Project Word Embeddings

The code bundle and GitHub code of the project:

launch binder

https://mybinder.org/v2/gh/chokrihamza/text-mining-word-embedding-project/HEAD

https://github.com/chokrihamza/text-mining-word-embedding-project.git

Realized by:

Hamza Chokri (mailto:chokrihamza.ia@gmail.com).

Malek Ghanem (mailto:malek.ghanem1920@gmail.com).

1. Introduction:

In information retrieval, techniques like TF-IDF (Term Frequency-Inverse Document Frequency) can be useful for extracting information about the frequency of a term within a document and the number of documents in which it occurs. However, these techniques do not take into account the context in which the words appear, which can be important for understanding their meaning and the overall structure of a sentence.

To better extract and understand the meaning of words in a given context, it can be helpful to use techniques that consider the relationships between words, such as word embeddings or other natural language processing techniques. These techniques can help to capture the meaning of words and the way they are used in different contexts, allowing for more accurate information retrieval and understanding of text.

For example:

Saying: "Jhon hugged his wife! So did Stive. "

If we take a look at the above sentence. Here, this sentence says Jhon hugged his wife and it also says so did Stive!

So, did Stive hug Jhon's wife?

Such ambiguity may exist in the data. So, it is necessary to have a clear understanding of the context of the text where each word is converted into a word vector which helps in finding similar words (George, 2022).

2. word embeddings (Word vectors):

Word vectors are numerical representations of words that capture their meaning and context.

Each number in the vector represents a particular aspect of the word's meaning, with higher values indicating a stronger association with that aspect.

While techniques like the Continuous Bag of Words (CBOW) and TF-IDF (Term Frequency-Inverse Document Frequency) can be effective for extracting features from text data, they do not take into account the structure, semantics, sequence, or context of the words.

Word embeddings, on the other hand, are able to capture this additional information, allowing for a more sophisticated and nuanced representation of the words in a document.

Therefore, while CBOW and TF-IDF can be useful for certain tasks, word embeddings may be a more appropriate choice in situations where a more comprehensive understanding of the words and their context is needed (George, 2022).

3. Difference between word embeddings and TF-IDF:

in this part we will list the difference between word embeddings and TF-IDF models.

	Word embeddings	TF-IDF matrix
Usage	These are vectors that capture words similar with other words in a multi-dimensional space	This is a sparse matrix where the words carry weight with no meaning
Training	Uses external large corpus for training	No external corpus of data involved
Memory usage	Very high memory usage	Low memory usage when compared to word embeddings
Application	Since it is applied to each word individually. it is ideal for places like machine translation.	Since it is applied for all the documents, it is ideal for larger files.

Table 1: Comparison of word embeddings and TF-IDF technique

4. Feature engineering include word embeddings:

There are many feature engineering techniques with word embeddings model like:

- Word2Vec
- Glove
- Fast Text
- BERT

In the next parts we will look to the three main feature engineering Word2Vec, Glove and Fast Text.

4.1. Word2Vec:

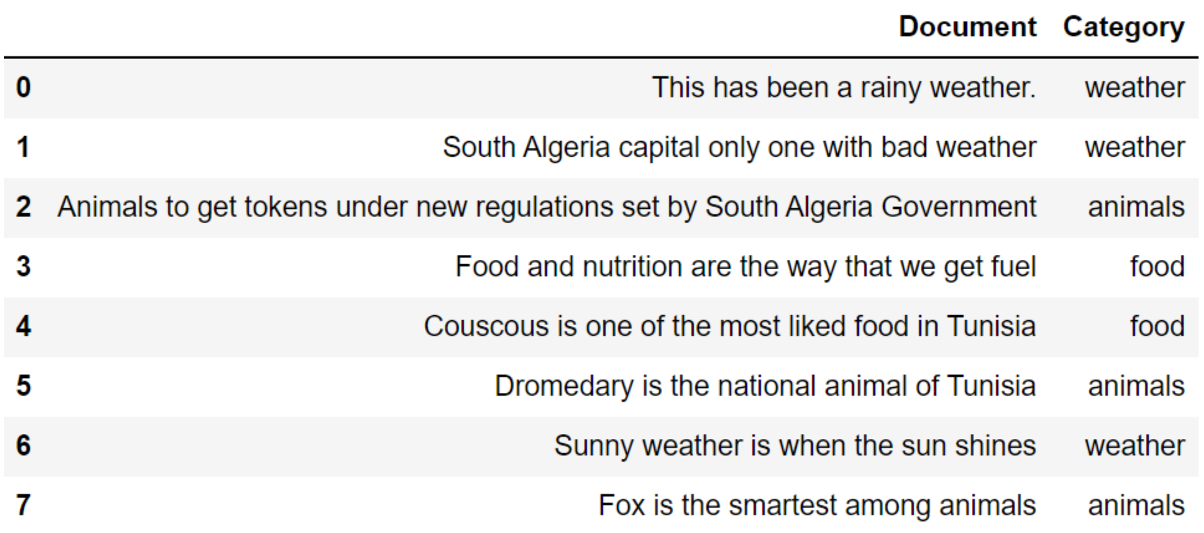
Created in 2013 by Google based on a pre-defined two-layer neural network trained to generate vector representations of words through which we can extract highly-contextual and semantic similarity.

It is an unsupervised method where a large text data is input from which we create a vocabulary and produce similar word embeddings for each word in the vocabulary (Mikolov, Kai , Greg , & Jeffrey , 2013).

Each word is represented in the form of a vector when it comes to word2vec embedding.

4.1.1. Project:

We will use the word2vec model from Gensim library and train it with our data set.



	Document	Category
0	This has been a rainy weather.	weather
1	South Algeria capital only one with bad weather	weather
2	Animals to get tokens under new regulations set by South Algeria Government	animals
3	Food and nutrition are the way that we get fuel	food
4	Couscous is one of the most liked food in Tunisia	food
5	Dromedary is the national animal of Tunisia	animals
6	Sunny weather is when the sun shines	weather
7	Fox is the smartest among animals	animals

Let us assume the widow size is 3 and Skip gram here is 1.

Given a specific word with window 3, we will start forming words inside the window with that specific word.

“This has been a rainy weather”: Document 0

has been a rainy weather – (this, has)

This been a rainy weather – (has, this), (has, been)

This **has a** rainy weather – (been, has), (been, a)

one word is selected on both sides of the specific word for the window size being 3.

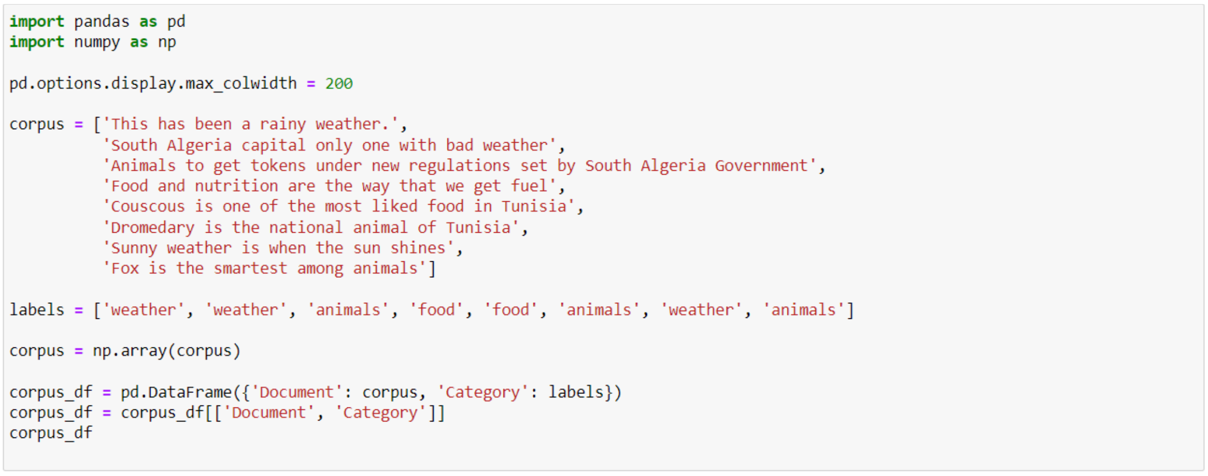
As shown in (south, food) and (south, Algeria), the pair of words (south, Algeria) will occur in many places together than (south, food).

Thus, when we look up south, then the probability for the output to be Algeria will be greater than the word food, as the word food can be associated with many other entities.

Steps of the project:

1. Preparing a corpus using NumPy and pandas.
2. Pre-processing using nltk stopwords and punkt tokenizer.
3. Fitting word2vec model to our data.
4. Visualization of the embeddings using t-SNE (t-Distributed Stochastic Neighbour Embedding).
5. Determining similarity between words and finding nearest words in the same direction.

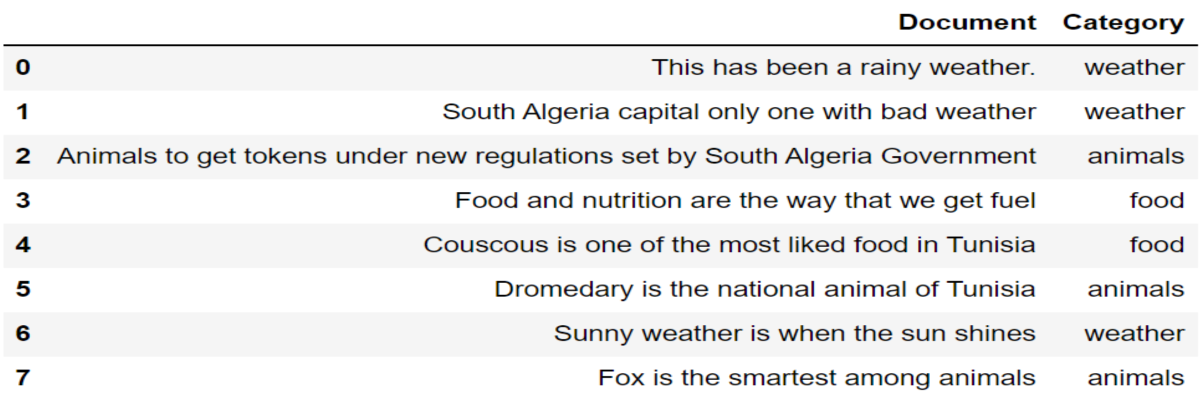
Step 1:



```python
import pandas as pd
import numpy as np

pd.options.display.max_colwidth = 200

corpus = ['This has been a rainy weather.',
          'South Algeria capital only one with bad weather',
          'Animals to get tokens under new regulations set by South Algeria Government',
          'Food and nutrition are the way that we get fuel',
          'Couscous is one of the most liked food in Tunisia',
          'Dromedary is the national animal of Tunisia',
          'Sunny weather is when the sun shines',
          'Fox is the smartest among animals']

labels = ['weather', 'weather', 'animals', 'food', 'food', 'animals', 'weather', 'animals']

corpus = np.array(corpus)

corpus_df = pd.DataFrame({'Document': corpus, 'Category': labels})
corpus_df = corpus_df[['Document', 'Category']]
corpus_df
```

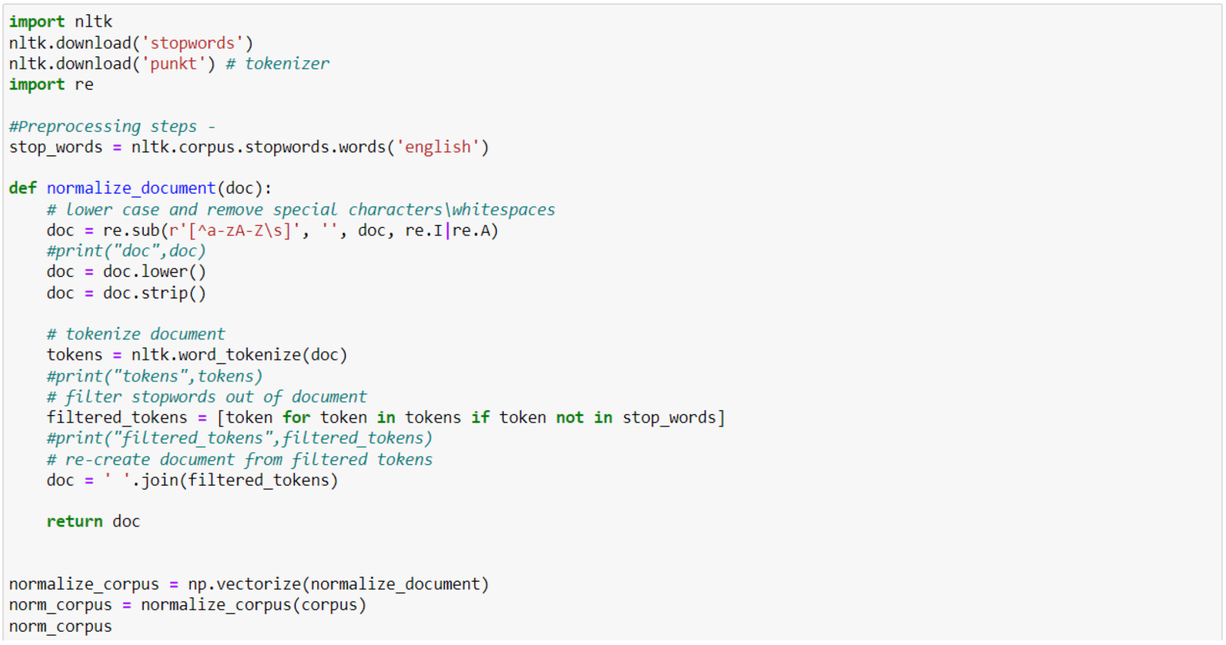
Image 1: Preparing a corpus using NumPy and pandas

Result:



	Document	Category
0	This has been a rainy weather.	weather
1	South Algeria capital only one with bad weather	weather
2	Animals to get tokens under new regulations set by South Algeria Government	animals
3	Food and nutrition are the way that we get fuel	food
4	Couscous is one of the most liked food in Tunisia	food
5	Dromedary is the national animal of Tunisia	animals
6	Sunny weather is when the sun shines	weather
7	Fox is the smartest among animals	animals

Step 2:



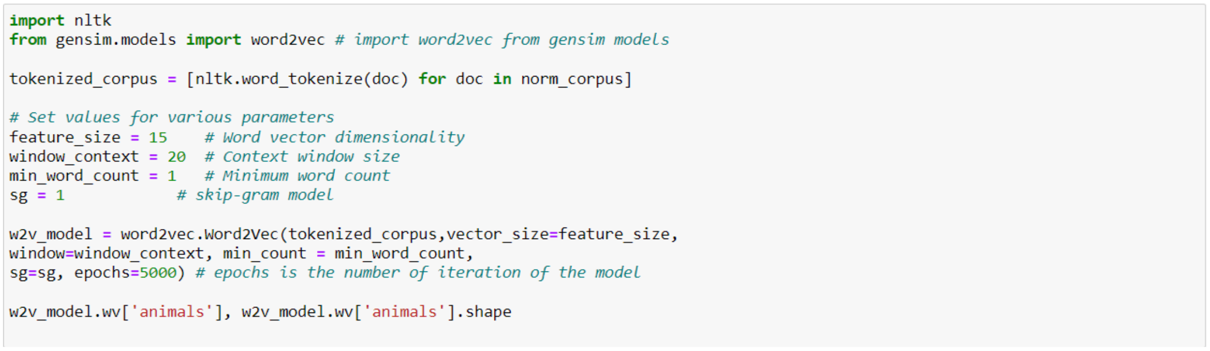
```python
import nltk
nltk.download('stopwords')
nltk.download('punkt') # tokenizer
import re

#Preprocessing steps -
stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
    # lower case and remove special characters\whitespaces
    doc = re.sub(r'[^a-zA-Z\s]', '', doc, re.I|re.A)
    #print("doc",doc)
    doc = doc.lower()
    doc = doc.strip()

    # tokenize document
    tokens = nltk.word_tokenize(doc)
    #print("tokens",tokens)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    #print("filtered_tokens",filtered_tokens)
    # re-create document from filtered tokens
    doc = ' '.join(filtered_tokens)

    return doc


normalize_corpus = np.vectorize(normalize_document)
norm_corpus = normalize_corpus(corpus)
norm_corpus
```

Image 2: Pre-processing using nltk stopwords and punkt tokenizer

Result:

```
array(['rainy weather', 'south algeria capital one bad weather',
       'animals get tokens new regulations set south algeria government',
       'food nutrition way get fuel', 'couscous one liked food tunisia',
       'dromedary national animal tunisia', 'sunny weather sun shines',
       'fox smartest among animals'], dtype='<U63')
```

Step 3:



```python
import nltk
from gensim.models import word2vec # import word2vec from gensim models

tokenized_corpus = [nltk.word_tokenize(doc) for doc in norm_corpus]

# Set values for various parameters
feature_size = 15    # Word vector dimensionality
window_context = 20  # Context window size
min_word_count = 1   # Minimum word count
sg = 1            # skip-gram model

w2v_model = word2vec.Word2Vec(tokenized_corpus,vector_size=feature_size,
window=window_context, min_count = min_word_count,
sg=sg, epochs=5000) # epochs is the number of iteration of the model

w2v_model.wv['animals'], w2v_model.wv['animals'].shape
```

Image 3: Fitting word2vec model to our data

Result:

```
(array([-0.5864642 ,  0.5505315 ,  1.2139299 , -0.42850158, -0.0179489 ,
        -0.22037804, -0.2871057 , -0.22489728, -0.23400211, -0.792308  ,
         0.276999  , -0.33136293, -0.59329915, -1.2955353 ,  0.53060514],
       dtype=float32),
 (15,))
```

Step 4:

```
import matplotlib.pyplot as plt

%matplotlib inline

# visualize embeddings
from sklearn.manifold import TSNE

words = w2v_model.wv.index_to_key

wvs = w2v_model.wv[words]

tsne = TSNE(n_components=2, random_state=42, n_iter=5000, perplexity=5)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(wvs)
labels = words

plt.figure(figsize=(12, 6))
plt.scatter(T[:, 0], T[:, 1], c='red', edgecolors='r')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
    plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset points')
```

Result:

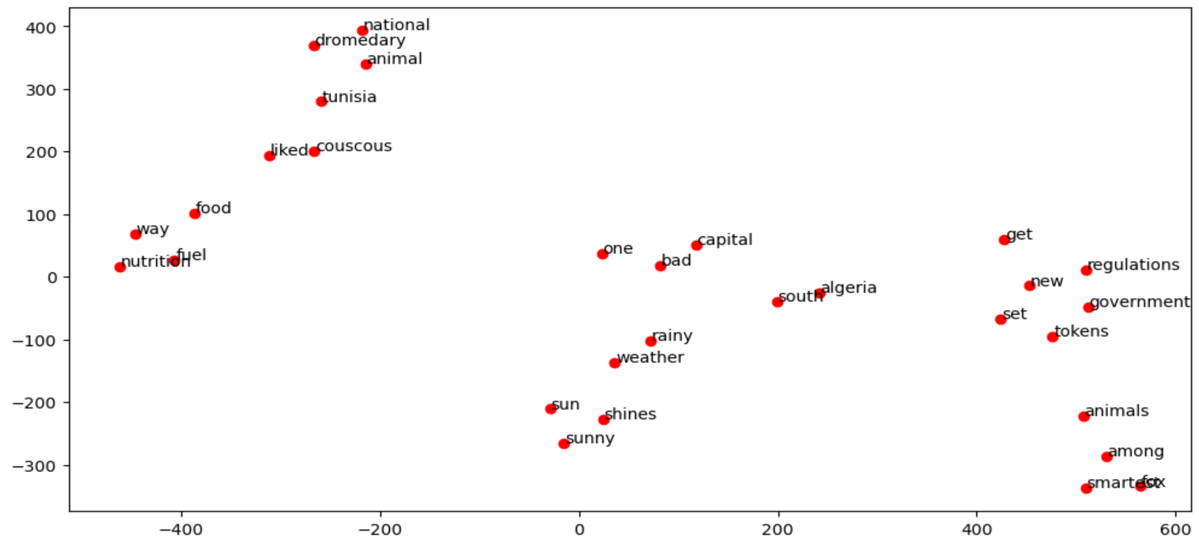
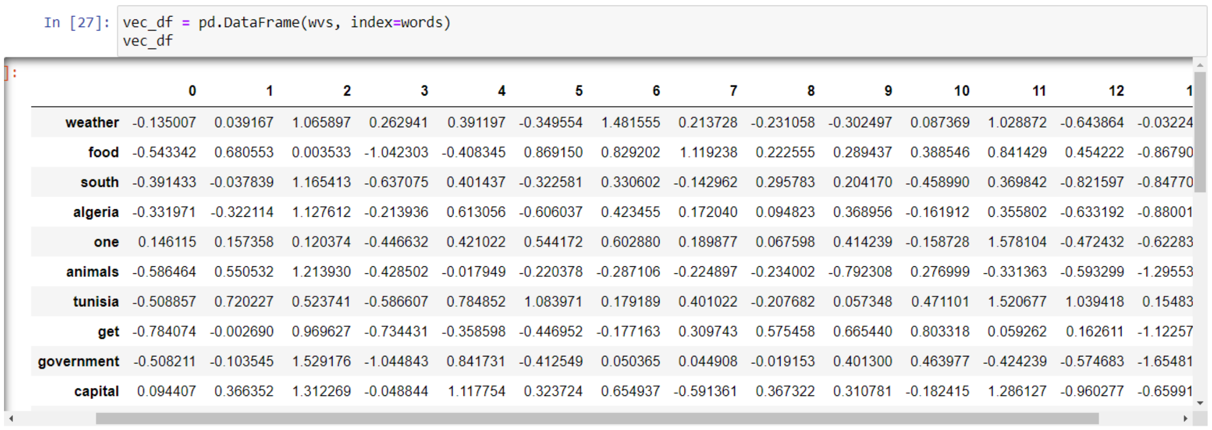


Image 4: Visualization of the embeddings using t-SNE

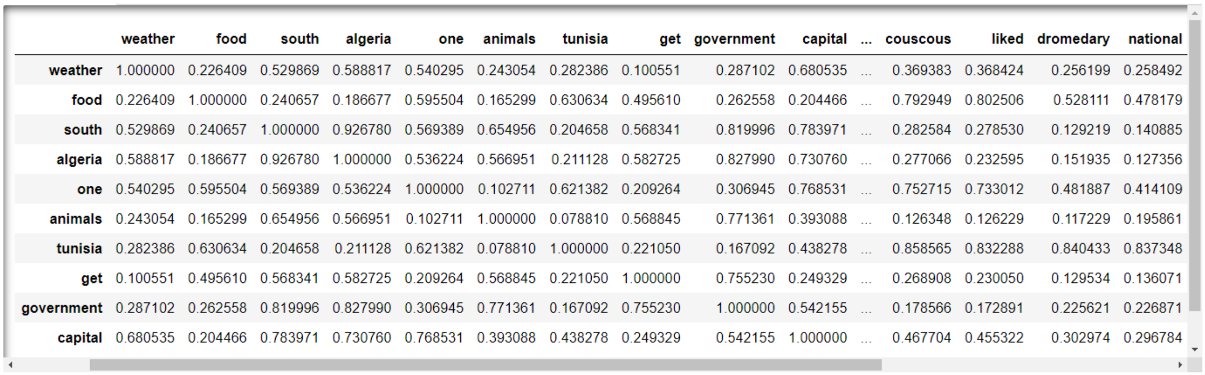


```
In [27]: vec_df = pd.DataFrame(wvs, index=words)
         vec_df
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	1
weather	-0.135007	0.039167	1.065897	0.262941	0.391197	-0.349554	1.481555	0.213728	-0.231058	-0.302497	0.087369	1.028872	-0.643864	-0.03224
food	-0.543342	0.680553	0.003533	-1.042303	-0.408345	0.869150	0.829202	1.119238	0.222555	0.289437	0.388546	0.841429	0.454222	-0.86790
south	-0.391433	-0.037839	1.165413	-0.637075	0.401437	-0.322581	0.330602	-0.142962	0.295783	0.204170	-0.458990	0.369842	-0.821597	-0.84770
algeria	-0.331971	-0.322114	1.127612	-0.213936	0.613056	-0.606037	0.423455	0.172040	0.094823	0.368956	-0.161912	0.355802	-0.633192	-0.88001
one	0.146115	0.157358	0.120374	-0.446632	0.421022	0.544172	0.602880	0.189877	0.067598	0.414239	-0.158728	1.578104	-0.472432	-0.62283
animals	-0.586464	0.550532	1.213930	-0.428502	-0.017949	-0.220378	-0.287106	-0.224897	-0.234002	-0.792308	0.276999	-0.331363	-0.593299	-1.29553
tunisia	-0.508857	0.720227	0.523741	-0.586607	0.784852	1.083971	0.179189	0.401022	-0.207682	0.057348	0.471101	1.520677	1.039418	0.15483
get	-0.784074	-0.002690	0.969627	-0.734431	-0.358598	-0.446952	-0.177163	0.309743	0.575458	0.665440	0.803318	0.059262	0.162611	-1.12257
government	-0.508211	-0.103545	1.529176	-1.044843	0.841731	-0.412549	0.050365	0.044908	-0.019153	0.401300	0.463977	-0.424239	-0.574683	-1.65481
capital	0.094407	0.366352	1.312269	-0.048844	1.117754	0.323724	0.654937	-0.591361	0.367322	0.310781	-0.182415	1.286127	-0.960277	-0.65991

Step 5:

```
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

similarity_matrix = cosine_similarity(vec_df.values)
similarity_df = pd.DataFrame(similarity_matrix, index=words, columns=words)
similarity_df.head(10)
```



	weather	food	south	algeria	one	animals	tunisia	get	government	capital	...	couscous	liked	dromedary	national
weather	1.000000	0.226409	0.529869	0.588817	0.540295	0.243054	0.282386	0.100551	0.287102	0.680535	...	0.369383	0.368424	0.256199	0.258492
food	0.226409	1.000000	0.240657	0.186677	0.595504	0.165299	0.630634	0.495610	0.262558	0.204466	...	0.792949	0.802506	0.528111	0.478179
south	0.529869	0.240657	1.000000	0.926780	0.569389	0.654956	0.204658	0.568341	0.819996	0.783971	...	0.282584	0.278530	0.129219	0.140885
algeria	0.588817	0.186677	0.926780	1.000000	0.536224	0.566951	0.211128	0.582725	0.827990	0.730760	...	0.277066	0.232595	0.151935	0.127356
one	0.540295	0.595504	0.569389	0.536224	1.000000	0.102711	0.621382	0.209264	0.306945	0.768531	...	0.752715	0.733012	0.481887	0.414109
animals	0.243054	0.165299	0.654956	0.566951	0.102711	1.000000	0.078810	0.568845	0.771361	0.393088	...	0.126348	0.126229	0.117229	0.195861
tunisia	0.282386	0.630634	0.204658	0.211128	0.621382	0.078810	1.000000	0.221050	0.167092	0.438278	...	0.858565	0.832288	0.840433	0.837348
get	0.100551	0.495610	0.568341	0.582725	0.209264	0.568845	0.221050	1.000000	0.755230	0.249329	...	0.268908	0.230050	0.129534	0.136071
government	0.287102	0.262558	0.819996	0.827990	0.306945	0.771361	0.167092	0.755230	1.000000	0.542155	...	0.178566	0.172891	0.225621	0.226871
capital	0.680535	0.204466	0.783971	0.730760	0.768531	0.393088	0.438278	0.249329	0.542155	1.000000	...	0.467704	0.455322	0.302974	0.296784

Image 5: Determining similarity between words and finding nearest words in the same direction.

```
feature_names = np.array(words)
similarity_df.apply(lambda row: feature_names[np.argsort(-row.values)[1:4]],
axis=1)
```

```
weather                    [sun, shines, sunny]
food                     [way, nutrition, fuel]
south                       [algeria, set, new]
algeria               [south, set, government]
one                    [bad, capital, couscous]
animals                  [among, tokens, set]
tunisia          [animal, couscous, dromedary]
get             [new, regulations, government]
government             [new, regulations, set]
capital                     [bad, south, rainy]
bad                     [capital, one, south]
tokens                 [set, new, government]
new             [set, government, regulations]
regulations        [new, government, tokens]
set                [new, tokens, government]
among               [smartest, fox, animals]
smartest               [among, fox, animals]
nutrition                  [way, fuel, food]
way                  [fuel, nutrition, food]
fuel                  [way, nutrition, food]
couscous              [liked, tunisia, food]
liked              [couscous, tunisia, food]
dromedary        [national, animal, tunisia]
national        [animal, dromedary, tunisia]
animal        [national, dromedary, tunisia]
sunny                    [shines, sun, rainy]
sun                    [shines, sunny, rainy]
shines                    [sun, sunny, rainy]
fox                [smartest, among, animals]
rainy                    [sunny, shines, sun]
dtype: object
```

4.2. Global Vector (GloVe) Model:

The Global Vector model is an unsupervised learning model that finds out how frequently words appear together in a bigger corpus. It is a count-based model to find semantic relationships between words from co-occurrence matrix.

4.2.1. why use Glove when Word2Vec is already in use?

GloVe (Global Vectors for Word Representation) is a method for generating word vectors, or word embeddings, which are numerical representations of words that capture semantic relationships between them. GloVe uses a global matrix factorization approach, where the goal is to learn a matrix that represents the word co-occurrence information in a corpus (Jeffrey , Richard , & Christopher D., 2014).
Word2Vec is another method for generating word vectors, which uses a neural network architecture to learn word vectors. Word2Vec represents each word as a vector and uses a sliding window approach to predict the context of a word based on its surrounding words.
One advantage of GloVe over Word2Vec is that GloVe considers the global context of a word, whereas Word2Vec only considers the local context within a sliding window. This means that GloVe may be able to capture more information about the relationships between words in a corpus, as it takes into account the co-occurrence of words across the entire corpus.
For example: "The food was the best".
both GloVe and Word2Vec would likely represent the word "The" as a stop word, as it is a common function word that does not convey much meaning on its own. However, GloVe may be able to capture more information about the relationships between the other words in the sentence, such as the fact that "food" and "best" are related because they both describe the quality of the food.
Spacy is an open-source natural language processing library that includes pre-trained word vectors based on the GloVe model. It can be used for tasks such as part-of-speech tagging, named entity recognition, and dependency parsing.

4.3. Fast-Text:

Fast-Text is an open-source library for learning word embeddings and performing text classification, created by Facebook's AI Research lab in 2016. It is a lightweight model that is designed to be efficient and easy to use. Fast-Text is an improvement on the vanilla Word2Vec model and has several features that make it well-suited for natural language processing tasks. These include the ability to generate state-of-the-art English word vectors and word vectors for 157 different languages, as well as models for language identification and various supervised tasks. Fast-Text is a popular choice for working with text data, due to its flexibility and ease of use (Armand , et al., 2016).

4.3.1. Why Fast-text?

Word2Vec is a popular technique for generating word embeddings, which are numerical representations of words that capture their meaning and context.

In Word2Vec, each word is treated as a single entity and the model ignores the morphological structure of the word.

i.e., the way it is constructed from smaller units such as prefixes, suffixes, and root words (Armand , et al., 2016).

Fast-Text, on the other hand, takes a different approach to representing words. It considers each word as a bag of n-grams, where an n-gram is a contiguous sequence of n characters within the word. This means that Fast-Text can capture information about the internal structure of words and how they are composed of smaller units. This can be particularly useful for handling rare words or words in languages with complex morphological structures, as the n-grams that make up these words may occur in other words, allowing for a better representation. For example:

Consider the word: Success with n=3.

Then, the Fast-Text would represent the word as < su, suc, ucc, cce, ces, ess, ss >

were the:

< represents the beginning of the word.

> represents the end of the word.

And the word <Success> is taken as a whole. This would help a lot in understanding the prefixes, suffixes, and the meaning of the shorter words better.

Once we have the n-gram representation of the word, we will then train a skip gram model to get the embeddings.

4.3.2. how is this helpful in understanding shorter words better?

Consider the word **<here>.** If we represent the word for **n=3,** we will get **<he, her, ere, re>** and the word **<her>** is different from the tri-gram derived from the word **<here>**.

The model suggests extracting n-grams of sizes 3 to 6 in order to create a more robust representation of a word.

This is particularly useful for representing rare words, as the character-based n-grams that make up these words may occur in other words, increasing the likelihood of a good representation. The approach is based on the paper "Enriching Word Vectors with Subword Information."

5. Conclusion:

word embeddings are a powerful tool for representing the meaning of words in a numerical form that can be used as input to machine learning models. Word2Vec, GloVe, and fastText are all popular methods for creating word embeddings, and each has its own strengths and limitations.

Word2Vec and GloVe are based on the idea of using the co-occurrence of words in a large text corpus to infer the meaning of each word. These methods take a large amount of text as input and create numerical representations of each word based on how often it appears in the text and how it is used in context.

Fast-Text is similar to Word2Vec and GloVe, but it also takes into account the context of shorter words by treating each word as a combination of n-grams (substrings of the word). This can be particularly useful for preserving the meaning of rare or out-of-vocabulary words that might not be captured by the other methods. Once you have created word embeddings, you can use them as input to a variety of natural language processing tasks, such as text classification or document categorization. In these tasks, you can use machine learning algorithms to learn patterns in the word embeddings that can be used to predict the class or category of a given text or documents (George, 2022).

References:

Armand , J., Edouard , G., Piotr , B., Matthijs , D., Herv´e , J., & Tomas , M. (2016). FASTTEXT.ZIP: COMPRESSING TEXT CLASSIFICATION MODELS. *ICLR 2017*.

Birajdar, N. (2021, mars 16). *Medium*. Retrieved from https://towardsdatascience.com/word2vec-research-paper-explained-205cb7eecc30

George, A. (2022). *python text mining .* BPB Publications, India.

Jeffrey , P., Richard , S., & Christopher D., M. (2014). GloVe: Global Vectors for Word Representation.

Mikolov, T., Kai , C., Greg , C., & Jeffrey , D. (2013). Efficient Estimation of Word Representations in Vector Space.