In [1]:

```python
# importing the libraries
from IPython.display import display
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
import nltk
from nltk.probability import FreqDist
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from gensim.models import Word2Vec, KeyedVectors
from datasets import load_dataset
import gensim.downloader as api
from sklearn.metrics.pairwise import cosine_similarity
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier, kneighbors_graph
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score
from sklearn.metrics import precision_recall_fscore_support
import warnings
from pandas.core.common import SettingWithCopyWarning
```

In [2]:

```python
warnings.simplefilter(action="ignore", category=SettingWithCopyWarning)
```

In [3]:

```python
# downloading nltk.punkt
try:
    nltk.data.find('tokenizers/punkt')
except LookupError:
    nltk.download('punkt')
```

Defining relevant functions

In [4]:

```python
def word_cloud_plot (data):
    """
    function that creates a word cloud from a specified column of a dataframe
    """
    # create set of stopwords
    stopwords = set(STOPWORDS)

    # Instantiate the word cloud object
    word_cloud = WordCloud(background_color='white',max_words=200,stopwords=stopwords, widt

    # generate the word cloud
    word_cloud.generate(' '.join(data))

    # To display the word cloud
    plt.figure( figsize=(20,10) )
    plt.imshow(word_cloud, interpolation='bilinear')
    plt.axis('off')
    plt.show()
```

In [5]:

```python
def regex_filter(sentence):
    """
    funtion that formats string to remove special characters
    """
    import re
    return re.sub('[^a-zA-Z]', ' ', sentence)
```

In [6]:

```python
def filter_stop_words(token):
    """
    function that removes stopwords from a word-tokenized sentence
    """
    stop_words = set(stopwords.words('english'))
    filtered_token = [word.lower() for word in token if word.lower() not in stop_words]
    return filtered_token
```

In [7]:

```python
def stem_words(token):
    """
    function that stems word-tokenized sentences
    """
    ps = PorterStemmer()
    stemmed_token = [ps.stem(word) for word in token]
    return stemmed_token
```

In [8]:

```
def lemmatize_words(token):
    """
    function that lemmatizes word-tokenized sentences
    """
    lem = WordNetLemmatizer()
    lemmatized_token = [lem.lemmatize(word, 'v') for word in token]
    return lemmatized_token
```

In [9]:

```
def join_token(token):
    """
    function that joins word-tokenized sentences back to single string
    """
    return ' '.join(token)
```

In [10]:

```
def get_embeddings(group, model):
    """
    Function for getting embeddings of words from a word2vec model
    """
    group_embedding = []
    group_labels = []

    unique_words = [word for sentence in group for word in sentence]
    unique_words = list(dict.fromkeys(unique_words))

    for word in unique_words:
        if model.wv.__contains__(word):
            group_embedding.append(list(model.wv.__getitem__(word)))
            group_labels.append(word)

    df_embedding = pd.DataFrame(group_embedding)
    df_word = pd.DataFrame(group_labels, columns = ["Word"])
    df = pd.concat([df_word, df_embedding], axis=1)
    return df
```

In [11]:

```
def similarity(words, stem_model=None, lem_model=None, W2V_pretrained=None, GloVe_pretraine
    """
    function that computes similarity between words for up to four models passed
    """
    if stem_model:
        ps = PorterStemmer()
        stemmed = [ps.stem(word) for word in words]
        try:
            print("Stemmed W2V model similarity between", words[0], "and", words[1], "=", r
        except:
            print("Error: Word not in stem model vocabulary")

    if lem_model:
        lem = WordNetLemmatizer()
        lemma = [lem.lemmatize(word, 'v') for word in words]
        try:
            print("Lemmatized W2V model similarity between", words[0], "and", words[1], "="
        except:
            print("Error: Word not in lemmatized model vocabulary")

    if W2V_pretrained:
        try:
            print("Word2vec pretrained model similarity between", words[0], "and", words[1]
        except:
            print("Error: Word not in Word2vec pretrained model vocabulary")

    if GloVe_pretrained:
        try:
            print("GloVe pretrained model similarity between", words[0], "and", words[1], "
        except:
            print("Error: Word not in GloVe pretrained model vocabulary")
```

In [12]:

```
def tsne_plot(df):
    """
    function that plots annotated scatter plot from a dataframe
    """
    plt.figure(figsize=(18, 18))
    for i in range(len(df)):
        plt.scatter(df.iloc[i,1],df.iloc[i,2])
        plt.annotate(df.iloc[i,0],
                     xy=(df.iloc[i,1], df.iloc[i,2]),
                     xytext=(5, 2),
                     textcoords='offset points',
                     ha='right',
                     va='bottom')
    plt.show()
```

In [13]:

```
def get_sentence_embedding(data, column, train_word_embedding, test_word_embedding):
    """
    function that creates a sentence embedding from the embeddings of the individual words
    sentence_embedding = average of word embeddings for all words in the sentence
    """
    data.reset_index(inplace=True, drop = True)
    sentence_embeddings = []
    for token in data[column]:
        embeddings = []
        for word in token:
            if word in train_word_embedding.index:
                embeddings.append(train_word_embedding.loc[word])
            else:
                embeddings.append(test_word_embedding.loc[word])

        embedding_array = np.array(embeddings)
        sentence_embedding = np.mean(embedding_array, axis=0)
        sentence_embeddings.append(list(sentence_embedding))

    features = len(sentence_embeddings[0])
    df = pd.DataFrame(sentence_embeddings, columns = ["feature_"+ str(i+1) for i in range(f
    df = pd.concat([data["claim"], df, data["claim_label"]], axis=1)
    return df
```

In [14]:

```
def get_most_similar_words(embedding, n_similar = 1):
    """
    function that returns n_similar most similar words to a particular word in an embedding
    embedding is n x n square matrix of relationship (similarity) between words
    """
    n_similar += 1
    similar = pd.DataFrame(columns = ['most_similar_'+ str(i) for i in range(1, n_similar)]

    embedding_T = embedding.T
    for word in embedding.index:
        most_similar = list(embedding_T.nlargest(n = n_similar, columns = word).index)
        if word in most_similar:
            most_similar.remove(word)
        else:
            most_similar = most_similar[:-1]

        similar.loc[word] = most_similar

    return similar
```

In [15]:

```
def precision_recall_fscore(y_true, y_pred):
    """
    function that computes the precision, recall and fscore between 2 dataframes across n_c
    returns the average precision, recall and fscore across the n_columns
    """
    if len(y_true) != len(y_pred):
        print("Error in dimensions of inputs")
        return

    n_columns = len(y_true)
    metrics = []

    for i in range(n_columns):
        metric = list(precision_recall_fscore_support(y_true.iloc[:,i], y_pred.iloc[:,i], a
        metrics.append(metric[:-1])

    metrics = np.mean(np.array(metrics), axis=0)

    print("Precision: ", round(metrics[0], 2))
    print("Recall: ", round(metrics[1], 2))
    print("F1_score: ", round(metrics[2], 2))
```

In [16]:

```
def run_knn_opt(X_train, X_val, X_test, y_train, y_val, y_test, k_values):
    """
    function that performs tunning of k_parameter in KNN classifier
    produces confusion matrix, accuracy, fscore and screeplots
    """
    # Developing the Classification Model
    classifier = KNeighborsClassifier()
    classifier.fit(X_train,y_train)

    # Predicting the test set result
    y_pred = classifier.predict(X_test)

    # Evaluating the Model
    cm = confusion_matrix(y_test,y_pred)

    accuracy_1 = round(100 * accuracy_score(y_test,y_pred), 2)
    f1_score_1 = round(f1_score(y_test, y_pred, average = "weighted"), 2)

    y_pred_train  = classifier.predict(X_train)

    # Making the Confusion Matrix
    cm_train = pd.DataFrame(confusion_matrix(y_train,y_pred_train))
    cm_test = pd.DataFrame(confusion_matrix(y_test,y_pred))

    print("***** Training Set Evaluation *****\n")
    print("confusion Matrix")
    display(cm_train)
    print("Accuracy: ", round(100 * accuracy_score(y_train, y_pred_train), 2))
    print("F1_score: ", round(f1_score(y_train, y_pred_train, average = 'weighted'), 2))

    print("\n\n***** Test Set Evaluation *****\n")
    print("confusion Matrix")
    display(cm_test)
    print("Accuracy: ", accuracy_1)
    print("F1_score: ", f1_score_1)

    accuracy = {}
    for k in k_values:
        classifier = KNeighborsClassifier(n_neighbors=k)
        classifier.fit(X_train,y_train)

        # Predicting the test set result
        y_pred = classifier.predict(X_val)

        model_accuracy = accuracy_score(y_val, y_pred)

        accuracy[k] = round(model_accuracy * 100, 2)

    # plotting the parameter vs accuracy graph
    sns.lineplot(x = k_values, y = accuracy.values())
```

Downloading the dataset

In [17]:

```
dataset = load_dataset('climate_fever')

df = dataset['test'].to_pandas()
df2 = pd.json_normalize(dataset['test'], 'evidences', ['claim', 'claim_id','claim_label'],

data1 = df[['claim', 'claim_label']]
data2 = df2[['evidence','evidence_label']]
```

```
Using custom data configuration default
Reusing dataset climate_fever (C:\Users\jubil\.cache\huggingface\datasets\cl
imate_fever\default\1.0.1\3b846b20d7a37bc0019b0f0dcbde5bf2d0f94f6874f7e4c398
c579f332c4262c)
```

Data preparation

Claim Data

In [18]:

```
# filter with regex
data1.loc[:, 'claim_token'] = data1.loc[:, 'claim'].apply(regex_filter)

# Tokenizing the claims
data1.loc[:, 'claim_token'] = data1.loc[:, 'claim_token'].apply(nltk.word_tokenize)

# Removing stop words from the claclaim_tokenim tokens
data1.loc[:,'claim_token'] = data1.loc[:,'claim_token'].apply(filter_stop_words)

# Stemming the words
data1.loc[:,'stemmed_words'] = data1.loc[:,'claim_token'].apply(stem_words)

# Lemmatizing the words
data1.loc[:,'lemmatized_words'] = data1.loc[:,'claim_token'].apply(lemmatize_words)
```

Evidence Data

In [19]:

```
# Adding the evidences to increase corpus size

# filer with regex
data2.loc[:, ('evidence_token')] = data2.loc[:, ('evidence')].apply(regex_filter)

# Tokenizing the claims
data2.loc[:, ('evidence_token')] = data2.loc[:, ('evidence_token')].apply(nltk.word_tokeniz

# Removing stop words from the evidence_token tokens
data2.loc[:,('evidence_token')] = data2.loc[:,('evidence_token')].apply(filter_stop_words)

# Stemming the words
data2.loc[:,('stemmed_words')] = data2.loc[:,('evidence_token')].apply(stem_words)

# Lemmatizing the words
data2.loc[:,('lemmatized_words')] = data2.loc[:,('evidence_token')].apply(lemmatize_words)
```

In [20]:

```
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(data1[['claim', 'stemmed_words', 'lemmatized_words
```

In [21]:

```
# creating the stemmed corpus and lemmatized corpus
corpus_stem = list(data1['stemmed_words']) + list(data2['stemmed_words'])
corpus_lem = list(data1['lemmatized_words']) + list(data2['stemmed_words'])
```

In [22]:

```
# Embeding with Word2Vec
model_stem = Word2Vec(corpus_stem, min_count=1)
model_lem = Word2Vec(corpus_lem, min_count=1)
print(model_stem)
print(model_lem)
```

```
Word2Vec(vocab=7433, size=100, alpha=0.025)
Word2Vec(vocab=8894, size=100, alpha=0.025)
```

In [23]:

```python
# Training set embedings [STEMMING]
train_embedding_stem = get_embeddings(list(train_data['stemmed_words']), model_stem)
train_embedding_stem.set_index("Word", inplace=True)
train_embedding_stem.head()
```

Out[23]:

	0	1	2	3	4	5	6	7	
Word									
pdo	-0.007563	0.044511	0.025590	-0.035496	-0.064830	-0.239282	-0.128552	-0.027723	0.
last	-0.130583	0.167503	0.617398	-0.159868	-0.395399	-1.272915	-0.145560	0.325738	0.8
switch	-0.031355	0.004839	0.028413	-0.048590	-0.052671	-0.080059	-0.106902	-0.054508	0.0
cool	-0.021669	0.253109	0.278511	0.182573	-0.496178	-0.906144	0.052943	0.026868	0.6
phase	-0.086800	0.123085	0.126501	-0.072909	-0.184174	-0.569626	-0.351786	-0.063643	0.4

5 rows × 100 columns

In [24]:

```python
# Training set embedings [LEMMATIZING]
train_embedding_lem = get_embeddings(list(train_data['lemmatized_words']), model_lem)
train_embedding_lem.set_index("Word", inplace=True)
train_embedding_lem.head()
```

Out[24]:

	0	1	2	3	4	5	6	7	
Word									
pdo	-0.074875	0.252571	-0.103052	0.053736	0.039467	-0.101775	0.024312	-0.153328	0.1
last	-0.307621	0.735764	0.276465	0.119051	-0.227690	-0.712704	0.301766	-0.359552	0.7
switch	-0.051504	0.115070	-0.033129	0.015233	0.010408	-0.019066	0.002020	-0.105870	0.0
cool	-0.285945	0.853142	-0.080562	0.459503	-0.192052	-0.517079	0.485993	-0.526372	0.7
phase	-0.267871	0.633408	-0.186877	0.156740	0.063929	-0.223141	0.027060	-0.373447	0.4

5 rows × 100 columns

Getting the test set embeddings

In [25]:

```
# Test set embedings [STEMMING]
test_embedding_stem = get_embeddings(list(test_data['stemmed_words']), model_stem)
test_embedding_stem.set_index("Word", inplace=True)
test_embedding_stem.head()
```

Out[25]:

	0	1	2	3	4	5	6	7
Word								
trenberth	-0.017038	0.013299	0.002177	-0.029537	-0.021646	-0.067447	-0.067976	-0.020581
view	-0.018748	0.064682	0.002448	-0.129314	-0.023861	-0.462953	-0.367762	-0.032959
clarifi	-0.001354	0.006540	0.003853	-0.003351	-0.002437	-0.017939	-0.014778	-0.002117
paper	-0.028626	0.070783	-0.072200	-0.289876	0.059990	-0.616429	-0.761886	-0.152431
imper	-0.012223	0.007623	-0.007936	-0.020599	-0.005858	-0.064724	-0.064233	-0.013333

5 rows × 100 columns

In [26]:

```
# Test set embedings [LEMMATIZING]
test_embedding_lem = get_embeddings(list(test_data['lemmatized_words']), model_lem)
test_embedding_lem.set_index("Word", inplace=True)
test_embedding_lem.head()
```

Out[26]:

	0	1	2	3	4	5	6	7
Word								
trenberth	-0.042692	0.106644	-0.039109	0.013503	0.017004	-0.025219	0.002146	-0.066980
view	-0.124928	0.501030	-0.212534	0.048093	0.140766	-0.189625	-0.004337	-0.323847
clarify	0.001797	0.007571	-0.004464	-0.004013	0.001436	-0.003795	0.002320	-0.002964
paper	-0.193147	0.830351	-0.447796	0.003107	0.331940	-0.203153	-0.192300	-0.552235
imperative	-0.002352	0.013623	-0.004010	0.004259	0.006063	-0.002932	0.000861	-0.010286

5 rows × 100 columns

LLE

Locally linear embedding (LLE) seeks a lower-dimensional projection of the data which preserves distances within local neighborhoods. LLE unfolds the non-linear manifold in a piecewise manner. The standard LLE algorithm comprises of 3 steps

1. Construct the KNN graph.
2. Calculate the reconstruction weights for reconstructing every point by its neighbours.

3. Use the obtained weights to embed the points in a low dimensional space.

Q.1

Using Stemming

In [27]:

```
from sklearn.manifold import LocallyLinearEmbedding
```

In [28]:

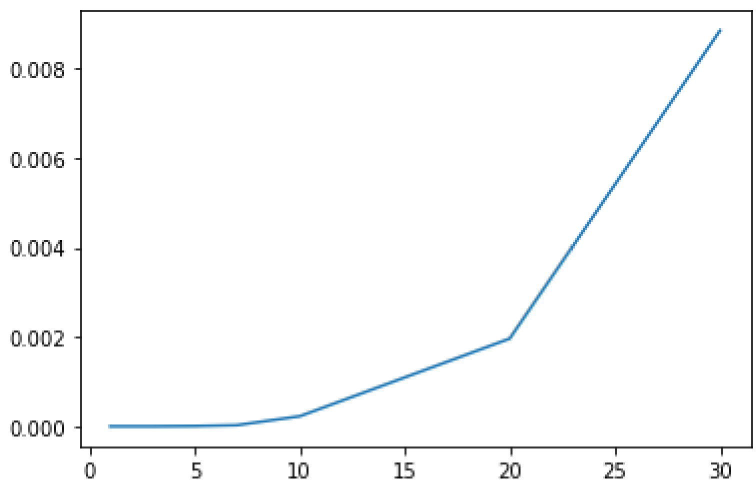
```
n_components = [1, 2, 3, 5, 7, 10, 20, 30]
reconstruction_error = []
for n in n_components:
    lle_model = LocallyLinearEmbedding(n_components=n, random_state=0)
    lle_test = lle_model.fit_transform(test_embedding_stem.iloc[:,:].values)
    reconstruction_error.append(lle_model.reconstruction_error_)
```

In [29]:

```
plt.plot(n_components, reconstruction_error)
```

Out[29]:

```
[<matplotlib.lines.Line2D at 0x220438a4788>]
```



In [30]:

```
%%time
lle_model  = LocallyLinearEmbedding(n_components=4, random_state=0)
lle_test = lle_model.fit_transform(test_embedding_stem.iloc[:,:].values)

lle_test = pd.DataFrame(lle_test, columns = ["feature1", "feature2", "feature3", "feature4"
lle_test.index = test_embedding_stem.index
```

```
Wall time: 338 ms
```

In [31]:

```
lle_test.head()
```

Out[31]:

	feature1	feature2	feature3	feature4
Word				
trenberth	-0.008398	-0.004796	0.005225	0.037093
view	-0.014039	0.016293	0.021334	-0.031066
clarifi	-0.011228	-0.066904	0.010288	-0.007664
paper	0.072247	-0.000074	0.001254	-0.011154
imper	-0.009847	-0.036586	-0.006689	0.015336

In [32]:

```
lle_model.reconstruction_error_
```

Out[32]:

```
7.764554481358607e-07
```

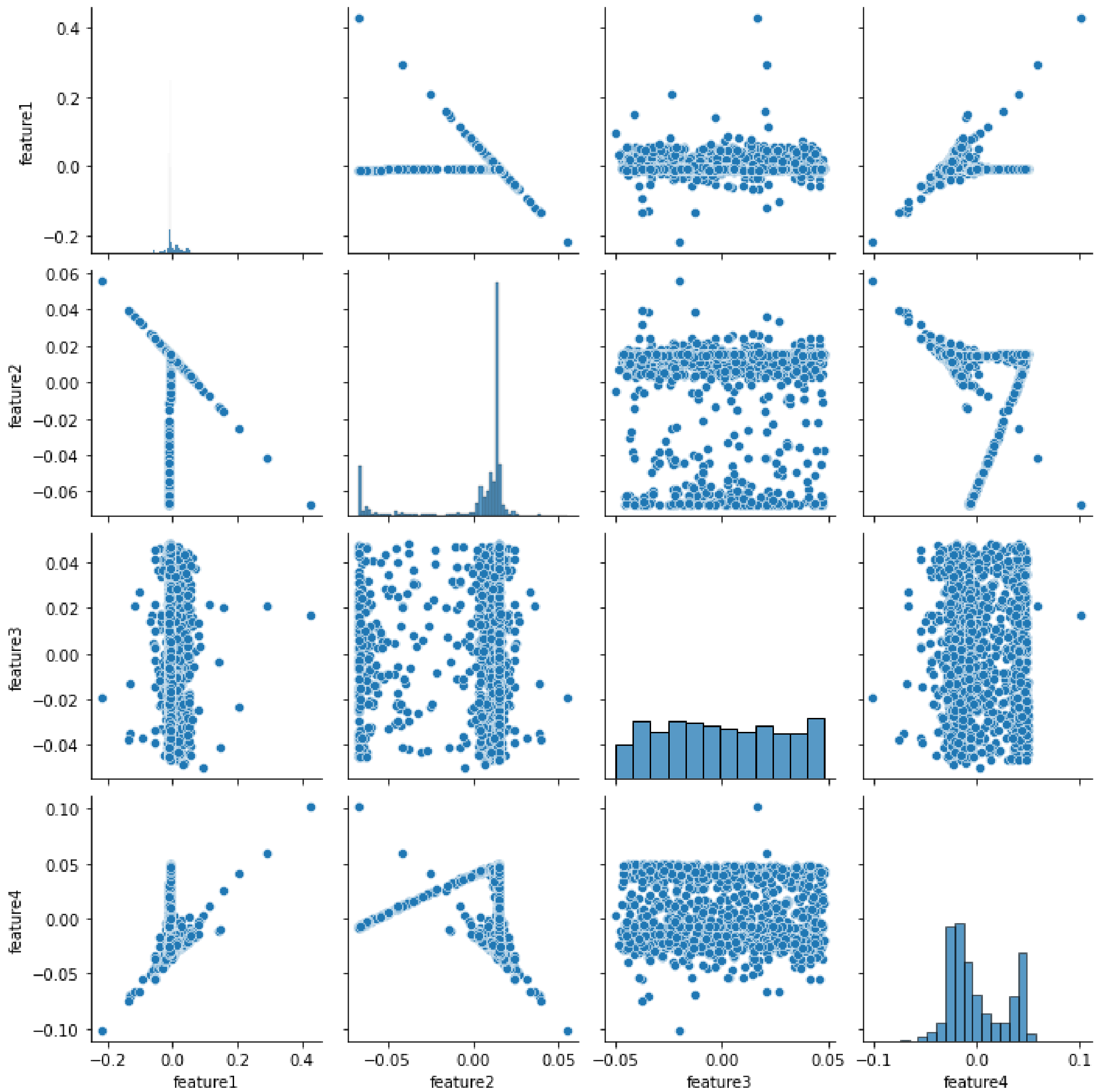
Q.2

In [33]:

```
sns.pairplot(lle_test)
```

Out[33]:

```
<seaborn.axisgrid.PairGrid at 0x22042f88208>
```



Discussions on the LLE Embeddings [STEMMING]

- The largest range of embeddings is 0.1
- The plots of all the features are arbitrary shapes.
- The distribution of the feature 2 is skewed to the right
- The training time for the LLE transformation was 338ms

Using Lemmatization

In [34]:

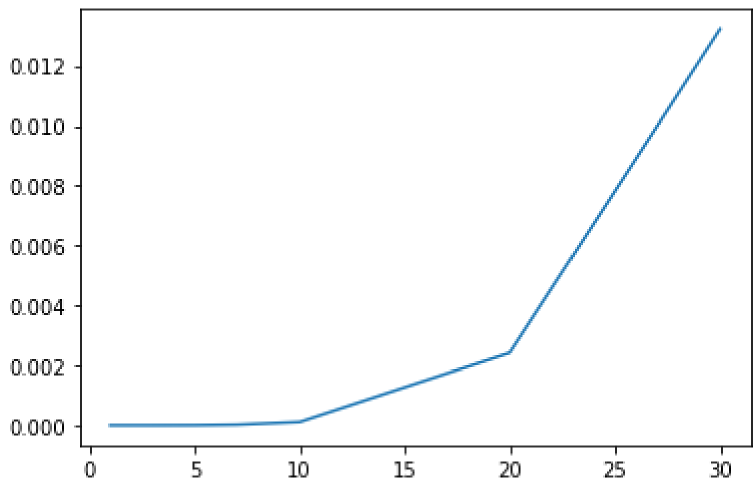
```
n_components = [1, 2, 3, 5, 7, 10, 20, 30]
reconstruction_error = []
for n in n_components:
    lle_model = LocallyLinearEmbedding(n_components=n, random_state=0)
    lle_test_lem = lle_model.fit_transform(test_embedding_lem.iloc[:,:].values)
    reconstruction_error.append(lle_model.reconstruction_error_)
```

In [35]:

```
plt.plot(n_components, reconstruction_error)
```

Out[35]:

```
[<matplotlib.lines.Line2D at 0x22045367988>]
```



In [36]:

```
%%time
lle_model = LocallyLinearEmbedding(n_components=4, random_state=0)
lle_test_lem = lle_model.fit_transform(test_embedding_lem.iloc[:,1:].values)

lle_test_lem = pd.DataFrame(lle_test_lem, columns = ["feature1", "feature2", "feature3", "f
lle_test_lem.index = test_embedding_lem.index
```

```
Wall time: 429 ms
```

In [37]:

```
lle_test_lem.head()
```

Out[37]:

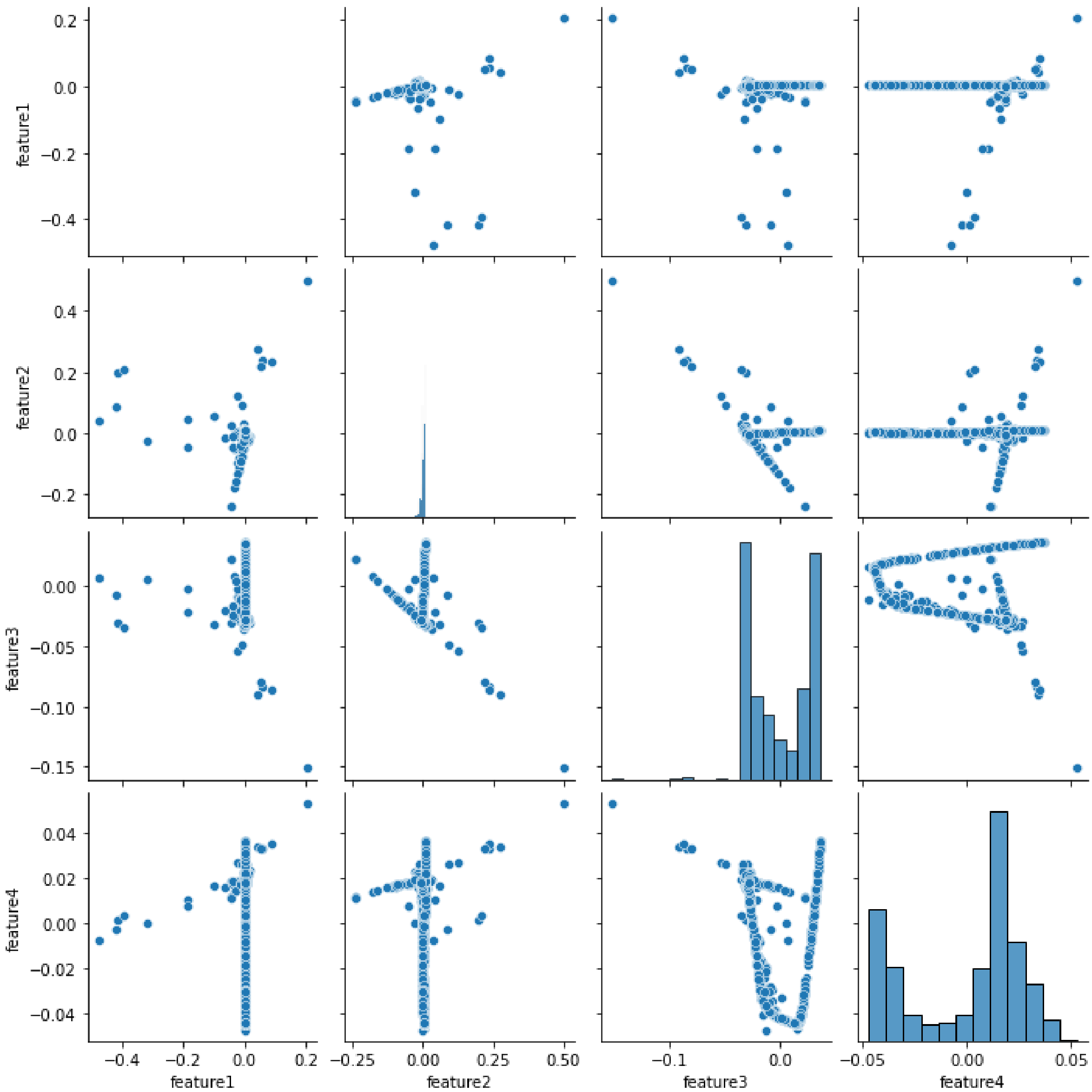
	feature1	feature2	feature3	feature4
Word				
trenberth	0.002264	0.003789	0.016586	-0.043139
view	0.000715	0.002328	-0.031027	0.019274
clarify	0.002293	0.006537	0.035892	0.036470
paper	0.001516	-0.018307	-0.027152	0.019633
imperative	0.002293	0.006522	0.035780	0.035712

In [38]:

```
sns.pairplot(lle_test_lem)
```

Out[38]:

```
<seaborn.axisgrid.PairGrid at 0x22045387108>
```



In [39]:

```
lle_model.reconstruction_error_
```

Out[39]:

```
1.3026465750707678e-06
```

Discussions on the LLE Embeddings [LEMMATIZING]

- The largest range of embeddings is 0.6
- The plots of all the features are arbitrary shapes.
- The distribution of the feature 3 and feature 2 are skewed to the right.
- The training time for the LLE transformation was 429ms

Q.3 Cosine Similarity LLE

Getting Cosine Similarity from word2vec Embeddings

Getting Cosine similarity between all words in test set [STEMMING]

In [40]:

```
# set cosine similarity threshold for defining similar words for comparing the different em
cos_threshold = 0.99
```

In [41]:

```
cos_sim_w2v = cosine_similarity(test_embedding_stem.iloc[:,:].values, Y=None, dense_output=
cos_sim_w2v.shape
```

Out[41]:

```
(1291, 1291)
```

In [42]:

```
cos_sim_w2v = pd.DataFrame(cos_sim_w2v,
                           columns = list(test_embedding_stem.index),
                           index = list(test_embedding_stem.index)
                          )
cos_sim_w2v.head()
```

Out[42]:

	trenberth	view	clarifi	paper	imper	climat	chang	plan	
trenberth	1.000000	0.976370	0.936243	0.966142	0.989991	0.754860	0.754498	0.996683	0.92
view	0.976370	1.000000	0.932005	0.978668	0.983556	0.815498	0.808264	0.967623	0.95
clarifi	0.936243	0.932005	1.000000	0.927367	0.932369	0.665868	0.674442	0.935049	0.91
paper	0.966142	0.978668	0.927367	1.000000	0.983515	0.792883	0.760241	0.959463	0.92
imper	0.989991	0.983556	0.932369	0.983515	1.000000	0.791888	0.786750	0.985714	0.92

5 rows × 1291 columns

In [43]:

```
# create a dataframe of similar words if cosine similarity > cos_threshold
cos_similar_stem = (cos_sim_w2v > cos_threshold).astype(int)
```

Getting the most similar word from cosine similarity [STEMMING]

In [44]:

```
cos_most_similar_stem = get_most_similar_words(cos_sim_w2v, n_similar = 5)
```

Getting Cosine similarity between all words in test set [LEMMATIZING]

In [45]:

```
cos_sim_w2v_lem = cosine_similarity(test_embedding_lem.iloc[:,:].values, Y=None, dense_outp
cos_sim_w2v_lem.shape
```

Out[45]:

```
(1364, 1364)
```

In [46]:

```
cos_sim_w2v_lem = pd.DataFrame(cos_sim_w2v_lem,
                           columns = list(test_embedding_lem.index),
                           index = list(test_embedding_lem.index)
                          )
```

In [47]:

```
# create a dataframe of similar words if cosine similarity > cos_threshold
cos_similar_lem = (cos_sim_w2v_lem > cos_threshold).astype(int)
cos_similar_lem.head()
```

Out[47]:

	trenberth	view	clarify	paper	imperative	climate	change	plan	track	earth	...
trenberth	1	0	0	0	0	0	0	1	0	0	...
view	0	1	0	0	0	0	0	0	0	0	...
clarify	0	0	1	0	0	0	0	0	0	0	...
paper	0	0	0	1	0	1	0	0	0	0	...
imperative	0	0	0	0	1	0	0	0	0	0	...

5 rows × 1364 columns

This sparse matrix of word similarity (from cosine similarity) of words from the word2vec embedding will be used as true values (labels) for evaluating the performance of the dimensionality reduction methods.

Getting the most similar word from cosine similarity [LEMMATIZING]

In [48]:

```
cos_most_similar_lem = get_most_similar_words(cos_sim_w2v_lem, n_similar=5)
```

Getting Cosine Similarity from LLE Embeddings

Getting Cosine similarity between all words LLE [STEMMING]

In [49]:

```
cos_sim_lle = cosine_similarity(lle_test.iloc[:,:].values, Y=None, dense_output=False)
```

In [50]:

```
cos_sim_lle.shape
```

Out[50]:

```
(1291, 1291)
```

In [51]:

```
cos_sim_lle = pd.DataFrame(cos_sim_lle, columns = list(lle_test.index), index = list(lle_te
cos_sim_lle
```

Out[51]:

	trenberth	view	clarifi	paper	imper	climat	chang	plan
trenberth	1.000000	-0.596368	0.069128	-0.358326	0.494314	-0.375531	-0.451399	-0.450840
view	-0.596368	1.000000	-0.158518	-0.202432	-0.599278	0.848199	0.762053	0.920518
clarifi	0.069128	-0.158518	1.000000	-0.140230	0.829486	-0.078503	-0.133199	-0.094328
paper	-0.358326	-0.202432	-0.140230	1.000000	-0.293283	-0.656797	-0.640885	0.005607
imper	0.494314	-0.599278	0.829486	-0.293283	1.000000	-0.322713	-0.321113	-0.570388
...	...	...	...	...	...	...	...	...
classic	0.687799	-0.779071	-0.230587	-0.271021	0.349949	-0.417323	-0.303633	-0.864100
feast	0.031519	-0.348548	0.930112	-0.147900	0.883881	-0.157627	-0.123134	-0.385678
follow	-0.453447	-0.382497	-0.235193	0.768392	-0.196601	-0.562961	-0.394744	-0.415291
coupl	-0.114668	0.002564	-0.700174	-0.294082	-0.407712	0.278480	0.454535	-0.306679
recoveri	0.811237	-0.640827	-0.399833	-0.282253	0.162862	-0.361987	-0.318233	-0.643907

1291 rows × 1291 columns

Comparing most similar words in LLE to Word2Vec most similar words [STEMMING]

In [52]:

```
cos_most_sim_lle_stem = get_most_similar_words(cos_sim_lle, n_similar=5)
cos_most_sim_lle_stem.head()
```

Out[52]:

	most_similar_1	most_similar_2	most_similar_3	most_similar_4	most_similar_5
trenberth	habit	closur	incorrect	kilimanjaro	accompani
view	cooler	tropospher	reconstruct	open	five
clarifi	harbour	bere	stalagmit	unspot	crap
paper	energi	panel	extrem	event	theori
imper	toxin	super	ran	pure	lesser

In [53]:

```
# create a dataframe of similar words if cosine similarity > cos_threshold
cos_sim_lle_label = (cos_sim_lle > cos_threshold).astype(int)
cos_sim_lle_label.head()
```

Out[53]:

	trenberth	view	clarifi	paper	imper	climat	chang	plan	track	earth	...	tropo
trenberth	1	0	0	0	0	0	0	0	0	0	...	
view	0	1	0	0	0	0	0	0	0	0	...	
clarifi	0	0	1	0	0	0	0	0	0	0	...	
paper	0	0	0	1	0	0	0	0	0	0	...	
imper	0	0	0	0	1	0	0	0	0	0	...	

5 rows × 1291 columns

In [54]:

```
cos_similar_stem.head()
```

Out[54]:

	trenberth	view	clarifi	paper	imper	climat	chang	plan	track	earth	...	tropospł
trenberth	1	0	0	0	0	0	0	1	0	0	...	
view	0	1	0	0	0	0	0	0	0	0	...	
clarifi	0	0	1	0	0	0	0	0	0	0	...	
paper	0	0	0	1	0	0	0	0	0	0	...	
imper	0	0	0	0	1	0	0	0	0	0	...	

5 rows × 1291 columns

In [55]:

```
cos_sim_lle
```

Out[55]:

	trenberth	view	clarifi	paper	imper	climat	chang	plan
trenberth	1.000000	-0.596368	0.069128	-0.358326	0.494314	-0.375531	-0.451399	-0.450840
view	-0.596368	1.000000	-0.158518	-0.202432	-0.599278	0.848199	0.762053	0.920518
clarifi	0.069128	-0.158518	1.000000	-0.140230	0.829486	-0.078503	-0.133199	-0.094328
paper	-0.358326	-0.202432	-0.140230	1.000000	-0.293283	-0.656797	-0.640885	0.005607
imper	0.494314	-0.599278	0.829486	-0.293283	1.000000	-0.322713	-0.321113	-0.570388
...	...	...	...	...	...	...	...	...
classic	0.687799	-0.779071	-0.230587	-0.271021	0.349949	-0.417323	-0.303633	-0.864100
feast	0.031519	-0.348548	0.930112	-0.147900	0.883881	-0.157627	-0.123134	-0.385678
follow	-0.453447	-0.382497	-0.235193	0.768392	-0.196601	-0.562961	-0.394744	-0.415291
coupl	-0.114668	0.002564	-0.700174	-0.294082	-0.407712	0.278480	0.454535	-0.306679
recoveri	0.811237	-0.640827	-0.399833	-0.282253	0.162862	-0.361987	-0.318233	-0.643907

1291 rows × 1291 columns

In [56]:

```
precision_recall_fscore(cos_similar_stem, cos_sim_lle_label)
```

```
Precision:  0.59
Recall:  0.63
F1_score:  0.52
```

Getting Cosine similarity between all words in test set [LEMMATIZING]

In [57]:

```
cos_sim_lle_lem = cosine_similarity(lle_test_lem.iloc[:,:].values, Y=None, dense_output=Fal
cos_sim_lle_lem.shape
```

Out[57]:

```
(1364, 1364)
```

In [58]:

```
cos_sim_lle_lem = pd.DataFrame(cos_sim_lle_lem,
                           columns = list(lle_test_lem.index),
                           index = list(lle_test_lem.index))
cos_sim_lle_lem
```

Out[58]:

	trenberth	view	clarify	paper	imperative	climate	change	pla
trenberth	1.000000	-0.785851	-0.395427	-0.768403	-0.387185	-0.444245	-0.698488	-0.7728
view	-0.785851	1.000000	-0.208370	0.843071	-0.216855	0.334274	0.719982	0.9533
clarify	-0.395427	-0.208370	1.000000	-0.189932	0.999960	-0.123983	-0.185594	-0.2463
paper	-0.768403	0.843071	-0.189932	1.000000	-0.198141	0.781052	0.978301	0.9575
imperative	-0.387185	-0.216855	0.999960	-0.198141	1.000000	-0.128723	-0.193083	-0.2548
...	...	...	...	...	...	...	...	
feast	-0.197338	-0.399592	0.978464	-0.375189	0.980274	-0.230709	-0.354389	-0.4370
river	-0.790882	0.938989	-0.223003	0.975414	-0.231575	0.626181	0.909980	0.9966
follow	-0.795620	0.958260	-0.230238	0.961115	-0.238884	0.585631	0.887106	0.9959
couple	-0.311476	-0.292556	0.995971	-0.271425	0.996734	-0.171004	-0.259911	-0.3305
recovery	-0.332022	-0.272400	0.997683	-0.251906	0.998252	-0.159753	-0.242120	-0.3103

1364 rows × 1364 columns

Comparing most similar words in LLE to Word2Vec most similar words [LEMMATIZING]

In [59]:

```
cos_most_sim_lle_lem = get_most_similar_words(cos_sim_lle_lem, n_similar=5)
cos_most_sim_lle_lem.head()
```

Out[59]:

	most_similar_1	most_similar_2	most_similar_3	most_similar_4	most_similar_5
trenberth	jones	difference	reconstructions	simple	barack
view	sunlight	turn	greater	five	half
clarify	cloudcover	gov	purely	pronounce	entitle
paper	peer	forest	u	wind	us
imperative	isotopes	destabilize	outstrip	disappearance	combine

In [60]:

```
# create a dataframe of similar words if cosine similarity > cos_threshold
cos_sim_lle_lem_label = (cos_sim_lle_lem > cos_threshold).astype(int)
cos_sim_lle_lem_label.head()
```

Out[60]:

	trenberth	view	clarify	paper	imperative	climate	change	plan	track	earth	...
trenberth	1	0	0	0	0	0	0	0	0	0	...
view	0	1	0	0	0	0	0	0	0	0	...
clarify	0	0	1	0	1	0	0	0	0	0	...
paper	0	0	0	1	0	0	0	0	0	0	...
imperative	0	0	1	0	1	0	0	0	0	0	...

5 rows × 1364 columns

In [61]:

```
precision_recall_fscore(cos_similar_lem, cos_sim_lle_lem_label)
```

```
Precision:  0.56
Recall:  0.66
F1_score:  0.5
```

Comparing Evaluation Metrics for Cosine Similarity of LLE embeddings

	Precision	Recall	F1 Score
LLE Embeddings of Stemmed Words	0.56	0.64	0.51
LLE Embeddings of Lemmatized Words	0.59	0.63	0.49

In [62]:

```
words_list = [['man', 'bear'],['heat', 'warm'],['earth', 'global'], ['cold', 'warm'], ['sum
for word in words_list:
    print("The Cos similarity of stemmed LLE embeddings between", word[0], "and", word[1],
    print("The Cos similarity of lemmatized LLE embeddings between", word[0], "and", word[1
    similarity(words = word,
               stem_model = model_stem,
               lem_model = model_lem
              )
    print("\n")
```

```
The Cos similarity of stemmed LLE embeddings between man and bear is 0.26
The Cos similarity of lemmatized LLE embeddings between man and bear is 0.98
Stemmed W2V model similarity between man and bear = 0.92
Lemmatized W2V model similarity between man and bear = 0.96


The Cos similarity of stemmed LLE embeddings between heat and warm is 0.35
The Cos similarity of lemmatized LLE embeddings between heat and warm is 0.8
7
Stemmed W2V model similarity between heat and warm = 0.62
Lemmatized W2V model similarity between heat and warm = 0.73


The Cos similarity of stemmed LLE embeddings between earth and global is 0.5
3
The Cos similarity of lemmatized LLE embeddings between earth and global is
0.96
Stemmed W2V model similarity between earth and global = 0.93
Lemmatized W2V model similarity between earth and global = 0.93


The Cos similarity of stemmed LLE embeddings between cold and warm is 0.97
The Cos similarity of lemmatized LLE embeddings between cold and warm is 0.9
9
Stemmed W2V model similarity between cold and warm = 0.68
Lemmatized W2V model similarity between cold and warm = 0.68


The Cos similarity of stemmed LLE embeddings between summer and ocean is 0.6
3
The Cos similarity of lemmatized LLE embeddings between summer and ocean is
0.67
Stemmed W2V model similarity between summer and ocean = 0.73
Lemmatized W2V model similarity between summer and ocean = 0.75


The Cos similarity of stemmed LLE embeddings between summer and winter is 0.
99
The Cos similarity of lemmatized LLE embeddings between summer and winter is
1.0
Stemmed W2V model similarity between summer and winter = 1.0
Lemmatized W2V model similarity between summer and winter = 0.99
```

Analysis of Cosine similarity

1. **Man and Bear**

These words are not similar, an ideal similarity should be 0.5 or less. The LLE embeddings of stemmed words produced a similarity of 0.26, while the LLE embeddings of lemmatized words produced a similarity of 0.98. The stemmed Word2Vec model produces a similarity of 0.92 while the lemmatized Word2Vec model produces a similarity of 0.96.

2. **Heat and Warm**

These words are similar, an ideal similarity value should be about 0.7 or 0.8. The LLE embeddings of stemmed words produced a similarity of 0.35, while the LLE embeddings of lemmatized words produced a similarity of 0.87. However, the stemmed Word2Vec model produces a similarity of 0.62 while the lemmatized Word2Vec model produces a similarity of 0.73.

3. **Earth and Global**

These words have a similar context, an ideal similarity value should be about 0.8. The LLE embeddings of stemmed words produced a similarity of 0.53, while the LLE embeddings of lemmatized words produced a similarity of 0.96. However, the stemmed Word2Vec model produces a similarity of 0.93 while the lemmatized Word2Vec model produces a similarity of 0.93. All the similarities here are slightly higher than our expectation.

4. **Cold and Warm**

These words are not similar, an ideal similarity should be 0.5 or less. The LLE embeddings of stemmed words produced a similarity of 0.97, while the LLE embeddings of lemmatized words produced a similarity of 0.99. The stemmed Word2Vec model produces a similarity of 0.68 while the lemmatized Word2Vec model produces a similarity of 0.68.

5. **Summer and Ocean**

These words are not similar, an ideal similarity should be 0.6 or less. The LLE embeddings of stemmed words produced a similarity of 0.63, while the LLE embeddings of lemmatized words produced a similarity of 0.67. The stemmed Word2Vec model produces a similarity of 0.73 while the lemmatized Word2Vec model produces a similarity of 0.75.

6. **Summer and Winter**

These words are opposites, an ideal similarity should be less than 0.5. The LLE embeddings of stemmed words produced a similarity of 0.99, while the LLE embeddings of lemmatized words produced a similarity of 1.0. The stemmed Word2Vec model produces a similarity of 1.0 while the lemmatized Word2Vec model produces a similarity of 0.99. This should not be the case considering that these words are not similar.

Summmary of Analysis

Words	Stemmed LLE	Lemmatized LLE	Stemmed Word2Vec	Lemmatized Word2Vec
Man, Bear	**0.26**	0.98	0.92	0.96
Heat, Warm	**0.35**	0.87	0.62	0.73
Earth, Global	0.53	**0.96**	0.93	0.93
Cold, Warm	0.97	0.99	**0.68**	**0.68**
Summer, Ocean	**0.63**	0.67	0.73	0.75
Summer, Winter	0.99	1.0	1.0	0.99

Best performing model in bold

KNN GRAPH (Word2Vec)

Using KNN on word embedding to get most similar word [STEMMING]

In [63]:

```
knn_similar_stem = kneighbors_graph(test_embedding_stem.iloc[:,:].values, 6, mode='connecti
```

In [64]:

```
knn_similar_stem = pd.DataFrame(knn_similar_stem.toarray(),
                    columns = list(test_embedding_stem.index),
                    index = list(test_embedding_stem.index)
                   )
knn_similar_stem.head()
```

Out[64]:

	trenberth	view	clarifi	paper	imper	climat	chang	plan	track	earth	...	troposph
trenberth	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	(
view	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	(
clarifi	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	(
paper	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	(
imper	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	(

5 rows × 1291 columns

In [65]:

```
knn_most_similar_stem = get_most_similar_words(knn_similar_stem, n_similar=5)
knn_most_similar_stem.head()
```

Out[65]:

	most_similar_1	most_similar_2	most_similar_3	most_similar_4	most_similar_5
trenberth	wherebi	refin	cheap	earthquak	fix
view	strong	statist	question	independ	repres
clarifi	blew	bere	mack	sussex	accret
paper	issu	univers	work	public	accord
imper	cook	steig	super	fashion	overpeck

Using KNN on word embedding to get most similar word [LEMMATIZING]

In [66]:

```
knn_similar_lem = kneighbors_graph(test_embedding_lem.iloc[:,:].values, 6, mode='connectivi
```

In [67]:

```
knn_similar_lem = pd.DataFrame(knn_similar_lem.toarray(),
                    columns = list(test_embedding_lem.index),
                    index = list(test_embedding_lem.index)
                   )
knn_similar_lem.head()
```

Out[67]:

	trenberth	view	clarify	paper	imperative	climate	change	plan	track	earth	...
trenberth	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
view	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
clarify	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
paper	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...
imperative	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...

5 rows × 1364 columns

In [68]:

```
knn_most_similar_lem = get_most_similar_words(knn_similar_lem, n_similar=5)
knn_most_similar_lem.head()
```

Out[68]:

	most_similar_1	most_similar_2	most_similar_3	most_similar_4	most_similar_5
trenberth	filter	jones	important	simple	debate
view	link	cosmic	strong	fund	man
clarify	apparently	occurence	deluge	relation	grandchildren
paper	u	peer	first	public	accord
imperative	outstrip	utility	climatologists	indicative	deluge

The KNN Neighbors of words from the word2vec embedding will be used as true labels for comparing dimensionality reduction methods

KNN GRAPH (LLE)

Using KNN on word embedding to get most similar word [STEMMING]

In [69]:

```
knn_similar_stem_lle = kneighbors_graph(lle_test.iloc[:,:].values, 6, mode='connectivity',
```

In [70]:

```
knn_similar_stem_lle = pd.DataFrame(knn_similar_stem_lle.toarray(),
                    columns = list(lle_test.index),
                    index = list(lle_test.index)
                  )
knn_similar_stem_lle.head()
```

Out[70]:

	trenberth	view	clarifi	paper	imper	climat	chang	plan	track	earth	...	tropospl
trenberth	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	(
view	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	'
clarifi	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	(
paper	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	(
imper	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	(

5 rows × 1291 columns

Comparing most similar words in LLE to Word2Vec most similar words [STEMMING]

In [71]:

```
knn_most_similar_stem_lle = get_most_similar_words(knn_similar_stem_lle, n_similar=5)
knn_most_similar_stem_lle.head()
```

Out[71]:

	most_similar_1	most_similar_2	most_similar_3	most_similar_4	most_similar_5
trenberth	kilimanjaro	habit	incorrect	closur	bigger
view	appear	five	cooler	reconstruct	tropospher
clarifi	unspot	bere	crap	stalagmit	harbour
paper	event	model	sun	wave	theori
imper	ran	super	pure	toxin	lesser

In [72]:

```
precision_recall_fscore(knn_similar_stem, knn_similar_stem_lle)
```

```
Precision:  0.63
Recall:  0.65
F1_score:  0.62
```

Using KNN on word embedding to get most similar word [LEMMATIZING]

In [73]:

```
knn_similar_lem_lle = kneighbors_graph(lle_test_lem.iloc[:,:].values, 6, mode='connectivity
```

In [74]:

```
knn_similar_lem_lle = pd.DataFrame(knn_similar_lem_lle.toarray(),
                    columns = list(lle_test_lem.index),
                    index = list(lle_test_lem.index)
                   )
knn_similar_lem_lle.head()
```

Out[74]:

	trenberth	view	clarify	paper	imperative	climate	change	plan	track	earth	...
trenberth	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
view	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
clarify	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
paper	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...
imperative	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...

5 rows × 1364 columns

Comparing most similar words in LLE to Word2Vec most similar words [LEMMATIZING]

In [75]:

```
knn_most_similar_lem_lle = get_most_similar_words(knn_similar_lem_lle, n_similar=5)
knn_most_similar_lem_lle.head()
```

Out[75]:

	most_similar_1	most_similar_2	most_similar_3	most_similar_4	most_similar_5
trenberth	barack	jones	difference	simple	reconstructions
view	half	turn	orbit	sunlight	greater
clarify	gov	cloudcover	purely	pronounce	entitle
paper	u	wind	peer	forest	us
imperative	outstrip	combine	isotopes	destabilize	disappearance

In [76]:

```
precision_recall_fscore(knn_similar_lem, knn_similar_lem_lle)
```

```
Precision:  0.69
Recall:  0.71
F1_score:  0.68
```

Comparing Evaluation Metrics for KNN Graph of LLE embeddings

LLE KNN Graph Evaluation

	Precision	Recall	F1 Score
LLE Embeddings of Stemmed Words	0.63	0.65	0.62
LLE Embeddings of Lemmatized Words	0.69	0.71	0.68

The lemmatized LLE model performs better that the stemmed LLE model.

Comparing with PCA KNN Graph evaluation from CM2

PCA KNN Graph Evaluation from CM2

	Precision	Recall	F1 Score
PCA Embeddings of Stemmed Words	0.94	0.95	0.94
PCA Embeddings of Lemmatized Words	0.99	0.7	0.76

From the tables, PCA performed better than LLE in both stemmed and lemmatized corpi for the KNN Graph.

In [ ]: