```
from collections import Counter
from tqdm import tqdm
from scipy.sparse import csr_matrix
import math
import operator
from sklearn.preprocessing import normalize
import numpy
```

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

```
def fit(corpus):
  #creating set for storing unique_words
  unique_words = set()
  if isinstance(corpus, list):
    #iterating through rows in corpus
    for row in corpus:
      #iterating through each words in row
      for word in row.split():
        #checking for length of words
        if len(word) < 2:
          continue
        #adding each words in set
        unique_words.add(word)
    #converting set into sorted list
    unique_words = sorted(list(unique_words))
    #creating vocab dict
    vocab = {j:i for i , j in enumerate(unique_words)}

    return vocab
  else:
    print("Send list of Sentences")
```

```
def transform(corpus, vocab):
  #creating empty lists for rows, column, values
  rows = []
  columns = []
  val = []
  #iterating through rows of corpus
  for idx, row in enumerate(tqdm(corpus)):
    values = []
    #creating word_frequency dict using counter
    word_freq = dict(Counter(row.split()))
    #iterating through words in vocab dict
    for word in vocab.keys():
      #calculating tfidf using formula
      tfidf = (word_freq.get(word, 0) / len(row.split())) * get_idf(word, corpus)
      col_index = vocab.get(word, 0)
```

```
      col_index = vocab.get(word, 0)

      if tfidf != 0:
        rows.append(idx)
        columns.append(col_index)
      values.append(tfidf)
    val.append(values)
  norm = normalize(val)
  #return csr_matrix((norm), shape=(len(corpus),len(vocab)))
  #return csr_matrix((norm, (rows, columns)), shape=(len(corpus),len(vocab)))
  return csr_matrix((norm))
```

```
def get_idf(word, corpus):
  count=0
  #iterating through rows in corpus
  for r in corpus:
    #if word in that row increament count by one
    if word in r:
      count += 1

  idf_key= 1 + math.log((1+len(corpus)) / (count+1))
  return idf_key
```

```
#calling fit method
vocab = fit(corpus)
#creating dictionary with words in vocab as keys and its idf value as values
vocab_idf = {word:get_idf(word, corpus) for word in vocab.keys()}
```

```
#printing all words in vocab dict
print(list(vocab.keys()))
```

```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

```
#printing idf values for words in vocab dict
print(list(vocab_idf.values()))
```

```
[1.916290731874155, 1.2231435513142097, 1.5108256237659907, 1.0, 1.916290731874155, 1
```

```
#shape of output matrix
print(numpy.shape(transform(corpus, vocab)))
```

```
100%|██████████| 4/4 [00:00<00:00, 7588.07it/s](4, 9)
```

```
tr = transform(corpus, vocab)
print(tr)
```

```
100%|██████████| 4/4 [00:00<00:00, 10094.59it/s]  (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
```

```
1 print(tr[0].toarray())
```

```
[[0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]]
```

```
1
```

✓ 0s completed at 15:32