In [1]:

```
# 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]:

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

In [3]:

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

Defining relevant functions

In [4]:

```
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]:

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

In [6]:

```
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]:

```
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]:

```python
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	-0.209349	0.211203	0.160982	-0.148733	0.181342	0.204856	0.113328	-0.094728	0.04
last	0.017848	0.331719	0.484374	-1.070892	0.500586	1.260484	0.120261	-1.046856	-0.43
switch	-0.161795	0.134954	0.090551	-0.060087	0.119985	0.098721	0.080246	0.023134	0.11
cool	-0.555943	0.407788	0.518327	-0.701247	0.826338	0.946836	0.472072	-0.533027	0.03
phase	-0.527429	0.497619	0.403759	-0.359568	0.391707	0.444784	0.257453	-0.208391	0.19

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	-0.196773	0.105953	0.052032	-0.057684	-0.055261	-0.089418	-0.070971	0.011750	-0.
last	-0.181438	0.002779	0.116914	-0.653354	-0.466708	0.176549	-0.491657	-0.701406	-1.
switch	-0.138131	0.067366	0.030634	-0.012261	-0.025369	-0.071405	-0.036417	0.059377	-0.
cool	-0.730841	0.236594	0.275793	-0.436811	0.014740	-0.062164	-0.093488	-0.084746	-0
phase	-0.518672	0.215381	0.151445	-0.173897	-0.205804	-0.258466	-0.202111	-0.017236	-0.

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	-0.099120	0.086698	0.063106	-0.041859	0.066776	0.062011	0.042706	-0.013826	0.
view	-0.421241	0.481964	0.310263	-0.238352	0.351689	0.373587	0.219227	-0.116195	0.
clarifi	-0.016760	0.014419	0.008773	-0.011625	0.012340	0.007927	0.006872	0.000053	0.
paper	-0.659361	0.900829	0.454997	-0.187653	0.406615	0.345930	0.277394	-0.107806	0.
imper	-0.067434	0.079196	0.047366	-0.028237	0.044952	0.046282	0.035077	-0.014935	0.

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	-0.110677	0.054654	0.027002	-0.013453	-0.020220	-0.052755	-0.030843	0.032404
view	-0.395669	0.254809	0.092313	-0.076944	-0.150078	-0.223747	-0.180401	0.060727
clarify	-0.006618	0.003608	0.003014	0.003484	-0.009510	-0.005484	-0.002837	-0.002074
paper	-0.640657	0.521411	0.113881	0.066554	-0.304611	-0.535869	-0.345126	0.171257
imperative	-0.014345	0.002710	0.003984	-0.003274	-0.005046	-0.013955	-0.011195	0.000035

5 rows × 100 columns

TSNE

t-SNE is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is not convex, i.e. with different initializations we can get different results.

Q.1

Using Stemming

In [27]:

```
from sklearn.manifold import TSNE
```

Choice of Dimensionality

tSNE is a dimensionality reduction method primarily used for visualization. It is difficult to visualize data beyond three dimensions, thus a reduction to 2 or 3 dimensions is most suitable for tSNE.

In [28]:

```
%%time
tsne_model = TSNE(n_components=2, init='pca', random_state=0)
tsne_vectors = tsne_model.fit_transform(test_embedding_stem.iloc[:,:].values)
```

```
Wall time: 10.4 s
```

In [29]:

```
df_tsne = pd.DataFrame(tsne_vectors, columns = ["feature1", "feature2"])
df_tsne.index = test_embedding_stem.index
```

In [30]:

```
df_tsne.head()
```

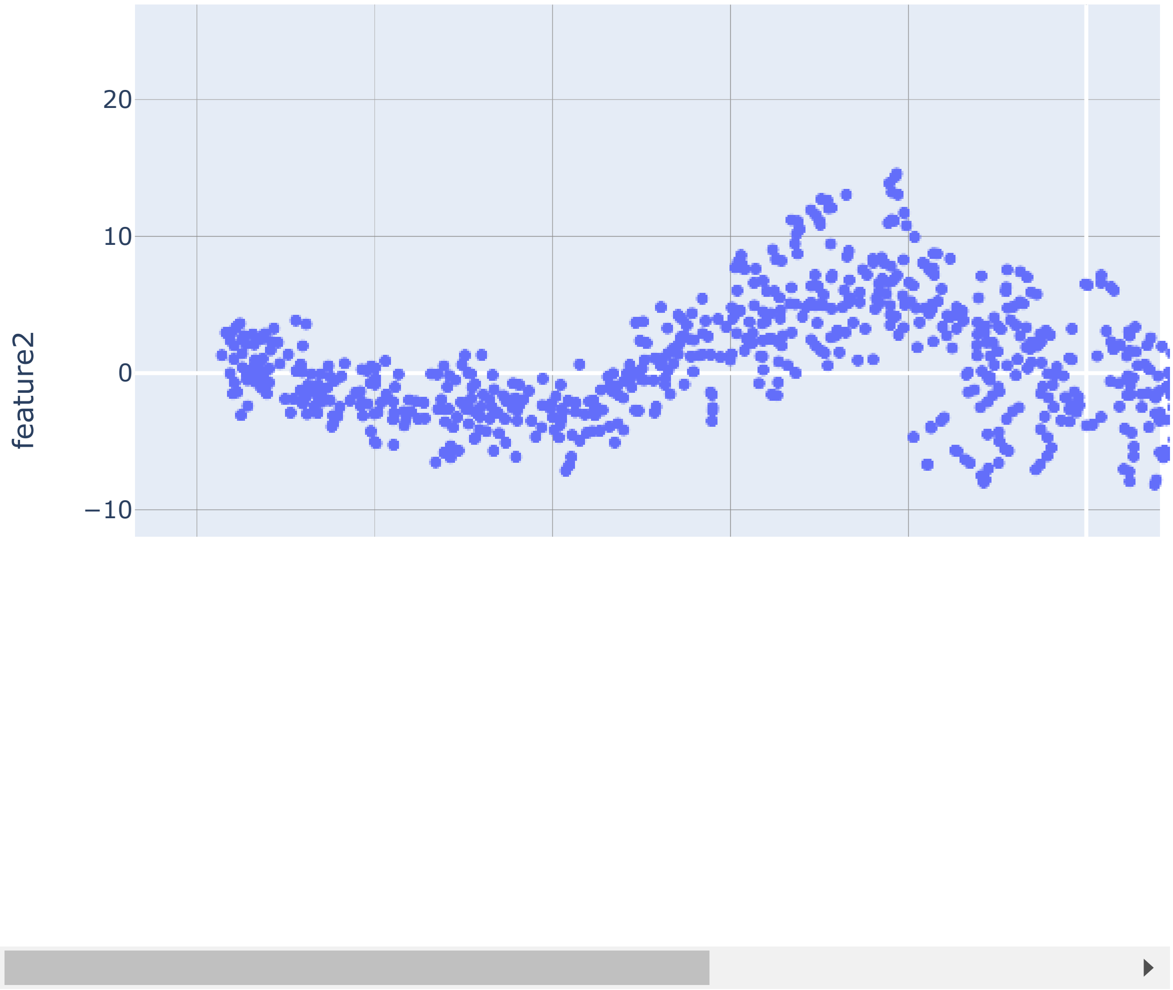
Out[30]:

	feature1	feature2
Word		
trenberth	-49.821774	0.291613
view	27.230789	2.090050
clarifi	-88.805252	0.111629
paper	57.674660	4.399313
imper	-56.297764	-3.059801

Q.2

In [31]:

```
fig = px.scatter(df_tsne, x="feature1", y="feature2",
                 hover_name=list(df_tsne.index))
fig.show("notebook")
```



Discussions on tSNE embedding [STEMMING]

- The range of embeddings are much higher than that of PCA and LLE, with the highest range being 170
- The scatter plot of features produces an S-shaped curve
- The plot also shows some relationship between the wrods as some similar words are close to each other in the plot
- The tSNE training time was 10.4s

In [32]:

```
tsne_model.kl_divergence_
```

Out[32]:

```
0.6374173164367676
```

In [33]:

```
tsne_model.n_iter_
```

Out[33]:

999

Using Lemmatization

In [34]:

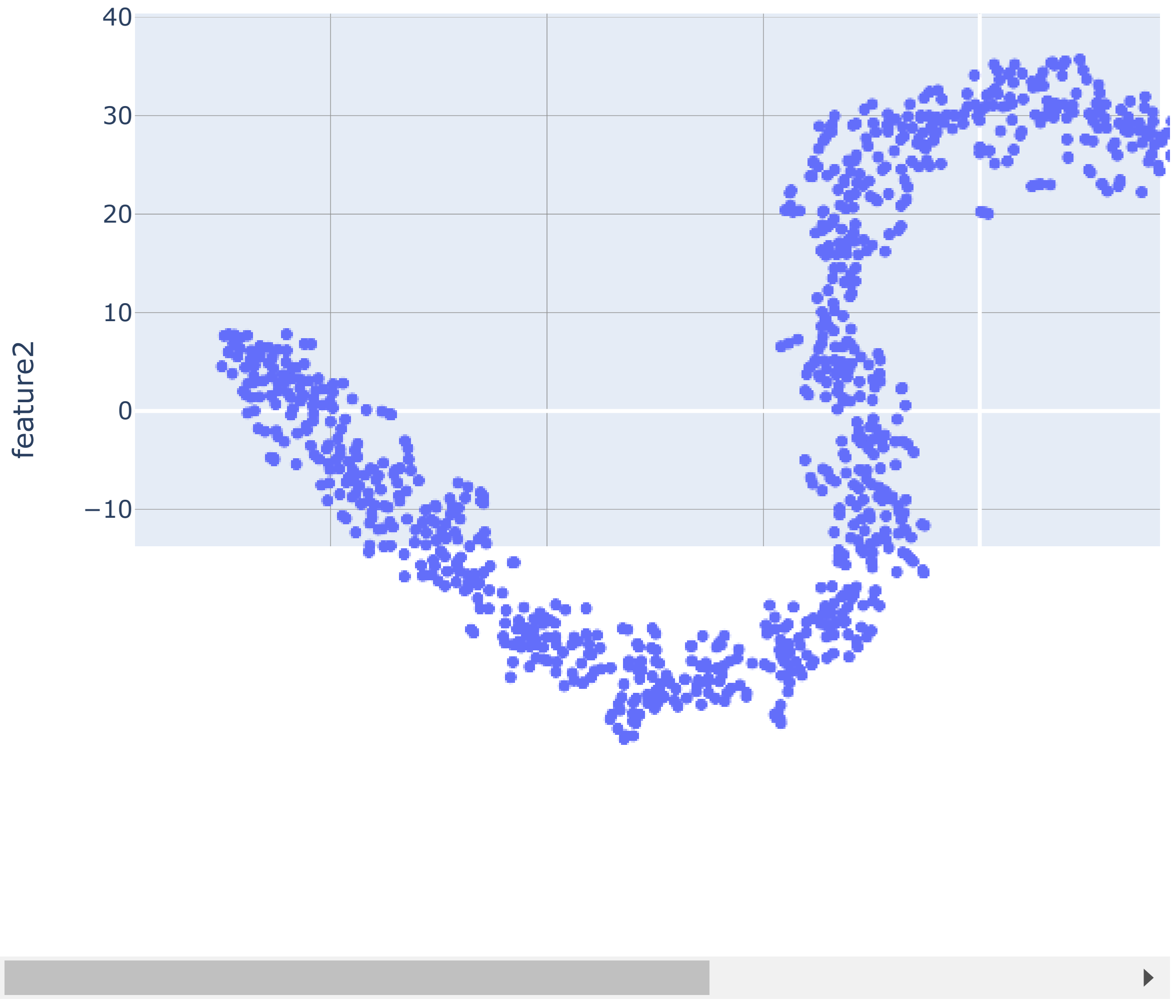
```
tsne_model_lem = TSNE(n_components=2, init='pca', random_state=0)
tsne_vectors_lem = tsne_model_lem.fit_transform(test_embedding_lem.iloc[:,:].values)
```

In [35]:

```
df_tsne_lem = pd.DataFrame(tsne_vectors_lem, columns = ["feature1", "feature2"])
df_tsne_lem.index = test_embedding_lem.index
```

In [36]:

```
fig = px.scatter(df_tsne_lem, x="feature1", y="feature2",
                 hover_name=list(df_tsne_lem.index))
fig.show("notebook")
```



In [37]:

```
tsne_model.kl_divergence_
```

Out[37]:

```
0.6374173164367676
```

In [38]:

```
tsne_model.n_iter_
```

Out[38]:

```
999
```

Discussions on tSNE embedding [LEMMATIZING]

- The range of embeddings are much higher than that of PCA and LLE, with the highest range being 170
- The scatter plot of features produces an S-shaped curve
- The plot also shows some relationship between the wrods as some similar words are close to each other in the plot

Q.3 Cosine Similarity TSNE

Getting Cosine Similarity from word2vec Embeddings

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

In [39]:

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

In [40]:

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

Out[40]:

```
(1291, 1291)
```

In [41]:

```
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[41]:

	trenberth	view	clarifi	paper	imper	climat	chang	plan	
trenberth	1.000000	0.979533	0.935634	0.965298	0.981510	0.730940	0.719631	0.992687	0.97
view	0.979533	1.000000	0.932067	0.966692	0.989023	0.816604	0.804787	0.968409	0.98
clarifi	0.935634	0.932067	1.000000	0.919884	0.935626	0.756286	0.749674	0.938733	0.90
paper	0.965298	0.966692	0.919884	1.000000	0.980106	0.811279	0.758742	0.946008	0.96
imper	0.981510	0.989023	0.935626	0.980106	1.000000	0.825483	0.807354	0.973662	0.97

5 rows × 1291 columns

In [42]:

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

Out[42]:

	trenberth	view	clarifi	paper	imper	climat	chang	plan	track	earth	...	tropospl
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

Getting the most similar word from cosine similarity [STEMMING]

In [43]:

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

Out[43]:

	most_similar_1	most_similar_2	most_similar_3	most_similar_4	most_similar_5
trenberth	need	obama	epa	list	fund
view	mani	un	reproduc	conclus	agreement
clarifi	sustain	threaten	econom	polit	irrevers
paper	research	articl	journal	publish	scienc
imper	testabl	physic	risk	disput	danger

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

In [44]:

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

Out[44]:

```
(1364, 1364)
```

In [45]:

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

Out[45]:

	trenberth	view	clarify	paper	imperative	climate	change	plan
trenberth	1.000000	0.979864	0.895123	0.975883	0.923429	0.984872	0.990285	0.996463
view	0.979864	1.000000	0.895560	0.975544	0.904608	0.980859	0.987679	0.971971
clarify	0.895123	0.895560	1.000000	0.904734	0.851118	0.913513	0.910849	0.883413
paper	0.975883	0.975544	0.904734	1.000000	0.921663	0.992056	0.985413	0.963723
imperative	0.923429	0.904608	0.851118	0.921663	1.000000	0.933157	0.936126	0.920909

5 rows × 1364 columns

In [46]:

```
# 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[46]:

	trenberth	view	clarify	paper	imperative	climate	change	plan	track	earth	...
trenberth	1	0	0	0	0	0	1	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 [47]:

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

Out[47]:

	most_similar_1	most_similar_2	most_similar_3	most_similar_4	most_similar_5
trenberth	phil	idea	support	amazon	action
view	note	contradict	suggest	agreement	drastic
clarify	climate	february	drought	cite	barrier
paper	research	journal	publish	latest	computer
imperative	newspaper	new	seal	vast	barrier

Getting Cosine Similarity from TSNE Embeddings

Getting Cosine similarity between all words [STEMMING]

In [48]:

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

In [49]:

```
cos_sim_tsne.shape
```

Out[49]:

```
(1291, 1291)
```

In [50]:

```
cos_sim_tsne = pd.DataFrame(cos_sim_tsne, columns = list(df_tsne.index), index = list(df_ts
cos_sim_tsne
```

Out[50]:

	trenberth	view	clarifi	paper	imper	climat	chang	plan
trenberth	1.000000	-0.996602	0.999989	-0.996641	0.998192	-0.986138	-0.985337	-0.982630
view	-0.996602	1.000000	-0.996970	1.000000	-0.999751	0.996454	0.996042	0.994576
clarifi	0.999989	-0.996970	1.000000	-0.997007	0.998457	-0.986890	-0.986111	-0.983473
paper	-0.996641	1.000000	-0.997007	1.000000	-0.999762	0.996414	0.996000	0.994527
imper	0.998192	-0.999751	0.998457	-0.999762	1.000000	-0.994329	-0.993812	-0.992009
...	...	...	...	...	...	...	...	...
classic	0.999999	-0.996706	0.999994	-0.996744	0.998267	-0.986346	-0.985552	-0.982863
feast	0.999655	-0.998421	0.999765	-0.998448	0.999426	-0.990154	-0.989477	-0.987163
follow	-0.941183	0.910155	-0.939620	0.910351	-0.919169	0.872069	0.869731	0.862129
coupl	-0.954708	0.926957	-0.953330	0.927135	-0.935095	0.892102	0.889943	0.882908
recoveri	0.964377	-0.939313	0.963151	-0.939475	0.946731	-0.907116	-0.905103	-0.898535

1291 rows × 1291 columns

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

In [51]:

```
cos_most_sim_tsne = get_most_similar_words(cos_sim_tsne, n_similar=5)
cos_most_sim_tsne.head()
```

Out[51]:

	most_similar_1	most_similar_2	most_similar_3	most_similar_4	most_similar_5
trenberth	anyway	unlik	debunk	pingo	twenti
view	research	abl	paper	repres	get
clarifi	nsw	earthquak	jonathan	travel	latest
paper	abl	repres	view	research	get
imper	rp	anyon	multitud	unclear	blatantli

In [52]:

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

Out[52]:

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

5 rows × 1291 columns

In [53]:

```
precision_recall_fscore(cos_similar_stem, cos_sim_tsne_label)
```

```
Precision:  0.55
Recall:  0.67
F1_score:  0.51
```

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

In [54]:

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

Out[54]:

```
(1364, 1364)
```

In [55]:

```
cos_sim_tsne_lem = pd.DataFrame(cos_sim_tsne_lem,
                               columns = list(df_tsne_lem.index),
                               index = list(df_tsne_lem.index))
cos_sim_tsne_lem
```

Out[55]:

	trenberth	view	clarify	paper	imperative	climate	change	pl
trenberth	1.000000	-0.915500	0.984407	-0.999807	0.977322	-0.997396	-0.997745	-0.9454
view	-0.915500	1.000000	-0.971995	0.923223	-0.979932	0.942133	0.940441	0.9966
clarify	0.984407	-0.971995	1.000000	-0.987671	0.999332	-0.994530	-0.993994	-0.9879
paper	-0.999807	0.923223	-0.987671	1.000000	-0.981292	0.998619	0.998870	0.9516
imperative	0.977322	-0.979932	0.999332	-0.981292	1.000000	-0.990050	-0.989332	-0.9929
...	...	...	...	...	...	...	...	
feast	0.886066	-0.997692	0.953793	-0.894997	0.964134	-0.917192	-0.915184	-0.9887
river	-0.989939	0.963215	-0.999393	0.992526	-0.997452	0.997566	0.997204	0.9820
follow	-0.998733	0.894092	-0.974306	0.997552	-0.965426	0.992502	0.993102	0.9278
couple	0.925505	-0.999672	0.977696	-0.932763	0.984717	-0.950411	-0.948840	-0.9984
recovery	0.941585	-0.997512	0.986144	-0.948016	0.991547	-0.963423	-0.962067	-0.9999

1364 rows × 1364 columns

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

In [56]:

```
cos_most_sim_tsne_lem = get_most_similar_words(cos_sim_tsne_lem, n_similar=5)
cos_most_sim_tsne_lem.head()
```

Out[56]:

	most_similar_1	most_similar_2	most_similar_3	most_similar_4	most_similar_5
trenberth	filter	debate	troposphere	argue	bias
view	sunlight	main	meet	understand	frequent
clarify	nimbus	pingoes	productivity	cherry	anticipate
paper	public	ipcc	report	peer	model
imperative	gov	australasian	slowdown	feb	insignificant

In [57]:

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

Out[57]:

	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	1	0	0	...
clarify	0	0	1	0	1	0	0	0	0	0	...
paper	0	0	0	1	0	1	1	0	0	1	...
imperative	0	0	1	0	1	0	0	0	0	0	...

5 rows × 1364 columns

In [58]:

```
precision_recall_fscore(cos_similar_lem, cos_sim_tsne_lem_label)
```

```
Precision:  0.54
Recall:  0.64
F1_score:  0.48
```

Comparing Evaluation Metrics for Cosine Similarity of tSNE embeddings

	Precision	Recall	F1 Score
tSNE Embeddings of Stemmed Words	0.55	0.67	0.51
tSNE Embeddings of Lemmatized Words	0.54	0.64	0.48

The model trained on Stemmed words performs better.

In [59]:

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

```
The Cos similarity of stemmed tSNE embeddings between man and bear is 0.89
The Cos similarity of lemmatized tSNE embeddings between man and bear is 0.9
Stemmed W2V model similarity between man and bear = 0.93
Lemmatized W2V model similarity between man and bear = 0.97


The Cos similarity of stemmed tSNE embeddings between heat and warm is 0.84
The Cos similarity of lemmatized tSNE embeddings between heat and warm is 0.
94
Stemmed W2V model similarity between heat and warm = 0.6
Lemmatized W2V model similarity between heat and warm = 0.68


The Cos similarity of stemmed tSNE embeddings between earth and global is 1.
0
The Cos similarity of lemmatized tSNE embeddings between earth and global is
1.0
Stemmed W2V model similarity between earth and global = 0.92
Lemmatized W2V model similarity between earth and global = 0.93


The Cos similarity of stemmed tSNE embeddings between cold and warm is 0.99
The Cos similarity of lemmatized tSNE embeddings between cold and warm is 1.
0
Stemmed W2V model similarity between cold and warm = 0.67
Lemmatized W2V model similarity between cold and warm = 0.7


The Cos similarity of stemmed tSNE embeddings between summer and ocean is 0.
98
The Cos similarity of lemmatized tSNE embeddings between summer and ocean is
1.0
Stemmed W2V model similarity between summer and ocean = 0.72
Lemmatized W2V model similarity between summer and ocean = 0.78


The Cos similarity of stemmed tSNE embeddings between summer and winter is
1.0
The Cos similarity of lemmatized tSNE embeddings between summer and winter i
s 1.0
Stemmed W2V model similarity between summer and winter = 0.99
Lemmatized W2V model similarity between summer and winter = 1.0
```

Analysis of Cosine similarity

1. **Man and Bear**

These words are not similar, an ideal similarity should be 0.5 or less. The tSNE embeddings of stemmed words produced a similarity of 0.89, while the tSNE embeddings of lemmatized words produced a similarity of 0.9. The stemmed Word2Vec model produces a similarity of 0.93 while the lemmatized Word2Vec model produces a similarity of 0.97. All the models did poorly here.

2. **Heat and Warm**

These words are similar, an ideal similarity value should be about 0.7 or 0.8. The tSNE embeddings of stemmed words produced a similarity of 0.84, while the tSNE embeddings of lemmatized words produced a similarity of 0.94. However, the stemmed Word2Vec model produces a similarity of 0.6 while the lemmatized Word2Vec model produces a similarity of 0.68.

3. **Earth and Global**

These words have a similar context, an ideal similarity value should be about 0.8. The tSNE embeddings of stemmed words produced a similarity of 1.0, while the tSNE embeddings of lemmatized words produced a similarity of 1.0. However, the stemmed Word2Vec model produces a similarity of 0.92 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 tSNE embeddings of stemmed words produced a similarity of 0.99, while the tSNE embeddings of lemmatized words produced a similarity of 1.0. The stemmed Word2Vec model produces a similarity of 0.67 while the lemmatized Word2Vec model produces a similarity of 0.7.

5. **Summer and Ocean**

These words are not similar, an ideal similarity should be 0.6 or less. The tSNE embeddings of stemmed words produced a similarity of 0.98, while the tSNE embeddings of lemmatized words produced a similarity of 1.0. The stemmed Word2Vec model produces a similarity of 0.72 while the lemmatized Word2Vec model produces a similarity of 0.78.

6. **Summer and Winter**

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

Summmary of Analysis

Words	Stemmed tSNE	Lemmatized tSNE	Stemmed Word2Vec	Lemmatized Word2Vec
Man, Bear	0.89	0.9	0.93	0.97
Heat, Warm	**0.84**	0.94	0.6	0.68
Earth, Global	1.0	1.0	**0.92**	**0.93**
Cold, Warm	0.99	1.0	**0.67**	0.7
Summer, Ocean	0.98	1.0	**0.72**	0.78

Words	Stemmed tSNE	Lemmatized tSNE	Stemmed Word2Vec	Lemmatized Word2Vec
Summer, Winter	1.0	1.0	0.99	1.0

Best performing model in bold

KNN GRAPH (Word2Vec)

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

In [60]:

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

In [61]:

```
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[61]:

	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

In [62]:

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

Out[62]:

	most_similar_1	most_similar_2	most_similar_3	most_similar_4	most_similar_5
trenberth	ben	nois	rooftop	mind	closur
view	statist	question	independ	repres	method
clarifi	remnant	instruct	computer	supernova	predetermin
paper	issu	univers	peer	public	accord
imper	ran	whose	profession	massachusett	watson

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

In [63]:

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

In [64]:

```
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[64]:

	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 [65]:

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

Out[65]:

	most_similar_1	most_similar_2	most_similar_3	most_similar_4	most_similar_5
trenberth	company	barack	filter	richard	debate
view	strong	know	throughout	longer	likely
clarify	simulations	combine	conducive	elliptical	pingoes
paper	scientist	peer	first	public	accord
imperative	australasian	reflection	gov	citation	indicative

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

KNN GRAPH

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

In [66]:

```
knn_similar_stem_tsne = kneighbors_graph(df_tsne.iloc[:,:].values, 6, mode='connectivity',
```

In [67]:

```
knn_similar_stem_tsne = pd.DataFrame(knn_similar_stem_tsne.toarray(),
                    columns = list(df_tsne.index),
                    index = list(df_tsne.index)
                   )
knn_similar_stem_tsne.head()
```

Out[67]:

	trenberth	view	clarifi	paper	imper	climat	chang	plan	track	earth	...	tropospł
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 TSNE to Word2Vec most similar words [STEMMING]

In [68]:

```
knn_most_similar_stem_tsne = get_most_similar_words(knn_similar_stem_tsne, n_similar=5)
knn_most_similar_stem_tsne.head()
```

Out[68]:

	most_similar_1	most_similar_2	most_similar_3	most_similar_4	most_similar_5
trenberth	ben	beard	sulphur	earthquak	fix
view	alreadi	job	feder	repres	anoth
clarifi	computer	nowher	gordon	supernova	readout
paper	studi	issu	peer	public	accord
imper	steig	whose	profession	massachusett	watson

In [69]:

```
precision_recall_fscore(knn_similar_stem, knn_similar_stem_tsne)
```

```
Precision:  0.83
Recall:  0.85
F1_score:  0.82
```

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

In [70]:

```
knn_similar_lem_tsne = kneighbors_graph(df_tsne_lem.iloc[:,:].values, 6, mode='connectivity
```

In [71]:

```
knn_similar_lem_tsne = pd.DataFrame(knn_similar_lem_tsne.toarray(),
                    columns = list(df_tsne_lem.index),
                    index = list(df_tsne_lem.index)
                   )
knn_similar_lem_tsne.head()
```

Out[71]:

	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 TSNE to Word2Vec most similar words [LEMMATIZING]

In [72]:

```
knn_most_similar_lem_tsne = get_most_similar_words(knn_similar_lem_tsne, n_similar=5)
knn_most_similar_lem_tsne.head()
```

Out[72]:

	most_similar_1	most_similar_2	most_similar_3	most_similar_4	most_similar_5
trenberth	company	filter	problems	richard	debate
view	fund	throughout	stream	longer	question
clarify	subsidize	prohibit	elliptical	nimbus	occurence
paper	scientist	publish	research	peer	public
imperative	persistent	australasian	thicker	hundred	disappearance

In [73]:

```
precision_recall_fscore(knn_similar_lem, knn_similar_lem_tsne)
```

```
Precision:  0.82
Recall:  0.84
F1_score:  0.81
```

Comparing Evaluation Metrics for KNN Graph of tSNE embeddings

tSNE KNN Graph Evaluation

	Precision	Recall	F1 Score
tSNE Embeddings of Stemmed Words	0.83	0.85	0.82
tSNE Embeddings of Lemmatized Words	0.82	0.85	0.81

The performance of both is almost the same. The model trained with Stemmed Words performs slightly better.

Comparing with PCA and LLE KNN Graph evaluation from CM2 and CM3

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

LLE KNN Graph Evaluation from CM3

	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

- From the tables, PCA performed better than both LLE and tSME in both stemmed and lemmatized corpi for the KNN Graph.
- Of the three reduction models, LLE had the least performance in terms of KNN graph

In [ ]:

```

```