```
Assignment
            What does tf-idf mean?
            Tf-idf stands for term frequency-inverse document frequency, and the tf-idf weight is a weight often used in information retrieval and text
            mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The
            importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in
            the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's
            relevance given a user query.
            One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions
            are variants of this simple model.
            Tf-idf can be successfully used for stop-words filtering in various subject fields including text summarization and classification.
            </font>
            How to Compute:
            Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a
            word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency
            (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific
            term appears.
              • TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is
                possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by
                the document length (aka. the total number of terms in the document) as a way of normalization:
                TF(t) = \frac{\text{Number of times term t appears in a document}}{\text{Number of terms in the document}}.

    IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally

                important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance.
                Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:
                $IDF(t) = \log_{e}\frac{\text{Total number of documents}} {\text{Number of documents with term t in it}}.$ for numerical stability
                we will be changing this formula little bit DF(t) = \log_{e}\frac{\text{Total number of documents}} {\text{Number of documents}}
                with term t in it+1.$
            Example
            Consider a document containing 100 words wherein the word cat appears 3 times. The term frequency (i.e., tf) for cat is then (3 / 100) =
            0.03. Now, assume we have 10 million documents and the word cat appears in one thousand of these. Then, the inverse document
            frequency (i.e., idf) is calculated as log(10,000,000 / 1,000) = 4. Thus, the Tf-idf weight is the product of these quantities: 0.03 * 4 = 0.12.
             </font>
            Task-1
            1. Build a TFIDF Vectorizer & compare its results with Sklearn:

    As a part of this task you will be implementing TFIDF vectorizer on a collection of text documents.

    You should compare the results of your own implementation of TFIDF vectorizer with that of sklearns implementation TFIDF

                vectorizer.

    Sklearn does few more tweaks in the implementation of its version of TFIDF vectorizer, so to replicate the exact results you would

                need to add following things to your custom implementation of tfidf vectorizer:
                  1. Sklearn has its vocabulary generated from idf sroted in alphabetical order
                  2. Sklearn formula of idf is different from the standard textbook formula. Here the constant "1" is added to the numerator and
                    denominator of the idf as if an extra document was seen containing every term in the collection exactly once, which prevents
                    zero divisions. IDF(t) = 1+\log_{e}\frac{1\cdot x}{1\cdot x} + text{Total number of documents in collection}{1+\cot}
                    documents with term t in it}}.$
                  3. Sklearn applies L2-normalization on its output matrix.
                  4. The final output of sklearn tfidf vectorizer is a sparse matrix.
                Steps to approach this task:
                  1. You would have to write both fit and transform methods for your custom implementation of tfidf vectorizer.
                  2. Print out the alphabetically sorted voacb after you fit your data and check if its the same as that of the feature names from sklearn
                  3. Print out the idf values from your implementation and check if its the same as that of sklearns tfidf vectorizer idf values.
                  4. Once you get your voacb and idf values to be same as that of sklearns implementation of tfidf vectorizer, proceed to the below
                  5. Make sure the output of your implementation is a sparse matrix. Before generating the final output, you need to normalize your
                    sparse matrix using L2 normalization. You can refer to this link https://scikit-
                    learn.org/stable/modules/generated/sklearn.preprocessing.normalize.html
                  6. After completing the above steps, print the output of your custom implementation and compare it with sklearns implementation
                    of tfidf vectorizer.
                  7. To check the output of a single document in your collection of documents, you can convert the sparse matrix related only to that
                    document into dense matrix and print it.
            Note-1: All the necessary outputs of sklearns tfidf vectorizer have been provided as reference in this notebook, you can compare your
            outputs as mentioned in the above steps, with these outputs.
            Note-2: The output of your custom implementation and that of sklearns implementation would match only with the collection of document
            strings provided to you as reference in this notebook. It would not match for strings that contain capital letters or punctuations, etc, because
            sklearn version of tfidf vectorizer deals with such strings in a different way. To know further details about how sklearn tfidf vectorizer
            works with such string, you can always refer to its official documentation.
            Note-3: During this task, it would be helpful for you to debug the code you write with print statements wherever necessary. But when you
            are finally submitting the assignment, make sure your code is readable and try not to print things which are not part of this task.
            Corpus
In [147]: ## SkLearn# Collection of string documents
            corpus = [
                   'this is the first document',
                   'this document is the second document',
                   'and this is the third one',
                  'is this the first document',
            SkLearn Implementation
In [148]:
            from sklearn.feature_extraction.text import TfidfVectorizer
             vectorizer = TfidfVectorizer()
            vectorizer.fit(corpus)
            skl_output = vectorizer.transform(corpus)
            print("Number of documents in corpus = ",len(corpus))
            Number of documents in corpus = 4
In [149]: # sklearn feature names, they are sorted in alphabetic order by default.
            print(vectorizer.get_feature_names())
            ['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
In [150]: # Here we will print the sklearn tfidf vectorizer idf values after applying the fit method
             # After using the fit function on the corpus the vocab has 9 words in it, and each has its i
            df value.
            print(vectorizer.idf_)
            [1.91629073 1.22314355 1.51082562 1.
                                                                   1.91629073 1.91629073
                          1.91629073 1.
             1.
In [151]: # shape of sklearn tfidf vectorizer output after applying transform method.
            skl_output.shape
Out[151]: (4, 9)
In [152]: # sklearn tfidf values for first line of the above corpus.
            # Here the output is a sparse matrix
            print(skl_output)
               (0, 8)
                                0.38408524091481483
                                0.38408524091481483
               (0, 6)
               (0, 3)
                               0.38408524091481483
                               0.5802858236844359
               (0, 2)
               (0, 1)
                               0.46979138557992045
               (1, 8)
                                0.281088674033753
               (1, 6)
                                0.281088674033753
                               0.5386476208856763
               (1, 5)
               (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
                               0.38408524091481483
               (3, 6)
               (3, 3)
                               0.38408524091481483
               (3, 2)
                               0.5802858236844359
               (3, 1)
                                0.46979138557992045
In [153]: # sklearn tfidf values for first line of the above corpus.
            # To understand the output better, here we are converting the sparse output matrix to dense
             matrix and printing it.
            # Notice that this output is normalized using L2 normalization. sklearn does this by defaul
            print(skl_output[0].toarray())
                            0.46979139 0.58028582 0.38408524 0.
            [[0.
                                                                                  Θ.
               0.38408524 0.
                                         0.38408524]]
            Your custom implementation
In [154]: # Write your code here.
            # Make sure its well documented and readble with appropriate comments.
            # Compare your results with the above sklearn tfidf vectorizer
            # You are not supposed to use any other library apart from the ones given below
            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 as np
             def custom_fit(corpus):
                 total_words_list={}
                 for i in range(len(corpus)):
                      words_in_each_document=corpus[i].split()
                      total_words_list=set(words_in_each_document).union(set(total_words_list))
                 total_words_list=sorted(list(total_words_list))
                 return total_words_list
            features=custom_fit(corpus)
            print("vocab list:")
            print(features)
            vocab list:
            ['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
In [155]: #transform funtion
            def custom_transform(features):
                 total=np.zeros(len(features))
                 tf=[]
                 for i in range(len(corpus)):
                      rowfre=dict(Counter(corpus[i].split()))
                      length=len(corpus[i].split())
                      rowcount = dict.fromkeys(features, 0)
                      for word in rowfre:
                           rowcount[word]=rowfre[word] # computing no of words in each document
                      for word in rowcount:
                           if(rowcount[word]>0):
                                total[k]+=1
                                                        # computing no of documents which contains word Wi and s
             toring it in total
                      rowtf=[]
                      for word in rowcount:
                           rowtf.append(rowcount[word]/length)
                                                                         # computing tf values
                      tf.append(rowtf)
                 for j in range(len(features)):
                           idf_=1+math.log((1+len(corpus))/float(1+total[j])) #computing idf values
                           idf.append(idf_)
                 for i in range(len(corpus)):
                      for j in range(len(features)):
                           tf[i][j]*=idf[j]
                 tf=normalize(csr_matrix(tf), norm='12')
                 return(tf)
            idf=[]
            tfidf=custom_transform(features)
In [156]: print("idf values of custom implementation:\n",idf)
            idf values of custom implementation:
             731874155, 1.0, 1.916290731874155, 1.0]
In [157]: | print("tfidf sparse matrix:\n",tfidf)
            tfidf sparse matrix:
                               0.4697913855799205
                (0, 1)
               (0, 2)
                                0.580285823684436
                                0.3840852409148149
               (0, 3)
                                0.3840852409148149
               (0, 6)
               (0, 8)
                                0.3840852409148149
               (1, 1)
                                0.6876235979836937
                                0.2810886740337529
               (1, 3)
               (1, 5)
                                0.5386476208856762
               (1, 6)
                                0.2810886740337529
               (1, 8)
                                0.2810886740337529
                                0.511848512707169
               (2, 0)
               (2, 3)
                                0.267103787642168
               (2, 4)
                                0.511848512707169
               (2, 6)
                                0.267103787642168
               (2, 7)
                                0.511848512707169
               (2, 8)
                                0.267103787642168
               (3, 1)
                                0.4697913855799205
                                0.580285823684436
               (3, 2)
               (3, 3)
                                0.3840852409148149
               (3, 6)
                                0.3840852409148149
               (3, 8)
                                0.3840852409148149
In [158]: print(tfidf[0].toarray())
                            0.46979139 0.58028582 0.38408524 0.
                                                                                  Θ.
                                         0.38408524]]
               0.38408524 0.
In [159]: tfidf.shape
Out[159]: (4, 9)
            Task-2
            2. Implement max features functionality:

    As a part of this task you have to modify your fit and transform functions so that your vocab will contain only 50 terms with top idf

    This task is similar to your previous task, just that here your vocabulary is limited to only top 50 features names based on their idf

                values. Basically your output will have exactly 50 columns and the number of rows will depend on the number of documents you
                have in your corpus.
              • Here you will be give a pickle file, with file name cleaned_strings. You would have to load the corpus from this file and use it as
                input to your tfidf vectorizer.

    Steps to approach this task:

                  1. You would have to write both fit and transform methods for your custom implementation of tfidf vectorizer, just like in the
                    previous task. Additionally, here you have to limit the number of features generated to 50 as described above.
                  2. Now sort your vocab based in descending order of idf values and print out the words in the sorted voacb after you fit your data.
                    Here you should be getting only 50 terms in your vocab. And make sure to print idf values for each term in your vocab.
                  3. Make sure the output of your implementation is a sparse matrix. Before generating the final output, you need to normalize your
                    sparse matrix using L2 normalization. You can refer to this link https://scikit-
                    learn.org/stable/modules/generated/sklearn.preprocessing.normalize.html
                  4. Now check the output of a single document in your collection of documents, you can convert the sparse matrix related only to
                    that document into dense matrix and print it. And this dense matrix should contain 1 row and 50 columns.
            # Below is the code to load the cleaned_strings pickle file provided
            # Here corpus is of list type
            import pickle
            with open('cleaned_strings', 'rb') as f:
                 corpus1 = pickle.load(f)
            # printing the length of the corpus loaded
             print("Number of documents in corpus = ",len(corpus1))
            Number of documents in corpus = 746
In [161]: 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 as np
            def custom_fit(corpus):
                 total_words_list={}
                 for i in range(len(corpus)):
                      words_in_each_document=corpus[i].split()
                      total_words_list=set(words_in_each_document).union(set(total_words_list))
                 """computing no of documents which contain the word Wi, the word which is in present les
            s no of documents have
                 highest idf value"""
                 vocab = dict.fromkeys(total_words_list, 0)
                 for i in range(len(corpus)):
                      res=Counter(corpus[i].split())
                      for word in res:
                           vocab[word]+=1
                 vocab=sorted(vocab.items(), key=lambda x: x[1])
                 first=[]
                 for i in range(50):
                      first.append(vocab[i][0])
                                                          # storing the top 50 idf valued features in first
                 first=sorted(list(first))
                 return first
            features=custom_fit(corpus1)
            print("top 50 vocab list with high idf values:\n", features)
            top 50 vocab list with high idf values:
            ['abandoned', 'abroad', 'angry', 'applause', 'atrocious', 'blue', 'borrowed', 'builders', 'c ard', 'coastal', 'concerning', 'confidence', 'debated', 'deep', 'elsewhere', 'english', 'evok es', 'explosion', 'fully', 'gallon', 'gone', 'ground', 'initially', 'instruments', 'intensit
            y', 'kidnapped', 'lower', 'marred', 'nun', 'obliged', 'plane', 'preservation', 'realized', 'r
            emoving', 'reporter', 'riveted', 'senior', 'sites', 'soldiers', 'sour', 'stereotypically', 's
            tinker', 'strokes', 'stuff', 'suffered', 'thomerson', 'trond', 'waylaid', 'within', 'witticis
            ms']
In [162]: def custom_transform(features):
                 total=np.zeros(len(features))
                 for i in range(len(corpus1)):
                      rowfre=dict(Counter(corpus1[i].split()))
                      length=len(corpus1[i].split())
                      rowcount = dict.fromkeys(features, 0)
                      k=0
                      for word in rowfre:
                           if word in rowcount:
                                rowcount[word]=rowfre[word]
                                                                     # computing no of words in each document
                      for word in rowcount:
                           if(rowcount[word]>0):
                                                                  # computing no of documents which contains word
                                total[k]+=1
              Wi and storing it in total
                           k+=1
                      rowtf=[]
                      for word in rowcount:
                           rowtf.append(rowcount[word]/length)
                                                                         # computing tf values
                      tf.append(rowtf)
                 for j in range(len(features)):
                           idf_=1+math.log((1+len(corpus1))/float(1+total[j]))
                                                                                                  # computing idf value
            S
                           idf.append(idf_)
                 for i in range(len(corpus1)):
                      for j in range(len(features)):
                           tf[i][j]*=idf[j]
                 tf=normalize(csr_matrix(tf), norm='12')
                 return(tf)
            idf=[]
            tfidf=custom_transform(features)
In [163]: print("idf values :\n",idf)
            idf values :
             [6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.92291800457287
            72, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.922918004572872, 6.92291800457
```

2872, 6.922918004572872, 6.92291

872, 6.922918004572872]

tfidf spare matrix:

(19, 5)

(19, 7)

(19, 35)

(37, 28)

(56, 12)

(57, 3)

(78, 40)

(101, 20)

(104, 42)

(107, 31)

(108, 18)

(130, 44)

(134, 19)

(134, 36)

(135, 2)

(135, 38)

(135, 39)

(135, 49)

(148, 47)

(216, 1)

(225, 14)

(259, 26)

(270, 0)

(277, 15)

(296, 46)

(299, 22)

(299, 37)

(328, 27)

(338, 8)

(342, 21) (342, 30)

15)

(355

In [164]: print("tfidf spare matrix:\n",tfidf)

1.0

1.0

1.0

1.0

1.0

1.0

1.0

1.0

1.0

0.5

0.5

0.5

0.5

1.0

1.0

1.0

1.0

1.0

1.0

1.0

1.0

1.0

0.5773502691896258

0.5773502691896258

0.5773502691896258

0.7071067811865475

0.7071067811865475

0.7071067811865475

0.7071067811865475

0.7071067811865476

0.7071067811865476