Word Embeddings/ vectorization

Words

Words make up most of the world most of us live in.

Text Vectorization: A fundamental step for NLP

In order for computers to be able to use language for predictive models…or you might say ‘understand’ words, the words must be translated into numbers. Our **documents need to be transformed from strings to vectors**
We have several ways to turn words into numbers. This process is called ‘Vectorizing’, turning language into vectors (one dimensional arrays). Word embedding is one way to do this.

Word Embeddings:

In word embedding, on the other hand, **words are ‘embedded’ in n-dimensional space, where n in a developer defined value**.

Word embeddings ideally preserve the semantic relationships between words and allow a deep learning model to consider a text in order.

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 is used to find word predictions, words similarities/semantics.

Word Embeddings/ vectorization

After the words are converted as vectors, we need to use some techniques such as **Euclidean distance, Cosine Similarity** to identify similar words.

There are several great strategies for turning a document into a vector. :

- A count-vector describes a document by how many of which words are in it,
- **TF-IDF** vectorization uses a floating point number to describe how specific each word is to that document

What is TF-IDF vectorizer?

Imp links: TF-IDF

The **term frequency** is the **number of occurrences of a specific term in a document**. Term frequency indicates how important a specific term in a document. Term frequency represents every text from the data as a matrix whose **rows are the number of documents and columns are the number of distinct** terms throughout all documents.

	and	but	hate	i	image	language	like	love	natural	processing	python	signal
Text 1	0	1	1	2	0	1	0	1	1	1	1	0
Text 2	0	0	0	1	1	0	1	0	0	1	0	0
Text 3	1	0	0	1	1	0	1	0	0	2	0	1

Document frequency is the **number of documents containing a specific term**. Document frequency indicates how common the term is.

Inverse document frequency (IDF) is the weight assigned to a particular word.
Represented as :

Word Embeddings/ vectorization

$$idf_i = \log(\frac{n}{df_i})$$

n = total no. of documents
Df =

So, suppose a word " The" is there and we have 10 documents in all. We also know that this word occurs in all 10 documents, then

Idf = log(10/10)
= log(1)
= 0

So we can say that this word appears in every document. It gives the rare term high weight and gives the common term low weight

The **TF-IDF score** as the name suggests is just a **multiplication of the term frequency matrix with its IDF**, it can be calculated as follow:

$$w_{i,j} = tf_{i,j} \times idf_i$$

For example:
Suppose we have 3 texts and we need to vectorize these texts using TF-IDF.

Text 1	i love natural language processing but i hate python
Text 2	i like image processing
Text 3	i like signal processing and image processing

STEP 1:

Creating **Term frequency matrix** of distinct, words appeared in each document (here text 1 , text 2 , text 3).

Word Embeddings/ vectorization

	and	but	hate	i	image	language	like	love	natural	processing	python	signal
Text 1	0	1	1	2	0	1	0	1	1	1	1	0
Text 2	0	0	0	1	1	0	1	0	0	1	0	0
Text 3	1	0	0	1	1	0	1	0	0	2	0	1

STEP 2:

Compute **inverse document frequency (IDF)** .

Term	and	but	hate	i	image	language	like	love	natural	processing	python	signal
IDF	0.47712	0.47712	0.4771	0	0.1760913	0.477121	0.1760913	0.477121	0.47712125	0	0.477121	0.477121

Like for suppose word " image" :

Number of documents: 3

Document frequency : 2

So, its IDF will be :

log(3/2) = 0.176 something.

STEP-3:

Multiply TF matrix with IDF respectively:

	and	but	hate	i	image	language	like	love	natural	processing	python	signal
Text 1	0	0.47712	0.4771	0	0	0.477121	0	0.477121	0.47712125	0	0.477121	0
Text 2	0	0	0	0	0.1760913	0	0.1760913	0	0	0	0	0
Text 3	0.47712	0	0	0	0.1760913	0	0.1760913	0	0	0	0	0.477121

…… We now can feed this into suitable ML Algorithms……

Word Embeddings/ vectorization

Limitations of TF-IDF:

1. It does not consider the **synonymity** of words
2. Also neglects word semantic. It does not group the semantic words each other. It often **fails to relate each words with each other** : for example : words like dog , cats , rabbits are similar words because they all are animals.

However we can include the word semantics by using word2vec text vectorization technique

Word embeddings coming from pre-trained methods such as,

1. **Word2Vec - (google)**
2. Fasttext -(Facebook)
3. Glove -(Stanford)

In this project Word2Vec will be implemented:

Word2Vec:

Important links:

1. Towardsdatascience.
2. Towardsdatascience
3. word-embedding-word2vec model
4. Intro to word embeddings and word2vec

It is a natural language processing method that **captures** a large number of precise **syntactic and semantic word relationships.**

It is a shallow two-layered neural network that can detect synonymous words and suggest additional words for partial sentences once it is trained

Word Embeddings/ vectorization

Word2vec represents words in vector space representation. Words are represented in the form of vectors and placement is done in such a way that similar meaning words appear together and dissimilar words are located far away.

Word2vec reconstructs the linguistic context of words.

For example: **“What is the temperature of India”**, here the context is the user wants to know **“temperature of India”** which is context. In short, the main objective of a sentence is context. Word or sentence surrounding spoken or written language (disclosure) helps in determining the meaning of context. **Word2vec learns vector representation of words through the contexts.**

How Word2vec works?

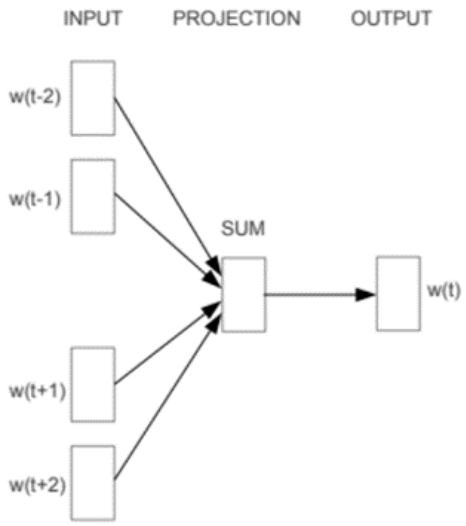
Two different model architectures that can be used by Word2Vec to create the word embeddings are the **Continuous Bag of Words (CBOW) model & the Skip-Gram model**

How does CBOW model works?

This model architecture essentially tries to predict a target word from a list of context words(key words)
Like given a phrase `"Have a great day"` , we will choose our **target word to be “a”** and our **context words** to be **[“have”, “great”, “day”]**.

What this model will do is take the distributed representations of the context words to try and predict the target word.

Word Embeddings/ vectorization



After that we match the output predicted by the model with our target word and compute the loss and then we perform backpropagation with each epoch to update the embedding layer in the process.

How does Skip-gram model works?

In the skip-gram model, given a target (centre) word, the context words are predicted.

Its the flip of CBOW model.
It takes the **current word as an input** and tries to accurately predict the words before and after this current word. This model essentially tries to learn and predict the **context words** around the specified input word.

Word Embeddings/ vectorization

Preparing the Data for Embedding:

1. Text Normalization Techniques:

**IMP LINKS: (python code implementation)

- stemming/ lemmatization using python
- Intro to text normalization

In any natural language, **words can be written or spoken in more than one form** depending on the situation. That's what makes the language such a thrilling part of our lives, right? For example:

- Lisa **ate** the food and washed the dishes.
- They were **eating** noodles at a cafe.
- Don't you want to **eat** before we leave?
- We have just **eaten** our breakfast.

In all these sentences, **we can see that the word eat has been used in multiple forms**. For us, it is easy to understand that eating is the activity here. So it doesn't really matter to us whether it is 'ate', 'eat', or 'eaten' – we know what is going on.

Unfortunately, that is not the case with machines. They treat these words

differently. Therefore, we need to convert them to their root word, which is

"eat" in our example

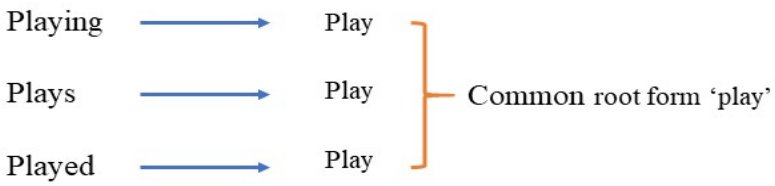
Text normalization is a process of transforming a word into a single canonical(root) form. This can be done by two processes, **stemming** and **lemmatization**.

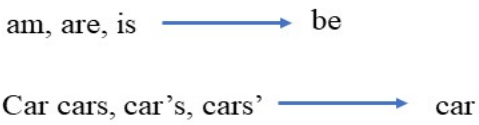
For example, the word **'play'** can be used as **'playing', 'played', 'plays',** etc.

STEMMING vs LEMMATIZATION:

Word Embeddings/ vectorization

- **Stemming** is a normalization technique that c**uts off the end or beginning** of a word by taking into account a list of common prefixes or suffixes basically removing “ly”,”ish” “ing” from the words.





Using above mapping a sentence could be normalized as follows:

- Lemmatization, on the other hand, is an **organized & step-by-step procedure of obtaining the root form of the word**. It makes use of dictionary importance of words and grammar relations)
- **Stemming** algorithm works by **cutting the suffix** or prefix from the word. **Lemmatization** is a more powerful operation as it takes into **consideration the morphological analysis** of the word
- **stemming** is a **quick** method of **chopping off words** to its root form while on the other hand, **lemmatization** is an intelligent operation that uses dictionaries which are created by in-depth linguistic knowledge. Hence, **Lemmatization helps in forming better features**
- **For example**
- Word “litigation “ ——————> litig by using stemming
  litigation ——————————>litigation by using lemmatization

Word Embeddings/ vectorization

For understanding python implementation use above **links

2. Removing stop words(if necessary based upon the task):

**Important link:

1. how-to-remove-stopwords-text-normalization-nltk-
2. Krish_Naik_how to remove_stop_words

Stopwords are the most common words in any natural language. For the purpose of analyzing text data and building NLP models, these stopwords might not add much value to the meaning of the document., e.g “an”, “on”, “the” etc..

Key benefits:

- On removing stopwords, dataset size decreases and the time to train the model also decreases
- Removing stopwords can potentially help improve the performance as there are fewer and only meaningful tokens left.
- Even search engines like Google remove stopwords for fast and relevant retrieval of data from the database

stopword removal doesn’t take off the punctuation marks or newline characters. We will need to remove them manually.

Stopword Removal using:

1. **NLTK**
   NLTK, or the Natural Language Toolkit, is a treasure trove of a library for text preprocessing.

Word Embeddings/ vectorization

2. **Gensim**
   Gensim is a pretty handy library to work with on NLP tasks.

   While pre-processing, gensim provides methods to remove stopwords as well. We can easily import the **remove_stopwords method** from the class **gensim.parsing.preprocessing.**

   For understanding its python implementation, refer to ** imp link above:)

3.Keyword Extraction Methods from Documents in NLP

*imp link: keyword-extraction-methods