Assignment_3_Solution

August 19, 2020

1 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.

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:

$TF(t) = \frac{\text{Number of times term t appears in a document}}{\text{Total number of terms in the document}}$.

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 stabiltiy we will be changing this formula little bit $IDF(t) = \log_e \frac{\text{Total number of documents}}{\text{Number of documents with term t in it+1}}$.

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.

1.1 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 implemenation 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:

Sklearn has its vocabulary generated from idf sroted in alphabetical order

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.

$IDF(t) = 1 + \log_e \frac{1 + \text{Total number of documents in collection}}{1+\text{Number of documents with term t in it}}$.

Sklearn applies L2-normalization on its output matrix.

The final output of sklearn tfidf vectorizer is a sparse matrix.

Steps to approach this task:

You would have to write both fit and transform methods for your custom implementation of tfidf vectorizer.

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 tfidf vectorizer.

Print out the idf values from your implementation and check if its the same as that of sklearns tfidf vectorizer idf values.

Once you get your voacb and idf values to be same as that of sklearns implementation of tfidf vectorizer, proceed to the below steps.

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

After completing the above steps, print the output of your custom implementation and compare it with sklearns implementation of tfidf vectorizer.

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.

1.1.1 Corpus

[92]:
```
## 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',
]
```

1.1.2 SkLearn Implementation

[93]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)
skl_output = vectorizer.transform(corpus)
```

[94]:
```
# 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']
```

[95]:
```
# 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 idf value.

print(vectorizer.idf_)
```

```
[1.91629073 1.22314355 1.51082562 1.         1.91629073 1.91629073
 1.         1.91629073 1.        ]
```

[96]:
```
# shape of sklearn tfidf vectorizer output after applying transform method.
```

```
skl_output.shape
```

[96]: (4, 9)

[97]:
```
# sklearn tfidf values for first line of the above corpus.
# Here the output is a sparse matrix

print(skl_output[0])
```

```
  (0, 8)        0.38408524091481483
  (0, 6)        0.38408524091481483
  (0, 3)        0.38408524091481483
  (0, 2)        0.5802858236844359
  (0, 1)        0.46979138557992045
```

[98]:
```
# 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 default.

print(skl_output[0].toarray())
```

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

1.1.3 Your custom implementation

[99]:
```
# 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
```

1.1.4 Step 1: Creating Fit Method to find all unique words and assigning dimension to each unique word.

[100]:
```
from tqdm import tqdm #To visualise progress of execution of an iterable.

#Fit method
def fit(dataset):
  unique_words = set() # set initialised to store unique words
  if isinstance(dataset,(list,)):
    for row in dataset: # For each sentence in Corpus list
        for word in row.split(" "):# For each word of the sentence of Corpus in␣
↪consideration
          if (len(word) > 1):
            unique_words.add(word) # Adding word of the sentence in␣
↪consideration to the set unique_words if the length of the word is greater␣
↪than 1.
    unique_words = sorted(list(unique_words)) # Sorting the unique words␣
↪collected alphabetically.
    vocab = {j:i for i,j in enumerate(unique_words)} #Assigning index number to␣
↪each unique words collected in the set unique_words

    return vocab # Returning dictionary of unqiue words collected from the␣
↪dataset along with it's index numbers
  else:
    print('Please pass list of sentence.')

#Reference: Assignment_3_Reference.ipynb given by Applied AI Course
```

1.1.5 Step 2: Printing out the alphabetically sorted voacb after it fits data given and comparing results of fit method with sklearns tfidf vectorizer

[101]:
```
#Fit Method Result
vocab = fit(corpus)
print(list(vocab.keys()))
```

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

1.1.6 SkLearn Implementation

[102]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)
print(vectorizer.get_feature_names())
```

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

1.1.7 Step 3: Creating function to generate idf of each words of the vocab

[103]:
```
# Formula for IDF:
# IDF of term t = 1 + log( (1 + Total Number of Documents in the Collection) /␣
 ↪(1 + Number of Documents with term t in it) )
def get_idf(dataset,vocab):
    vocab_idf = {}
    if isinstance(dataset,(list,)):
        total_doc = len(dataset)  #Total number of Documents in dataset
        for word in list(vocab.keys()):
            doc_with_word = 0
            for doc in dataset:
                if word in doc:
                    doc_with_word += 1 #Total number of Document with word in it
            vocab_idf[word] = (1 + math.log( (1 + total_doc) / (1 +␣
 ↪doc_with_word) ) ) #Calculation of IDF for each word and storing in the dict

        return vocab_idf
    else:
        print('Please pass list of sentence.')
```

1.1.8 Step 2: Printing idf values of the unique words in corpus and comparing result with Sklearn Implementation of tfidf vectorizer idf values

[104]:
```
vocab_idf = get_idf(corpus,vocab)
print(list(vocab_idf.values()))
print(vocab)
```

```
[1.916290731874155, 1.2231435513142097, 1.5108256237659907, 1.0,
1.916290731874155, 1.916290731874155, 1.0, 1.916290731874155, 1.0]
{'and': 0, 'document': 1, 'first': 2, 'is': 3, 'one': 4, 'second': 5, 'the': 6,
'third': 7, 'this': 8}
```

1.1.9 SkLearn Implementation of IDF

[105]:
```
print(vectorizer.idf_)
```

```
[1.91629073 1.22314355 1.51082562 1.         1.91629073 1.91629073
 1.         1.91629073 1.        ]
```

1.1.10 Step 3:Creating transform method to get the vector

[106]:
```
#Formula for TF:
#TF of term t = (Number of times t appeared in the document)/(Total number of␣
 ↪documents in the collection)
#Tranform method for custom tfidf vectortizer
def transform(dataset,vocab):
```

```
    rows = []
    columns = []
    values = []
    if isinstance(dataset,(list,)):
        vocab_idf = get_idf(dataset,vocab)
        total_doc = len(corpus)
        for idx,row in enumerate(dataset):
            word_freq = dict(Counter(row.split(" ")))
            for word,frequency in word_freq.items():
                if (len(word) > 1):
                    column_index = vocab.get(word, -1)
                    if (column_index != -1):
                        rows.append(idx)
                        columns.append(column_index)
                        #Calculating TF of word
                        word_tf = frequency/total_doc
                        #Calculating TF-IDF valye of word
                        tf_idf_value = (word_tf * vocab_idf[word])
                        values.append(tf_idf_value)
        sparse_matrix = csr_matrix((values, (rows, columns)),␣
↪shape=(len(dataset),len(vocab)))
        sparse_matrix_normalized = normalize(sparse_matrix, norm='l2')
        return sparse_matrix_normalized
    else:
        print('Please pass list of sentence.')
```

1.1.11 Getting shape of the sparse matrix through custom tfidf vectorizer

[107]:
```
# shape of custom tfidf vectorizer output after applying transform method.
tf_idf_vectorizer = transform(corpus, vocab)

print(tf_idf_vectorizer.shape)
```

```
(4, 9)
```

1.1.12 Sklearn implementation to get shape of the sparse matrix

[108]:
```
# shape of sklearn tfidf vectorizer output after applying transform method.
skl_output.shape
```

[108]: (4, 9)

1.1.13 Getting custom tfidf values and sklearn tf idf values for firstline of the above corpus and it's comparison

[109]:
```
print(tf_idf_vectorizer[0])
```

```
  (0, 1)        0.46979138557992045
  (0, 2)        0.5802858236844359
  (0, 3)        0.38408524091481483
  (0, 6)        0.38408524091481483
  (0, 8)        0.38408524091481483
```

[110]:
```
# sklearn tfidf values for first line of the above corpus.
# Here the output is a sparse matrix

print(skl_output[0])
```

```
  (0, 8)        0.38408524091481483
  (0, 6)        0.38408524091481483
  (0, 3)        0.38408524091481483
  (0, 2)        0.5802858236844359
  (0, 1)        0.46979138557992045
```

1.2 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 scores.

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:

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.

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.

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

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.

[111]:
```
# 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:
    corpus = pickle.load(f)

# printing the length of the corpus loaded
print("Number of documents in corpus = ",len(corpus))
```

```
Number of documents in corpus =  746
```

[112]:
```
# Write your code here.
# Try not to hardcode any values.
# Make sure its well documented and readble with appropriate comments.
```

1.2.1 Fit Method to get the unique words in the dataset given

[113]:
```
#Fit Method to get the unique words and it's index value
def fit(dataset):
    if isinstance(dataset,(list,)):
         unique_words = set()
         for doc in dataset:
             for word in doc.split(" "):
                 if (len(word)>1):
                     unique_words.add(word)
         unique_words = sorted(list(unique_words))
         vocab = {j:i for i,j in enumerate(unique_words)}

         return vocab
    else:
        print('Please pass list of sentence.')
```

1.2.2 Implementation of Custom fit method on pickle file to get all unique words from the pickle file

[114]:
```
#custome fit method implementation result
vocab = fit(corpus)
print(list(vocab.keys()))
```

```
['aailiyah', 'abandoned', 'ability', 'abroad', 'absolutely', 'abstruse',
'abysmal', 'academy', 'accents', 'accessible', 'acclaimed', 'accolades',
'accurate', 'accurately', 'accused', 'achievement', 'achille', 'ackerman',
'act', 'acted', 'acting', 'action', 'actions', 'actor', 'actors', 'actress',
'actresses', 'actually', 'adams', 'adaptation', 'add', 'added', 'addition',
```

'admins', 'admiration', 'admitted', 'adorable', 'adrift', 'adventure', 'advise',
'aerial', 'aesthetically', 'affected', 'affleck', 'afraid', 'africa',
'afternoon', 'age', 'aged', 'ages', 'ago', 'agree', 'agreed', 'aimless', 'air',
'aired', 'akasha', 'akin', 'alert', 'alexander', 'alike', 'allison', 'allow',
'allowing', 'almost', 'along', 'alongside', 'already', 'also', 'although',
'always', 'amateurish', 'amaze', 'amazed', 'amazing', 'amazingly', 'america',
'american', 'americans', 'among', 'amount', 'amusing', 'amust', 'anatomist',
'angel', 'angela', 'angeles', 'angelina', 'angle', 'angles', 'angry', 'anguish',
'angus', 'animals', 'animated', 'animation', 'anita', 'ann', 'anne',
'anniversary', 'annoying', 'another', 'anthony', 'antithesis', 'anyone',
'anything', 'anyway', 'apart', 'appalling', 'appealing', 'appearance',
'appears', 'applauded', 'applause', 'appreciate', 'appropriate', 'apt',
'argued', 'armageddon', 'armand', 'around', 'array', 'art', 'articulated',
'artiness', 'artist', 'artistic', 'artless', 'arts', 'aside', 'ask', 'asleep',
'aspect', 'aspects', 'ass', 'assante', 'assaulted', 'assistant',
'astonishingly', 'astronaut', 'atmosphere', 'atrocious', 'atrocity', 'attempt',
'attempted', 'attempting', 'attempts', 'attention', 'attractive', 'audience',
'audio', 'aurv', 'austen', 'austere', 'author', 'average', 'aversion', 'avoid',
'avoided', 'award', 'awarded', 'awards', 'away', 'awesome', 'awful',
'awkwardly', 'aye', 'baaaaaad', 'babbling', 'babie', 'baby', 'babysitting',
'back', 'backdrop', 'backed', 'bad', 'badly', 'bag', 'bailey', 'bakery',
'balance', 'balanced', 'ball', 'ballet', 'balls', 'band', 'barcelona', 'barely',
'barking', 'barney', 'barren', 'based', 'basic', 'basically', 'bat', 'bates',
'baxendale', 'bear', 'beautiful', 'beautifully', 'bec', 'became', 'bechard',
'become', 'becomes', 'began', 'begin', 'beginning', 'behind', 'behold', 'bela',
'believable', 'believe', 'believed', 'bell', 'bellucci', 'belly', 'belmondo',
'ben', 'bendingly', 'bennett', 'bergen', 'bertolucci', 'best', 'better',
'betty', 'beware', 'beyond', 'bible', 'big', 'biggest', 'billy', 'biographical',
'bipolarity', 'bit', 'bitchy', 'black', 'blah', 'blake', 'bland', 'blandly',
'blare', 'blatant', 'blew', 'blood', 'blown', 'blue', 'blush', 'boasts', 'bob',
'body', 'bohemian', 'boiling', 'bold', 'bombardments', 'bond', 'bonding',
'bonus', 'bonuses', 'boobs', 'boogeyman', 'book', 'boost', 'bop', 'bordered',
'borderlines', 'borders', 'bore', 'bored', 'boring', 'borrowed', 'boss',
'bother', 'bothersome', 'bought', 'box', 'boyfriend', 'boyle', 'brain',
'brainsucking', 'brat', 'breaking', 'breeders', 'brevity', 'brian', 'brief',
'brigand', 'bright', 'brilliance', 'brilliant', 'brilliantly', 'bring',
'brings', 'broad', 'broke', 'brooding', 'brother', 'brutal', 'buddy', 'budget',
'buffalo', 'buffet', 'build', 'builders', 'buildings', 'built', 'bullock',
'bully', 'bunch', 'burton', 'business', 'buy', 'cable', 'cailles', 'california',
'call', 'called', 'calls', 'came', 'cameo', 'camera', 'camerawork', 'camp',
'campy', 'canada', 'cancan', 'candace', 'candle', 'cannot', 'cant', 'captain',
'captured', 'captures', 'car', 'card', 'cardboard', 'cardellini', 'care',
'carol', 'carrell', 'carries', 'carry', 'cars', 'cartoon', 'cartoons', 'case',
'cases', 'cast', 'casted', 'casting', 'cat', 'catchy', 'caught', 'cause',
'ceases', 'celebration', 'celebrity', 'celluloid', 'centers', 'central',
'century', 'certain', 'certainly', 'cg', 'cgi', 'chalkboard', 'challenges',
'chance', 'change', 'changes', 'changing', 'channel', 'character',
'characterisation', 'characters', 'charisma', 'charismatic', 'charles',

'charlie', 'charm', 'charming', 'chase', 'chasing', 'cheap', 'cheaply', 'check',
'checking', 'cheek', 'cheekbones', 'cheerfull', 'cheerless', 'cheesiness',
'cheesy', 'chemistry', 'chick', 'child', 'childhood', 'children', 'childrens',
'chills', 'chilly', 'chimp', 'chodorov', 'choice', 'choices', 'choked',
'chosen', 'chow', 'christmas', 'christopher', 'church', 'cinema', 'cinematic',
'cinematographers', 'cinematography', 'circumstances', 'class', 'classic',
'classical', 'clear', 'clearly', 'clever', 'clich', 'cliche', 'clients',
'cliff', 'climax', 'close', 'closed', 'clothes', 'club', 'co', 'coach', 'coal',
'coastal', 'coaster', 'coherent', 'cold', 'cole', 'collect', 'collective',
'colored', 'colorful', 'colours', 'columbo', 'come', 'comedic', 'comedy',
'comes', 'comfortable', 'comforting', 'comical', 'coming', 'commands',
'comment', 'commentary', 'commented', 'comments', 'commercial', 'community',
'company', 'compelling', 'competent', 'complete', 'completed', 'completely',
'complex', 'complexity', 'composed', 'composition', 'comprehensible',
'compromise', 'computer', 'concentrate', 'conception', 'conceptually',
'concerning', 'concerns', 'concert', 'conclusion', 'condescends', 'confidence',
'configuration', 'confirm', 'conflict', 'confuses', 'confusing', 'connections',
'connery', 'connor', 'conrad', 'consequences', 'consider', 'considerable',
'considered', 'considering', 'considers', 'consistent', 'consolations',
'constant', 'constantine', 'constructed', 'contained', 'containing', 'contains',
'content', 'continually', 'continuation', 'continue', 'continuity',
'continuously', 'contract', 'contrast', 'contributing', 'contributory',
'contrived', 'control', 'controversy', 'convention', 'convey', 'convince',
'convincing', 'convoluted', 'cool', 'coppola', 'cords', 'core', 'corn', 'corny',
'correct', 'cost', 'costs', 'costumes', 'cotton', 'could', 'couple', 'course',
'court', 'courtroom', 'cover', 'cowardice', 'cox', 'crackles', 'crafted',
'crap', 'crash', 'crashed', 'crayon', 'crayons', 'crazy', 'create', 'created',
'creates', 'creative', 'creativity', 'creature', 'credible', 'credit',
'credits', 'crew', 'crime', 'crisp', 'critic', 'critical', 'crocdodile',
'crocs', 'cross', 'crowd', 'crowe', 'cruel', 'cruise', 'cry', 'cult', 'culture',
'curtain', 'custer', 'cute', 'cutest', 'cutie', 'cutouts', 'cuts', 'cutting',
'dads', 'damian', 'damn', 'dance', 'dancing', 'dangerous', 'dark', 'darren',
'daughter', 'daughters', 'day', 'days', 'de', 'dead', 'deadly', 'deadpan',
'deal', 'dealt', 'death', 'debated', 'debbie', 'debits', 'debut', 'decay',
'decent', 'decidely', 'decipher', 'decisions', 'dedication', 'dee', 'deep',
'deeply', 'defensemen', 'defined', 'definitely', 'delete', 'delight',
'delightful', 'delights', 'deliver', 'delivered', 'delivering', 'delivers',
'dependant', 'depending', 'depends', 'depicted', 'depicts', 'depressing',
'depth', 'derivative', 'describe', 'describes', 'desert', 'deserved',
'deserves', 'deserving', 'design', 'designed', 'designer', 'desperately',
'desperation', 'despised', 'despite', 'destroy', 'detailing', 'details',
'develop', 'development', 'developments', 'di', 'diabetic', 'dialog', 'dialogs',
'dialogue', 'diaper', 'dickens', 'difference', 'different', 'dignity',
'dimensional', 'direct', 'directed', 'directing', 'direction', 'director',
'directorial', 'directors', 'disappointed', 'disappointing', 'disappointment',
'disaster', 'disbelief', 'discomfort', 'discovering', 'discovery', 'disgrace',
'disgusting', 'dislike', 'disliked', 'disney', 'disparate', 'distant',
'distinction', 'distorted', 'distract', 'distressed', 'disturbing', 'diving',

'doctor', 'documentaries', 'documentary', 'dodge', 'dogs', 'dollars',
'dominated', 'done', 'donlevy', 'dont', 'doomed', 'dose', 'doubt', 'downs',
'dozen', 'dr', 'dracula', 'draft', 'drag', 'drago', 'drama', 'dramatic',
'drawings', 'drawn', 'dream', 'dreams', 'dreary', 'dribble', 'drift',
'drifting', 'drive', 'drooling', 'dropped', 'dry', 'due', 'duet', 'dull',
'dumb', 'dumbest', 'duper', 'duris', 'dustin', 'dvd', 'dwight', 'dysfunction',
'earlier', 'early', 'earth', 'easily', 'easy', 'eating', 'ebay', 'ebola',
'eccleston', 'ed', 'edge', 'editing', 'edition', 'educational', 'edward',
'effect', 'effective', 'effects', 'effort', 'efforts', 'egotism', 'eighth',
'eiko', 'either', 'elaborately', 'elderly', 'elegant', 'element', 'elias',
'eloquently', 'else', 'elsewhere', 'embarrassed', 'embarrassing', 'embassy',
'emerge', 'emilio', 'emily', 'emoting', 'emotion', 'emotionally', 'emotions',
'emperor', 'empowerment', 'emptiness', 'empty', 'en', 'enchanting', 'end',
'endearing', 'ended', 'ending', 'endlessly', 'ends', 'energetic', 'energy',
'engaging', 'english', 'enhanced', 'enjoy', 'enjoyable', 'enjoyed', 'enjoyment',
'enough', 'enter', 'enterprise', 'entertained', 'entertaining', 'entire',
'entirely', 'entrance', 'episode', 'episodes', 'equivalent', 'era', 'errol',
'errors', 'escalating', 'escapism', 'especially', 'essence', 'establish',
'established', 'estate', 'estevez', 'etc', 'european', 'evaluate', 'even',
'events', 'ever', 'every', 'everybody', 'everyone', 'everything', 'everywhere',
'evidently', 'evil', 'evinced', 'evokes', 'exactly', 'exaggerating', 'example',
'excellent', 'excellently', 'except', 'exceptional', 'exceptionally',
'excerpts', 'excessively', 'exchange', 'exciting', 'excruciatingly', 'excuse',
'excuses', 'executed', 'exemplars', 'existent', 'existential', 'expansive',
'expect', 'expectations', 'expected', 'expecting', 'experience', 'experiences',
'expert', 'explain', 'explains', 'explanation', 'exploit', 'explorations',
'explosion', 'expression', 'exquisite', 'extant', 'exteriors', 'extraneous',
'extraordinary', 'extremely', 'eye', 'eyes', 'fabulous', 'face', 'faces',
'facial', 'facing', 'fact', 'factory', 'failed', 'fails', 'fair', 'fairly',
'faithful', 'fall', 'falling', 'falls', 'falsely', 'falwell', 'fame', 'famed',
'family', 'famous', 'fan', 'fanciful', 'fans', 'fantastic', 'fantasy', 'far',
'farce', 'fare', 'fascinated', 'fascinating', 'fascination', 'fashioned',
'fast', 'faster', 'fat', 'father', 'faultless', 'fausa', 'faux', 'favorite',
'favourite', 'fear', 'feature', 'features', 'feel', 'feeling', 'feelings',
'feet', 'feisty', 'fellowes', 'felt', 'female', 'females', 'ferry', 'fest',
'fi', 'fields', 'fifteen', 'fifties', 'fill', 'film', 'filmed', 'filmiing',
'filmmaker', 'filmography', 'films', 'final', 'finale', 'finally', 'financial',
'find', 'finds', 'fine', 'finest', 'fingernails', 'finished', 'fire', 'first',
'fish', 'fishnet', 'fisted', 'fit', 'five', 'flag', 'flakes', 'flaming',
'flashbacks', 'flat', 'flaw', 'flawed', 'flaws', 'fleshed', 'flick', 'flicks',
'florida', 'flowed', 'flying', 'flynn', 'focus', 'fodder', 'follow',
'following', 'follows', 'foolish', 'footage', 'football', 'force', 'forced',
'forces', 'ford', 'foreign', 'foreigner', 'forever', 'forget', 'forgettable',
'forgetting', 'forgot', 'forgotten', 'form', 'format', 'former', 'fort',
'forth', 'forwarded', 'found', 'four', 'fox', 'foxx', 'frances', 'francis',
'frankly', 'free', 'freedom', 'freeman', 'french', 'fresh', 'freshness',
'friends', 'friendship', 'frightening', 'front', 'frontier', 'frost',
'frustration', 'fulci', 'fulfilling', 'full', 'fully', 'fumbling', 'fun',

'function', 'fundamental', 'funniest', 'funny', 'future', 'fx', 'gabriel',
'gadget', 'gain', 'gake', 'galley', 'gallon', 'game', 'games', 'garage',
'garbage', 'garbo', 'garfield', 'gas', 'gaudi', 'gave', 'gay', 'geek', 'gem',
'general', 'generally', 'generates', 'generic', 'genius', 'genre', 'gently',
'genuine', 'george', 'gerardo', 'gere', 'get', 'gets', 'getting', 'ghibili',
'giallo', 'gibberish', 'gifted', 'giovanni', 'girl', 'girlfriend', 'girls',
'girolamo', 'give', 'given', 'gives', 'giving', 'glad', 'glance', 'glasses',
'gloriously', 'go', 'goalies', 'god', 'goes', 'going', 'gone', 'gonna', 'good',
'gore', 'goremeister', 'gorman', 'gosh', 'got', 'goth', 'gotta', 'gotten',
'government', 'grace', 'grade', 'gradually', 'grainy', 'granted', 'graphics',
'grasp', 'grates', 'great', 'greatest', 'greatness', 'green', 'greenstreet',
'grew', 'grim', 'grimes', 'gripping', 'groove', 'gross', 'ground', 'guards',
'guess', 'guests', 'guilt', 'gung', 'guy', 'guys', 'hackneyed', 'haggis',
'hair', 'hairsplitting', 'half', 'halfway', 'ham', 'hand', 'handle', 'handled',
'handles', 'hands', 'hang', 'hankies', 'hanks', 'happen', 'happened',
'happiness', 'happy', 'hard', 'harris', 'hate', 'hated', 'hatred', 'havilland',
'hay', 'hayao', 'hayworth', 'hbo', 'head', 'heads', 'hear', 'heard', 'heart',
'hearts', 'heartwarming', 'heaven', 'heche', 'heels', 'heist', 'helen', 'hell',
'hellish', 'helms', 'help', 'helping', 'helps', 'hence', 'hendrikson',
'hernandez', 'hero', 'heroes', 'heroine', 'heroism', 'hes', 'hide', 'high',
'higher', 'highest', 'highlights', 'highly', 'hilarious', 'hill', 'hilt', 'hip',
'history', 'hitchcock', 'ho', 'hockey', 'hoffman', 'hold', 'holding', 'holds',
'holes', 'hollander', 'hollow', 'hollywood', 'home', 'homework', 'honest',
'honestly', 'hoot', 'hope', 'hopefully', 'hopeless', 'horrendous',
'horrendously', 'horrible', 'horrid', 'horrified', 'horror', 'horse', 'hosting',
'hot', 'hour', 'hours', 'house', 'houses', 'howdy', 'howe', 'howell', 'however',
'huge', 'hugo', 'human', 'humanity', 'humans', 'hummh', 'humor', 'humorous',
'humour', 'hurt', 'huston', 'hype', 'hypocrisy', 'idea', 'idealogical',
'identified', 'identifies', 'identify', 'idiot', 'idiotic', 'idyllic', 'iffy',
'im', 'imaginable', 'imagination', 'imaginative', 'imagine', 'imdb',
'imitation', 'impact', 'imperial', 'implausible', 'important', 'impossible',
'impressed', 'impression', 'impressive', 'improved', 'improvement',
'improvisation', 'impulse', 'inappropriate', 'incendiary', 'includes',
'including', 'incomprehensible', 'inconsistencies', 'incorrectness',
'incredible', 'incredibly', 'indeed', 'indescribably', 'indication',
'indictment', 'indie', 'individual', 'indoor', 'indulgent', 'industry',
'ineptly', 'inexperience', 'inexplicable', 'initially', 'innocence', 'insane',
'inside', 'insincere', 'insipid', 'insomniacs', 'inspiration', 'inspiring',
'instant', 'instead', 'instruments', 'insulin', 'insult', 'intangibles',
'integral', 'integration', 'intelligence', 'intelligent', 'intense',
'intensity', 'intentions', 'interacting', 'interest', 'interested',
'interesting', 'interim', 'interplay', 'interpretations', 'interview',
'intoning', 'intrigued', 'inventive', 'involved', 'involves', 'involving', 'iq',
'ireland', 'ironically', 'irons', 'ironside', 'irritating', 'ishioka', 'iso',
'issue', 'issues', 'istagey', 'italian', 'ive', 'jack', 'jaclyn', 'james',
'jamie', 'japanese', 'jason', 'jay', 'jealousy', 'jean', 'jennifer', 'jerky',
'jerry', 'jessica', 'jessice', 'jet', 'jim', 'jimmy', 'job', 'jobs', 'joe',
'john', 'joins', 'joke', 'jokes', 'jonah', 'jones', 'journey', 'joy', 'joyce',

'juano', 'judge', 'judging', 'judith', 'judo', 'julian', 'june', 'junk',
'junkyard', 'justice', 'jutland', 'kanaly', 'kathy', 'keep', 'keeps', 'keira',
'keith', 'kept', 'kevin', 'kid', 'kidnapped', 'kids', 'kieslowski', 'kill',
'killer', 'killing', 'killings', 'kind', 'kinda', 'kirk', 'kitchy', 'knew',
'knightley', 'knocked', 'know', 'known', 'knows', 'koteas', 'kris',
'kristoffersen', 'kudos', 'la', 'labute', 'lack', 'lacked', 'lacks', 'ladies',
'lady', 'lame', 'lance', 'landscapes', 'lane', 'lange', 'largely', 'laselva',
'lassie', 'last', 'lasting', 'latched', 'late', 'later', 'latest', 'latifa',
'latin', 'latter', 'laugh', 'laughable', 'laughed', 'laughs', 'layers', 'lazy',
'lead', 'leading', 'leap', 'learn', 'least', 'leave', 'leaves', 'leaving',
'lee', 'left', 'legal', 'legendary', 'length', 'leni', 'less', 'lesser',
'lestat', 'let', 'lets', 'letting', 'level', 'levels', 'lewis', 'lid', 'lie',
'lies', 'lieutenant', 'life', 'lifetime', 'light', 'lighting', 'like', 'liked',
'likes', 'lilli', 'lilt', 'limitations', 'limited', 'linda', 'line', 'linear',
'lines', 'lino', 'lion', 'list', 'literally', 'littered', 'little', 'lived',
'lives', 'living', 'loads', 'local', 'location', 'locations', 'loewenhielm',
'logic', 'london', 'loneliness', 'long', 'longer', 'look', 'looked', 'looking',
'looks', 'loose', 'loosely', 'lord', 'los', 'losing', 'lost', 'lot', 'lots',
'lousy', 'lovable', 'love', 'loved', 'lovely', 'loves', 'low', 'lower',
'loyalty', 'lucio', 'lucy', 'lugosi', 'lust', 'luv', 'lyrics', 'macbeth',
'machine', 'mad', 'made', 'magnificent', 'main', 'mainly', 'major', 'make',
'maker', 'makers', 'makes', 'making', 'male', 'males', 'malta', 'man',
'managed', 'manages', 'manna', 'mansonites', 'many', 'marbles', 'march',
'marine', 'marion', 'mark', 'marred', 'marriage', 'martin', 'masculine',
'masculinity', 'massive', 'master', 'masterful', 'masterpiece', 'masterpieces',
'material', 'matrix', 'matter', 'matthews', 'mature', 'may', 'maybe',
'mchattie', 'mclaglen', 'meagre', 'mean', 'meanders', 'meaning', 'meanings',
'meant', 'medical', 'mediocre', 'meld', 'melodrama', 'melville', 'member',
'members', 'memorable', 'memories', 'memorized', 'menace', 'menacing',
'mention', 'mercy', 'meredith', 'merit', 'mesmerising', 'mess', 'messages',
'meteorite', 'mexican', 'michael', 'mickey', 'microsoft', 'middle', 'might',
'mighty', 'mind', 'mindblowing', 'miner', 'mini', 'minor', 'minute', 'minutes',
'mirrormask', 'miserable', 'miserably', 'mishima', 'misplace', 'miss', 'missed',
'mistakes', 'miyazaki', 'modern', 'modest', 'mollusk', 'moment', 'moments',
'momentum', 'money', 'monica', 'monolog', 'monotonous', 'monster', 'monstrous',
'monumental', 'moral', 'morgan', 'morons', 'mostly', 'mother', 'motion',
'motivations', 'mountain', 'mouse', 'mouth', 'move', 'moved', 'movements',
'moves', 'movie', 'movies', 'moving', 'ms', 'much', 'muddled', 'muppets',
'murder', 'murdered', 'murdering', 'murky', 'music', 'musician', 'must',
'mystifying', 'naked', 'narration', 'narrative', 'nasty', 'national',
'nationalities', 'native', 'natural', 'nature', 'naughty', 'nearly', 'necklace',
'need', 'needed', 'needlessly', 'negative', 'negulesco', 'neighbour', 'neil',
'nerves', 'nervous', 'net', 'netflix', 'network', 'never', 'nevertheless',
'nevsky', 'new', 'next', 'nice', 'nicola', 'night', 'nimoy', 'nine', 'no',
'noble', 'nobody', 'noir', 'non', 'none', 'nonetheless', 'nonsense', 'nor',
'normally', 'northern', 'nostalgia', 'not', 'notable', 'notch', 'note',
'noteworthy', 'nothing', 'novella', 'number', 'numbers', 'nun', 'nuns', 'nurse',
'nut', 'nuts', 'obliged', 'obsessed', 'obvious', 'obviously', 'occasionally',

'occupied', 'occur', 'occurs', 'odd', 'offend', 'offensive', 'offer', 'offers',
'often', 'oh', 'okay', 'old', 'olde', 'older', 'ole', 'olivia', 'omit', 'one',
'ones', 'open', 'opened', 'opening', 'operas', 'opinion', 'ordeal', 'oriented',
'original', 'originality', 'origins', 'ortolani', 'oscar', 'others',
'otherwise', 'ought', 'outlandish', 'outlets', 'outside', 'outward',
'overacting', 'overall', 'overcome', 'overdue', 'overly', 'overs', 'overt',
'overwrought', 'owed', 'owls', 'owned', 'owns', 'oy', 'pace', 'paced', 'pacing',
'pack', 'paid', 'painful', 'painfully', 'paint', 'painted', 'pair', 'palance',
'pandering', 'pans', 'paolo', 'pap', 'paper', 'par', 'parents', 'park',
'parker', 'part', 'partaking', 'particular', 'particularly', 'parts', 'passed',
'passion', 'past', 'patent', 'pathetic', 'patriotism', 'paul', 'pay', 'peaking',
'pearls', 'peculiarity', 'pedestal', 'pencil', 'people', 'perabo', 'perfect',
'perfected', 'perfectly', 'performance', 'performances', 'perhaps', 'period',
'perplexing', 'person', 'personalities', 'personally', 'peter', 'pg',
'phantasm', 'phenomenal', 'philippa', 'phony', 'photograph', 'photography',
'phrase', 'physical', 'pi', 'picked', 'picture', 'pictures', 'piece', 'pieces',
'pile', 'pillow', 'pitch', 'pitiful', 'pixar', 'place', 'places', 'plain',
'plane', 'planned', 'plants', 'play', 'played', 'player', 'players', 'playing',
'plays', 'pleasant', 'pleased', 'pleaser', 'pleasing', 'pledge', 'plenty',
'plmer', 'plot', 'plug', 'plus', 'pm', 'poet', 'poetry', 'poignant', 'point',
'pointillistic', 'pointless', 'poised', 'poler', 'political', 'politically',
'politics', 'ponyo', 'poor', 'poorly', 'popcorn', 'popular', 'portrayal',
'portrayals', 'portrayed', 'portraying', 'positive', 'possible', 'possibly',
'post', 'potted', 'power', 'powerful', 'powerhouse', 'practical', 'practically',
'pray', 'precisely', 'predict', 'predictable', 'predictably', 'prejudice',
'prelude', 'premise', 'prepared', 'presence', 'presents', 'preservation',
'president', 'pretentious', 'pretext', 'pretty', 'previous', 'primal',
'primary', 'probably', 'problem', 'problems', 'proceedings', 'process',
'produce', 'produced', 'producer', 'producers', 'product', 'production',
'professionals', 'professor', 'progresses', 'promote', 'prompted', 'prone',
'propaganda', 'properly', 'proud', 'proudly', 'provided', 'provokes',
'provoking', 'ps', 'pseudo', 'psychological', 'psychotic', 'public', 'pull',
'pulling', 'pulls', 'punched', 'punches', 'punish', 'punishment', 'puppet',
'puppets', 'pure', 'purity', 'put', 'putting', 'puzzle', 'pyromaniac', 'qu',
'quaid', 'qualities', 'quality', 'question', 'questioning', 'quick', 'quicker',
'quiet', 'quinn', 'quite', 'race', 'racial', 'racism', 'radiant', 'raging',
'random', 'range', 'ranks', 'rare', 'rate', 'rated', 'rather', 'rating',
'ratings', 'raver', 'raw', 'ray', 'reactions', 'readers', 'reading', 'ready',
'real', 'realised', 'realistic', 'reality', 'realize', 'realized', 'really',
'reason', 'reasonable', 'reasons', 'receive', 'received', 'recent', 'recently',
'recommend', 'recommended', 'reconciliation', 'recover', 'recurring',
'redeemed', 'redeeming', 'reenactments', 'references', 'reflected',
'refreshing', 'regardless', 'regret', 'regrettable', 'regrettably', 'rejection',
'relate', 'related', 'relation', 'relations', 'relationship', 'relationships',
'relatively', 'relaxing', 'release', 'released', 'relief', 'relying',
'remaining', 'remake', 'remarkable', 'remember', 'reminded', 'remotely',
'removing', 'rendering', 'rendition', 'renowned', 'rent', 'repair', 'repeated',
'repeating', 'repeats', 'repertory', 'reporter', 'represents', 'require',

'rescue', 'researched', 'resounding', 'respecting', 'rest', 'restrained',
'result', 'results', 'resume', 'retarded', 'retreat', 'return', 'revealing',
'revenge', 'revere', 'reverse', 'review', 'reviewer', 'reviewers', 'reviews',
'rice', 'rickman', 'ridiculous', 'ridiculousness', 'right', 'riot', 'rips',
'rise', 'rita', 'rivalry', 'riveted', 'riz', 'road', 'robert', 'robotic',
'rochon', 'rocked', 'rocks', 'roeg', 'role', 'roles', 'roller', 'rolls',
'romantic', 'room', 'roosevelt', 'roth', 'rough', 'round', 'routine', 'row',
'rpg', 'rpger', 'rubbish', 'rubin', 'rumbles', 'run', 'running', 'ruthless',
'ryan', 'ryans', 'sabotages', 'sack', 'sacrifice', 'sad', 'said', 'sake',
'salesman', 'sam', 'sample', 'sand', 'sandra', 'sappiest', 'sarcophage', 'sat',
'satanic', 'savalas', 'savant', 'save', 'savor', 'saw', 'say', 'says', 'scale',
'scamp', 'scare', 'scared', 'scares', 'scary', 'scene', 'scenery', 'scenes',
'schilling', 'schizophrenic', 'school', 'schoolers', 'schrader', 'schultz',
'sci', 'science', 'scientist', 'score', 'scot', 'scream', 'screamy', 'screen',
'screened', 'screenplay', 'screenwriter', 'scrimm', 'script', 'scripted',
'scripting', 'scripts', 'sculpture', 'sea', 'seamless', 'seamlessly', 'sean',
'season', 'seat', 'second', 'secondary', 'secondly', 'see', 'seeing', 'seem',
'seemed', 'seems', 'seen', 'selections', 'self', 'sells', 'semi', 'senior',
'sense', 'senses', 'sensibility', 'sensitivities', 'sentiment', 'seperate',
'sequel', 'sequels', 'sequence', 'sequences', 'series', 'serious', 'seriously',
'served', 'set', 'sets', 'setting', 'settings', 'seuss', 'several', 'sex',
'shakespear', 'shakespears', 'shallow', 'shame', 'shameful', 'share', 'sharing',
'sharply', 'shatner', 'shattered', 'shed', 'sheer', 'shelf', 'shell', 'shelves',
'shenanigans', 'shepard', 'shined', 'shirley', 'shocking', 'shooting', 'short',
'shortlist', 'shot', 'shots', 'show', 'showcasing', 'showed', 'shows', 'shut',
'sibling', 'sick', 'side', 'sidelined', 'sign', 'significant', 'silent',
'silly', 'simmering', 'simplifying', 'simply', 'since', 'sincere', 'sing',
'singing', 'single', 'sinister', 'sink', 'sinking', 'sister', 'sisters', 'sit',
'sitcoms', 'site', 'sites', 'sits', 'situation', 'situations', 'skilled',
'skip', 'slackers', 'slavic', 'sleep', 'slideshow', 'slightest', 'slightly',
'slimy', 'sloppy', 'slow', 'slurs', 'smack', 'small', 'smart', 'smells',
'smile', 'smiling', 'smith', 'smoothly', 'snider', 'snow', 'soap', 'sobering',
'social', 'soldiers', 'sole', 'solid', 'solidifying', 'solving', 'someone',
'something', 'sometimes', 'somewhat', 'son', 'song', 'songs', 'soon',
'sophisticated', 'sorrentino', 'sorry', 'sort', 'soul', 'sound', 'sounded',
'sounds', 'soundtrack', 'sour', 'south', 'southern', 'space', 'spacek',
'spacey', 'span', 'speak', 'speaking', 'special', 'speed', 'spend', 'spent',
'spew', 'sphere', 'spiffy', 'splendid', 'spock', 'spoil', 'spoiled', 'spoiler',
'spoilers', 'spot', 'spy', 'squibs', 'stable', 'stage', 'stagy', 'stand',
'standout', 'stanwyck', 'star', 'starlet', 'starring', 'stars', 'start',
'started', 'starts', 'state', 'stay', 'stayed', 'stealing', 'steamboat',
'steele', 'step', 'stephen', 'stereotypes', 'stereotypically', 'steve',
'stewart', 'stick', 'still', 'stinker', 'stinks', 'stocking', 'stockings',
'stoic', 'store', 'stories', 'storm', 'story', 'storyline', 'storytelling',
'stowe', 'strange', 'stranger', 'stratus', 'straw', 'street', 'strident',
'string', 'strives', 'strokes', 'strong', 'struck', 'structure', 'struggle',
'stuart', 'student', 'students', 'studio', 'study', 'stuff', 'stunning',
'stupid', 'stupidity', 'style', 'stylized', 'sub', 'subject', 'subjects',

'sublime', 'sublimely', 'subplots', 'subtitles', 'subtle', 'subversive',
'subverting', 'succeeded', 'succeeds', 'success', 'suck', 'sucked', 'sucks',
'suffered', 'suffering', 'suggest', 'suggests', 'suited', 'sum', 'summary',
'sundays', 'super', 'superb', 'superbad', 'superbly', 'superficial',
'superlative', 'supernatural', 'supporting', 'supposed', 'supposedly', 'sure',
'surely', 'surf', 'surface', 'surprised', 'surprises', 'surprising',
'surprisingly', 'surrounding', 'surroundings', 'survivors', 'suspense',
'suspension', 'sven', 'swamp', 'sweep', 'sweet', 'switched', 'swords', 'sydney',
'sympathetic', 'syrupy', 'system', 'tacky', 'taelons', 'take', 'taken', 'takes',
'taking', 'tale', 'talent', 'talented', 'talents', 'talk', 'tanks', 'taped',
'tardis', 'task', 'taste', 'taxidermists', 'taylor', 'teacher', 'teaches',
'team', 'tear', 'tears', 'technically', 'teddy', 'tedium', 'teen', 'teenagers',
'teeth', 'telephone', 'television', 'tell', 'telly', 'temperaments', 'ten',
'tender', 'tension', 'tensions', 'terminology', 'terms', 'terrible', 'terribly',
'terrific', 'terror', 'th', 'thanks', 'theater', 'theatre', 'theatres',
'theatrical', 'theme', 'themes', 'therapy', 'thick', 'thing', 'things', 'think',
'thinking', 'thomerson', 'thoroughly', 'thorsen', 'though', 'thought',
'thoughts', 'thousand', 'thread', 'three', 'threshold', 'thrilled', 'thriller',
'thrillers', 'throughout', 'throwback', 'thrown', 'thug', 'thumper',
'thunderbirds', 'thus', 'ticker', 'tickets', 'tightly', 'time', 'timeless',
'timely', 'timers', 'times', 'timing', 'tiny', 'tired', 'title', 'titta',
'today', 'together', 'told', 'tolerable', 'tolerate', 'tom', 'tomorrow', 'tone',
'tongue', 'tonight', 'tons', 'tony', 'took', 'toons', 'top', 'tops', 'torture',
'tortured', 'total', 'totally', 'touch', 'touches', 'touching', 'tough',
'towards', 'towers', 'townsend', 'track', 'tract', 'traditional', 'traffic',
'trailer', 'train', 'tranquillity', 'transcend', 'transfers', 'translate',
'translating', 'trap', 'trash', 'trashy', 'treachery', 'treasure', 'treat',
'treatments', 'trek', 'tremendous', 'tremendously', 'tries', 'trilogy',
'trinity', 'trip', 'triumphed', 'trond', 'trooper', 'trouble', 'truck', 'true',
'truly', 'trumbull', 'trumpeter', 'truth', 'try', 'trying', 'trysts', 'tsunami',
'tuneful', 'turkey', 'turn', 'turned', 'turns', 'tv', 'twice', 'twirling',
'twist', 'twists', 'two', 'tying', 'type', 'typical', 'ue', 'ugliest', 'ugly',
'uhura', 'ultra', 'um', 'unaccompanied', 'unbearable', 'unbearably',
'unbelievable', 'uncalled', 'unconditional', 'unconvincing', 'underacting',
'underappreciated', 'underbite', 'underlines', 'underlying', 'underneath',
'underrated', 'understand', 'understanding', 'understated', 'understatement',
'understood', 'undertone', 'underwater', 'undoubtedly', 'uneasy', 'unemployed',
'unethical', 'unfaithful', 'unfolds', 'unforgettable', 'unfortunate',
'unfortunately', 'unfunny', 'unintentionally', 'uninteresting', 'union',
'unique', 'uniqueness', 'universal', 'universe', 'unless', 'unlockable',
'unmatched', 'unmitigated', 'unmoving', 'unnecessary', 'unneeded', 'unoriginal',
'unpleasant', 'unpredictability', 'unpredictable', 'unrealistic',
'unrecognizable', 'unrecommended', 'unremarkable', 'unrestrained',
'unsatisfactory', 'unwatchable', 'upa', 'uplifting', 'upper', 'ups', 'uptight',
'ursula', 'us', 'use', 'used', 'user', 'uses', 'using', 'ussr', 'usual',
'utter', 'utterly', 'valentine', 'value', 'values', 'vampire', 'vandiver',
'variation', 'vehicles', 'ventura', 'verbal', 'verbatim', 'versatile',
'version', 'versus', 'vessel', 'veteran', 'vey', 'vibe', 'victor', 'video',

```
'view', 'viewer', 'viewing', 'views', 'villain', 'villains', 'violence',
'violin', 'virtue', 'virus', 'vision', 'visual', 'visually', 'vitally',
'vivian', 'vivid', 'vocal', 'voice', 'volatile', 'volcano', 'vomit', 'vomited',
'voyage', 'vulcan', 'waiting', 'waitress', 'walk', 'walked', 'wall', 'want',
'wanted', 'wanting', 'wants', 'war', 'warmth', 'warn', 'warning', 'wartime',
'warts', 'washed', 'washing', 'waste', 'wasted', 'waster', 'wasting', 'watch',
'watchable', 'watched', 'watching', 'water', 'watkins', 'watson', 'wave', 'way',
'waylaid', 'wayne', 'ways', 'wb', 'weak', 'weaker', 'weariness', 'weaving',
'website', 'wedding', 'weight', 'weird', 'well', 'welsh', 'went', 'whatever',
'whatsoever', 'whenever', 'whether', 'whine', 'whiny', 'white', 'whites',
'whoever', 'whole', 'wholesome', 'wide', 'widmark', 'wife', 'wih', 'wild',
'wilkinson', 'william', 'willie', 'wily', 'win', 'wind', 'wise', 'wish',
'within', 'without', 'witticisms', 'witty', 'woa', 'women', 'wonder',
'wondered', 'wonderful', 'wonderfully', 'wong', 'wont', 'woo', 'wooden', 'word',
'words', 'work', 'worked', 'working', 'works', 'world', 'worry', 'worse',
'worst', 'worth', 'worthless', 'worthwhile', 'worthy', 'would', 'wouldnt',
'woven', 'wow', 'wrap', 'write', 'writer', 'writers', 'writing', 'written',
'wrong', 'wrote', 'yardley', 'yawn', 'yeah', 'year', 'years', 'yelps', 'yes',
'yet', 'young', 'younger', 'youthful', 'youtube', 'yun', 'zillion', 'zombie',
'zombiez']
```

1.2.3 Method to get top 50 idf value

[115]:
```
# Formula for IDF:
# IDF of term t = 1 + log( (1 + Total Number of Documents in the Collection) /␣
↪(1 + Number of Documents with term t in it) )
def get_top50_idf(dataset,vocab):
    vocab_idf = {} # Empty dict inititated to store all idf values present in␣
↪the vocab
    top_50_vocab_idf = {} # Empty dict inititate to store 50 words with max idf␣
↪value
    if isinstance(dataset,(list,)):
        total_doc = len(dataset)  #Total number of Documents in dataset

        #The following calculated idf value and store in vocab_idf. vocab_idf␣
↪is the dict of all words present in vocab obtained by fit method
        for word in list(vocab.keys()):
            doc_with_word = 0
            for doc in dataset:
                if word in doc:
                    doc_with_word += 1 #Total number of Document with word in it
            vocab_idf[word] = (1 + math.log( (1 + total_doc) / (1 +␣
↪doc_with_word) ) ) #Calculation of IDF for each word and storing in the dict

        #Sorting vocab_idf dict by value in descending order and storing it in␣
↪vocab_idf_list as a list of tuples. Reference: https://www.w3resource.com/
↪python-exercises/dictionary/python-data-type-dictionary-exercise-1.php
```

```
        vocab_idf_list = sorted(vocab_idf.items(),key = operator.
↪itemgetter(1),reverse=True)

        #The following loop filters the top 50 tuples of the reverse order  and␣
↪store in the top_50_vocab_idf dict
        count = 0
        for item in vocab_idf_list:
            if count<50 :
               key = item[0]
               value = item[1]
               top_50_vocab_idf[key] = value
               count += 1

        return top_50_vocab_idf
    else:
        print('Please pass list of sentence.')
```

1.2.4 Implementation of get_top50_idf method to print a dict of top 50 idf key and value pair

[116]:
```
vocab_idf_50 = get_top50_idf(corpus,vocab)
print(vocab_idf_50)
```

```
{'aailiyah': 6.922918004572872, 'abandoned': 6.922918004572872, 'abroad':
6.922918004572872, 'abstruse': 6.922918004572872, 'academy': 6.922918004572872,
'accents': 6.922918004572872, 'accessible': 6.922918004572872, 'acclaimed':
6.922918004572872, 'accolades': 6.922918004572872, 'accurately':
6.922918004572872, 'achille': 6.922918004572872, 'ackerman': 6.922918004572872,
'adams': 6.922918004572872, 'added': 6.922918004572872, 'admins':
6.922918004572872, 'admiration': 6.922918004572872, 'admitted':
6.922918004572872, 'adrift': 6.922918004572872, 'adventure': 6.922918004572872,
'aesthetically': 6.922918004572872, 'affected': 6.922918004572872, 'affleck':
6.922918004572872, 'afternoon': 6.922918004572872, 'agreed': 6.922918004572872,
'aimless': 6.922918004572872, 'aired': 6.922918004572872, 'akasha':
6.922918004572872, 'alert': 6.922918004572872, 'alike': 6.922918004572872,
'allison': 6.922918004572872, 'allowing': 6.922918004572872, 'alongside':
6.922918004572872, 'amateurish': 6.922918004572872, 'amazed': 6.922918004572872,
'amazingly': 6.922918004572872, 'amusing': 6.922918004572872, 'amust':
6.922918004572872, 'anatomist': 6.922918004572872, 'angela': 6.922918004572872,
'angelina': 6.922918004572872, 'angry': 6.922918004572872, 'anguish':
6.922918004572872, 'angus': 6.922918004572872, 'animals': 6.922918004572872,
'animated': 6.922918004572872, 'anita': 6.922918004572872, 'anniversary':
6.922918004572872, 'anthony': 6.922918004572872, 'antithesis':
6.922918004572872, 'anyway': 6.922918004572872}
```

1.2.5 Creating Method to get vocabulary of unique words having top 50 idf value

[117]:
```
#Method to get 50 unique words in vocab having top idf values
def get_top50idfvocab(dataset,vocab):
    vocab_idf = {}
    vocab_list = []
    if isinstance(dataset,(list,)):
        total_doc = len(dataset)  #Total number of Documents in dataset
        for word in list(vocab.keys()):
            doc_with_word = 0
            for doc in dataset:
                if word in doc:
                    doc_with_word += 1 #Total number of Document with word in it
            vocab_idf[word] = (1 + math.log( (1 + total_doc) / (1 +␣
↪doc_with_word) ) ) #Calculation of IDF for each word and storing in the dict
        vocab_idf = dict(sorted(vocab_idf.items(),key = operator.
↪itemgetter(1),reverse=True)) #Sorting dict by valye
        #Reference: https://www.w3resource.com/python-exercises/dictionary/
↪python-data-type-dictionary-exercise-1.php
        count = 0
        for item in list(vocab_idf.keys()):
            if(count < 50):
                vocab_list.append(item)
                count += 1
        vocab = {j:i for i,j in enumerate(vocab_list)}
        return vocab
    else:
        print('Please pass list of sentence.')
```

1.2.6 Implementatio of get_top50idfvocab method to print 50 unique words in vocab having max idf values

[118]:
```
vocab = get_top50idfvocab(corpus,vocab)
print(vocab)
```

```
{'aailiyah': 0, 'abandoned': 1, 'abroad': 2, 'abstruse': 3, 'academy': 4,
'accents': 5, 'accessible': 6, 'acclaimed': 7, 'accolades': 8, 'accurately': 9,
'achille': 10, 'ackerman': 11, 'adams': 12, 'added': 13, 'admins': 14,
'admiration': 15, 'admitted': 16, 'adrift': 17, 'adventure': 18,
'aesthetically': 19, 'affected': 20, 'affleck': 21, 'afternoon': 22, 'agreed':
23, 'aimless': 24, 'aired': 25, 'akasha': 26, 'alert': 27, 'alike': 28,
'allison': 29, 'allowing': 30, 'alongside': 31, 'amateurish': 32, 'amazed': 33,
'amazingly': 34, 'amusing': 35, 'amust': 36, 'anatomist': 37, 'angela': 38,
'angelina': 39, 'angry': 40, 'anguish': 41, 'angus': 42, 'animals': 43,
'animated': 44, 'anita': 45, 'anniversary': 46, 'anthony': 47, 'antithesis': 48,
'anyway': 49}
```

1.2.7 Writing transform method to get the sparse matrix from the vocabulary having unique words with top 50 idf values

[119]:
```
#Formula for TF:
#TF of term t = (Number of times t appeared in the document)/(Total number of␣
↪documents in the collection)
#Tranform method for custom tfidf vectortizer
def transform(dataset,vocab):
    rows = []
    columns = []
    values = []
    if isinstance(dataset,(list,)):
        vocab_idf = get_top50_idf(dataset,vocab) #Gets a dict having top 50 idf␣
↪values
        total_doc = len(corpus) #gets the total number of documents present in␣
↪the dataset
        for idx,row in enumerate(dataset):
            word_freq = dict(Counter(row.split(" ")))
            for word,frequency in word_freq.items():
                if ((len(word) > 1) and (word in list(vocab.keys()))):
                    column_index = vocab.get(word, -1)
                    if (column_index != -1):
                        rows.append(idx)
                        columns.append(column_index)
                        #Calculating TF of word
                        word_tf = frequency/total_doc
                        #Calculating TF-IDF valye of word
                        tf_idf_value = (word_tf * vocab_idf[word])
                        values.append(tf_idf_value)
        sparse_matrix = csr_matrix((values, (rows, columns)),␣
↪shape=(len(dataset),len(vocab)))
        sparse_matrix_normalized = normalize(sparse_matrix, norm='l2')
        return sparse_matrix_normalized
    else:
        print('Please pass list of sentence.')
```

1.2.8 Implementing transform method to finally get the sparse matrix of the pickle file with vocab limited to 50 unique words having top 50 idf values

[126]:
```
max_vectorizer = transform(corpus,vocab)
print('Shape of sparse matrix: ',max_vectorizer.shape)

print('')
print('Document: ',corpus[0])
print('Vector of Document: \n',max_vectorizer[0].toarray())
print('Shape: ',max_vectorizer[0].shape)
```

```
print('')
print('Document: ',corpus[461])
print('Vector of Document: \n',max_vectorizer[461].toarray())
print('Shape: ',max_vectorizer[461].shape)
```

```
Shape of sparse matrix:  (746, 50)

Document:  slow moving aimless movie distressed drifting young man
Vector of Document:
 [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0.]]
Shape:  (1, 50)

Document:  witty delightful adaptation dr seuss book brilliantly animated upa
finest thoroughly deserving academy award
Vector of Document:
 [[0.         0.         0.         0.         0.70710678 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.70710678 0.         0.         0.
  0.         0.        ]]
Shape:  (1, 50)
```