Istanbul Technical University - Science and Letters Faculty

Mathematics Engineering Program

Graduation Project

Student Name: Arif Çakır

Student Number: 090190355

Course: MAT4091E

Advisor: Prof. Dr. Atabey Kaygun

Submission Date: June 9, 2023

Contents

1 Introduction

Language is a fundamental aspect of human communication and one of the most common ways to pass on information and culture throughout history. Due to their nature of encapsulating information, different languages are studied by linguists throughout the years. Thanks to this scientific foundation, machine learning, and computer science are also advanced upon this topic. Natural Language Processing (NLP) is the subfield of machine learning that studies the processing of human language by computers. One of the most popular NLP techniques in recent years is Word Embedding, which represents words as vectors in semantic space that allows applying mathematical operations on them to analyse. Word Embedding can be used on various subjects such as text classification, translation, and sentiment analysis with promising results. It is aimed in this paper to study several Word Embedding techniques and their applications. The aim is to create a model by applying various Word Embedding models and with the help of this model, describing the relationships and similarities of languages.

The rest of the paper is organised as follows: In Section 2, word embedding techniques are discussed. The word embedding techniques that analysed on the paper are Term Frequency (TF), Inverse Document Frequency (IDF), Skip-Gram, Continuous Bag of Words (CBOW), Global Vectors of Word Representation (GloVe), and Bidirectional Encoder Representations from Transformers (BERT). Section 2 is followed by Section 3 which discusses the Word2Vec method. After that, Section 4 introduces the dataset The Divine Comedy by Dante Alighieri. Also in Section 4, Skip-Gram and CBOW models are applied to the dataset. This application aims to create a dendrogram of cosine similarity of languages, which clusters languages based on their similarities. This is followed by Section 5, which discusses the output of Section 4. Finally, the paper is concluded in Section 6.

2 Word Embedding Techniques

Word Embedding is one of the most popular natural language processing techniques due to its vector representation of the words. As Agarwal stated, capturing the semantic meaning

of the words in a vector of text is the ambition of the word embedding techniques (Agarwal, 2022). With respect to that, some of the most popular word embedding techniques will be studied in this section. Those techniques are TF, which counts the rarity of words; IDF, which counts the rarity of words; Skip-Gram method, which is used by Word2Vec; Continuous Bag of Words method, also used by Word2Vec; GloVe, which captures co-occurrence of words; and BERT, a family of masked-language models introduced by Google.

2.1 Term Frequency - Inverse Document Frequency

Term Frequency (TF) is a word embedding technique which counts the occurrence of the words in a document. TF can be shown as

$$TF(i) = \frac{\log(Frequency(i,j))}{\log(TotalNumber(j))}. \tag{1}$$

where $Frequency(i,j)$ is the frequency of a word that occurred in a j word document and $TotalNumber(j)$ is the total number of the words in the document.

On the other hand, Inverse Document Frequency (IDF) is practically the opposite of the TF method. In this method, the algorithm relies on the information gained from the words which are rarely used. IDF can be written as

$$IDF(i) = \log(\frac{TotalNumber(j)}{Frequency(j,i)}). \tag{2}$$

where $Frequency(i,j)$ is the frequency of a word that occurred in a j word document and $TotalNumber(j)$ is the total number of the words in the document.

TF-IDF mainly shows the degree of relevancy of word i in document j, while the main disadvantage of TF-IDF is it does not grasp the contextual relationship between the words. As Kınık and Güran stated, TF-IDF does not capture semantic relationships between words and accepts them as independent values (Kınık and Güran, 2021). Due to TF-IDF's lack of capturing the semantic relationship of the words, TF-IDF is mainly used to detect stop words.

2.2 Skip-Gram Algorithm

The Skip-Gram model is used by word embedding tasks of the Word2Vec model, which will be discussed in the following chapters. Dive Into Deep Learning (n.d) states that a word can be used for generating its surrounding words in the Skip-Gram model. For example, if the sentence "the man loves his son" is taken and "loves" is chosen as the center word, then the Skip-Gram model considers the conditional probability for generating the words. This architecture can be seen in **Figure 1**. Due to this approach, each word has a two-dimensional vector representation in the Skip-Gram model.

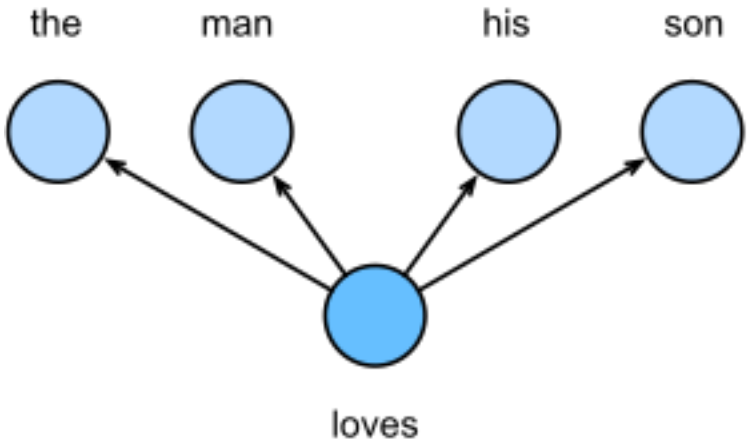

Figure 1: Skip-Gram model architecture, from Dive Into Deep Learning

Dive Into Deep Learning continues that, for any word with index i in the directory, $\mathbf{v}_i \in \mathbb{R}^d$ and $\mathbf{u}_i \in \mathbb{R}^d$ are its two vector representations, where $\mathbf{v}_i$ is when the word is used as the "center word" and $\mathbf{u}_i$ is when the word is used as the "context word". If w_o is any context word and w_c is given center word, then conditional probability of generating w_o can be modelled as

$$P(w_o|w_c) = \frac{exp({\mathbf{u}_o}^\top \mathbf{v}_c)}{\sum_{i\in\mathcal{V}} exp({\mathbf{u}_i}^\top \mathbf{v}_c)}. \tag{3}$$

where $\mathcal{V}$ is the vocabulary index set. If a text sequence of length T is given and word at

time step t is denoted as w^t, then the likelihood function of the Skip-Gram model for context widow size m is as follows:

$$\prod_{t=1}^{T} \prod_{-m \leq j \leq m, j \neq 0} P(w^{t+j}|w^t). \tag{4}$$

The Skip-Gram model parameters are the center word and context word vector for each word in the corpus. In order to train the Skip-Gram model, the given loss function must be minimized:

$$-\sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} logP(w^{(t+j)}|w^{(t)}).$$

Moreover, while using (stochastic) gradient descent to minimize the loss function, gradients of log conditional probability must be obtained. For center word w_c and context word w_o, log conditional probability is

$$logP(w_o|w_c) = \mathbf{u}_o^\top \mathbf{v}_c - log(\sum_{i \in \mathcal{V}} exp(\mathbf{u}_i^\top \mathbf{v}_c)). \tag{5}$$

And the gradient of center word $\mathbf{v}_c$ can be obtained as

$$\frac{\partial P(w_o|w_c)}{\partial \mathbf{v}_c} = \mathbf{u}_o - \sum_{j \in \mathcal{V}} P(w_o|w_c)\mathbf{u}_j. \tag{6}$$

After this calculation, center word vector $\mathbf{v}_i$ and context word vector $\mathbf{u}_i$ are obtained for index i in the dictionary.

2.3 Continuous Bag of Words

The other method for word embedding in Word2Vec is the Continuous Bag of Words (CBOW) method. The main difference between Skip-Gram and CBOW is instead of generating surrounding words with respect to the center word, CBOW generates the center word with the help of surrounding words. If the same example "the man loves his son" is taken for the CBOW model, instead of generating surrounding words based on the center

word "loves", the model generates the center word "loves" from its surroundings. This architecture can be seen in **Figure 2**.

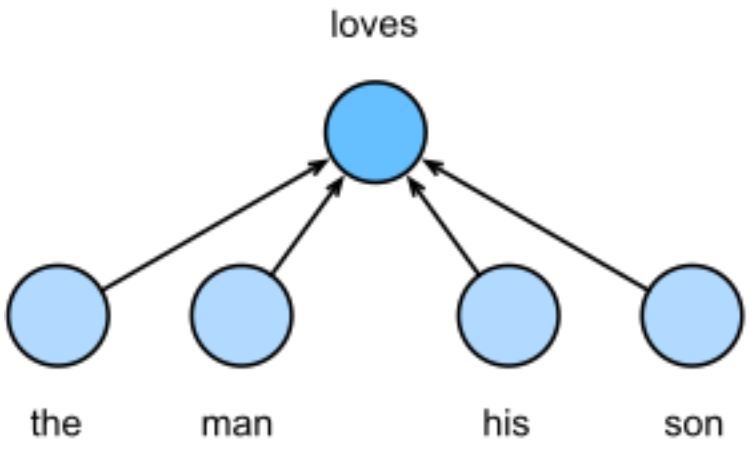

Figure 2: CBOW model architecture, from Dive Into Deep Learning

According to Dive Into Deep Learning (n.d.), in order to calculate conditional probability, context word vectors are averaged because there are multiple words. For any word with index i in the dictionary, $\mathbf{v}_i \in \mathbb{R}^d$ is the context word while $\mathbf{u}_i \in \mathbb{R}^d$ is the center word. It can be seen that meanings are switched compared to the Skip-Gram model. While $w_o, ..., w_{o_{2m}}$ the conditional probability of generating center word w_c can be modelled as following:

$$P(w_c|w_o, ..., w_{o_{2m}}) = \frac{exp(\frac{1}{2m}u_c^\top(\mathbf{v}_{o_1} + ... + \mathbf{v}_{o_{2m}}))}{\sum_{i\in\mathcal{V}} exp(\frac{1}{2m}u_c^\top(\mathbf{v}_{o_1} + ... + \mathbf{v}_{o_{2m}}))}. \tag{7}$$

For simplicity, $\mathcal{W}_o = \{w_o, ..., w_{o_{2m}}\}$ and $\bar{\mathbf{v}}_o = (\mathbf{v}_{o_1} + ... + \mathbf{v}_{o_{2m}})/(2m)$. The equation given above is simplified as follows:

$$P(w_c|\mathcal{W}_o) = \frac{exp(u_c^\top \bar{\mathbf{v}}_o)}{\sum_{i\in\mathcal{V}} exp(u_i^\top \bar{\mathbf{v}}_o)}. \tag{8}$$

Furthermore, Dive Into Deep Learning continues as if the length of a text sequence is T and the word at time t is $w^{(t)}$, then for context window of size m the likelihood function of CBOW model is

$$\prod_{t=1}^{T} P(w^{(t)}|w^{(t-m)}, ..., w^{(t-1)}, w^{(t+1)}, ..., w^{(t+m)}). \tag{9}$$

Due to CBOW and skip-gram models being similar, training them is also almost the same. According to Dive Into Deep Learning, to train the CBOW model, the maximum likelihood estimation of the CBOW model is equal to the minimization of the following loss function

$$-\sum_{t=1}^{T} logP(w^{(t)}|w^{t-m)}, ..., w^{(t-1)}, w^{(t+1)}, ..., w^{(t+m)}). \tag{10}$$

where

$$logP(w_c|\mathcal{W}_o) = \mathbf{u}_c^\top \bar{\mathbf{v}}_o - log(\sum_{i \in \mathcal{V}} exp(\mathbf{u}_i^\top \bar{\mathbf{v}}_o)). \tag{11}$$

Through differentiation, the following gradient can be obtained

$$\frac{\partial logP(w_c|\mathcal{W}_o)}{\partial \mathbf{v}_{o_i}} = \frac{1}{2m}(\mathbf{u}_c - \sum_{j \in \mathcal{V}} P(w_j|\mathcal{W}_o \mathbf{u}_j)). \tag{12}$$

Which is gradient with respect to any context word vector $\mathbf{v}_{o_i}$.

2.4 Global Vectors for Word Representation

Global Vectors for Word Representation (GloVe) is an unsupervised learning algorithm. Unlike Word2Vec, GloVe captures global contextual information of words by calculating a global word-word co-occurrence matrix. For example, in a large corpus, the word "liquid" is more likely to co-occur with "water" than "ice", but the word "solid" is more likely to co-occur with "ice" than "steam". According to Agarwal, only the local context of words is captured by Word2Vec (Agarwal, 2022). On the other hand, the entire corpus is considered by GloVe, and a large matrix that can capture the co-occurrence of words within the corpus is created.

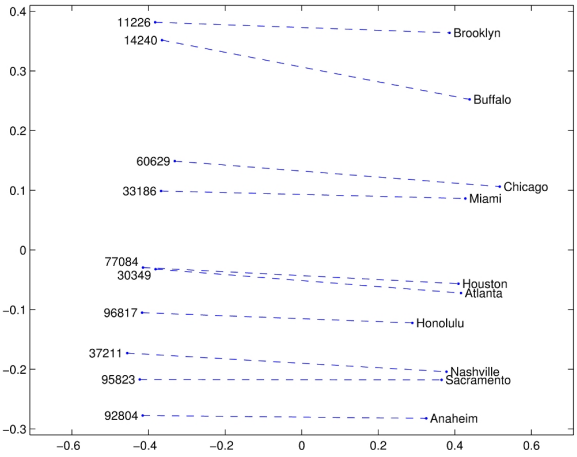

Figure 3: GloVe vectors capturing the relation between city and zip code, from Stanford University

Agarwal continues by GloVe has the combination of the advantages of two-word vector learning methods: matrix factorization like latent semantic analysis (LSA) and local context window method (like Skip-Gram or CBOW). LSA is the technique that analyses the relationship between a set of documents and the terms they contain by using singular value decomposition.

Furthermore, the GloVe method's computational time is reduced by a rather simpler least square error function. As it is stated in Dive Into Deep Learning, Glove makes three changes to the Skip-Gram model square loss (n.d.). where vectors $\mathbf{v}_i \in \mathbb{R}^d$ and $\mathbf{u}_i \in \mathbb{R}^d$ keep same representations as the Skip-Gram model, those their changes are as following:

1. Using variables $p'_{ij} = x_{ij}$ and $q'_{ij} = exp(\mathbf{u}_j^\top \mathbf{v}_i)$ that are not probabilistic distributions. After taking their logarithms, the squared loss term becomes
$(log(p'_{ij}) - log(q'_{ij}))^2 = (\mathbf{u}_j^\top \mathbf{v}_i - log(x_{ij}))^2$.

2. For each word w_i, adding two scalar model parameters: the center word bias b_i and context word bias c_i.

3. Replacing the weight of each loss term $h(x_{ij}$ where $h(x)$ is increasing in the interval of $[0, 1]$.

Therefore, the loss function of GloVe is:

$$\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{V}} h(x_{ij})(u_j^\top \mathbf{v}_i + b_i + c_j - log(x_{ij}))^2 \tag{13}$$

Where suggested choice for $h(x)$ is if $x < c$, then $h(x) = (x/c)^\alpha$, else $h(x) = 1$.

Finally, when compared to Word2Vec GloVe handles out of vocabulary words better. Due to that, GloVe performs better in word analogy and named entity recognition tasks.

2.5 Bidirectional Encoder Representations from Transformers

Bidirectional Encoder Representations from Transformers (BERT) is a family of masked-language models. Before discussing BERT; the terms context-independent, context-sensitive, task-specific, and task-agnostic must be discussed.

A context-independent function $f(x)$ only takes token x as its input. Due to the complex semantics and synonyms in natural languages, context-independent representations miss the meaning of the words in some cases. For example, the word "bank" can be used in both the sentence "I sat by the bank and enjoyed the view of the river." and the sentence I went to the bank to deposit some money". A context-independent algorithm might miss the difference between these cases.

On the other hand, the context-sensitive function $f(x, c(x))$ is dependent on both token x and context $c(x)$. According to Dive Into Deep Learning, some of the most popular context-sensitive representations are language-model-augmented sequence tagger (TagLM), Context Vectors (CoVe), and ELMo (Embeddings from Language Models).

When it comes to task-specific and task-agnostic representation, task-specific representation is when a model is optimized for a specific task, while task-agnostic representation is when a model is independent of a task-based architecture. For instance, ELMo is a task-specific solution while it is not necessary to create specific architecture for each NLP task. The Generative Pre-Training (GPT) model is a context-sensitive, task-agnostic representation. However, according to Dive Into Deep Learning, GPT only looks left to right because of the autoregressive nature of natural languages. For example, the sentences "A crane is

flying." and "A crane is crashed." is taken, because of word "crane" is sensitive to the context on its left, and GPT will return the same representation of "crane".

Both ELMo and GPT fail in some cases. According to Dive Into Deep Learning (n.d.), the combination of both representations BERT, encodes context bidirectionally and requires minimal architecture changes for a wide range of natural language processing tasks. Differences between ELMo, GPT, and BERT can be seen in **Figure 4**.

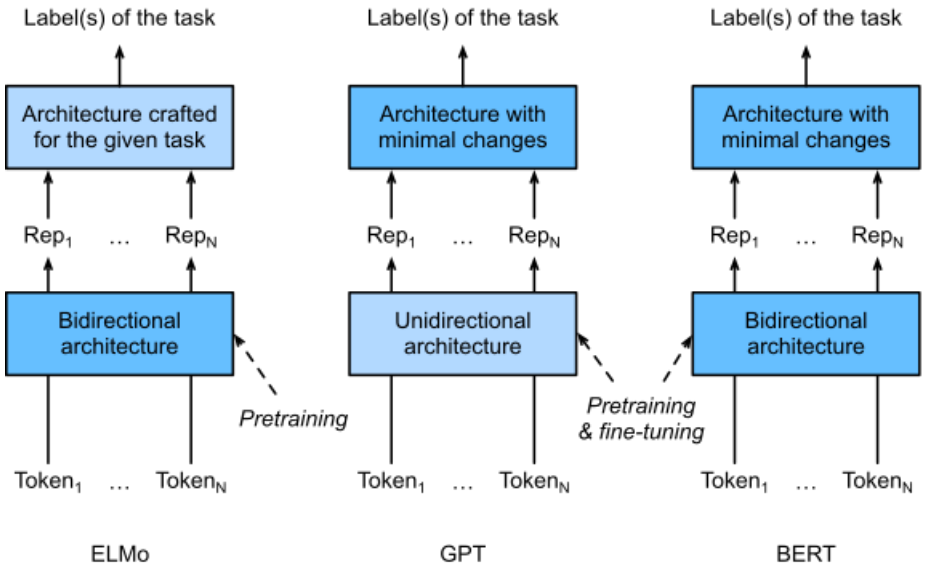

Figure 4: Difference between ELMo, GPT, and BERT, from Dive Into Deep Learning

Furthermore, Dive Into Deep Learning continues by using a pretrained transformer encoder, any token based on its bidirectional context can be represented by BERT. Furthermore, BERT is similar to GPT in two aspects during supervised learning of downstream tasks.

1. BERT representations will be fed into an added output layer with minimal changes to the model architecture.

2. while the additional output layer will be trained from scratch, all the parameters of the pre-trained transformer encoder are fine-tuned.

Finally, Agarwal states that there are two variants of BERT: BERT-Base and BERT-Large. While BERT-Base has 110 million parameters, BERT-Large has 340 million parameters.

3 Word2Vec

In this section, another natural language processing technique Word2Vec is discussed. As stated in Gensim documentations (n.d.), words are embedded in lower-dimensional vector space by Work2Vec. The method used to compress data into lower dimensions is called autoencoding. After that, in this vector space, vectors which have similarities of context between them are close to each other while words that have different meanings are distant to each other. Word2Vec Model calculates the cosine similarity of those words in order to find relations between them. Due to that, the term autoencoders are discussed and it is followed by a discussion of the Word2Vec model in this section.

3.1 Autoencoders

In order to understand the Word2Vec model, the term autoencoders must be discussed. An autoencoder reduces input into lower dimensional data by using neural networks, basically compressing them. After that, this reduced form code can be used in code, which might contain mathematical operations or data analysis. Finally, processed data is decoded by a decoder to get an output that has the same size as the input. Simplilearn states that there are five types of autoencoders (Simplilearn, 2023). Those autoencoders are under complete autoencoders, sparse autoencoders, contractive autoencoders, denoising autoencoders, and variational autoencoders. Due to being a method for processing non-mathematical data, Autoencoders are mainly used in tasks like anomaly detection, feature detection, facial recognition, and word embedding. The typical structure of an autoencoder can be seen in **Figure 5**.

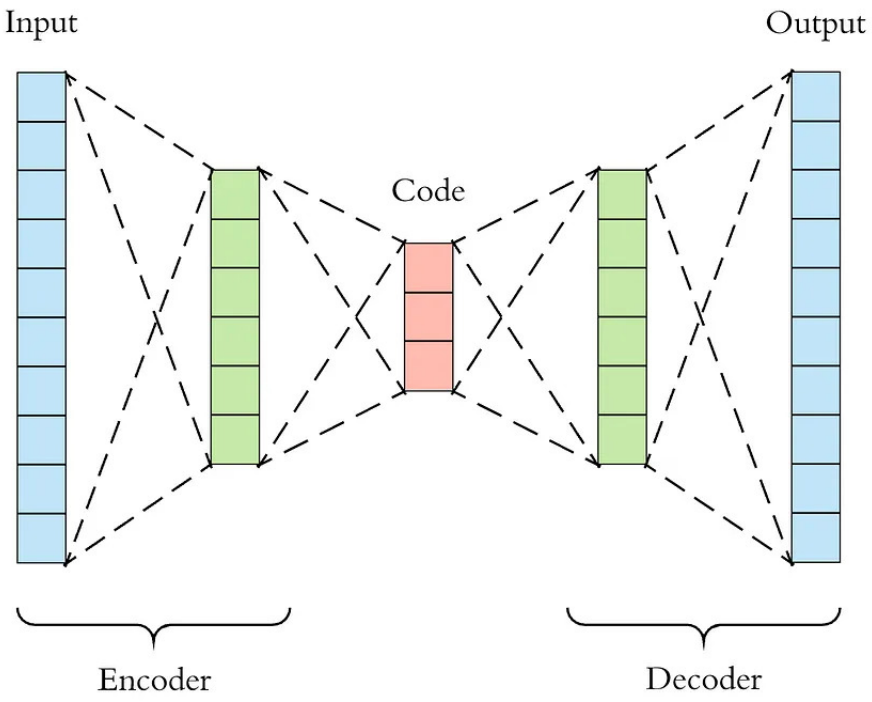

Figure 5: Typical autoencoding structure, from Towards Data Science

3.2 Word2Vec

After obtaining vectors, Word2Vec uses the cosine similarity metric to measure the similarity of the words. Due to that, this architecture is similar to autoencoders, where there are encoder and decoder layers. If the cosine value of two words is 0, then words do not hold similarity. If the cosine value of two words is 1, then the words are overlapping. Due to that, the Word2Vec model is mostly used in semantic analysis. On the other hand, the main disadvantage of the Word2Vec model is that it can not handle vocabulary words well. The words that were not present in training data are called out of vocabulary words. As Chandran stated (2020), a random vector representation is assigned for out-of-vocabulary words by Word2Vec, and they can be not optimal. Word2Vec constructs vectors with two methods: Skip-Gram and CBOW methods. Those two methods are discussed in previous chapters. Briefly, Skip-Gram generates surrounding words by the

center word, while CBOW does the opposite: generating the center word by surrounding words. In the following application section, both Skip-Gram and CBOW models are used.

4 Application

In this section, Skip-Gram and CBOW are applied to The Divine Comedy by Dante Alighieri. Before applying models to The Divine Comedy, brief information about the dataset is given and then the dataset is prepared for the models.

4.1 Information About the Dataset

The Divine Comedy is an epic poem written by Italian poet Dante Alighieri in the 14th century. While it is considered one of the greatest works of literature, the poem is divided into three parts. Those three parts represent different realms of the afterlife and consist of Inferno (Hell), Purgatorio (Purgatory), and Paradiso (Paradise).

The Divine Comedy is chosen for this paper due to several reasons. Firstly and most significantly, The Divine Comedy is a voluminous work that consists of a wide range of characters, concepts, and names. Due to its rich vocabulary, it provides a great spectrum of words to use for the model. Secondly, The Divine Comedy has a great cultural significance for both European and world literature. Due to this importance, it has countless adaptations in various languages, and finding these adaptations is easier compared to other works of literature. Finally, The Divine Comedy explores fundamental aspects of human nature like sin, love, and redemption. The concepts explored in The Divine Comedy are mostly universal and due to that appear in different languages. Thanks to that, working on The Divine Comedy can show how language reflects universal concepts. For this paper, versions of The Divine Comedy in different languages are used and the data is taken from Project Gutenberg. Six different versions of The Divine Comedy in different languages are used and those languages are Dutch, English, Finnish, German, and Italian.

4.2 Application

Firstly, required libraries are imported. pandas library is used for data manipulation and analysis, while functions from the nltk library are used for text processing: word_tokenize is used for tokenization, stopwords is used for removing stopwords, and WordNetLemmatizer is used for lemmatization. In the following, gensim is used for the Word2Vec model. matplotlib.pyplot is used for plotting. Also, cosine_similarity is used for calculating cosine similarity between two vectors while dendrogram and linkage are used for hierarchical clustering.

In [ ]:
```
import pandas as pd
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import gensim
import gensim.downloader as api
from gensim.models import word2vec
from simalign import SentenceAligner
import matplotlib.pyplot as plt
from sklearn.metrics.pairwise import cosine_similarity
from scipy.cluster.hierarchy import dendrogram, linkage
```

After importing the required libraries, data is imported. Data is acquired by Gutenberg Project as discussed in previous chapters and saved as txt files.

In [ ]:
```
Dutch = open('Dutch.txt').read()
English = open('English.txt').read()
Finnish = open('Finnish.txt').read()
German = open('German.txt').read()
Italian = open('Italian.txt').read()
languages = [Dutch, English, Finnish, German, Italian]
names = ['Dutch', 'English', 'Finnish', 'German', 'Italian']
```

This is followed by lemmatization and tokenization processes. Lemmatization is grouping the inflected forms of words to analyze them as a single item, while tokenization is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens. Tokenization is followed by the removal of stopwords from the dataset. These processed datasets are saved as tokenized_"language_name" variables.

In [ ]:
```
def data_tokenizer(language, language_name, encoding = 'utf-8'):
    lemmatized = WordNetLemmatizer().lemmatize(language)
    tokenized = nltk.word_tokenize(lemmatized)
    f=[word.lower() for word in tokenized if word.isalpha()]
    stop_words = set(nltk.corpus.stopwords.words(language_name))
    [stopped] = [[i for i in j if i not in stop_words] for j in [f]]
    return stopped

tokenized_Dutch = data_tokenizer(Dutch,'Dutch')
tokenized_English = data_tokenizer(English,'English')
tokenized_Finnish = data_tokenizer(Finnish,'Finnish')
tokenized_German = data_tokenizer(German,'German')
tokenized_Italian = data_tokenizer(Italian,'Italian')
tokenized_names = 'tokenized_'+pd.Series(names)
tokenized_languages = [tokenized_Dutch, tokenized_English, tokenized_Finnish, toke
```

For more consistent models, only the most common 50 words are selected and used for further analysis. This is done by using most_common_words function. This function takes a list and transforms it into a dataset, then counts the number of words and sorts them in descending order. Finally, it returns the most common 50 words.

In [ ]:
```
def most_common_words(lang):
    df = pd.DataFrame(lang, columns = ['Language'])
    df_sorted = df.groupby(['Language'])['Language'].count().reset_index(
                        name='Count').sort_values(['Count'], ascending=False)
    return df_sorted.Language[:50].reset_index(drop=True)

Dutch_most_common = most_common_words(tokenized_Dutch)
English_most_common = most_common_words(tokenized_English)
Finnish_most_common = most_common_words(tokenized_Finnish)
German_most_common = most_common_words(tokenized_German)
Italian_most_common = most_common_words(tokenized_Italian)
most_common_names = 'most_common_'+pd.Series(names)
most_common_languages = [Dutch_most_common, English_most_common, Finnish_most_comm
most_common_words_ = pd.DataFrame(most_common_languages).T
most_common_words_.columns = most_common_names
```

The most common words can are as following:

In [ ]:
```
most_common_words_
```

Out[ ]:

	most_common_Dutch	most_common_English	most_common_Finnish	most_common_German	n
0	den	thou	ma	sprach	
1	gij	one	mi	sah	
2	zoo	thee	mut	drum	
3	zóó	unto	näin	schon	
4	wanneer	upon	sa	mehr	
5	zeide	said	mun	wohl	
6	wij	thy	jo	ward	
7	waar	us	mulle	licht	
8	mijne	made	min	gleich	
9	oogen	eyes	kaikki	wer	
10	voorts	doth	mua	wort	
11	gelijk	may	ett	o	
12	eene	thus	vain	geist	
13	waarom	saw	näät	blick	
14	zag	see	sun	einst	
15	zijne	shall	toinen	sieh	
16	welke	still	sitten	sei	
17	gaan	first	ois	schien	
18	o	even	myös	kraft	
19	weg	turned	niinkuin	erst	
20	zie	would	siks	ganz	
21	zien	great	vielä	macht	
22	boven	good	ennen	allein	
23	aldus	within	taas	welt	
24	alle	love	kuinka	gesang	
25	uwe	make	ken	fort	
26	gaat	light	tää	nie	
27	dien	round	hälle	drauf	
28	des	er	oi	voll	
29	achter	little	voi	gott	
30	hen	come	ynnä	bald	
31	licht	mine	tään	je	
32	weinig	world	laulu	meister	
33	hemel	people	ol	himmel	
34	zon	much	virkkoi	glanz	
35	anderen	time	siellä	zeit	

	most_common_Dutch	most_common_English	most_common_Finnish	most_common_German	r
36	zijt	heaven	silloin	augen	
37	liefde	art	ettei	kreis	
38	ziel	without	täällä	grund	
39	komt	god	maan	glut	
40	weet	well	sua	liebe	
41	eenen	forth	lausui	gut	
42	berg	far	alas	beim	
43	wel	like	nähdä	leben	
44	goede	already	kaiken	kaum	
45	goed	every	nää	muß	
46	onze	came	sinne	rief	
47	noch	whence	sulle	eh	
48	groote	way	taivaan	stand	
49	één	place	päällä	auge	

After that, the most common 50 words are aligned with each other to be used in the word2vec model. This is done by the alignment function. This function takes a language and aligns it with English. It returns aligned_"language_name" variables. While aligning, the mwmf key is used because it has the best results.

In [ ]:
```
aligner =  SentenceAligner(model="bert", token_type="bpe", matching_methods="mai")
def alingment(language):
    aligned = aligner.get_word_aligns(English_most_common.to_list(), language.to_l
    mwmf = pd.DataFrame(aligned['mwmf'])
    return language.reindex(mwmf[0]).reset_index(drop=True)
```

```
Some weights of the model checkpoint at bert-base-multilingual-cased were not used
when initializing BertModel: ['cls.predictions.decoder.weight', 'cls.seq_relations
hip.bias', 'cls.seq_relationship.weight', 'cls.predictions.bias', 'cls.prediction
s.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias', 'cls.pr
edictions.transform.dense.weight', 'cls.predictions.transform.dense.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a mode
l trained on another task or with another architecture (e.g. initializing a BertFo
rSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a
model that you expect to be exactly identical (initializing a BertForSequenceClass
ification model from a BertForSequenceClassification model).
2023-06-09 00:57:36,598 - simalign.simalign - INFO - Initialized the EmbeddingLoad
er with model: bert-base-multilingual-cased
```

In [ ]:
```
Dutch_aligned = alingment(Dutch_most_common)
English_aligned = alingment(English_most_common)
Finnish_aligned = alingment(Finnish_most_common)
German_aligned = alingment(German_most_common)
Italian_aligned = alingment(Italian_most_common)
aligned_names = 'aligned_'+pd.Series(names)
```

```
aligned_languages = [Dutch_aligned, English_aligned, Finnish_aligned, German_align
Aligned_DataFrame = pd.DataFrame(aligned_languages).T
Aligned_DataFrame.columns = names
```

After aligning, the most common words after being aligned are as follows:

In [ ]:
```
Aligned_DataFrame
```

Out[ ]:

	Dutch	English	Finnish	German	Italian
0	den	thou	ma	sprach	ch
1	den	one	ma	sprach	ch
2	gij	thee	mi	sah	sì
3	zoo	unto	mut	drum	de
4	zóó	upon	näin	drum	d
5	wanneer	said	sa	schon	d
6	zeide	thy	mun	schon	s
7	wij	us	jo	mehr	quel
8	wij	made	jo	wohl	me
9	waar	eyes	mulle	ward	poi
10	mijne	doth	min	ward	così
11	oogen	may	kaikki	licht	là
12	voorts	thus	mua	gleich	quando
13	voorts	saw	mua	wer	quando
14	gelijk	see	ett	wort	m
15	eene	shall	vain	wort	già
16	waarom	still	näät	o	tanto
17	zag	first	sun	geist	son
18	zijne	even	toinen	blick	altro
19	welke	turned	sitten	einst	occhi
20	gaan	would	ois	sieh	qual
21	o	great	myös	sei	ben
22	weg	good	niinkuin	schien	disse
23	zie	within	siks	kraft	sé
24	zien	love	vielä	erst	lor
25	boven	make	ennen	ganz	ché
26	aldus	light	taas	macht	qui
27	alle	round	kuinka	allein	fa
28	uwe	er	ken	welt	né
29	gaat	little	tää	gesang	or
30	dien	come	hälle	fort	com
31	des	mine	oi	nie	vidi
32	achter	world	voi	drauf	ogne
33	hen	people	ynnä	voll	elli
34	licht	much	tään	gott	pur
35	weinig	time	laulu	bald	però

	Dutch	English	Finnish	German	Italian
36	hemel	heaven	ol	je	esser
37	zon	art	virkkoi	meister	ciò
38	anderen	without	siellä	himmel	giù
39	zijt	god	silloin	glanz	altra
40	liefde	well	silloin	zeit	tal
41	ziel	forth	ettei	augen	prima
42	komt	far	täällä	kreis	n
43	weet	like	maan	grund	ancor
44	eenen	already	sua	glut	poco
45	berg	every	lausui	liebe	mondo
46	wel	came	alas	gut	te
47	goede	whence	nähdä	beim	onde
48	goed	way	kaiken	leben	sù
49	onze	place	nää	kaum	mai
50	noch	NaN	sinne	muß	terra
51	groote	NaN	sulle	rief	fuor
52	één	NaN	taivaan	eh	sanza
53	NaN	NaN	päällä	eh	NaN
54	NaN	NaN	NaN	stand	NaN
55	NaN	NaN	NaN	auge	NaN

These aligned words are going to be used in the Word2Vec model. There will be 2 Word2Vec models for each language. One will be trained with Skip-Gram, while the other will be trained with CBOW.

In [ ]:
```
def skipgram(language):
    return gensim.models.Word2Vec(language, vector_size = 50, sg = 1).wv
def cbow(language):
    return gensim.models.Word2Vec(language, vector_size = 50,sg = 0).wv

skipgram_Dutch = skipgram(Dutch_aligned)
skipgram_English = skipgram(English_aligned)
skipgram_Finnish = skipgram(Finnish_aligned)
skipgram_German = skipgram(German_aligned)
skipgram_Italian = skipgram(Italian_aligned)

cbow_Dutch = cbow(Dutch_aligned)
cbow_English = cbow(English_aligned)
cbow_Finnish = cbow(Finnish_aligned)
cbow_German = cbow(German_aligned)
cbow_Italian = cbow(Italian_aligned)
```

For using the Word2Vec model in clustering, each word must be represented by a vector instead of a matrix. Due to that, the following flat() function is for flattening the language matrices. This function takes the language matrix and transforms it into a list. After that, it flattens the list and returns it.

In [ ]:
```
def flat(model):
    vocab = list(model.index_to_key)
    vectors = model[vocab]
    vectors_flatten = vectors.flatten()
    return vectors_flatten
```

In this part, the Skip-Gram model will be used for clustering. Firstly, each language is flattened and saved as an array. After that, those vectors are combined as a dataframe named skipgram. Names of the languages are the index of this dataset and columns are corresponding vectors. NaN values are dropped to be able to use the dataframe in clustering.

In [ ]:
```
flat_skipgram_Dutch = flat(skipgram_Dutch)
flat_skipgram_English = flat(skipgram_English)
flat_skipgram_Finnish = flat(skipgram_Finnish)
flat_skipgram_German = flat(skipgram_German)
flat_skipgram_Italian = flat(skipgram_Italian)
skipgram = pd.DataFrame([flat_skipgram_Dutch, flat_skipgram_English, flat_skipgram_
```

After creating the skipgram dataframe, cosine similarity is calculated for the dataset. This metric returns the cosine value of the angle between two vectors. If this cosine value is 1, it means that the two vectors are identical. If it is 0, it means that two vectors are orthogonal. If it is -1, it means that the two vectors are opposite of each other. After calculating cosine similarity, the linkage is used for hierarchical clustering. This linkage function takes cosine similarity as input and returns a linkage matrix. A linkage matrix is a matrix that contains information about hierarchical clustering. This is followed by plotting the dendrogram. Dendrogram is a tree diagram that shows the arrangement of the clusters produced by hierarchical clustering. The x label of the dendrogram is the languages, while the Y label is the distance between clusters. The dendrogram can be seen below.

In [ ]:
```
skipgram_similarity = cosine_similarity(skipgram)
Z = linkage(skipgram_similarity, 'ward')
plt.figure(figsize=(16, 9))
dendrogram(Z, leaf_rotation=90, leaf_font_size=7., labels = skipgram.index)
plt.title('Dendrogram Created by Skip-Gram')
plt.ylabel('Distance')
plt.xlabel('Language')
plt.xticks(rotation = 45, fontsize = 10)
plt.show()
print("Figure 6: Dendrogram Created by Skip-Gram")
```

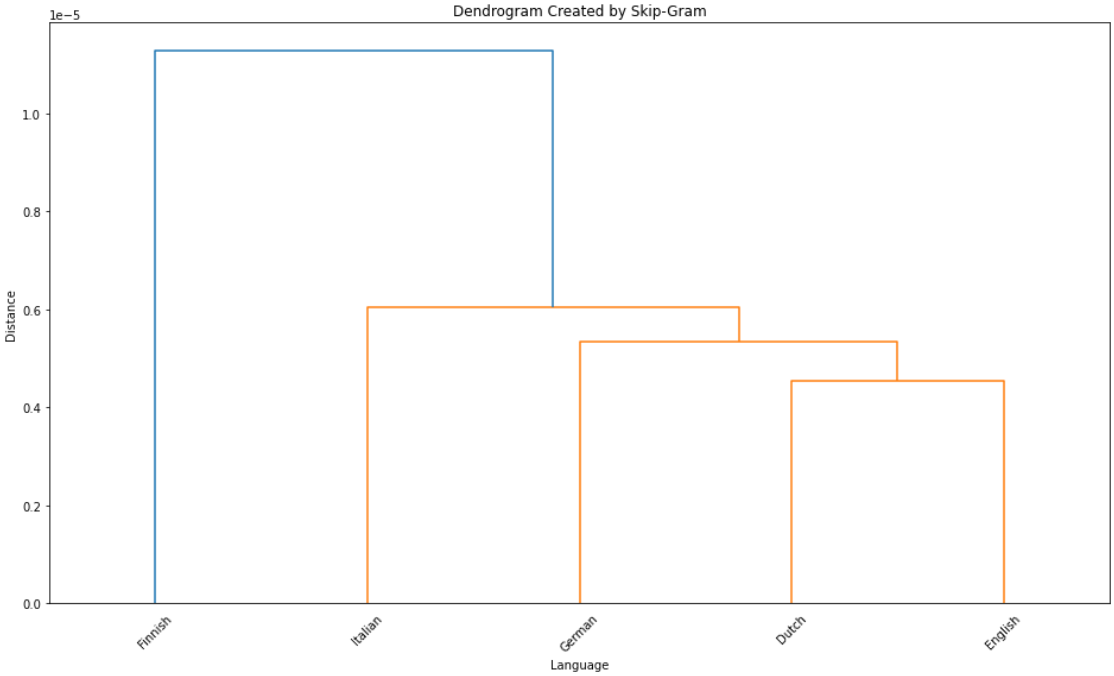

Figure 6: Dendrogram Created by Skip-Gram

As can be seen in the dendrogram, Finnish is clustered differently from the other 4 languages. This is caused by the fact that Finnish is not an Indo-European language. Finnish is a Uralic language, which is a language family that contains languages such as Hungarian and Estonian. Furthermore, Italian is also clustered differently from the other 3 languages. This is caused by the fact that Italian is a Romance language (National Geographic, 2022), which is a language family that contains languages such as Spanish and French. Finally, Dutch and English are clustered together instead of German. This might be caused by the fact that German is a High Germanic language while the other two aren't.

After Skip-Gram, the CBOW model will be used for clustering. Firstly, each language is flattened and saved as an array. After that, those vectors are combined as a dataset named cbow. Names of the

languages are the index of this dataset and columns are corresponding vectors. To use in clustering, NaN values are dropped.

In [ ]:
```
flat_cbow_Dutch = flat(cbow_Dutch)
flat_cbow_English = flat(cbow_English)
flat_cbow_Finnish = flat(cbow_Finnish)
flat_cbow_German = flat(cbow_German)
flat_cbow_Italian = flat(cbow_Italian)
cbow = pd.DataFrame([flat_cbow_Dutch, flat_cbow_English, flat_cbow_Finnish, flat_cl
```

cbow dataframe is used to calculate cosine similarity with the function cosine_similarity. While this section of the code is the same as the previous one, it is repeated to be able to compare results. After calculating cosine similarity, the linkage is used for hierarchical clustering. This linkage function takes cosine similarity as input and returns a linkage matrix. This is followed by plotting a dendrogram. Dendrogram is a tree diagram that shows the arrangement of the clusters produced by hierarchical clustering. Labels are the same as Skip-Gram dendrogram: The x label of the dendrogram is the languages, while the y label is the distance between clusters. The dendrogram can be seen below.

In [ ]:
```
cbow_similarity = cosine_similarity(cbow)
Z_cbow = linkage(cbow_similarity, 'ward')
plt.figure(figsize=(16, 9))
dendrogram(Z_cbow, leaf_rotation=90, leaf_font_size=7., labels = cbow.index)
plt.title('Dendrogram Created by CBOW')
plt.ylabel('Distance')
plt.xlabel('Language')
plt.xticks(rotation = 45, fontsize = 10)
plt.show()
print("Figure 7: Dendrogram Created by CBOW")
```

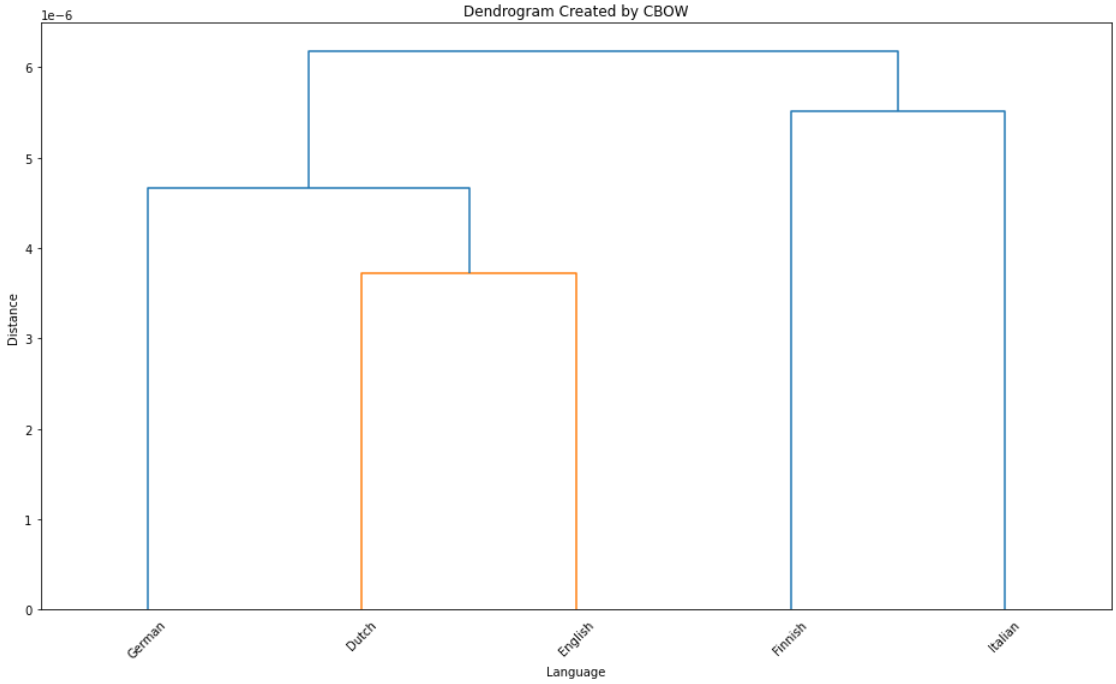

Figure 7: Dendrogram Created by CBOW

In this dendrogram, it can be seen that Dutch and English are clustered close to each other and German is the closest language to them. This relationship is caused by the same reason as the Skip-Gram dendrogram: While Dutch and English are West Germanic languages, German is a High Germanic language. Furthermore, instead of being clustered with other Indo-European languages, Italian is clustered with the Uralic language Finnish in this dendrogram. If Italian is excluded, the clustering is the same as the Skip-Gram dendrogram, it can be said that clustering by the CBOW algorithm is partially successful.

5 Discussion

All in all, two different dendrograms are obtained by Skip-Gram and CBOW in order. **Figure 7** visualize a dendogram as desired, while **Figure 8** partially succeeds this task. If **Figure 7** is examined, It can be seen that Finnish, the only Uralic language in the dataset, is clustered differently from the rest of the dataset, while the other four Indo-European languages are clustered together. If these four languages are examined further, the only romance language Italian is also clustered further than the other three languages which are all Germanic. This clustering shows the relationship of languages correctly and It can be said that the Skip-Gram model succeeded in this task. In addition, English and Dutch are clustered closer to each other rather than German.
When moved to **Figure 8**, the only difference from the Skip-Gram dendrogram is the clustering of Italian. Instead of being clustered with other Indo-European languages, Italian is clustered with the Uralic language Finnish. Even though culturally and geographically there is no close relation between Finland and Italy, this clustering might be caused by the characteristic of the dataset or word-co-occurrence patters in the dataset. This can be tested by constructing a model with different datasets. Other than Italy, the CBOW model clustered languages the same as the Skip-Gram model.

6 Conclusion

To conclude, Dante Alighieri's Divine Comedy is studied in this paper as a dataset, and different word embedding techniques are used on it. Relationships of languages are investigated with the help of the most common words that occured in the dataset. To take this study further, one can study a dataset that contains more languages and investigate why Finnish and Italian are clustered together in the CBOW model.

References

[1] Agarwal, N. (2022). *The Ultimate Guide To Different Word Embedding Techniques In NLP*. KDnuggets. Retrieved from https://www.kdnuggets.com/2021/11/guide-word-embedding-techniques-nlp.html

[2] Alighieri, D. (1997). Divine Comedy, Longfellow's Translation, Complete (H. W. Longfellow, Trans.). Retrieved from https://www.gutenberg.org/cache/epub/1004/pg1004-images.html

[3] Alighieri, D. (1997). La Divina Commedia di Dante: Complete. Project Gutenberg. Retrieved from https://www.gutenberg.org/cache/epub/1000/pg1000-images.html

[4] Alighieri, D. (2004). Jumalainen näytelmä. E. Leino (Trans.). Project Gutenberg. Retrieved from https://www.gutenberg.org/cache/epub/12546/pg12546.html

[5] Alighieri, D. (2005). Die Göttliche Komödie. Project Gutenberg. Retrieved from https://www.gutenberg.org/cache/epub/8085/pg8085.html

[6] Alighieri, D. (2012). Dante's Louteringsberg (H. J. Boeken, Trans.). Project Gutenberg. Retrieved from https://www.gutenberg.org/cache/epub/39181/pg39181-images.html

[7] Chandran, S. (2020). *Introduction to Text Representations for Language Processing — Part 2*. Towards Data Science. Retrieved from https://towardsdatascience.com/introduction-to-text-representations-for-language-processing-part-2-54fe6907868

[8] Dive Into Deep Learning. (n.d.). *The Skip Gram Model*. Retrieved from https://d2l.ai/chapter_natural-language-processing-pretraining/word2vec.html#the-skip-gram-model

[9] Dive Into Deep Learning. (n.d.). *The Continuous Bag of Words (CBOW) Model*. Retrieved from https://d2l.ai/chapter_natural-language-processing-pretraining/word2vec.html#the-continuous-bag-of-words-cbow-model

[10] Dive Into Deep Learning. (n.d.). *Bidirectional Encoder Representations from Transformers (BERT)*. Retrieved from https://d2l.ai/chapter_natural-language-processing-pretraining/bert.html#bidirectional-encoder-representations-from-transformers-bert

[11] Gensim. (n.d.). *Introducing: the Word2Vec Model*. Retrieved from https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html#sphx-glr-auto-examples-tutorials-run-word2vec-py

[12] Pennington, J. Socher R., Manning, C. D. (n.d.). *GloVe: Global Vectors for Word Representation*. Retrieved from https://nlp.stanford.edu/projects/glove/

[13] Kınık, D., Güran, A. (2021). TF-IDF ve Doc2Vec Tabanlı Türkçe Metin Sınıflandırma Sisteminin Başarım Değerinin Ardışık Kelime Grubu Tespiti ile Arttırılması, *Avrupa Bilim ve Teknoloji Dergisi*, 21, 323-332.

[14] National Geographic. (2022). Retrieved from *Family of Language*. https://education.nationalgeographic.org/resource/family-language/

[15] Simplilearn. (2023). *What are Autoencoders? Introduction to Autoencoders in Deep Learning*. Retrieved from https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-are-autoencoders-in-deep-learning