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
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
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

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 alignment(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 = alignment(Dutch_most_common)
English_aligned = alignment(English_most_common)
Finnish_aligned = alignment(Finnish_most_common)
German_aligned = alignment(German_most_common)
Italian_aligned = alignment(Italian_most_common)
aligned_names = 'aligned_'+pd.Series(names)
```

```
aligned_languages = [Dutch_aligned, English_aligned, Finnish_aligned, German_aligne
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.

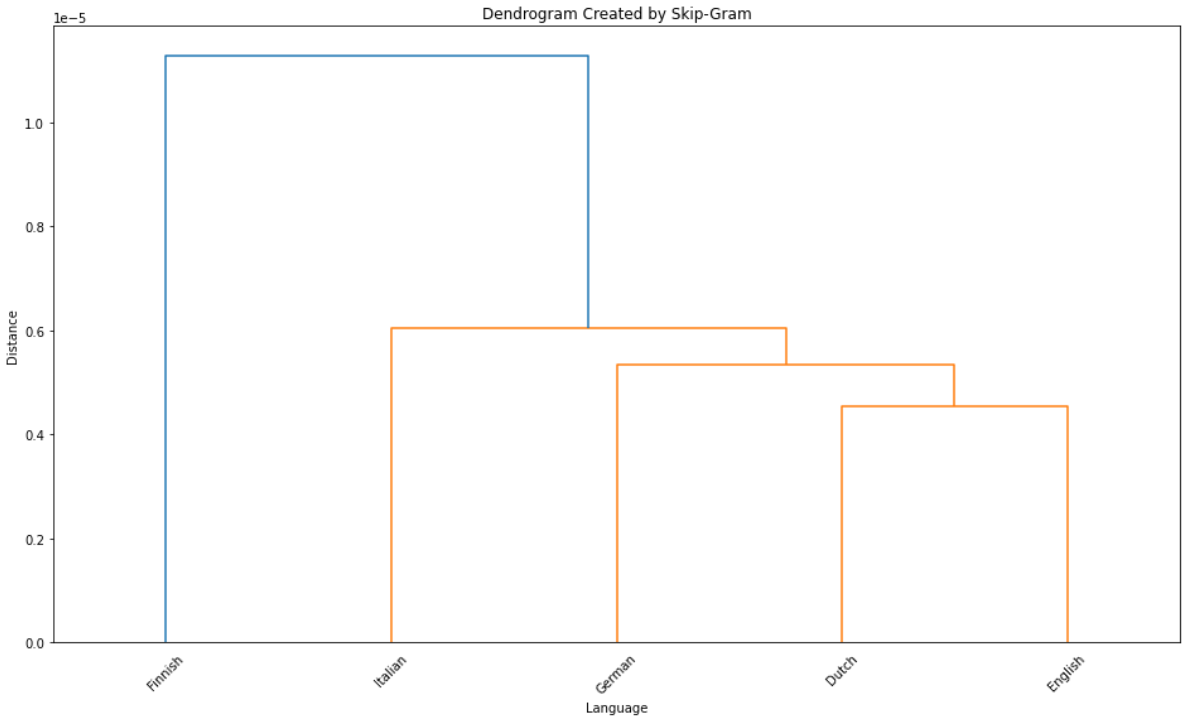
In [ ]:
```
def flat(model):
    vocab = list(model.index_to_key)
    vectors = model[vocab]
    vectors_flatten = vectors.flatten()
    return vectors_flatten
```

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")
```

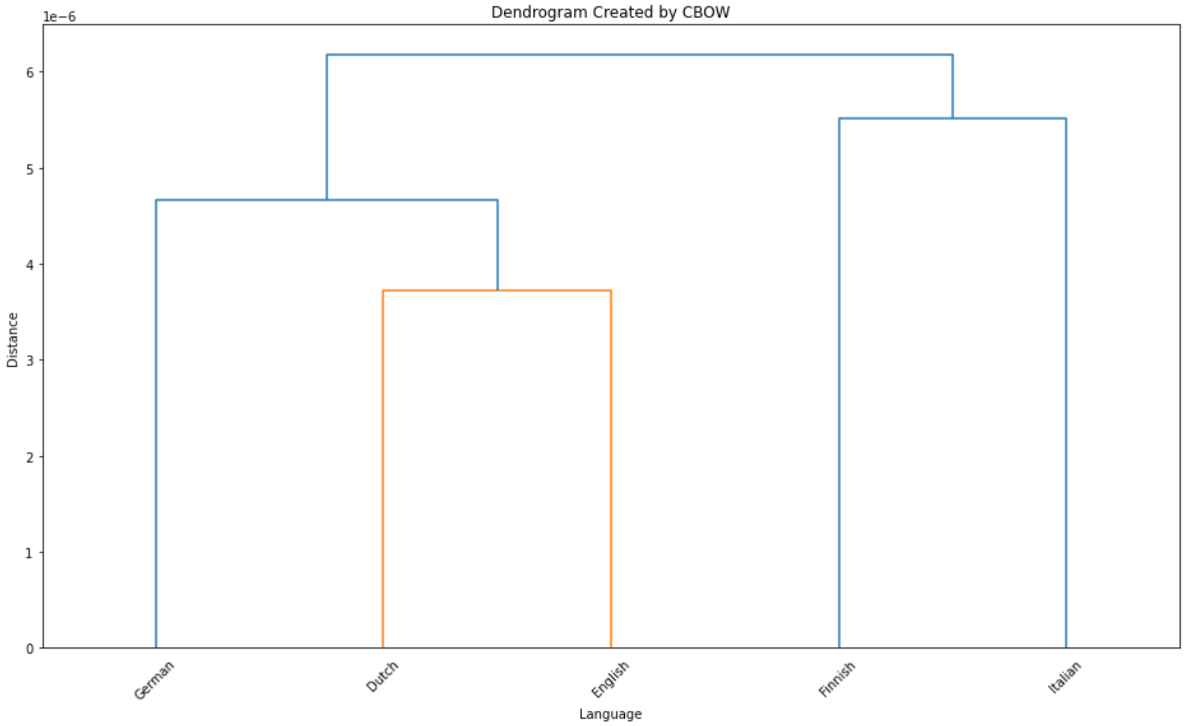


Figure 7: Dendrogram Created by CBOW

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.