Implementing Tfidf vectorizer

Task1

In [15]:
```
from collections import Counter
from scipy.sparse import csr_matrix
import math
import operator
from sklearn.preprocessing import normalize
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [16]:
```
corpus = ['this is the first document',
      'this document is the second document',
      'and this is the third one',
      'is this the first document', ]
```

In [63]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)
skl_output = vectorizer.transform(corpus)
```

Fit function for Tfidf

In [17]:
```
def IDF(corpus, unique_words):
    idf_dict={}
    N=len(corpus)
    for i in unique_words:
        count=0
        for sen in corpus:
            if i in sen.split():
                count=count+1
            idf_dict[i]=(math.log((1+N)/(count+1)))+1
    return idf_dict
```

In [18]:
```
def fit(whole_data):
    unique_words = set()
    if isinstance(whole_data, (list,)):
        for x in whole_data:
            for y in x.split():
                if len(y)<2:
                    continue
                unique_words.add(y)
        unique_words = sorted(list(unique_words))
        vocab = {j:i for i,j in enumerate(unique_words)}
    Idf_values_of_unique_words=IDF(whole_data,unique_words)
    return vocab, Idf_values_of_unique_words
```

In [19]:
```
Vocab, idf_of_vocab=fit(corpus)
print(Vocab)
```
```
{'and': 0, 'document': 1, 'first': 2, 'is': 3, 'one': 4, 'second': 5, 'the': 6, 'third': 7,
'this': 8}
```

In [64]:
```
print(vectorizer.get_feature_names())
```
```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

Both the unique words are same

Transform function :

In [65]:
```
def transform(dataset,vocab,idf_values):
    sparse= csr_matrix( (len(dataset), len(vocab)), dtype=np.float64)
    for row  in range(0,len(dataset)):
        number_of_words_in_sentence=Counter(dataset[row].split())
        for word in dataset[row].split():
            if word in  list(vocab.keys()):
                tf_idf_value=(number_of_words_in_sentence[word]/len(dataset[row].split()))*(
idf_values[word])
                sparse[row,vocab[word]]=tf_idf_value
    output =normalize(sparse, norm='l2', axis=1, copy=True, return_norm=False)
    return output
```

In [67]:
```
fin_out=transform(corpus,Vocab,idf_of_vocab)
print(fin_out)
print("the shape of the sparse matrix",fin_out.shape)
```
```
  (0, 1)	0.4697913855799205
  (0, 2)	0.580285823684436
  (0, 3)	0.3840852409148149
  (0, 6)	0.3840852409148149
  (0, 8)	0.3840852409148149
  (1, 1)	0.6876235979836937
  (1, 3)	0.2810886740337529
  (1, 5)	0.5386476208856762
  (1, 6)	0.2810886740337529
  (1, 8)	0.2810886740337529
  (2, 0)	0.511848512707169
  (2, 3)	0.267103787642168
  (2, 4)	0.511848512707169
  (2, 6)	0.267103787642168
  (2, 7)	0.511848512707169
  (2, 8)	0.267103787642168
  (3, 1)	0.4697913855799205
  (3, 2)	0.580285823684436
  (3, 3)	0.3840852409148149
  (3, 6)	0.3840852409148149
  (3, 8)	0.3840852409148149
the shape of the sparse matrix (4, 9)
```

In [22]:
```
print(fin_out[0].toarray())
```
```
[[0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]]
```

Sklearn's Countvectorizer

In [23]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer=TfidfVectorizer()
vectorizer.fit(corpus)
skl_output=vectorizer.transform(corpus)
```

In [24]:
```
print(skl_output)
```
```
  (0, 8)	0.38408524091481483
  (0, 6)	0.38408524091481483
  (0, 3)	0.38408524091481483
  (0, 2)	0.5802858236844359
  (0, 1)	0.46979138557992045
  (1, 8)	0.281088674033753
  (1, 6)	0.281088674033753
  (1, 5)	0.5386476208856763
  (1, 3)	0.281088674033753
  (1, 1)	0.6876235979836938
  (2, 8)	0.267103787642168
  (2, 7)	0.511848512707169
  (2, 6)	0.267103787642168
  (2, 4)	0.511848512707169
  (2, 3)	0.267103787642168
  (2, 0)	0.511848512707169
  (3, 8)	0.38408524091481483
  (3, 6)	0.38408524091481483
  (3, 3)	0.38408524091481483
  (3, 2)	0.5802858236844359
  (3, 1)	0.46979138557992045
```

In [25]:
```
print(skl_output[0].toarray())
```
```
[[0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]]
```

TFIDF values for all the words are the same for both programs.

Task 2

Consider only the top 50 feature names based on their IDF values

In [30]:
```
import pickle
with open('cleaned_strings', 'rb') as f:
    corpus1 = pickle.load(f)
print("Number of documents in corpus = ",len(corpus1))
```
```
Number of documents in corpus =  746
```

Fit function

In [31]:
```
def IDF(corpus, unique_words):
    a=0
    idf_dict={}
    idfsorted={}
    unique_words50={}
    idf50={}
    N=len(corpus)
    for i in unique_words:
        count=0
        for sen in corpus:
            if i in sen.split():
                count=count+1
            idf_dict[i]=(math.log((1+N)/(count+1)))+1
    return idf_dict
```

In [32]:
```
def fit(whole_data):
    unique_words = set()
    if isinstance(whole_data, (list,)):
        for x in whole_data:
            for y in x.split():
                if len(y)<2:
                    continue
                unique_words.add(y)
        unique_words = sorted(list(unique_words))

    Idf_values_of_all_unique_words=IDF(whole_data,unique_words)

    top50_idf_value_indices = np.argsort(list(Idf_values_of_all_unique_words.values()))[::-1
][:50]

    top50_idf_values = np.take(list(Idf_values_of_all_unique_words.values()), top50_idf_valu
e_indices)
    top50_idf_words = np.take(list(Idf_values_of_all_unique_words.keys()), top50_idf_value_i
ndices)

    wanted_idf_vocabulary = dict(zip(top50_idf_words, top50_idf_values))
    wanted_vocabulary = {j: i for i, j in enumerate(list(wanted_idf_vocabulary.keys()))}
    return wanted_vocabulary, wanted_idf_vocabulary
```

In [33]:
```
wanted_vocabulary, wanted_idf_vocabulary = fit(corpus1)
print(wanted_vocabulary)
```
```
{'zombiez': 0, 'havilland': 1, 'hearts': 2, 'heads': 3, 'hbo': 4, 'hayworth': 5, 'hayao': 6,
'hay': 7, 'hatred': 8, 'heche': 9, 'harris': 10, 'happy': 11, 'happiness': 12, 'hanks': 13,
'hankies': 14, 'hang': 15, 'heartwarming': 16, 'heels': 17, 'gone': 18, 'hero': 19, 'higher':
20, 'hide': 21, 'hes': 22, 'heroism': 23, 'heroine': 24, 'heroes': 25, 'hernandez': 26, 'heis
t': 27, 'hendrikson': 28, 'helping': 29, 'help': 30, 'helms': 31, 'hellish': 32, 'helen': 33,
'handles': 34, 'handle': 35, 'ham': 36, 'grade': 37, 'grates': 38, 'grasp': 39, 'graphics': 4
0, 'granted': 41, 'grainy': 42, 'gradually': 43, 'government': 44, 'halfway': 45, 'gotten': 4
6, 'gotta': 47, 'goth': 48, 'gosh': 49}
```

In [34]:
```
print(wanted_idf_vocabulary)
```
```
{'zombiez': 6.922918004572872, 'havilland': 6.922918004572872, 'hearts': 6.922918004572872,
'heads': 6.922918004572872, 'hbo': 6.922918004572872, 'hayworth': 6.922918004572872, 'hayao':
6.922918004572872, 'hay': 6.922918004572872, 'hatred': 6.922918004572872, 'heche': 6.92291800
4572872, 'harris': 6.922918004572872, 'happy': 6.922918004572872, 'happiness': 6.922918004572
872, 'hanks': 6.922918004572872, 'hankies': 6.922918004572872, 'hang': 6.922918004572872, 'he
artwarming': 6.922918004572872, 'heels': 6.922918004572872, 'gone': 6.922918004572872, 'her
o': 6.922918004572872, 'higher': 6.922918004572872, 'hide': 6.922918004572872, 'hes': 6.92291
8004572872, 'heroism': 6.922918004572872, 'heroine': 6.922918004572872, 'heroes': 6.922918004
572872, 'hernandez': 6.922918004572872, 'heist': 6.922918004572872, 'hendrikson': 6.922918004
572872, 'helping': 6.922918004572872, 'help': 6.922918004572872, 'helms': 6.922918004572872,
'hellish': 6.922918004572872, 'helen': 6.922918004572872, 'handles': 6.922918004572872, 'hand
le': 6.922918004572872, 'ham': 6.922918004572872, 'grade': 6.922918004572872, 'grates': 6.922
918004572872, 'grasp': 6.922918004572872, 'graphics': 6.922918004572872, 'granted': 6.9229180
04572872, 'grainy': 6.922918004572872, 'gradually': 6.922918004572872, 'government': 6.922918
004572872, 'halfway': 6.922918004572872, 'gotten': 6.922918004572872, 'gotta': 6.922918004572
872, 'goth': 6.922918004572872, 'gosh': 6.922918004572872}
```

Transform Function

In [35]:
```
def transform(dataset,vocab,idf_values):
    sparse= csr_matrix( (len(dataset), len(vocab)), dtype=np.float64)
    for row  in range(0,len(dataset)):
        number_of_words_in_sentence=Counter(dataset[row].split())
        for word in dataset[row].split():
            if word in  list(vocab.keys()):
                tf_idf_value=(number_of_words_in_sentence[word]/len(dataset[row].split()))*(
idf_values[word])
                sparse[row,vocab[word]]=tf_idf_value
    output =normalize(sparse, norm='l2', axis=1, copy=True, return_norm=False)
    return output
```

In [39]:
```
final_output = transform(corpus1, wanted_vocabulary, wanted_idf_vocabulary)
print(final_output.shape)
print(final_output)
```
```
(746, 50)
  (19, 9)	0.25819888974716115
  (19, 13)	0.25819888974716115
  (19, 17)	0.25819888974716115
  (19, 30)	0.25819888974716115
  (19, 31)	0.25819888974716115
  (19, 38)	0.25819888974716115
  (19, 40)	0.7745966692414833
  (94, 49)	1.0
  (101, 18)	1.0
  (104, 35)	1.0
  (109, 0)	0.7071067811865476
  (109, 32)	0.7071067811865476
  (132, 25)	1.0
  (135, 12)	0.5773502691896258
  (135, 43)	0.5773502691896258
  (135, 46)	0.5773502691896258
  (180, 20)	1.0
  (191, 7)	0.7071067811865475
  (191, 24)	0.7071067811865475
  (197, 5)	1.0
  (222, 22)	1.0
  (225, 8)	1.0
  (232, 21)	1.0
  (234, 45)	1.0
  (236, 28)	1.0
  (253, 15)	1.0
  (270, 27)	1.0
  (277, 48)	1.0
  (323, 37)	1.0
  (343, 41)	1.0
  (371, 47)	1.0
  (421, 23)	1.0
  (430, 1)	1.0
  (437, 19)	1.0
  (459, 33)	1.0
  (462, 29)	1.0
  (475, 14)	1.0
  (532, 34)	1.0
  (533, 11)	1.0
  (539, 36)	1.0
  (572, 10)	1.0
  (610, 42)	1.0
  (625, 3)	1.0
  (628, 26)	1.0
  (633, 39)	1.0
  (644, 16)	1.0
  (660, 4)	1.0
  (681, 2)	1.0
  (703, 44)	1.0
  (714, 6)	1.0
```

In [61]:
```
print(final_output[19])
```
```
  (0, 9)	0.25819888974716115
  (0, 13)	0.25819888974716115
  (0, 17)	0.25819888974716115
  (0, 30)	0.25819888974716115
  (0, 31)	0.25819888974716115
  (0, 38)	0.25819888974716115
  (0, 40)	0.7745966692414833
```

In [60]:
```
print(final_output[19].toarray())
```
```
[[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.25819889 0.         0.
  0.         0.25819889 0.         0.         0.         0.25819889
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.25819889 0.25819889 0.         0.         0.         0.
  0.         0.         0.25819889 0.         0.77459667 0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]]
```

sklearn's countvectorizer

In [58]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer=TfidfVectorizer(norm = 'l2',max_features=50)
vect=vectorizer.fit(corpus1)
skl_output=vectorizer.transform(corpus1)
print(skl_output)
print(skl_output.shape)
```
```
  (0, 25)	1.0
  (1, 30)	0.572445699981522
  (1, 8)	0.8199426324888012
  (2, 32)	0.5699230219852334
  (2, 25)	0.35548195762874873
  (2, 10)	0.5273651853196742
  (2, 0)	0.5202944244602254
  (3, 20)	1.0
  (4, 25)	0.5110768901174535
  (4, 4)	0.859534997767905
  (5, 25)	1.0
  (7, 25)	0.33776010837475723
  (7, 17)	0.9412322291499969
  (11, 25)	0.32556342048857684
  (11, 23)	0.5366405664621315
  (11, 21)	0.5533799348242362
  (11, 4)	0.5475363087512142
  (12, 4)	1.0
  (14, 25)	1.0
  (15, 37)	0.7118629038754104
  (15, 0)	0.7023184506234106
  (16, 23)	0.6478335937139895
  (16, 14)	0.4007783298922675
  (16, 11)	0.6478335937139895
  (17, 25)	0.48899424176935075
  :	:
  (727, 44)	1.0
  (728, 25)	0.5830722125629216
  (728, 17)	0.8124203314399383
  (729, 38)	0.7482164865280394
  (729, 30)	0.4838684403668721
  (729, 25)	0.45392006091884396
  (733, 46)	0.8346729885507431
  (733, 25)	0.5507458598879985
  (734, 37)	0.6615139054580876
  (734, 33)	0.7499328989220224
  (735, 35)	0.7391763947283081
  (735, 32)	0.6735118836935696
  (736, 43)	1.0
  (738, 18)	1.0
  (739, 47)	0.6094124262000808
  (739, 40)	0.5963565650047202
  (739, 19)	0.5224704222907762
  (740, 28)	0.8668308595259199
  (740, 25)	0.4986023074290322
  (741, 44)	1.0
  (742, 48)	0.6520428640510826
  (742, 36)	0.6520428640510826
  (742, 14)	0.3868852631984358
  (744, 3)	1.0
  (745, 31)	1.0
(746, 50)
```