In [1]:

```
import nltk

import numpy as np
import pandas as pd
import os,re
from wordcloud import STOPWORDS
from bs4 import BeautifulSoup
from nltk.tokenize import word_tokenize
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\91958\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[1]:

```
True
```

In [2]:

```
glove_dir="glove.twitter.27B"
glove_embedding="glove.twitter.27B.25d.txt"
```

In [3]:

```
embedding_index = {}

f = open(os.path.join(os.getcwd(),os.path.join(glove_dir,glove_embedding)),encoding="utf8")
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:],dtype='float32')
    embedding_index[word] = coefs
f.close()

print('found word vecs: ',len(embedding_index))
```

```
found word vecs:  1193514
```

In [4]:

```
data=pd.read_excel('#KingKohli_keyword_5000_tweets_until3rdNov2022.xlsx')
```

In [5]:

```
data=data[data.language=='en']
```

data cleaning

In [6]:

```
# Remove mentions
regex_mentions = r"@[A-Za-z0-9_]+"
data.Text = data.Text.apply(lambda x: re.sub(regex_mentions, " ", str(x).strip()))
```

In [7]:

```
# Remove hashtags
regex_hashtags=r"(#+[a-zA-Z0-9(_)]{1,})"
data.Text = data.Text.apply(lambda x: re.sub(regex_hashtags, " ", str(x).strip()))
```

In [8]:

```
data.shape
```

Out[8]:

```
(3455, 11)
```

In [9]:

```
data.head()
```

Out[9]:

	Datetime	Tweet Id	Text	Username	replyCount	retweetCount	likeCount	quo
0	2022-11-02	1.587950e+18	I have seen Kohli for so many years now. I hav...	slogoverindia	0	1	1	
1	2022-11-02	1.587938e+18	Despite all the hue and cry you can't deny the...	Mian_ilysm	0	0	0	
2	2022-11-02	1.587928e+18	Punjab Kings announce Shikhar Dhawan as their ...	slogoverindia	0	0	0	
3	2022-11-02	1.587920e+18	We are deeply saddened by the news that our CT...	slogoverindia	0	0	0	
5	2022-11-02	1.587911e+18	I think this was also a cheating or maybe umpi...	Vaishal97704321	1	1	6	

In [10]:

```
def remove_urls(text):
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    return url_pattern.sub(r'', text.lower())
```

In [11]:

```
def unicode_remover(text):
    unicode_pattern = re.compile('/u\S{4-6}')
    return unicode_pattern.sub(r'', text.lower())
```

In [12]:

```
puncts = [',', '.', '"', ':', ')', '(', '-', '!', '?', '|', ';', "'", '$', '&', '/', '[', '
 '·', '_', '{', '}', '©', '^', '®', '`',  '<', '→', '°', '€', '™', '›',  '♥', '←', '×', '§'
 '″', '★', '”', '–', '●', 'â', '►', '−', '¢', '²', '¬', '░', '¶', '↑', '±', '¿', '▾', '=',
 '▒', ': ', '¼', '⊕', '▼', '▪', '†', '■', '’', '■', '¨', '▄', '♫', '☆', 'é', '¯', '♦', '¤'
 '·', ') ', '↓', '、', '|', ' (', '»', '，', '♪', '╩', '╚', '³', '・', '╦', '╣', '╔', '╗', '

def punctuation_replacer(x):
    x = str(x)
    for punct in puncts:
        if punct in x:
            x = x.replace(punct, f'{punct}')
    return x

def punctuation_cleaner(x):
    pattern = re.compile(r'[^a-zA-z0-9\s]')
    text = re.sub(pattern, '', x)
    return x
```

In [13]:

```
def word_level_punctuation_cleaning_implementor(text):
    new_text=[]
    #break sentences based on dot,exclamatio,question mark and comma
    sen_token = re.split('[,.!?]',text.lower().strip())
    for sen in sen_token:
        word_token = nltk.tokenize.TreebankWordTokenizer().tokenize(sen)
        new_sen=[]
        for word in word_token:
            word=re.compile('[@_!#$%^&*()<>?/\|}{~:+-=]').sub(' ',word).strip()
            if word in puncts:
                word=''
            if ( word not in puncts):
                word = punctuation_replacer(word)
                word = punctuation_cleaner(word)
            new_sen.append(word)
        new_sentence = ' '.join(new_sen)
        new_text.append(new_sentence)
    cleaned_text ='.'.join(new_text)
    return cleaned_text
```

In [14]:

```
contraction_dict = {"ain't": "is not", "aren't": "are not","can't": "can not", "'cause": "b

def _get_contractions(contraction_dict):
    contraction_re = re.compile('(%s)' % '|'.join(contraction_dict.keys()))
    return contraction_dict, contraction_re

contractions, contractions_re = _get_contractions(contraction_dict)

def replace_contractions(text):
    def replace(match):
        return contractions[match.group(0)]
    return contractions_re.sub(replace, text)
```

In [15]:

```
def date_extractor (text):
    date_pattern = re.compile('((\d{2})(/|-)(\S+)(/|-)(\d{2,4})) | [a-z]{3}-\d{2,4}')
    return date_pattern.sub(r'',text.lower())

def time_extractor (text):
    time_pattern = re.compile('((\d{2})\:(\d{2})\:?(\d{2})?)')
    return time_pattern.sub(r'', text.lower())

def amount_extractor (text):
    amount_pattern = re.compile('(\-?\d+(\,\d+)*(\.)\d+)|(\-?\d+(\,\d+)+\d+)|(rs(\.|\S|\,)*
    return amount_pattern.sub(r'',text.lower())
```

In [16]:

```
def clean_numbers(x):
    if bool(re.search(r'\d', x)):
        x = re.sub('[0-9]{5,}', '', x)
        x = re.sub('[0-9]{4}', '', x)
        x = re.sub('[0-9]{3}', '', x)
        x = re.sub('[0-9]{2}', '', x)
    return x
```

In [17]:

```
slang_abbrev_dict = {
    'AFAIK': 'As Far As I Know','AFK': 'Away From Keyboard','ASAP': 'As Soon As Possible','
    'BAK': 'Back At Keyboard','BBL': 'Be Back Later','BBS': 'Be Back Soon','BFN': 'Bye For
    'BTW': 'By The Way','B4': 'Before','B4N': 'Bye For Now','CU': 'See You','CUL8R': 'See Y
    'FC': 'Fingers Crossed','FWIW': 'For What It\'s Worth','FYI': 'For Your Information','G
    'GR8': 'Great!','G9': 'Genius','IC': 'I See','ICQ': 'I Seek you','ILU': 'I Love You','I
    'IRL': 'In Real Life','KISS': 'Keep It Simple, Stupid','LDR': 'Long Distance Relationsh
    'LTNS': 'Long Time No See','L8R': 'Later','MTE': 'My Thoughts Exactly','M8': 'Mate','NR
    'PITA': 'Pain In The Ass','PRT': 'Party','PRW': 'Parents Are Watching','QPSA?': 'Que Pa
    'ROTFLMAO': 'Rolling On The Floor Laughing My Ass Off','SK8': 'Skate','STATS': 'Your se
    'TTYL': 'Talk To You Later','U': 'You','U2': 'You Too','U4E': 'Yours For Ever','WB': 'W
    'WUF': 'Where Are You From?','W8': 'Wait','7K': 'Sick:-D Laugher'}
```

In [18]:

```
mispell_dict = {"aren't" : "are not","can't" : "can not","couldn't" : "could not","couldnt"
"doesnt" : "does not","don't" : "do not","hadn't" : "had not","hasn't" : "has not","haven't
"he'll" : "he will","he's" : "he is","i'd" : "I would","i'd" : "I had","i'll" : "I will","i
"it'll":"it will","i've" : "I have","let's" : "let us","mightn't" : "might not","mustn't" :
"she'll" : "she will","she's" : "she is","shouldn't" : "should not","shouldnt" : "should no
"theres" : "there is","they'd" : "they would","they'll" : "they will","they're" : "they are
"we're" : "we are","weren't" : "were not","we've" : "we have","what'll" : "what will","what
"where's" : "where is","who'd" : "who would","who'll" : "who will","who're" : "who are","wh
"wouldn't" : "would not","you'd" : "you would","you'll" : "you will","you're" : "you are","
"we'll":" will","didn't": "did not","tryin'":"trying","'cause":"because"}
```

In [19]:

```
def _get_mispell(mispell_dict):
    mispell_re = re.compile('(%s)' % '|'.join(mispell_dict.keys()))
    return mispell_dict, mispell_re
```

In [20]:

```
def replace_typical_misspell(text):
    mispellings, mispellings_re = _get_mispell(mispell_dict)

    def replace(match):
        return mispellings[match.group(0)]

    return mispellings_re.sub(replace, text)
```

In [21]:

```
def unslang(text):
    """Converts text like "OMG" into "Oh my God"
    """
    text = [slang_abbrev_dict[w.upper()] if w.upper() in slang_abbrev_dict.keys() else w fo
    return " ".join(text)
```

In [22]:

```
def remove_emoji(string):
    emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', string)
```

In [23]:

```
def remove_html(text):
    return BeautifulSoup(text, "lxml").text
```

In [24]:

```
def data_preprocessing(df):
    df = df.apply(lambda x : remove_urls(x))
    df = df.apply(lambda x: remove_html(x))
    df = df.apply(lambda x : replace_typical_misspell(x.lower()))
    df = df.apply(lambda x : unslang(x.lower()))
    df = df.apply(lambda x : remove_emoji(x.lower()))
    df = df.apply(lambda x : unicode_remover(x))
    df = df.apply(lambda x : date_extractor(x))
    df = df.apply(lambda x : time_extractor(x))
    df = df.apply(lambda x : amount_extractor(x))
    df = df.apply(lambda x : word_level_punctuation_cleaning_implementor(x))
    df = df.apply(lambda x : clean_numbers(x))
    return df
```

In [25]:

```
cleaned_data = pd.DataFrame(data_preprocessing(data.Text))
```

In [26]:

```
cleaned_data['len']=cleaned_data.Text.apply(lambda x: len(x.split(' ')))
```

In [27]:

```
cleaned_data.Text[0]
```

Out[27]:

```
'i have seen kohli for so many years now.i have rarely seen him so relaxed.i
will not be surprised if he finds another high'
```

Analysis Purpose Only. Will Use vocabulary from count-vectorizer function

In [28]:

```
word_to_index={}
index_to_word={}
word_frequency_counter={}
```

In [29]:

```
index=0
for corpus in cleaned_data.Text:
    sentence_token=corpus.strip().split('.')
    for sentence in sentence_token:
        word_token=sentence.strip().split(' ')
        for word in word_token:
            if word in word_to_index:
                word_frequency_counter[word]=word_frequency_counter.get(word)+1
                continue
            else:
                word_frequency_counter[word]=1
                word_to_index[word]=index
                index_to_word[index]=word
                index+=1
```

In [30]:

```
len(word_to_index)
```

Out[30]:

```
4944
```

In [31]:

```
import enchant
bad_english_word=[]
for word in word_to_index.keys():
    if word=='':
        bad_english_word.append((word,word_to_index[word]))
    elif (not enchant.Dict("en_US").check(word)):
        bad_english_word.append((word,word_to_index[word]))
```

In [32]:

```
len(bad_english_word)
```

Out[32]:

```
1458
```

In [33]:

```
not_in_glove_word=[]
for word in word_to_index.keys():
    if word not in embedding_index.keys():
        not_in_glove_word.append((word,word_to_index[word]))
```

In [34]:

```
len(not_in_glove_word)
```

Out[34]:

```
587
```

In [35]:

```
#bad english words that are present in glove
print(len(set(bad_english_word)-set(not_in_glove_word)))
set(bad_english_word)-set(not_in_glove_word)
```

```
 ('achi', 2609),
 ('adam', 1703),
 ('adelaide', 485),
 ('adina', 1109),
 ('af', 4589),
 ('afg', 913),
 ('afghanistan', 1392),
 ('africa', 517),
 ('african', 2220),
 ('afridi', 919),
 ('afterall', 2693),
 ('agle', 3827),
 ('agm', 3614),
 ('agnst', 1964),
 ('aheadd', 3598),
 ('aiden', 2859),
 ('aise', 4379),
 ('aithe', 4035),
 ('aj', 2608),
 ('ajam', 3374),
```

In [36]:

```
embedding_index[list(set(bad_english_word)-set(not_in_glove_word))[0][0]]
```

Out[36]:

```
array([-0.10047 , -0.76205 ,  1.0121  , -0.88258 , -1.2462  ,  0.55595 ,
        0.020013,  1.1555  ,  0.12481 ,  0.98591 , -0.77185 , -0.37652 ,
       -0.77298 ,  1.7311  , -0.45004 ,  0.52332 ,  0.30354 ,  0.18746 ,
       -0.019968, -0.14049 , -0.66515 ,  1.0727  , -1.0966  ,  0.43803 ,
       -0.76985 ], dtype=float32)
```

In [37]:

```
#good english words but not present in glove
set(not_in_glove_word)-set(bad_english_word)
```

Out[37]:

```
{('activeness', 3173),
 ('blissfulness', 4825),
 ('demolisher', 701),
 ('multifold', 2474),
 ('neutralizes', 3069),
 ('shote', 3528)}
```

In [38]:

```
embedding_index.get('activeness')
```

In [39]:

```
temp_word_df=pd.DataFrame.from_dict(word_frequency_counter,orient='index').reset_index()
temp_word_df.columns=['word','word_frequency']
temp_word_df.word_frequency.value_counts()[:15]
```

Out[39]:

```
1     2577
2      677
3      373
4      208
5      191
6      106
7       73
8       68
9       58
10      56
11      41
13      36
14      31
12      30
15      24
Name: word_frequency, dtype: int64
```

Mittens Implementation

Combination of experiment in possible

- Hyper parameters : - Xmax, alpha, mew, max_iterations, learning_rate
- Initialization can be - zero, random, xavier
- Co-occurence : normalized, count-based

In [40]:

```
doc=[]
sentence_count=0
for tweet in cleaned_data.Text:
    temp=tweet.split('.')
    sentence_count+=len(temp)
    doc.extend(temp)
```

In [41]:

```
from sklearn.feature_extraction.text import CountVectorizer
import scipy.sparse as sp
count_model = CountVectorizer(ngram_range=(1,1)) # default unigram model
X = count_model.fit_transform(doc)
Xc = (X.T * X)
g = sp.diags(1./Xc.diagonal())
Xc_norm = g * Xc # normalized co-occurence matrix
```

In [42]:

```
word_to_index=count_model.vocabulary_
```

In [43]:

```
embedding_length=25
vocab=list(word_to_index.keys())
```

In [60]:

```
#xavier Initialization
def randmatrix(m, n, random_seed=1000):
    """Creates an m x n matrix of random values drawn using
    the Xavier Glorot method."""
    val = np.sqrt(6.0 / (m + n))
    np.random.seed(random_seed)
    return np.random.uniform(-val, val, size=(m, n))
```

In [61]:

```
Word_weight=randmatrix(len(vocab),embedding_length) # word weight matrix intialized with xa
Context_weight=randmatrix(len(vocab),embedding_length) # context weight matrix initialized
```

In [62]:

```
mask_embeddings=np.zeros((len(vocab)),dtype='bool') # mask for vocabulary with embeddings i
original_embeddings=np.zeros((len(vocab),embedding_length)) # initializing original embeddi
np.random.seed(1000)
for index,word in enumerate(vocab):
    if word in embedding_index:
        mask_embeddings[index]=1
        original_embeddings[index]=embedding_index[word]#update original embedding with glo

        #xavier initialization modified for word having glove embeddings--it has been appro
#         Word_weight[index]=0.5*embedding_index[word]+np.random.normal(loc=0, scale=0.01,
#         Context_weight[index]=0.5*embedding_index[word]+np.random.normal(loc=0, scale=0.0

Word_bias=randmatrix(len(vocab),1)
Context_bias=randmatrix(len(vocab),1)
```

In [63]:

```


print(Context_bias.shape)
print(Word_bias.shape)
print(Context_weight.shape)
print(Word_weight.shape)
print(original_embeddings.shape)
print(mask_embeddings.shape)
```

```
(4688, 1)
(4688, 1)
(4688, 25)
(4688, 25)
(4688, 25)
(4688,)
```

In [48]:

```
def log_occurrence_func(coocurrence_matrix):
    m,n=coocurrence_matrix.shape
    for row_ind in range(m):
        for col_ind in range(n):
            ele=coocurrence_matrix[row_ind,col_ind]
            if ele==0:
                coocurrence_matrix[row_ind,col_ind]=0
            else:
                coocurrence_matrix[row_ind,col_ind]=np.log(ele)


    return coocurrence_matrix
```

In [49]:

```
log_occurrence=log_occurrence_func(Xc.todense()) # word-word count based co-occurence matri
```

In [50]:

```
(log_occurrence>0).sum()
```

Out[50]:

```
34207
```

In [51]:

```
## Xmax=100,alpha=0.75 (refer glove paper)
def weighting_function(coocurrence_matrix):
    m,n=coocurrence_matrix.shape
    for row_ind in range(m):
        for col_ind in range(n):
            ele=coocurrence_matrix[row_ind,col_ind]
            if ele<100:
                coocurrence_matrix[row_ind,col_ind]=(ele/100)**(0.75)
            else:
                coocurrence_matrix[row_ind,col_ind]=1
    return coocurrence_matrix
```

In [52]:

```
weights=weighting_function(Xc.todense()) # weights for each element of error to be calculat
```

In [53]:

```
(weights>0).sum()
```

Out[53]:

```
310
```

In [64]:

```
def prediction_glove(Word_weight,Context_weight,Word_bias,Context_bias):
    """
    Word_weight : weight embedding to be learnt. dimensions=(n_word,len(glove_embedding_vec
    Contex_weight : context weight embedding to be learnt. dimensions=(n_word,len(glove_emb
    Word_bias : bias term to be learnt for each word. dimesnion=(n_word,1)
    Context_bias : bias term to  be learnt for each context word. dimesnion=(n_word,1)

    """

    pred=np.dot(Word_weight,Context_weight.T)+Word_bias+Context_bias

    return pred
```

In [65]:

```
def cost_function_glove_error(Word_weight,Context_weight,Word_bias,Context_bias,weights,log
    """

    Word_weight : weight embedding to be learnt. dimensions=(n_word,len(glove_embedding_vec
    Contex_weight : context weight embedding to be learnt. dimensions=(n_word,len(glove_emb
    Word_bias : bias term to be learnt for each word. dimesnion=(n_word,1)
    Context_bias : bias term to  be learnt for each context word. dimesnion=(n_word,1)
    weights : weights for each element in count based co-occurence matrix. dimesnion=(n_wor
    log_coocurrence : log of count based word-word co-occurence matrix. dimesnion=(n_word,n

    """
    # prediction
    prediction=prediction_glove(Word_weight,Context_weight,Word_bias,Context_bias)
    #difference between prediction and log of co-occurence
    difference = prediction - log_coocurrence
    # weighted difference of prediction and log(co-occurence matrix)
    weighted_difference=np.multiply(weights,difference)# element wise multiplication for we
    Word_weight_grad=np.dot(weighted_difference,Context_weight)
    Context_weight_grad=np.dot(weighted_difference.T,Word_weight)
    Word_bias_grad=weighted_difference.sum(axis=1).reshape(-1, 1)
    Context_bias_grad=weighted_difference.sum(axis=0).reshape(-1, 1)
    error=(np.multiply(weights,np.square(difference))).sum() # element wise multiplication
    learned_embedding=Word_weight+Context_weight


    return error,learned_embedding,Word_weight_grad,Word_bias_grad,Context_weight_grad,Cont
```

In [66]:

```
prediction=prediction_glove(Word_weight,Context_weight,Word_bias,Context_bias)
```

In [67]:

```
#mew value=0.1, it has good result in the paper
def cost_func_mitten_error(glove_cost,learned_embedding,Word_weight_grad,Word_bias_grad,Con
    distance=learned_embedding[mask_embeddings]-original_embeddings[mask_embeddings]

    #update of grad for words having embedding in glove
    Word_weight_grad[mask_embeddings]=Word_weight_grad[mask_embeddings]+2*mew*distance
    Context_weight_grad[mask_embeddings]=Context_weight_grad[mask_embeddings]+2*mew*distanc
    #mitten error
    mitten_cost=glove_cost+mew*(np.linalg.norm(distance, ord=2, axis=1) ** 2).sum()

    return mitten_cost,Word_weight_grad,Word_bias_grad,Context_weight_grad,Context_bias_gra
```

In [68]:

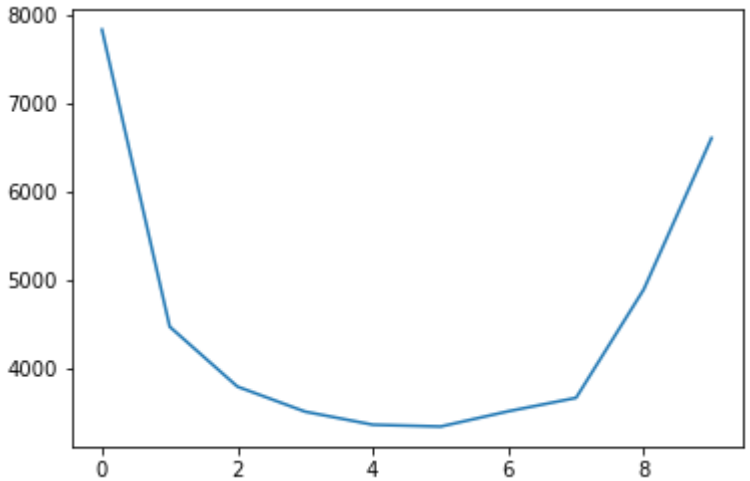
```
max_iterations=10
learning_rate=0.05
mitten_cost_list=[]
glove_cost_list=[]
for iter in range(max_iterations):
    glove_cost,learned_embedding,Word_weight_grad,Word_bias_grad,Context_weight_grad,Contex
    glove_cost_list.append(glove_cost)
    mitten_cost,Word_weight_grad,Word_bias_grad,Context_weight_grad,Context_bias_grad=cost_
    mitten_cost_list.append(mitten_cost)
    Word_weight=Word_weight-learning_rate*Word_weight_grad
    Context_weight=Context_weight-learning_rate*Context_weight_grad
    Word_bias=Word_bias-learning_rate*Word_bias_grad
    Context_biast=Context_bias-learning_rate*Context_bias_grad
```

In [69]:

```
from matplotlib import pyplot as plt
plt.plot(mitten_cost_list)
```

Out[69]:

```
[<matplotlib.lines.Line2D at 0x1ed9fe7d160>]
```



In [ ]: