**Syracuse University**

**IST-736 Assignment 2**

ThulasiRam RuppaKrishnan

IST 736

Professor Ami Gates

Contents

[Introduction 3](#_Toc14044417)

[Analysis and Models 5](#_Toc14044418)

[**About the data** 5](#_Toc14044419)

[**Models/Tools** 8](#_Toc14044420)

[Results 16](#_Toc14044421)

[Conclusion 19](#_Toc14044422)

## 

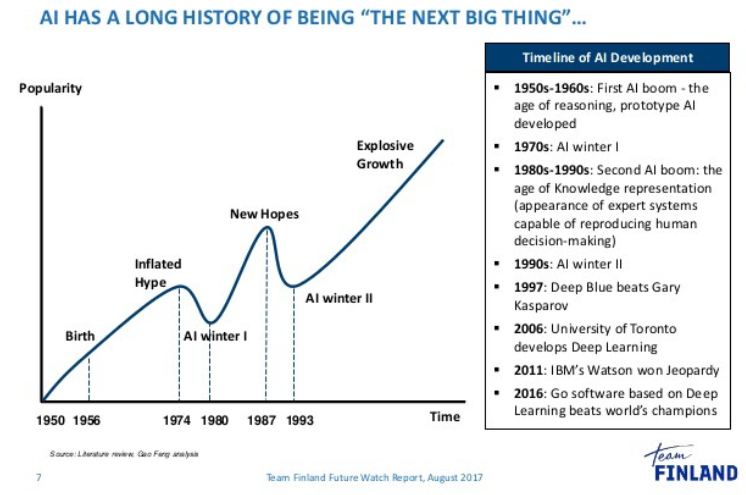
## **Introduction**

Artificial intelligence (AI) is the simulation of human intelligence processes by machines, especially computer systems. These processes include learning (the acquisition of information and rules for using the information), reasoning (using rules to reach approximate or definite conclusions) and self-correction. Applications of AI include expert systems, speech recognition and machine vision.

AI can be categorized as either weak or strong. Weak AI, also known as narrow AI, is an AI system that is designed and trained for a task. Virtual personal assistants, such as Apple's Siri, are a form of weak AI. Strong AI, also known as artificial general intelligence, is an AI system with generalized human cognitive abilities. When presented with an unfamiliar task, a strong AI system can find a solution without human intervention.

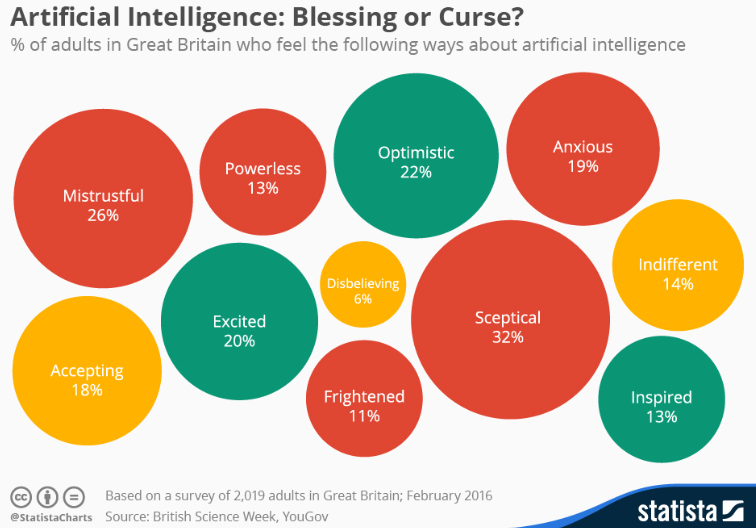


Because hardware, software and staffing costs for AI can be expensive, many vendors are including AI components in their standard offerings, as well as access to Artificial Intelligence as a Service (AIaaS) platforms. AI as a Service allows individuals and companies to experiment with AI for various business purposes and sample multiple platforms before making a commitment. Popular AI cloud offerings include Amazon AI services, IBM Watson Assistant, Microsoft Cognitive Services and Google AI services.



Source: <https://medium.com/@bboynton97/why-isnt-ai-the-buzzword-that-it-used-to-be-ffa3199324fd>

Advances in artificial intelligence (AI) could impact nearly all aspects of society: the labor market, transportation, healthcare, education, and national security. AI’s effects may be profoundly positive, but the technology entails risks and disruptions that warrant attention. While technologists and policymakers have begun to discuss AI and applications of machine learning more frequently, public opinion has not shaped much of these conversations. In the U.S., public sentiments have shaped many policy debates, including those about immigration, free trade, international conflicts, and climate change mitigation. As in these other policy domains, we expect the public to become more influential over time. It is thus vital to have a better understanding of how the public thinks about AI and the governance of AI. Such understanding is essential to crafting informed policy and identifying opportunities to educate the public about AI’s character, benefits, and risks.



A recent survey commissioned by the British Science Association once again showed that the public’s view of artificial intelligence is by no means unequivocally positive. In fact, the attributes mistrustful, skeptical and anxious were among the most cited when 2,019 Britons were asked how they feel about AI. 36 percent of the respondents even believe that intelligent machines pose a threat to the long-term survival of humanity – a view that may at least in part be influenced by Hollywood’s popular theme of robots turning on humans.

Source: <https://www.statista.com/chart/4503/views-on-artificial-intelligence/>

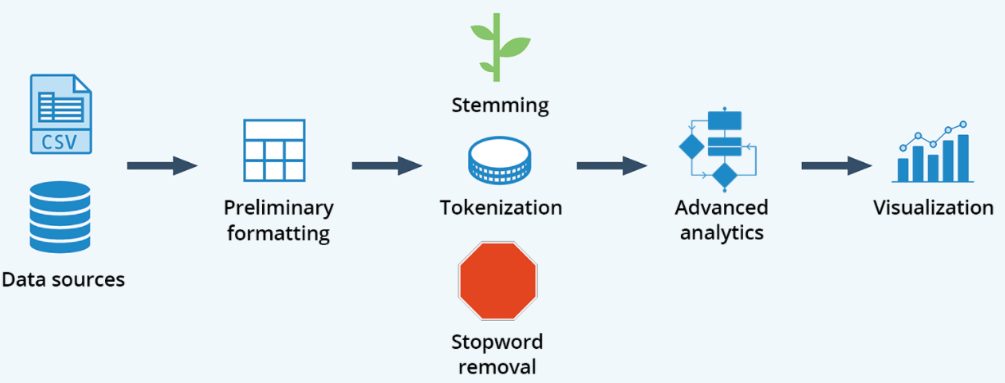
## **Analysis and Models**

### **About the data**

Microblogging today has become a very popular communication tool among Internet users. Millions of users share opinions on different aspects of life every day. Therefore, microblogging web-sites are rich sources of data for opinion mining and sentiment analysis. Because microblogging has appeared relatively recently, there are a few research works that were devoted to this topic. Twitter, the most popular microblogging platform, is used here for the task of sentiment analysis. Tweets are collected using twitter API from python and sentiment analysis is performed using tools like NLTK and some GUI based tools like sentistrength (<http://sentistrength.wlv.ac.uk/>), text-processing.com(<http://text-processing.com/demo/sentiment/>

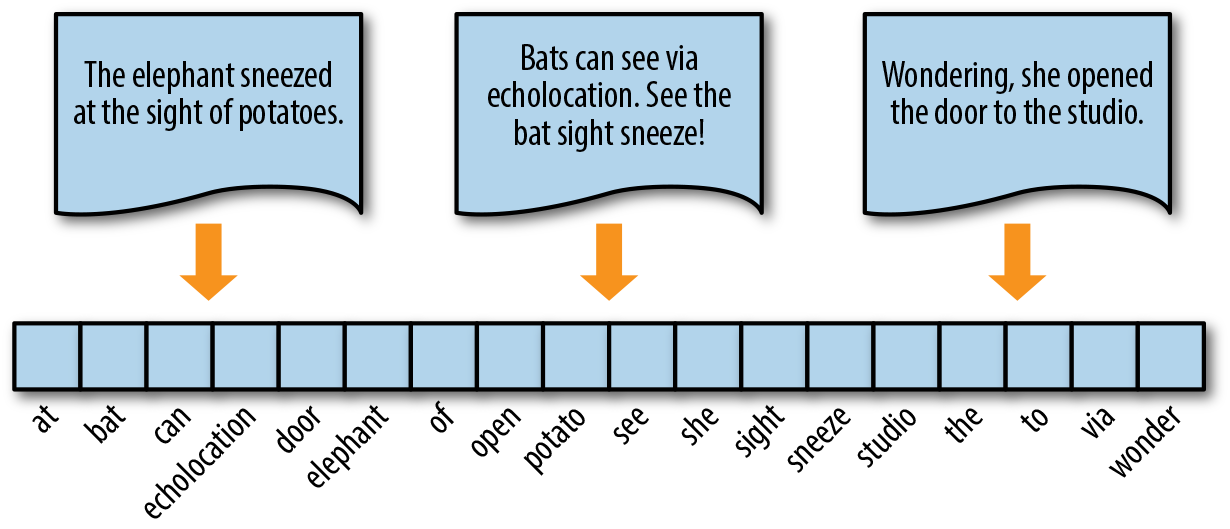
) etc... These tools can determine positive, negative and neutral sentiments for a tweet and these are analyzed to view public opinion on AI. A sample of 1000 tweets containing Artificial Intelligence keywords are collected and analyzed for this purpose.

**Figure 1.1** shows the basic text processing operations that will be carried out for sentiment analysis



**Figure 1.1 Basic text processing operations**

Tokenization and Vectorization is a must task before performing any sentiment analysis on the tweets, to vectorize a corpus with a bag-of-words (BOW) approach, we represent every document from the corpus as a vector whose length is equal to the vocabulary of the corpus. We can simplify the computation by sorting token positions of the vector into alphabetical order, as shown in **Figure1.**[**2**](https://www.oreilly.com/library/view/applied-text-analysis/9781491963036/ch04.html#atap_ch04_vector_encoding). Alternatively, we can keep a dictionary that maps tokens to vector positions. Either way, we arrive at a vector mapping of the corpus that enables us to uniquely represent every document.



**Figure 1.2 Vectorization of Documents**

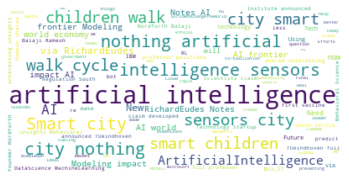
Source<https://www.oreilly.com/library/view/applied-text-analysis/9781491963036/ch04.html>

**Table 1.1** represents the samples dataset of tweets



**Table 1.1 Tweets data**

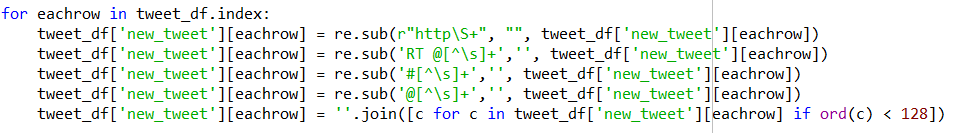
**Figure 1.3** shows the word cloud on AI tweets; word cloud talks more about the technology by itself rather than public opinions which infers that there are more tweets about AI technology than its opinion in the view of public.



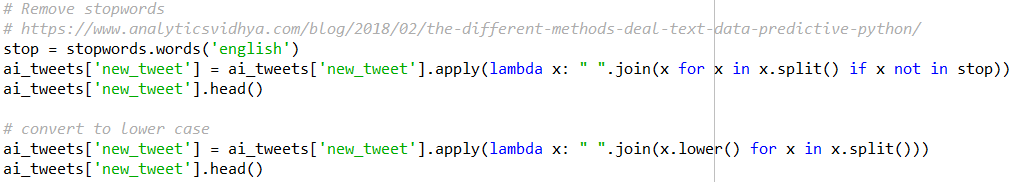
**Figure 1.3 Word Cloud on AI Tweets**

**Data Munging**

Most of the noise occurring in tweets such as links, users, hashtag and emojis can be removed using regular expression as these are not required for sentiment analysis.



It is required to convert all the document(tweets) to convert into lower case to give right frequency count for positive and negative words. Also, the stopwords are removed from the document which doesnot have any impact on positivity or negativity of the document and to reduce the dimensions and noise



### **Models/Tools**

In this exercise, tools used to vectorize the documents are as follows

* NLTK
* Sklearn

Vector encoding methods that are used for this analysis with the help of using NLTK and Sklearn packages are as follows

* **Frequency**
* **Term Frequency**
* **TF-IDF**

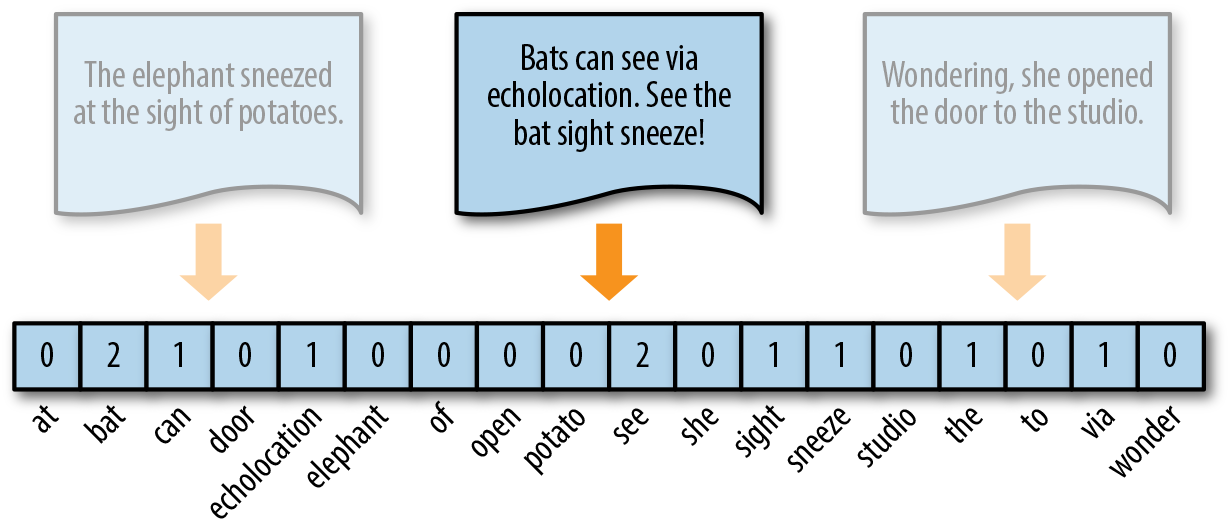
A small corpus of 7 sentences are used as an example to see how it works for vectorization across different tools. **Table 2.1** represents the sample data for the purpose of understanding vectorization in NLTK and Sklearn



**Table 2.1 Sample data to understand vectorization**

#### **Frequency Encoding**

The simplest vector encoding model is to simply fill in the vector with the frequency of each word as it appears in the document. In this encoding scheme, each document is represented as the multiset of the tokens that compose it and the value for each word position in the vector is its count. This representation can either be a straight count (integer) encoding as shown in **Figure 2.1** or a normalized encoding where each word is weighted by the total number of words in the document.



**Figure 2.1 Frequency Encoding**

Vectors can become extremely sparse, particularly as vocabularies get larger, which can have a significant impact on the speed and performance of machine learning models. For very large corpora, it is recommended to use the Scikit-Learn HashingVectorizer, which uses a hashing trick to find the token string name to feature index mapping. This means it uses very low memory and scales to large datasets as it does not need to store the entire vocabulary and it is faster to pickle and fit since there is no state. However, there is no inverse transform (from vector to text), there can be collisions, and there is no inverse document frequency weighting.

#### **TF Encoding**

Term Frequency is defined as how frequently the word appear in the document or corpus. As each sentence is not the same length so it may be possible a word appears in long sentence occur more time as compared to word appear in sorter sentence. Term frequency can be defined as:

**tf(t,d) = count of t in d / number of words in d**

#### **TF-IDF Encoding**

The bag-of-words representations that we have explored so far only describe a document in a standalone fashion, not considering the context of the corpus. A better approach would be to consider the relative frequency or rareness of tokens in the document against their frequency in other documents. The central insight is that meaning is most likely encoded in the rarer terms from a document. For example, in a corpus of sports text, tokens such as “umpire,” “base,” and “dugout” appear more frequently in documents that discuss baseball, while other tokens that appear frequently throughout the corpus, like “run,” “score,” and “play,” are less important.

Document Frequency measures the importance of document in whole set of corpus, this is very similar to TF. The only difference is that TF is frequency counter for a term t in document d, whereas DF is the count of occurrences of term t in the document set N. In other words, DF is the number of documents in which the word is present. We consider one occurrence if the term consists in the document at least once, we do not need to know the number of times the term is present.

**df(t) = occurrence of t in documents**

To keep this also in a range, we normalize by dividing with the total number of documents. To know the informativeness of a term, and DF is the exact inverse of it. Inverse Document Frequency (IDF) is the inverse of the document frequency which measures the informativeness of term t. It will be very low for the most occurring words such as stop words (because stop words such as “is” is present in almost all of the documents, and N/df will give a very low value to that word). This finally gives what we want, a relative weightage.

**idf(t) = N/df**

Now there are few other problems with the IDF, in case of a large corpus, say 10,000, the IDF value explodes. So, to dampen the effect we take log of IDF.

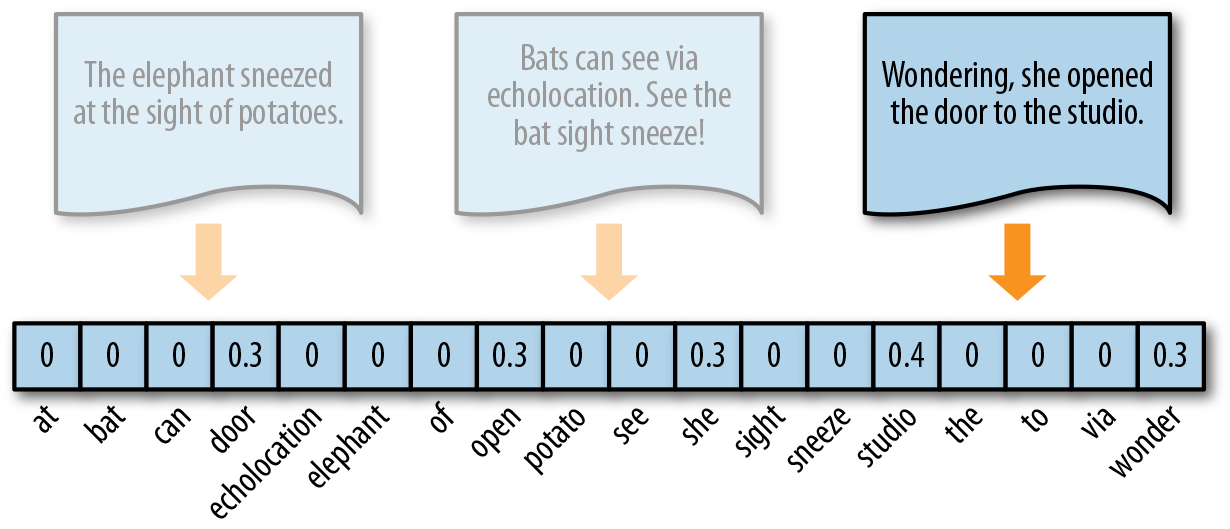
During the query time, when a word which is not in vocab occurs, the df will be 0. As we cannot divide by 0, we smoothen the value by adding 1 to the denominator.

**idf(t) = log(N/(df + 1))**

Finally, by taking a multiplicative value of TF and IDF, TF-IDF score is measured.

**tf-idf(t, d) = tf(t, d) \* log(N/(df + 1))**

As shown in **Figure 2.5**, where the token studio has a higher relevance to this document since it only appears there.



**Figure 2.5 TF-IDF Encoding**

Source <https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089>

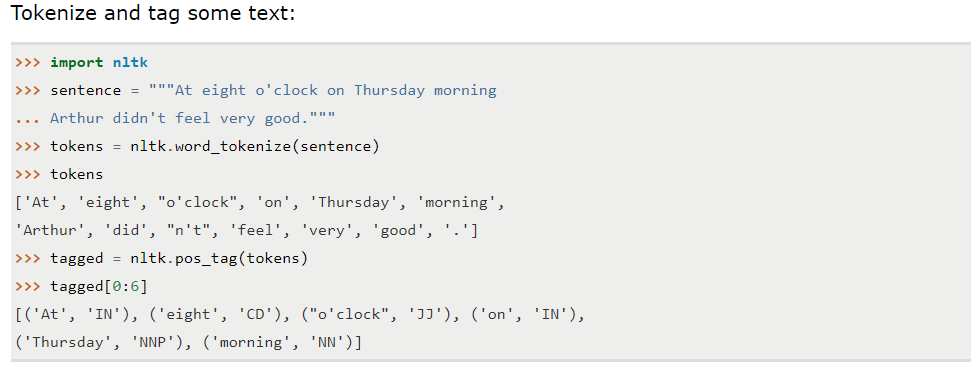
#### **NLTK (Natural Language Toolkit)**

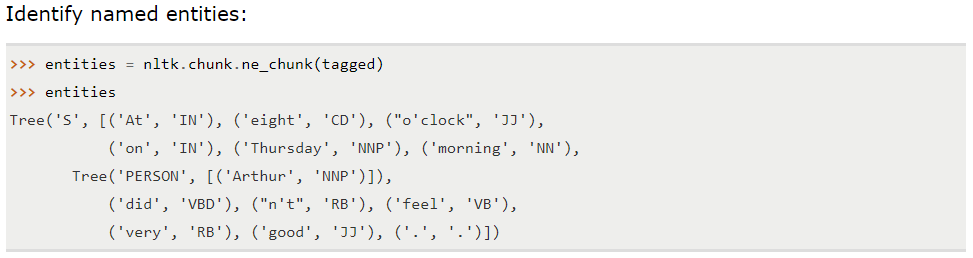
NLTK is an excellent library for machine-learning based NLP, written in Python by experts from both academia and industry. Python allows you to create rich data applications rapidly, iterating on hypotheses. The combination of Python + NLTK means that you can easily add language-aware data products to your larger analytical workflows and applications. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, vectorization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum. It was developed by Steven Bird and Edward Loper in the Department of Computer and Information Science at the University of Pennsylvania. NLTK includes graphical demonstrations and sample data. It is accompanied by a book that explains the underlying concepts behind the language processing tasks supported by the toolkit, plus a cookbook.

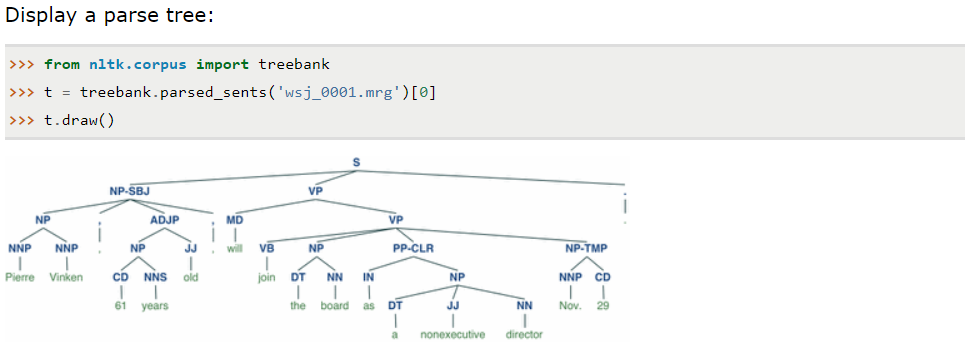
NLTK is intended to support research and teaching in NLP or closely related areas, including empirical linguistics, cognitive science, artificial intelligence, information retrieval, and machine learning. NLTK has been used successfully as a teaching tool, as an individual study tool, and as a platform for prototyping and building research systems. There are 32 universities in the US and 25 countries using NLTK in their courses. NLTK supports classification, tokenization, stemming, tagging, parsing, and semantic reasoning functionalities.

Source: <https://en.wikipedia.org/wiki/Natural_Language_Toolkit>

Some of the functions of NLTK are shown below

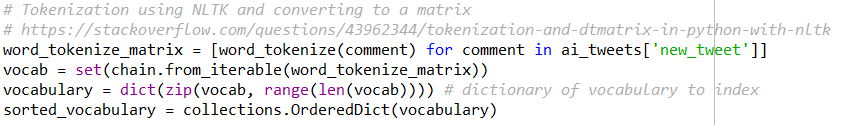


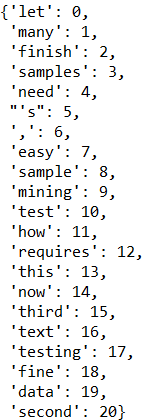




**Vectorization using NLTK**

NLTK expects features as a dict object whose keys are the names of the features and whose values are boolean or numeric. To encode documents in this way, individual tokens from the sentences are extracted and a dictionary is created using dict object whose keys are the tokens in the document and values represents the index of the tokens.

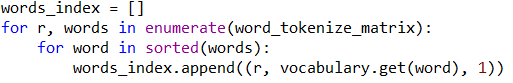


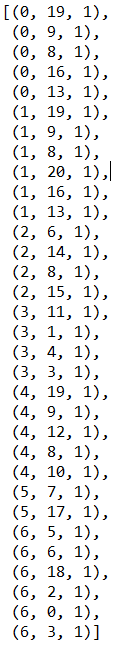


**Figure 2.1 Tokens represented in key, value pair**

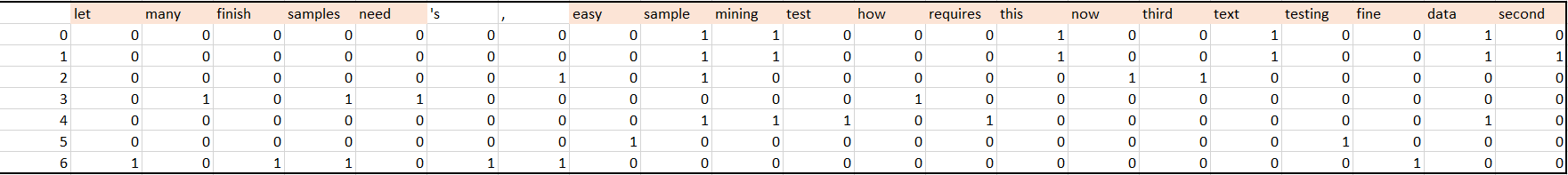
#### **Frequency Encoding**

A map of tokens and the number of occurrences to every item in the corpus using the above indexed dict object is created which represents the vectorized documents as shown in **Figure 2.2.** A Data frame or a sparse matrix object as shown in **Figure 2.3** can be then created from this map which can be used for other purpose.





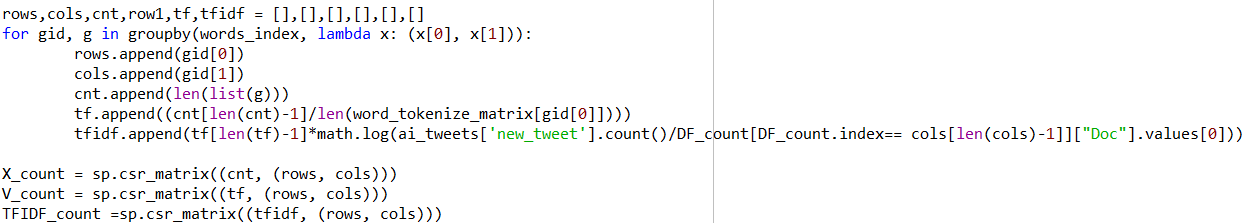
**Figure 2.2 Token Index, Document Index and Frequency map (vectorized document)**

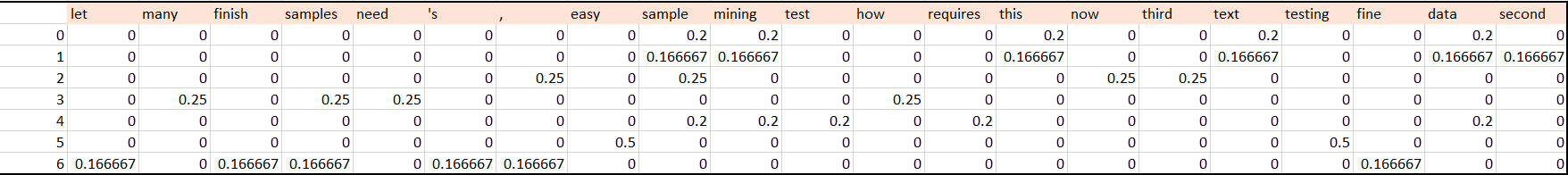


**Figure 2.3 Frequency Encoded data frame of corpus**

#### **TF Encoding**

Term Frequency matrix can be calculated by using the frequency encoded matrix/df and by deriving the total number of terms in a sentence/document as shown in the **Figure 2.4**

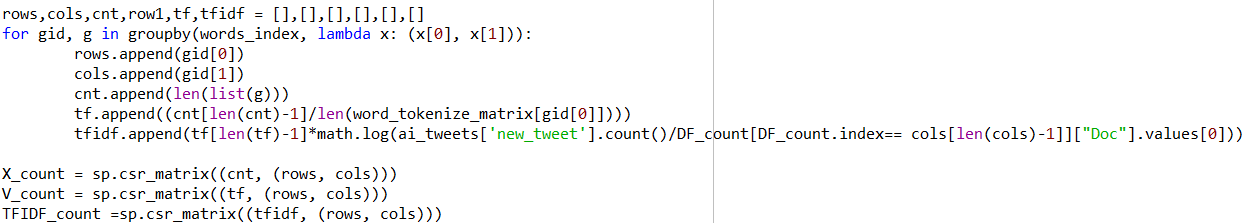


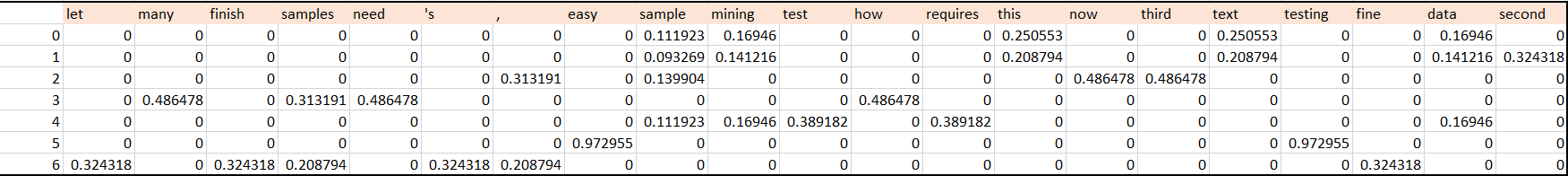


**Figure 2.4 TF Encoded data frame of corpus**

#### **TF-IDF Encoding**

TF-IDF matrix can be calculated by using the frequency encoded matrix/df and by deriving the total number of terms in a sentence/document as shown in the **Figure 2.4**





**Figure 2.6 TF-IDF Encoded data frame of corpus**

#### **Scikit-Learn (sklearn)**

Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language.It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

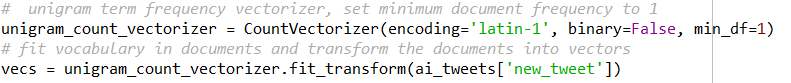
The scikit-learn project started as scikits.learn, a Google Summer of Code project by David Cournapeau. Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately-developed and distributed third-party extension to SciPy. The original codebase was later rewritten by other developers. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel, all from the French Institute for Research in Computer Science and Automation in Rocquencourt, France, took leadership of the project and made the first public release on February the 1st 2010. Of the various scikits, scikit-learn as well as scikit-image were described as "well-maintained and popular" in November 2012.

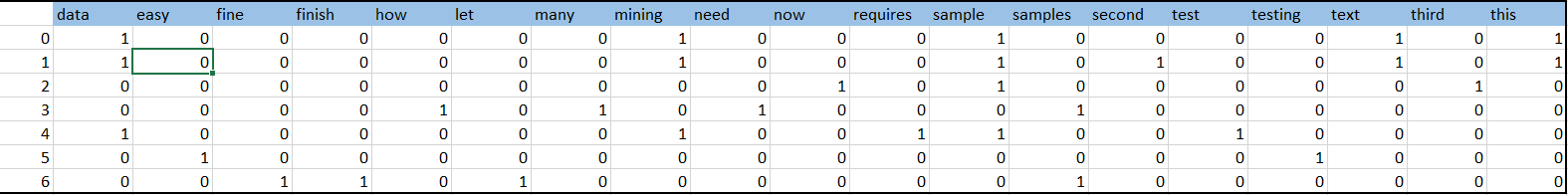
Source: <https://en.wikipedia.org/wiki/Scikit-learn>

**Vectorization using sklearn**

#### **Frequency Encoding**

The CountVectorizer transformer from the sklearn.feature\_extractionmodel has its own internal tokenization and normalization methods. The fit method of the vectorizer expects an iterable or list of strings or file objects and creates a dictionary of the vocabulary on the corpus. When transform is called, each individual document is transformed into a sparse array whose index tuple is the row (the document ID) and the token ID from the dictionary, and whose value is the count:



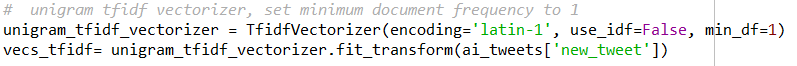


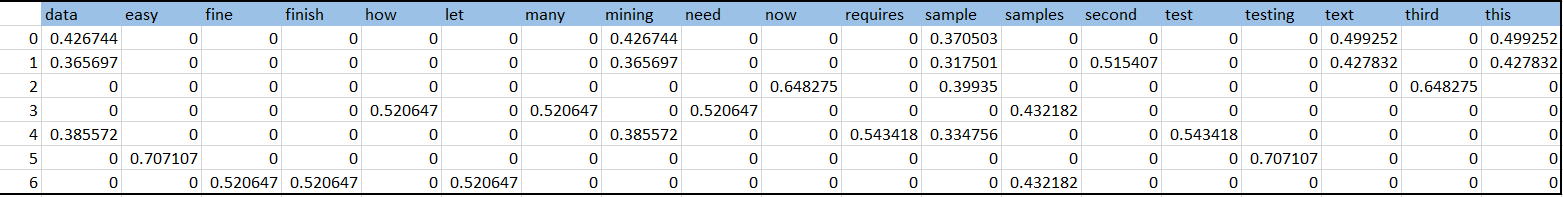
**Figure 2.7 Frequency Encoded data frame of corpus**

#### **TF-IDF Encoding**

Scikit-Learn provides a transformer called the TfidfVectorizer in the module called feature\_extraction.text for vectorizing documents with TF–IDF scores. Under the hood, the TfidfVectorizer uses the CountVectorizer estimator we used to produce the bag-of-words encoding to count occurrences of tokens, followed by a TfidfTransformer, which normalizes these occurrence counts by the inverse document frequency.

The input for a TfidfVectorizer is expected to be a sequence of filenames, file-like objects, or strings that contain a collection of raw documents, similar to that of the CountVectorizer. As a result, a default tokenization and preprocessing method is applied unless other functions are specified. The vectorizer returns a sparse matrix representation in the form of ((doc, term), tfidf) where each key is a document and term pair and the value is the TF–IDF score.





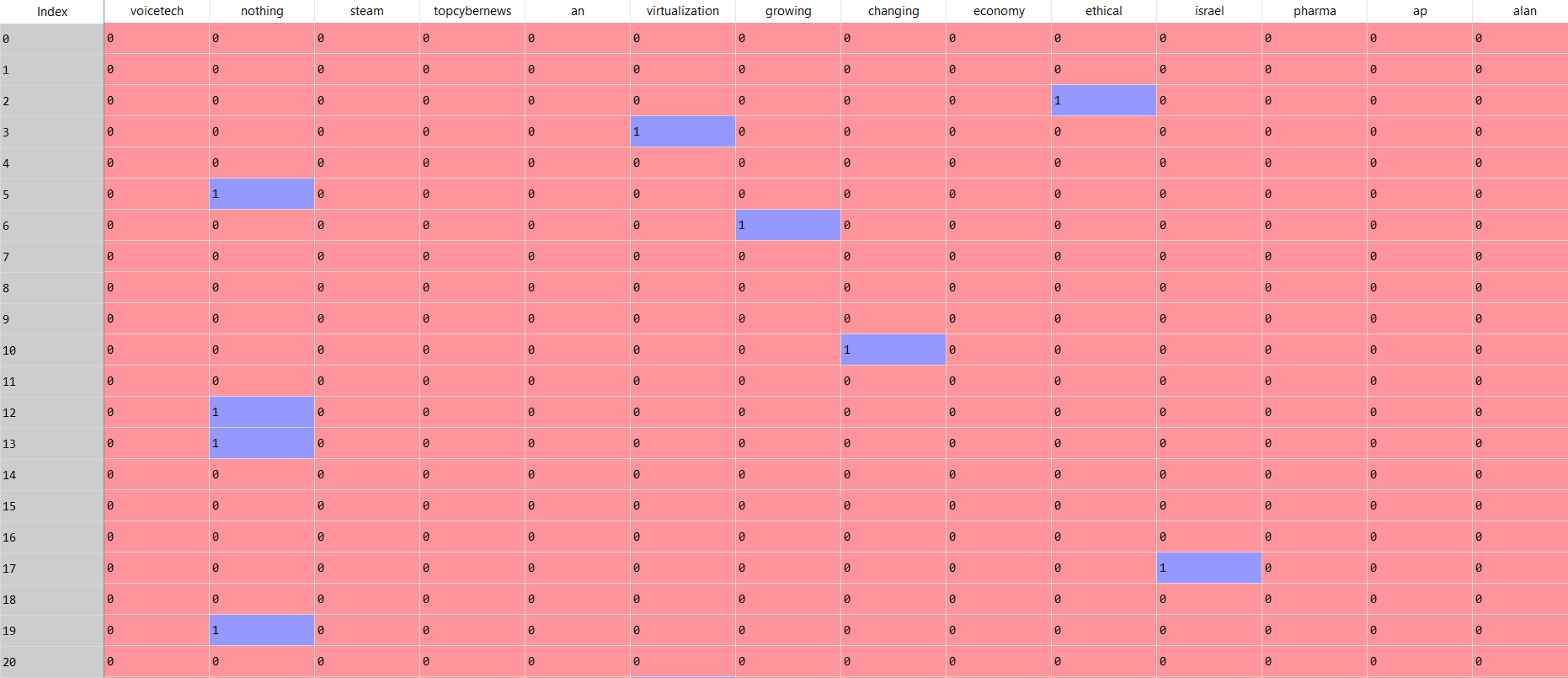
**Figure 2.8 TF-IDF Encoded data frame of corpus**

## **Results**

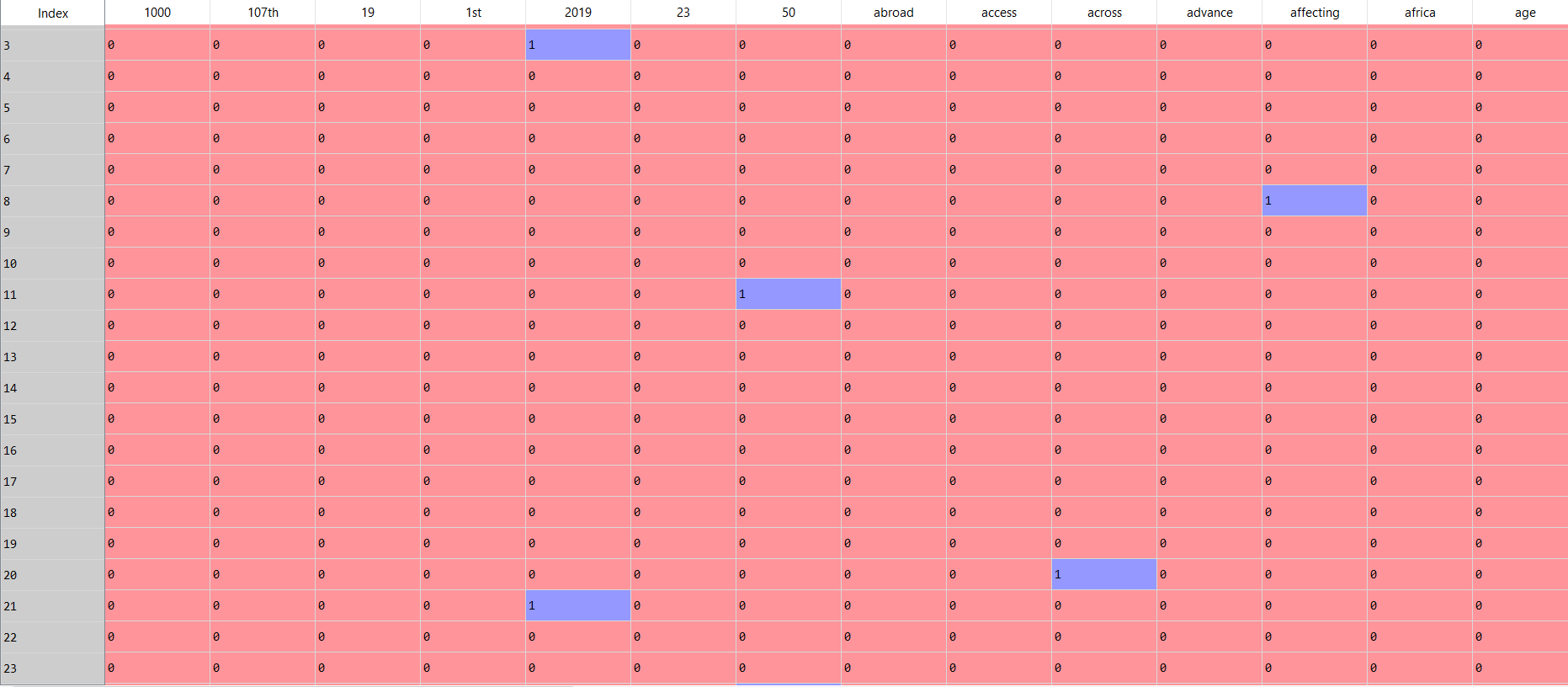
Vectorization results for tweets using NLTK and sklearn package are given below

#### **Frequency Encoding**

Frequency encoded matrix is derived for 1000 tweets using NLTK and sklearn package and the outcome of the matrix are shown in **Figure 3.1 and 3.2.** The matrix is very much scattered and heavily dispersed as there are large number of documents and fewer words matching across documents and hence most of the values are 0 representing the absence of a term in that document and any number above zero is the frequency of occurrences of that term in that document.



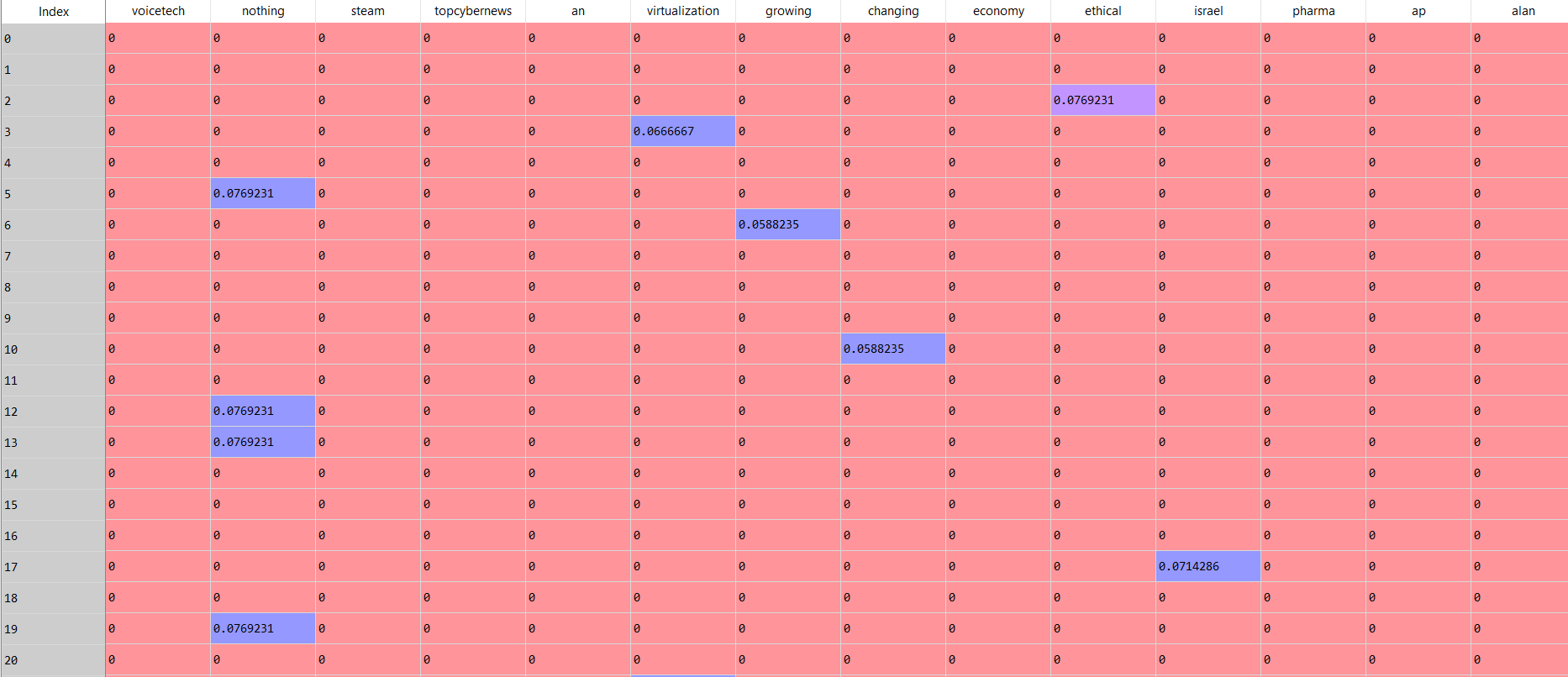
**Figure 3.1 Frequency Encoding using NLTK**



**Figure 3.2 Frequency Encoding using sklearn**

#### **Term Frequency (TF) Encoding**

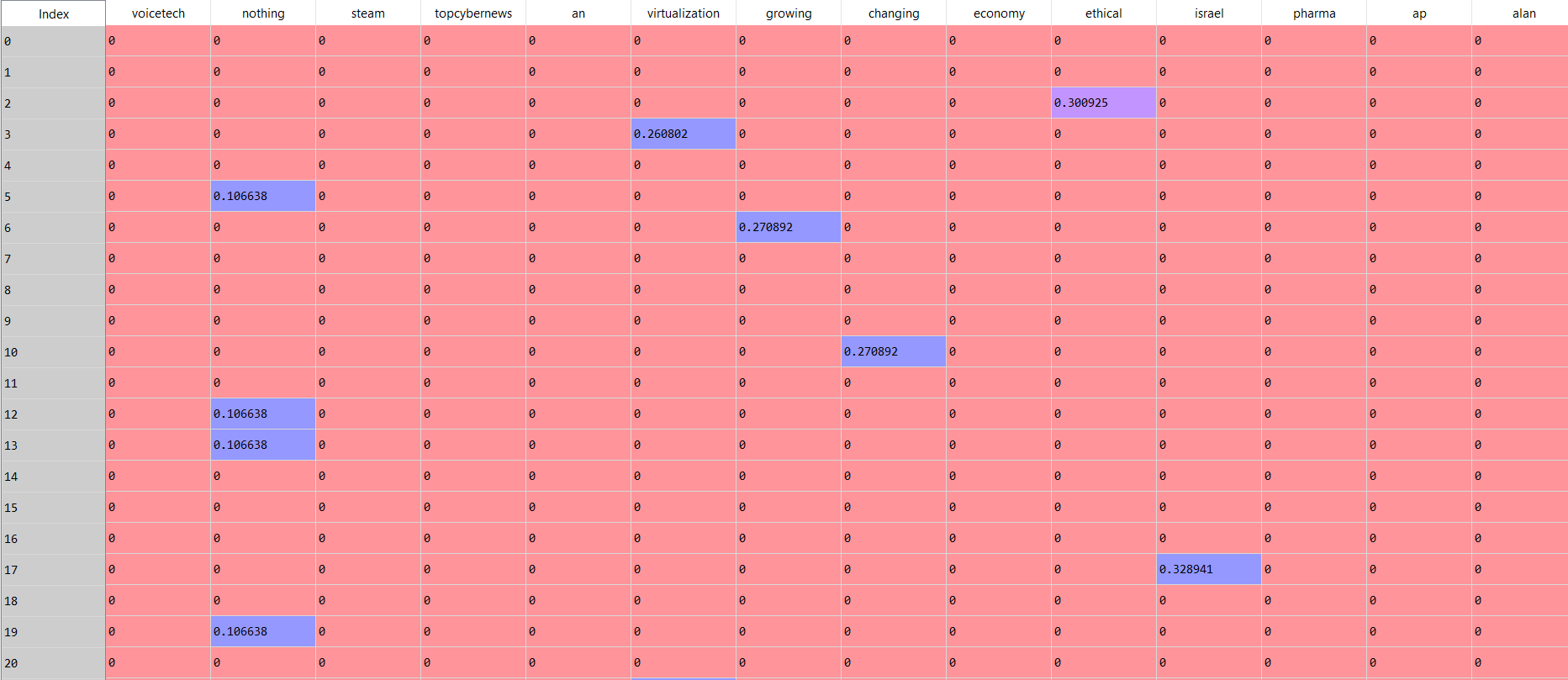
TF encoded matrix is derived for 1000 tweets using NLTK but not with sklearn package (due to some technical difficulties) and the outcome of the matrix are shown in **Figure 3.3.** The matrix is very much scattered and heavily dispersed as there are large number of documents and fewer words matching across documents and hence most of the values are 0 representing the absence of a term in that document and any number above zero is the ratio of frequency of occurrences of that term to the total number of terms in that document.



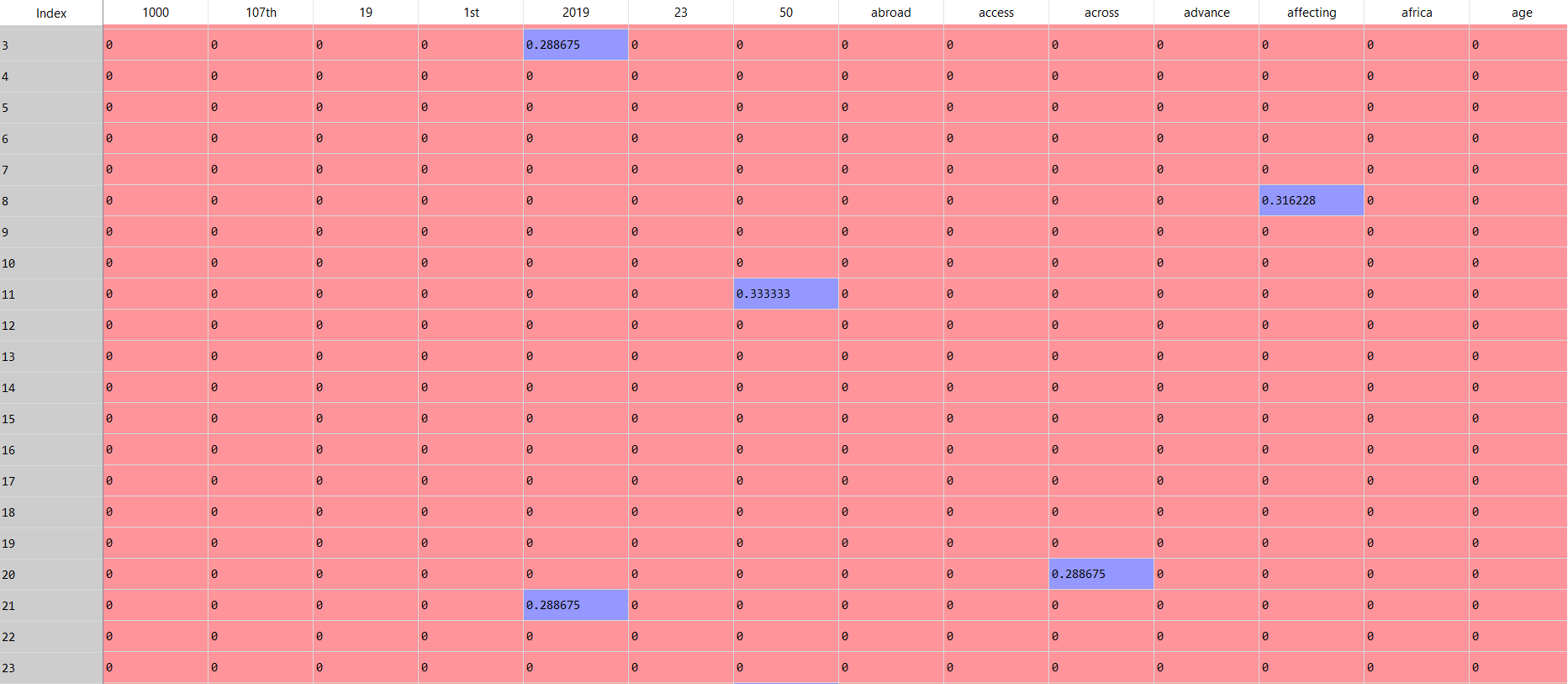
**Figure 3.3 TF Encoding using NLTK**

#### **TF-IDF Encoding**

TF-IDF encoded matrix is derived for 1000 tweets using NLTK and sklearn package and the outcome of the matrix are shown in **Figure 3.4 and 3.5.** The matrix is very much scattered and heavily dispersed as there are large number of documents and fewer words matching across documents and hence most of the values are 0 representing the absence of a term in that document. Any number above zero and close to zero represents the most common word which occurs in every document like (a, the) by heavily penalizing their presence in the document. Rare words occurring in fewer documents are given more importance and hence their value shows close to 1. From the figure it appears that the word “**nothing**” occurs in more than 1 document and has the value of 0.10 whereas the word “**ethical or Israel**” occurs in 1 document and has the value of 0.3 in NLTK encoded matrix. We are seeing similar pattern from the matrix produced by sklearn package.



**Figure 3.4 TF-IDF Encoding using NLTK**

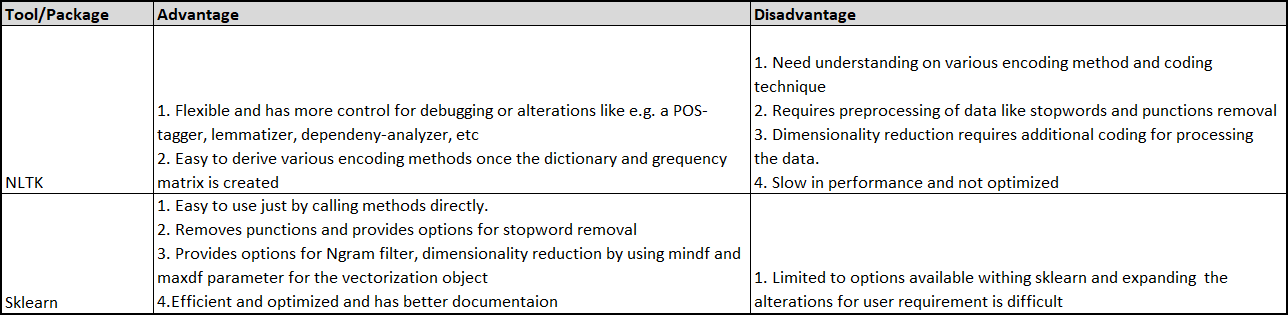


**Figure 3.5 TF-IDF Encoding using sklearn**

## **Conclusion**

Human language is elaborate, with nearly infinite grammatical variations, misspellings, slang and other challenges making accurate automated analysis of natural language quite difficult. English is difficult to analyze because of its complicated sentence structure. Since the beginning of the brief history of Natural Language Processing (NLP), there has been the need to transform text into something a machine can understand. That is, transforming text into a meaningful vector (or array) of numbers. The standard way of doing this in the deep learning and text mining is to use vectorization techniques.

**Table 4.1** Evaluates the vectorization techniques using NLTK and sklearn.



**Table 4.1 Evaluation of different tools for vectorization**

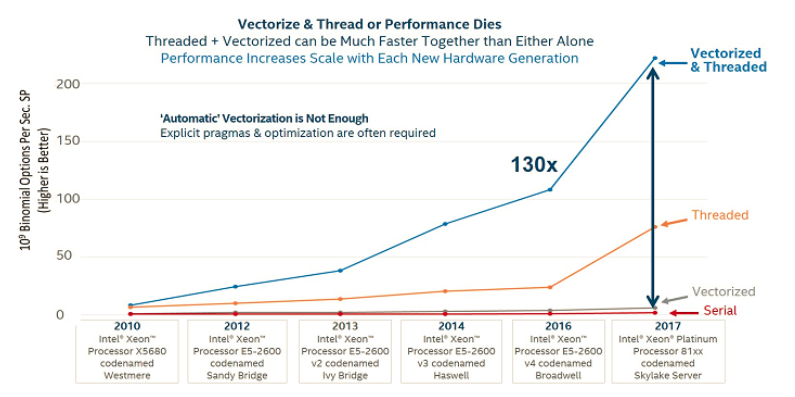
**NLTK** is specialized on gathering and classifying unstructured texts. If you need e.g. a POS-tagger, lemmatizer, dependeny-analyzer, etc, you'll find them there, and sometimes nowhere else. It offers a quit broad range of tools developed mainly in academic research. But: most often it is not very well optimized - involving NLTK libraries often means to accept a huge performance loss. If you do text-gathering or -preprocessing, its fine to begin with - until you found some faster alternatives.

**SKLEARN** offers a very systematic, efficient framework for machine-learning, analyzing, ensemble methods, evaluation and validation, and hyper-parameter optimization. It is very well documented (with a lot of ready to use recipes and examples), well optimized, and covers a broad range of ‘state of the art’ machine learning and statistical methods, the latter especially for evaluation purposes. Due to its integrity, it is ideal to start learning ‘machine learning’.

Vectorization offers potential speedups in codes with significant array-based computations Click to Tweet. Vectorization offers potential speedups in codes with significant array-based computations—speedups that amplify the improved performance obtained through higher-level, parallel computations using threads and distributed execution on clusters. Key features for vectorization include tunable array sizes to reflect various processor cache and instruction capabilities and stride-1 accesses within inner loops.

The importance of vectorization to increase performance will continue to grow as hardware designers extend the number of vector registers to eight doubles (and hopefully more) on emerging processors to overcome plateauing clock rates and thread scalability.

**Figure 4.1** shows how Intel could improve the performance of their systems after vectorization.



**Figure 4.1 Comparison of Vectorization with performance @ Intel**

Source: <http://www.datascienceblog.pw/the-importance-of-vectorization-resurfaces.html>