Text similiraty

31 Mart 2020 Salı 17:04

How similar are these two words?", "How similar are these two sentences?" , "How similar are these two documents?"

From <https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50>

I have found that Levenshtein and Jaro Winkler are great for small differences betwen strings such as:

- Spelling mistakes; or
- **ö** instead of **o** in a persons name.

However when comparing something like article titles where significant chunks of the text would be the same but with "noise" around the edges, Smith-Waterman-Gotoh has been fantastic:

compare these 2 titles (that are the same but worded differently from different sources):
An **endonuclease from Escherichia coli that introduces single polynucleotide chain scissions in ultraviolet -irradiated DNA**
Endonuclease III: **An Endonuclease from Escherichia coli That Introduces Single Polynucleotide Chain Scissions in Ultraviolet-Irradiated DNA**
This site that provides algorithm comparison of the strings shows:

- Levenshtein: 81
- Smith-Waterman Gotoh **94**
- Jaro Winkler 78

Jaro Winkler and Levenshtein are not as competent as Smith Waterman Gotoh in detecting the similarity. If we compare two titles that are not the same article, but have some matching text:
Fat metabolism in higher plants. The function of acyl thioesterases in the metabolism of **acyl-coenzymes** A and **acyl-acyl carrier protein**s
Fat metabolism in higher plants. The determination of **acyl-acyl carrier protein** and **acyl coenzyme** A in a complex lipid mixture
Jaro Winkler gives a false positive, but Smith Waterman Gotoh does not:

- Levenshtein: 54
- Smith-Waterman Gotoh **49**
- Jaro Winkler 89

From <https://stackoverflow.com/questions/9453731/how-to-calculate-distance-similarity-measure-of-given-2-strings>

1-leventshtein:yazım hataları tespiti için, genelde tekil kelimelerde
edit distance da denir: The edit distance is the number of characters that need to be substituted, inserted, or deleted, to transform s1 into s2

2-cosine:cümle ve doküman karşılaştırması için. 3 aşaması var
-Remove punctuations from a given string
-Lowercase the string
-Remove stopwords

Text similarity has to determine how 'close' two pieces of text are both in surface closeness **[lexical similarity]** and meaning **[semantic similarity]**.

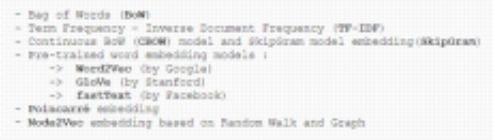
For instance, how similar are the phrases ***"the cat ate the mouse"*** with ***"the mouse ate the cat food"*** by just looking at the words?

Since **differences in word order** often go hand in hand with **differences in meaning** (compare the dog bites the man with the man bites the dog), we'd like our sentence embeddings to **be sensitive to this variation**.

But lucky we are, word vectors have evolved over the years **to know the difference** between record the play vs play the record

- **Jaccard Similarity** ☹☹☹
- Different embeddings **+ K-means** ☹☹
- Different embeddings **+ Cosine Similarity** ☹
- **Word2Vec + Smooth Inverse Frequency + Cosine Similarity** ☺
- Different embeddings **+LSI + Cosine Similarity** ☹
- Different embeddings **+ LDA + Jensen-Shannon distance** ☺
- Different embeddings **+ Word Mover Distance** ☺☺
- Different embeddings **+ Variational Auto Encoder (VAE)** ☺ ☺
- Different embeddings **+ Universal sentence encoder** ☺☺
- Different embeddings **+ Siamese Manhattan LSTM** ☺☺☺
- BERT embeddings + Cosine Similarity ♥
- **Knowledge-based Measures** ♥

Different Embeddingler:



Edit distance

01 Nisan 2020 Çarşamba 09:50

edit distance is a way of quantifying how dissimilar two strings (e.g., words) are to one another by counting the minimum number of operations required to transform one string into the other.

From <https://en.wikipedia.org/wiki/Edit_distance>

Klavye oto correction önerileri, DNA dizilimlerinde v.s kullanılır.

Different types of edit distance allow different sets of string operations. For instance:

- The Levenshtein distance allows deletion, insertion and substitution.
- The Longest common subsequence (LCS) distance allows only insertion and deletion, not substitution.
- The Hamming distance allows only substitution, hence, it only applies to strings of the same length.
- The Damerau–Levenshtein distance allows insertion, deletion, substitution, and the transposition of two adjacent characters.
- The Jaro distance allows only transposition. (transposition: *co**st*** → *co**ts)***

From <https://en.wikipedia.org/wiki/Edit_distance>

The Levenshtein distance between "kitten" and "sitting" is 3. A minimal edit script that transforms the former into the latter is:

1. **k**itten → **s**itten (substitute "s" for "k")
2. sitt**e**n → sitt**i**n (substitute "i" for "e")
3. sittin → sittin**g** (insert "g" at the end)

LCS distance (insertions and deletions only) gives a different distance and minimal edit script:

4. **k**itten → itten (delete "k" at 0)
5. itten → **s**itten (insert "s" at 0)
6. sitt**e**n → sittn (delete "e" at 4)
7. sittn → sitt**i**n (insert "i" at 4)
8. sittin → sittin**g** (insert "g" at 6)

for a total cost/distance of 5 operations.

From <https://en.wikipedia.org/wiki/Edit_distance>

Levenshteinn

01 Nisan 2020 Çarşamba 09:44

Informally, the Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other.

From <https://en.wikipedia.org/wiki/Levenshtein_distance>

kitt**e**n and **s**itt**i**n**g** have a distance of 3.

1. **k**itten → **s**itten (substitution of "s" for "k")
2. sitt**e**n → sitt**i**n (substitution of "i" for "e")
3. sittin → sittin**g** (insertion of "g" at the end).

From <https://en.wikipedia.org/wiki/String_metric>

Upper and lower bounds[edit]

The Levenshtein distance has several simple upper and lower bounds. These include:

- It is at least the difference of the sizes of the two strings.
- It is at most the length of the longer string.
- It is zero if and only if the strings are equal.
- If the strings are the same size, the Hamming distance is an upper bound on the Levenshtein distance.
- The Levenshtein distance between two strings is no greater than the sum of their Levenshtein distances from a third string (triangle inequality).

An example where the Levenshtein distance between two strings of the same length is strictly less than the Hamming distance is given by the pair "flaw" and "lawn". Here the Levenshtein distance equals 2 (delete "f" from the front; insert "n" at the end). The Hamming distance is 4.

From <https://en.wikipedia.org/wiki/Levenshtein_distance>

Hamming

01 Nisan 2020 Çarşamba 10:48

In information theory, the **Hamming distance** between two strings of equal length is the number of positions at which the corresponding symbols are different. In other words, it measures the minimum number of *substitutions* required to change one string into the other, or the minimum number of *errors* that could have transformed one string into the other. In a more general context, the Hamming distance is one of several string metrics for measuring the edit distance between two sequences.

From <https://en.wikipedia.org/wiki/Hamming_distance>

The Hamming distance between:

- "**karolin**" and "**kathrin**" is 3.
- "**karolin**" and "**kerstin**" is 3.
- **1011101** and **1001001** is 2.
- **2173896** and **2233796** is 3.

From <https://en.wikipedia.org/wiki/Hamming_distance>

Damerau–Levenshtein distance

01 Nisan 2020 Çarşamba 09:48

Informally, the Damerau–Levenshtein distance between two words is the minimum number of operations (consisting of insertions, deletions or substitutions of a single character, or transposition of two adjacent characters) required to change one word into the other.

From <https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance>

In his seminal paper,[5] Damerau stated that more than 80% of all human misspellings can be expressed by a single error of one of the four types. Damerau's paper considered only misspellings that could be corrected with at most one edit operation. While the original motivation was to measure distance between human misspellings to improve applications such as spell checkers, Damerau–Levenshtein distance has also seen uses in biology to measure the variation between protein sequences.

From <https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance>

Jaro winkler

01 Nisan 2020 Çarşamba 09:46

Only transposition yani komşu karkterlerin eyr değişimine izin var, bu harekt 1 işlemdir, her ne kadar iki karater yer değşitirmiş olsa da. Özellikle spelling hataları için olsa gerek.

JaroWinklerDist("MARTHA","MARHTA") =

$$d_j = \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) = \frac{1}{3}\left(\frac{6}{6} + \frac{6}{6} + \frac{6-\frac{2}{2}}{6}\right) = 0.944$$

- m is the number of *matching characters*;
- t is half the number of *transpositions*(`"MARTHA"[3]!=H, "MARHTA"[3]!=T`).

Si=size of string

Although often referred to as a *distance metric*, the Jaro–Winkler distance is not a metric in the mathematical sense of that term because it does not obey the triangle inequality.

From <https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance>

Cosine similarity

01 Nisan 2020 Çarşamba 10:39

It is important to note, however, that this is not a proper distance metric as it does not have the triangle inequality property—or, more formally, the Schwarz inequality—and it violates the coincidence axiom; to repair the triangle inequality property while maintaining the same ordering, it is necessary to convert to angular distance

From <https://en.wikipedia.org/wiki/Cosine_similarity>

Soft cosine measure[edit]
A soft cosine or ("soft" similarity) between two vectors considers similarities between pairs of features. [1][2] The traditional cosine similarity considers the vector space model (VSM) features as independent or completely different, while the soft cosine measure proposes considering the similarity of features in VSM, which help generalize the concept of cosine (and soft cosine) as well as the idea of (soft) similarity.
For example, in the field of natural language processing (NLP) the similarity among features is quite intuitive. Features such as words, *n*-grams, or syntactic *n*-grams[3] can be quite similar, though formally they are considered as different features in the VSM. For example, words "play" and "game" are different words and thus mapped to different points in VSM; yet they are semantically related. In case of *n*-grams or syntactic *n*-grams, Levenshtein distance can be applied (in fact, Levenshtein distance can be applied to words as well).
For calculating soft cosine, the matrix **s** is used to indicate similarity between features. It can be calculated through Levenshtein distance, WordNet similarity, or other similarity measures. Then we just multiply by this matrix.

From <https://en.wikipedia.org/wiki/Cosine_similarity>

With cosine similarity, we need to convert sentences into vectors. One way to do that is to use **bag of words with either TF** (term frequency) **or TF-IDF** (term frequency- inverse document frequency). The choice of TF or TF-IDF depends on application and is immaterial to how cosine similarity is actually performed — which just needs vectors. *TF is good for text similarity in general, but TF-IDF is good for search query relevance.*

Another way is to use Word2Vec or our own custom word embeddings to convert words into vectors

From <https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50>

There are two main difference between tf/ tf-idf with bag of words and word embeddings:
1. tf/ tf-idf creates one number per word, word embeddings typically creates one vector per word.
2. tf/ tf-idf is good for classification documents as a whole, but word embeddings is good for identifying contextual content.

From <https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50>

Jaccard

01 Nisan 2020 Çarşamba 12:53

The Jaccard similarity index (sometimes called the Jaccard similarity *coefficient*) compares members for two sets to see which members are shared and which are distinct. It's a measure of similarity for the two sets of data, with a range from 0% to 100%. The higher the percentage, the more similar the two populations. Although it's easy to interpret, it is extremely sensitive to small samples sizes and may give erroneous results, especially with very small samples or data sets with missing observations.

From <https://www.statisticshowto.datasciencecentral.com/jaccard-index/>

Jaccard Index = (the number in both sets) / (the number in either set) * 100
J(X,Y) = |X∩Y| / |X∪Y|

From <https://www.statisticshowto.datasciencecentral.com/jaccard-index/>

In Steps, that's:

1. Count the number of members which are shared between both sets.
2. Count the total number of members in both sets (shared and un-shared).
3. Divide the number of shared members (1) by the total number of members (2).
4. Multiply the number you found in (3) by 100.

This percentage tells you how similar the two sets are.

- Two sets that share all members would be 100% similar. the closer to 100%, the more similarity (e.g. 90% is more similar than 89%).
- If they share no members, they are 0% similar.
- The midway point — 50% — means that the two sets share half of the members.

From <https://www.statisticshowto.datasciencecentral.com/jaccard-index/>

Examples

A simple example using set notation: How similar are these two sets?

- A = {0,1,2,5,6}
- B = {0,2,3,4,5,7,9}

Solution: J(A,B) = |A∩B| / |A∪B| = |{0,2,5}| / |{0,1,2,3,4,5,6,7,9}| = 3/9 = 0.3

From <https://www.statisticshowto.datasciencecentral.com/jaccard-index/>

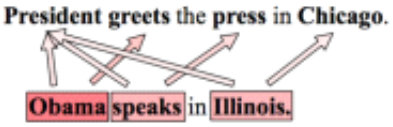
Differences between Jaccard Similarity and Cosine Similarity:

1. Jaccard similarity takes only **unique set of words** for each sentence / document while cosine similarity takes **total length of the vectors**. (these vectors could be made from bag of words term frequency or tf-idf)
2. This means that if you repeat the word "friend" in Sentence 1 several times, cosine similarity **changes** but Jaccard similarity does not. For ex, if the word "friend" is repeated in the first sentence 50 times, cosine similarity drops to 0.4 but Jaccard similarity remains at 0.5.
3. Jaccard similarity is good for cases where duplication does not matter, cosine similarity is good for cases where duplication matters while analyzing text similarity. For two product descriptions, it will be better to use Jaccard similarity as repetition of a word does not reduce their similarity.

From <https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50>

Let's take another example of two sentences having a similar meaning:

Sentence 1: President greets the press in Chicago
Sentence 2: Obama speaks in Illinois



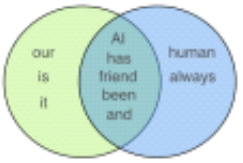
Why Jaccard Similarity is not efficient?

My 2 sentences **have no common words** and will have a **Jaccard score of 0**. This is a **terrible distance** score because **the 2 sentences have very similar meanings**. Here Jaccard similarity is neither able to capture **semantic similarity** nor **lexical semantic** of these two sentences.

Moreover, this approach has an **inherent flaw**. That is, as the **size of the document increases**, the number of common words tend to increase even if the documents talk about different topics.

Sentence 1: Al is our friend and it has been friendly
Sentence 2: Al and humans have always been friendly

In order to calculate similarity using Jaccard similarity, we will first perform **lemmatization** to reduce words to the same root word. In our case, "friend" and "friendly" will both become "friend", "has" and "have" will both become "has". Drawing a Venn diagram of the two sentences we get:



Venn Diagram of the two sentences for Jaccard similarity

For the above two sentences, we get Jaccard similarity of **5/(5+3+2) = 0.5** which is size of intersection of the set divided by total size of set.
The code for Jaccard similarity in Python is:

```
def get_jaccard_sim(str1, str2):
    a = set(str1.split())
    b = set(str2.split())
    c = a.intersection(b)
    return float(len(c)) / (len(a) + len(b) - len(c))
```

One thing to note here is that since we use sets, "friend" appeared twice in Sentence 1 but it did not affect our calculations — this will change with Cosine Similarity.

From <https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50>

Glossary

31 Mart 2020 Salı 17:28

Bag of words:
Tokenazation:
n-gram:
Tfidf:

Wordnet:
Kelime ağları, bir dildeki kelimeler arasındaki bağlantıları, eş anlam kümeleri oluşturarak ve bu kümeleri birbirine çeşitli anlamsal bağıntılar ile bağlayarak temsil eden bir çizge veri yapısıdır. Doğal dil işleme alanındaki en yaygın bilinen kelime ağı WordNet 1990 yılında İngilizce için oluşturulmuşken, Türkçe için en kapsamlı ağ, 2018 yılında oluşturulan KeNet'tir. Bildiğimiz kadarıyla, içinde 80000 eş anlam kümesi ve 25 farklı anlamsal bağlantı bulunan KeNet için şu ana kadar geliştirilen bir kullanıcı arayüzü yoktur. Bu çalışmada, KeNet çizgesinde, anlamsal bağlantıları kullanarak eş anlam kümeleri arasında çevrimiçi olarak gezinmeyi sağlayan bir arayüz sunuyoruz. Bu arayüz sayesinde, bir söz öbeği KeNet'te aranabilir ve eş anlam kümeleri arasındaki üst/alt anlam, parça-bütün ilişkileri gibi ilişkiler kullanılarak KeNet üzerinde gezilebilir. Ayrıca, herhangi bir eş anlam kümesinin, varsa, İngilizce karşılığının kimliği de görüntülenebilir ve bu kümeye WordNet'e ait internet sayfasından erişilebilir.

From <https://www.researchgate.net/publication/332786790_Turkce_Kelime_Agi_KeNet_icin_Arayuz>

---

a **string metric** (also known as a **string similarity metric** or **string distance function**)

Simplistic string metrics such as Levenshtein distance have expanded to include phonetic, token, grammatical and character-based methods of statistical comparisons

From <https://en.wikipedia.org/wiki/String_metric>

approximate string matching (often colloquially referred to as **fuzzy string searching**) is the technique of finding strings that match a pattern approximately (rather than exactly).

From <https://en.wikipedia.org/wiki/Approximate_string_matching>

Kod kıyas

31 Mart 2020 Salı 18:00

Python

- Difflib->sequencematcher: sanırım levenshtein yapıyor, c#taki levensthein ile karşılaştıralım
- sklearn.metrics.pairwise import cosine_similarity:
- nltk.edit_distance: levensthein

C#

- Simetrics:

Aşamalar

31 Mart 2020 Salı 18:10

1. **Removing punctuation**: When trying to train a machine learning model, it helps to reduce overfitting by removing punctuation (like !,* etc.). However, be careful to not remove something important, for example, question marks (?) help to recognize questions.
2. **Removing emojis:** Sometimes people attach emojis to words without spaces (for example: you❤) making it difficult to interpret such words. Removing emojis can help with such cases. Again, be careful while removing these as emojis might actually be really useful for tasks like sentiment analysis and topic classification.
3. **Removing stop words:** For tasks like data exploration and trend analysis, it might not be very useful to see common words like 'the', 'and', 'of' etc. The sklearn package actually has a collection of commonly used English stop words that we can use to remove these.
4. **Making all text lowercase:** This is the simplest way to normalize text. (after all, BeTTer and better do have the same semantic implication)
5. **Stemming words:** Another way of normalizing is by replacing derived words with their root form (eg: 'posting', 'posted', 'posts' are all replaced by 'post'). To stem words we use the PorterStemmer util provided by nltk.
6. **Extracting/Removing hashtags and mentions:** Hashtags and mentions can be very useful in identifying trends in your data. It helps to extract them out of your text and analyze them separately.

From <https://towardsdatascience.com/getting-started-with-natural-language-processing-nlp-2c482420cc05>

7- Parantez v.s uçurmak?

Duplikasyon kontrolü için ilaveten

- Alias ile replace
- Daha önce hayır dediklerimi almasın

Semantic similarity between sentences

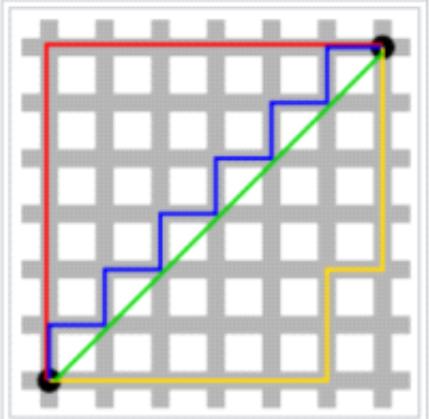
Given two sentences, the measurement determines how similar the meaning of two sentences is. The higher the score, the more similar the meaning of the two sentences.

Here are the steps for computing semantic similarity between two sentences:

- First, each sentence is partitioned into a list of tokens.
- Part-of-speech disambiguation (or tagging).
- Stemming words.
- Find the most appropriate sense for every word in a sentence (Word Sense Disambiguation).
- Finally, compute the similarity of the sentences based on the similarity of the pairs of words.

Manhatan-taxicab

01 Nisan 2020 Çarşamba 10:45



Taxicab geometry versus Euclidean distance: In taxicab geometry, the red, yellow, and blue paths all have the same shortest path length of 12. In Euclidean geometry, the green line has length $6\sqrt{2} \approx 8.49$ and is the unique shortest path.

A **taxicab geometry** is a form of geometry in which the usual distance function or metric of Euclidean geometry is replaced by a new metric in which the distance between two points is the sum of the absolute differences of their Cartesian coordinates. The **taxicab metric** is also known as **rectilinear distance**, L_1 **distance**, L^1 **distance** or **norm** (see L^p space), **snake distance**, **city block distance**, **Manhattan distance** or **Manhattan length**, with corresponding variations in the name of the geometry.[1] The latter names allude to the grid layout of most streets on the island of Manhattan, which causes the shortest path a car could take between two intersections in the borough to have length equal to the intersections' distance in taxicab geometry. The geometry has been used in regression analysis since the 18th century, and today is often referred to as LASSO. The geometric interpretation dates to non-Euclidean geometry of the 19th century and is due to Hermann Minkowski.

From <https://en.wikipedia.org/wiki/Taxicab_geometry>

TF-IDF

01 Nisan 2020 Çarşamba 12:57

short for **term frequency–inverse document frequency**, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

From <https://en.wikipedia.org/wiki/Tf%E2%80%93idf>

Wordnet

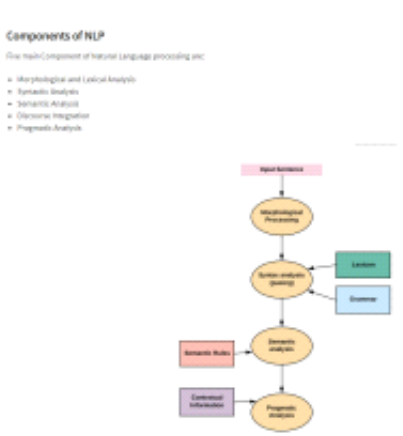
03 Nisan 2020 Cuma 12:49

Wordnet is an NLTK corpus reader, a lexical database for English. It can be used to find the meaning of words, synonym or antonym. One can define it as a semantically oriented dictionary of English.

From <https://www.guru99.com/wordnet-nltk.html>

NLP

03 Nisan 2020 Cuma 09:54



Word2vec ve word embedding

03 Nisan 2020 Cuma 16:24

Word2vec is the technique/model to produce word embedding for better word representation.

Word2vec is better and more efficient that latent semantic analysis model.

From <https://www.guru99.com/word-embedding-word2vec.html>

Bag of Words, bundan öncekiymiş sanırım. Ancak bagof wordste keliemlerin sırası ihmale edilir.

Shortcoming of Bag of Words method

- It ignores the order of the word, for example, this is bad = bad is this.
- It ignores the context of words. Suppose If I write the sentence "He loved books. Education is best found in books". It would create two vectors one for "He loved books" and other for "Education is best found in books." It would treat both of them orthogonal which makes them independent, but in reality, they are related to each other

To overcome these limitation word embedding was developed and word2vec is an approach to implement such.

From <https://www.guru99.com/word-embedding-word2vec.html>

Before Word Embedding

It is important to know which approach is used before word embedding and what are its demerits and then we will move to the topic of how demerits are overcome by Word embedding using word2vec approach. Finally, we will move how word2vec works because it is important to understand it's working.

Approach for Latent Semantic Analysis

This is the approach which was used before word embedding. It used the concept of Bag of words where words are represented in the form of encoded vectors. It is a sparse vector representation where the dimension is equal to the size of vocabulary. If the word occurs in the dictionary, it is counted, else not.

From <https://www.guru99.com/word-embedding-word2vec.html>

Burada word2vec ile wordembedding eşanlamlı gibi.

What is Word Embedding?

Word Embedding is a type of word representation that allows words with similar meaning to be understood by machine learning algorithms. Technically speaking, it is a mapping of words into vectors of real numbers using the neural network, probabilistic model, or dimension reduction on word co-occurrence matrix. It is language modeling and feature learning technique. Word embedding is a way to perform mapping using a neural network. There are various word embedding models available such as word2vec (Google), Glove (Stanford) and fastest (Facebook).
Word Embedding is also called as distributed semantic model or distributed represented or semantic vector space or vector space model. As you read these names, you come across the word semantic which means categorizing similar words together. For example fruits like apple, mango, banana should be placed close whereas books will be far away from these words. In a broader sense, word embedding will create the vector of fruits which will be placed far away from vector representation of books.

Word embedding helps in feature generation, document clustering, text classification, and natural language processing tasks.

From <https://www.guru99.com/word-embedding-word2vec.html>

Ne nerde

03 Nisan 2020 Cuma 12:03

	durum	tokenazasiton	vectorizationon	punctuationon	regex	Türkçe stem	Türkçe lemma	Türkçe synoym	Word embedding/word2vec	Açıklamalar
regex	ok				1					
string	ok			1						
nltk	Downladlar eksik	1	1	1	1					Lemması wordnetten
spacy	ok									
Gensim (ra-re tech)	ok									*topic modelling*, *document indexing* and *similarity retrieval*
zemberek	ok									
stanfordcore	????									
Glove(standford)	ok								1	
Fastest(facebook)									1	
fuuywuzzy	ok									