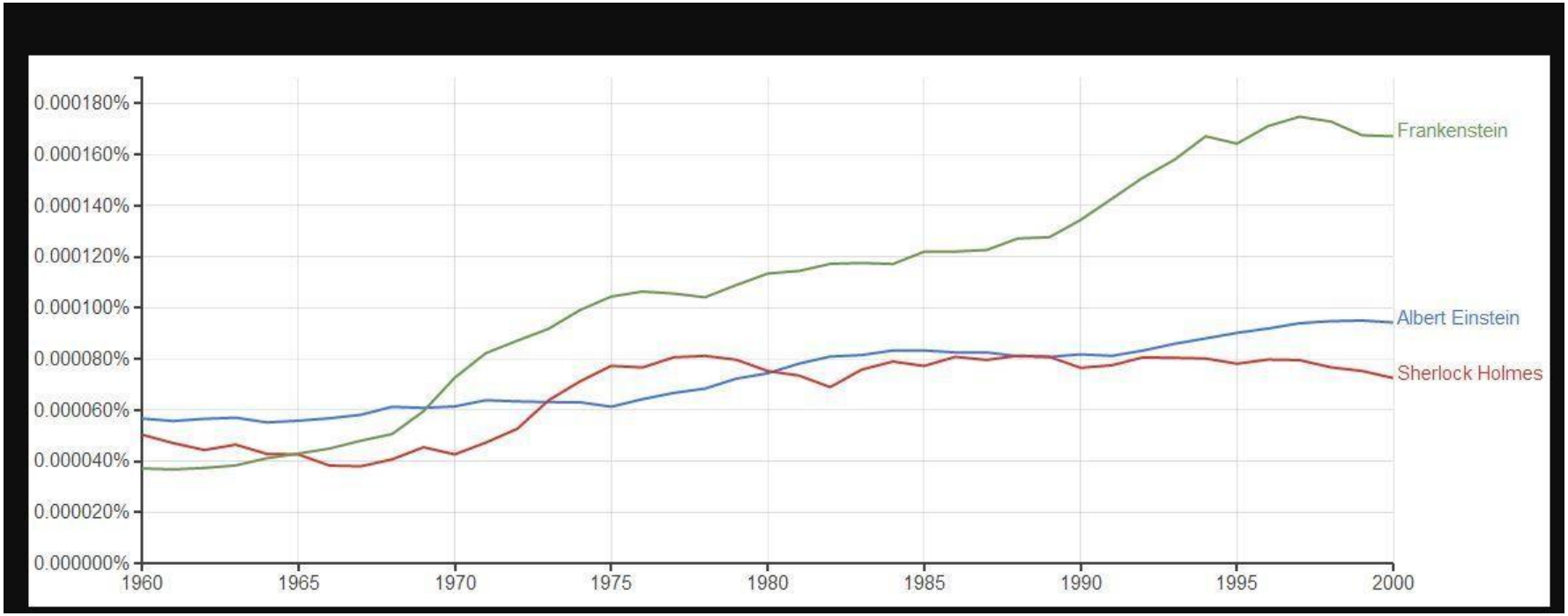
- **Summarize large amounts of text.** Automatically highlight key terms in a series of texts, and categorize text by topic, sentiment, and more, saving hours of reading time. How long would it take you to read 500 online reviews? With a word cloud or data visualization dashboard, you can understand text data at a glance.
- **Make text data easy to understand.** The human brain loves visual data. In fact, we are able to process images much faster than text. Text visualization is an effective way of simplifying complex data and communicating ideas and concepts to team managers.

Text Visualization.. Use

- **Find insights in qualitative data.** Customer feedback holds a trove of insights. Through text visualization, you can get an overview of the features, products, and topics that are most important to your customers. Learn what their pain points are and what you're doing right.
- **Discover hidden trends and patterns.** Analyze and visualize insights over time to detect fluctuations, and quickly find the root cause.

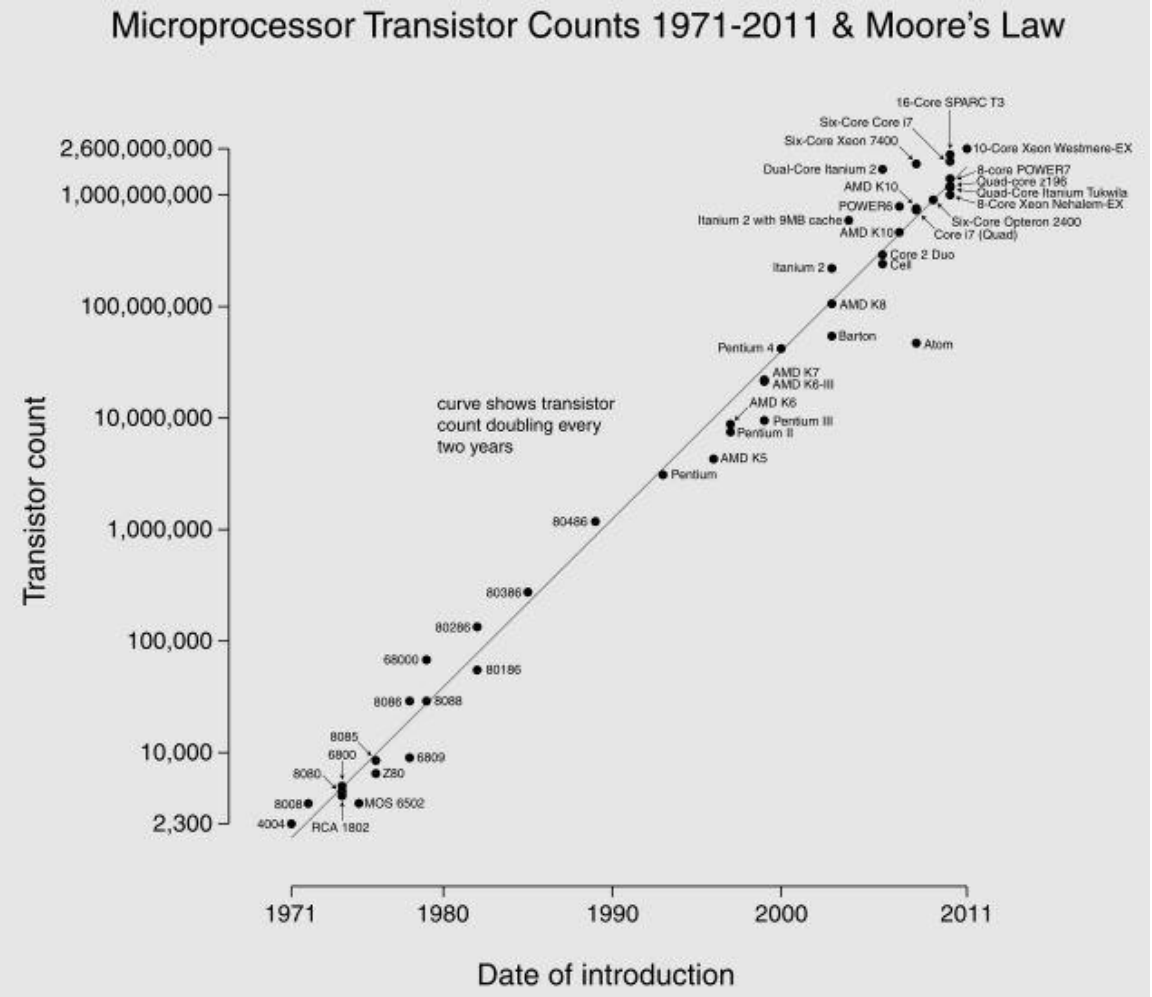
Text Visualization

- *A graph generated by Google Ngram Viewer with key words "Frankenstein" "Albert Einstein" and "Sherlock Holmes" to show the development of them.*



Text Visualization

- *CPU transistor counts v. s. date of introduction*



Text Visualization

- *Word Cloud of all my weekly blogs generated by Wordle*