IST-736 Text Mining
Homework-2&3

Sharat Sripada (vssripad@syr.edu)

Introduction

This week combines homework from week-2 and week-3 and involves the following topics:

- Word tokenization
- Stemming and Lemmatization
- Vectorization
- K-means clustering for text classification

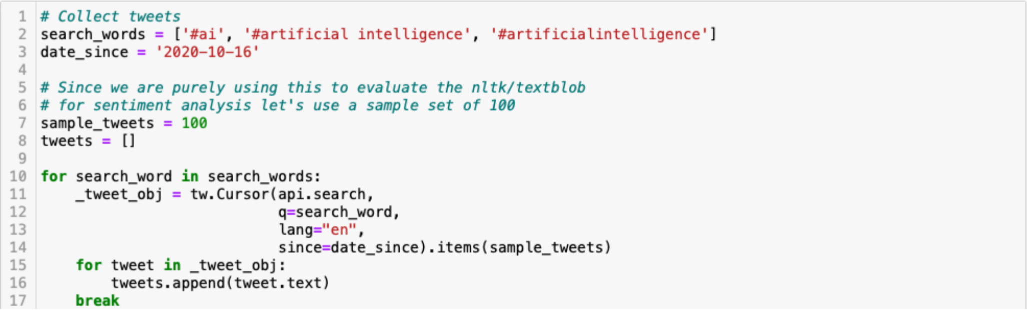
The homework will extend the usage of the dataset used in week-1 from Twitter on topic AI/artificial intelligence for most part of the tasks and use an additional Kaggle dataset related to review of a Pink Floyd album for the clustering or classification exercise.

Data source and collection

Data or tweets would be gathered from Twitter using the Python tweepy package/library. The code shows having to use consumer_key, consumer_secret, an access_token and access_token_secret to setup an OAuthHandler and api handle (row 13).

We would continue to limit (via tweepy api knobs) to one hundred most recent tweets at the time/date of writing this document.

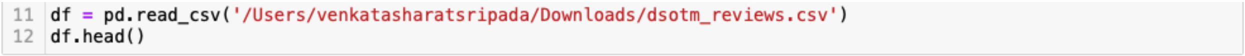
Using filters like search string, date, and number of tweets to retrieve, a sample of tweets tagged with word *#ai* is stored in a python list *tweets = []*:



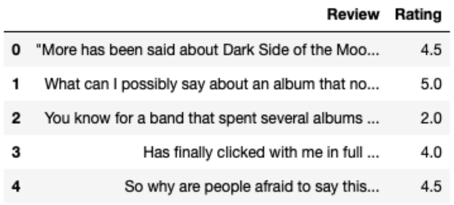
```
 1 # Collect tweets
 2 search_words = ['#ai', '#artificial intelligence', '#artificialintelligence']
 3 date_since = '2020-10-16'
 4 
 5 # Since we are purely using this to evaluate the nltk/textblob
 6 # for sentiment analysis let's use a sample set of 100
 7 sample_tweets = 100
 8 tweets = []
 9 
10 for search_word in search_words:
11     _tweet_obj = tw.Cursor(api.search,
12                            q=search_word,
13                            lang="en",
14                            since=date_since).items(sample_tweets)
15     for tweet in _tweet_obj:
16         tweets.append(tweet.text)
17     break
```

Further, for the clustering or classification exercise we will obtain the dataset or csv from the following URL: https://www.kaggle.com/michaelbryantds/reviews-of-pink-floyds-the-dark-side-of-the-moon?select=dsotm_reviews.csv

A snapshot of the dataset in its raw form is shown here:



```
df = pd.read_csv('/Users/venkatasharatsripada/Downloads/dsotm_reviews.csv')
df.head()
```

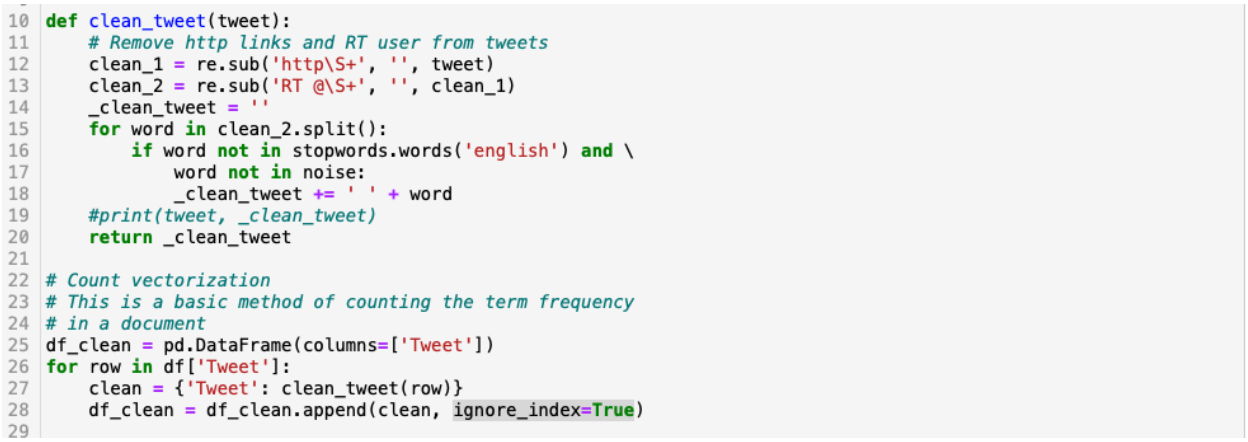


	Review	Rating
0	"More has been said about Dark Side of the Moo...	4.5
1	What can I possibly say about an album that no...	5.0
2	You know for a band that spent several albums ...	2.0
3	Has finally clicked with me in full ...	4.0
4	So why are people afraid to say this...	4.5

Fig: Music album review (csv -> pd dataframe)

Cleanup

As part of the data ingestion, we will clean up individual tweets for stop-words and punctuation marks:



```
def clean_tweet(tweet):
    # Remove http links and RT user from tweets
    clean_1 = re.sub('http\S+', '', tweet)
    clean_2 = re.sub('RT @\S+', '', clean_1)
    _clean_tweet = ''
    for word in clean_2.split():
        if word not in stopwords.words('english') and \
            word not in noise:
            _clean_tweet += ' ' + word
    #print(tweet, _clean_tweet)
    return _clean_tweet

# Count vectorization
# This is a basic method of counting the term frequency
# in a document
df_clean = pd.DataFrame(columns=['Tweet'])
for row in df['Tweet']:
    clean = {'Tweet': clean_tweet(row)}
    df_clean = df_clean.append(clean, ignore_index=True)
```

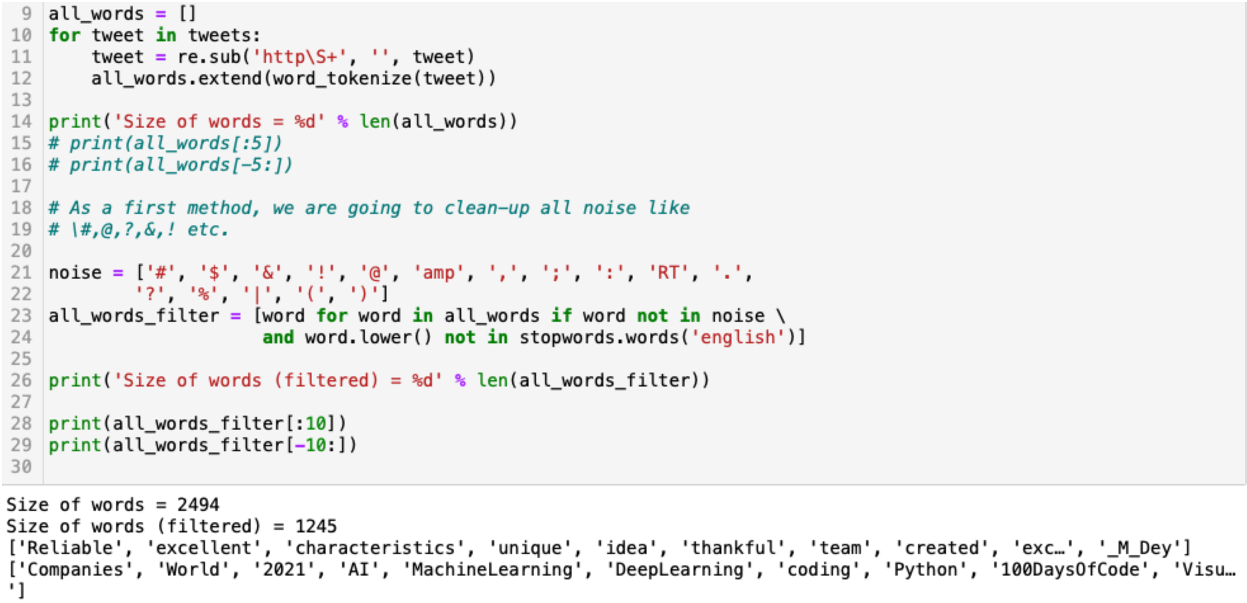
Fig: A custom method to clean tweets

Word Tokenization

In this exercise, we will use the nltk tokenize library and specifically consume the word_tokenize function. The idea is, given text or a sentence we want to tokenize it into individual words. For this we first iterate through the tweets remove punctuation and other meta characters (like @, # etc.) apart from urls beginning with http/https in tweets. These are likely to have any impact for use-cases that may choose to tokenize text.

When removing the http/https url from tweet we will use a regex as show below – essentially parsing for words beginning with http and replacing it with null:

See output below showing a sample of the tokenized words:



```
all_words = []
for tweet in tweets:
    tweet = re.sub('http\S+', '', tweet)
    all_words.extend(word_tokenize(tweet))

print('Size of words = %d' % len(all_words))
# print(all_words[:5])
# print(all_words[-5:])

# As a first method, we are going to clean-up all noise like
# \#,@,?,&,! etc.

noise = ['#', '$', '&', '!', '@', 'amp', ',', ';', ':', 'RT', '.',
         '?', '%', '|', '(', ')']
all_words_filter = [word for word in all_words if word not in noise \
                    and word.lower() not in stopwords.words('english')]

print('Size of words (filtered) = %d' % len(all_words_filter))

print(all_words_filter[:10])
print(all_words_filter[-10:])

```

```
Size of words = 2494
Size of words (filtered) = 1245
['Reliable', 'excellent', 'characteristics', 'unique', 'idea', 'thankful', 'team', 'created', 'exc…', '_M_Dey']
['Companies', 'World', '2021', 'AI', 'MachineLearning', 'DeepLearning', 'coding', 'Python', '100DaysOfCode', 'Visu…
']
```

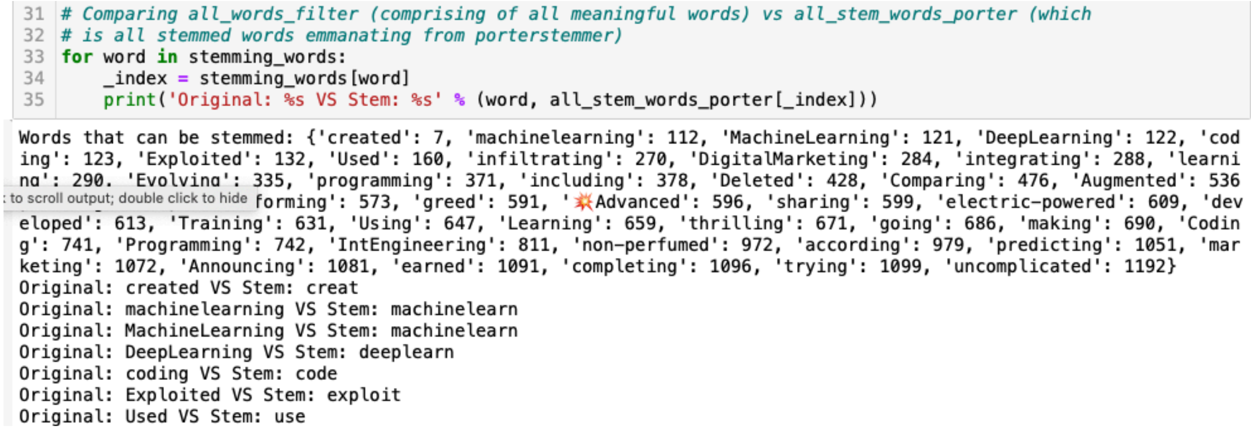
Fig: Tokenized tweets or sentences

Stemming and Lemmatization

Stem is part of the word to which you can add inflectional affixes such as (-ed, -ize, -s, -de). Stemming is the process of removing the inflectional affixes. This is particularly useful in search engines and information retrieval where stemming is used to index words. And therefore, instead of storing all forms of words, the index can comprise merely stem words.

For this homework, we will use PorterStemmer.

To see the working of the stemmer, code was written to find words with affixes like ‘ed’ and ‘ing’ and what the stemmer would translate it into:



```
# Comparing all_words_filter (comprising of all meaningful words) vs all_stem_words_porter (which
# is all stemmed words emmanating from porterstemmer)
for word in stemming_words:
    _index = stemming_words[word]
    print('Original: %s VS Stem: %s' % (word, all_stem_words_porter[_index]))
```

```
Words that can be stemmed: {'created': 7, 'machinelearning': 112, 'MachineLearning': 121, 'DeepLearning': 122, 'cod
ing': 123, 'Exploited': 132, 'Used': 160, 'infiltrating': 270, 'DigitalMarketing': 284, 'integrating': 288, 'learni
ng': 290, 'Evolving': 335, 'programming': 371, 'including': 378, 'Deleted': 428, 'Comparing': 476, 'Augmented': 536
: to scroll output; double click to hide forming': 573, 'greed': 591, '💥Advanced': 596, 'sharing': 599, 'electric-powered': 609, 'dev
eloped': 613, 'Training': 631, 'Using': 647, 'Learning': 659, 'thrilling': 671, 'going': 686, 'making': 690, 'Codin
g': 741, 'Programming': 742, 'IntEngineering': 811, 'non-perfumed': 972, 'according': 979, 'predicting': 1051, 'mar
keting': 1072, 'Announcing': 1081, 'earned': 1091, 'completing': 1096, 'trying': 1099, 'uncomplicated': 1192}
Original: created VS Stem: creat
Original: machinelearning VS Stem: machinelearn
Original: MachineLearning VS Stem: machinelearn
Original: DeepLearning VS Stem: deeplearn
Original: coding VS Stem: code
Original: Exploited VS Stem: exploit
Original: Used VS Stem: use
```

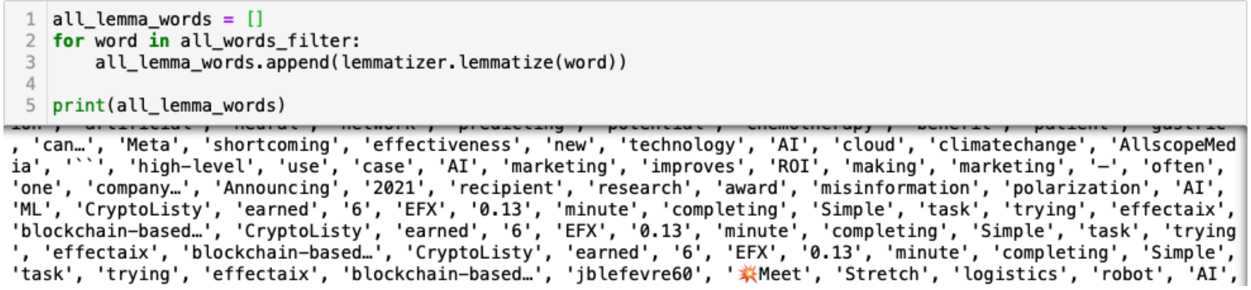
Fig: Illustration of working of PorterStemmer on dataset

Next, Lemmatization works like stemmer and has a similar use-case with search engines and information retrieval systems but particularly looks for meaning of a word. The output we will get after lemmatization is called ‘lemma’ which is a root word rather root stem, the output of stemming.

An example to bring out the difference between the two is as below:
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize('believes'))
print(word_stemmer.stem('believes'))

Output:
Belief <- Related to lemmatization
Believ <- Related to stemmer

Finally, we use the nltk lemmatizer.lemmatize function to iterate through the tokenized words and translate them to lemma.



```
all_lemma_words = []
for word in all_words_filter:
    all_lemma_words.append(lemmatizer.lemmatize(word))

print(all_lemma_words)
```

```
, 'can…', 'Meta', 'shortcoming', 'effectiveness', 'new', 'technology', 'AI', 'cloud', 'climatechange', 'AllscopeMed
ia', '``', 'high-level', 'use', 'case', 'AI', 'marketing', 'improves', 'ROI', 'making', 'marketing', '—', 'often',
'one', 'company…', 'Announcing', '2021', 'recipient', 'research', 'award', 'misinformation', 'polarization', 'AI',
'ML', 'CryptoListy', 'earned', '6', 'EFX', '0.13', 'minute', 'completing', 'Simple', 'task', 'trying', 'effectaix',
'blockchain-based…', 'CryptoListy', 'earned', '6', 'EFX', '0.13', 'minute', 'completing', 'Simple', 'task', 'trying
', 'effectaix', 'blockchain-based…', 'CryptoListy', 'earned', '6', 'EFX', '0.13', 'minute', 'completing', 'Simple',
'task', 'trying', 'effectaix', 'blockchain-based…', 'jblefevre60', '💥Meet', 'Stretch', 'logistics', 'robot', 'AI',
```

Fig: Illustration of working of Lemmatizer on dataset

Vectorization

Machine learning algorithms can operate only on numeric feature space, often expecting input as 2-dimensional array where rows are instances and columns are features. To perform machine learning on text, we need to transform our documents or text into vector representations such that we can apply numeric machine learning. This process is called vectorization and is the first step towards language-aware analysis.

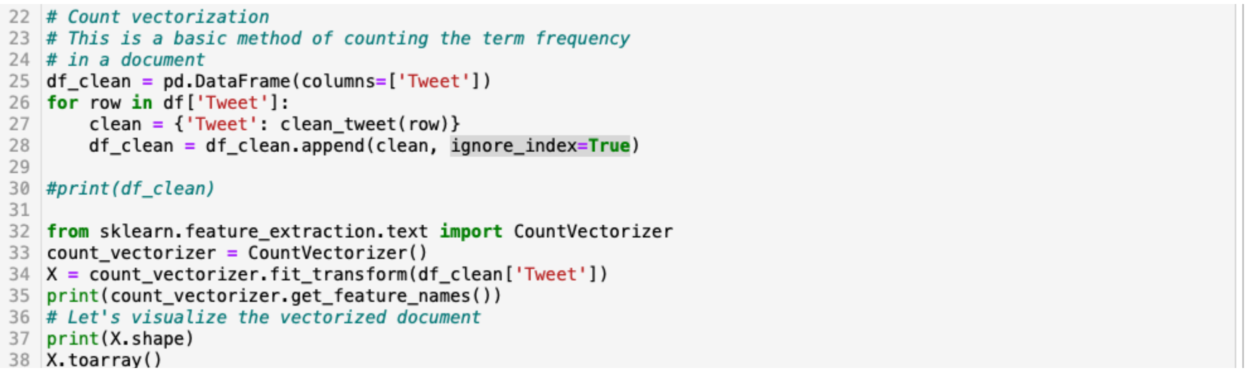
Here we experiment with the following vectorization techniques:

- Count vectorization – Unigram and N-gram/Bi-gram term frequencies
- TF-IDF

For this we will import the *CountVectorizer* or *TfidfVectorizer* from the sklearn.feature_extraction.text library.

Count vectorization

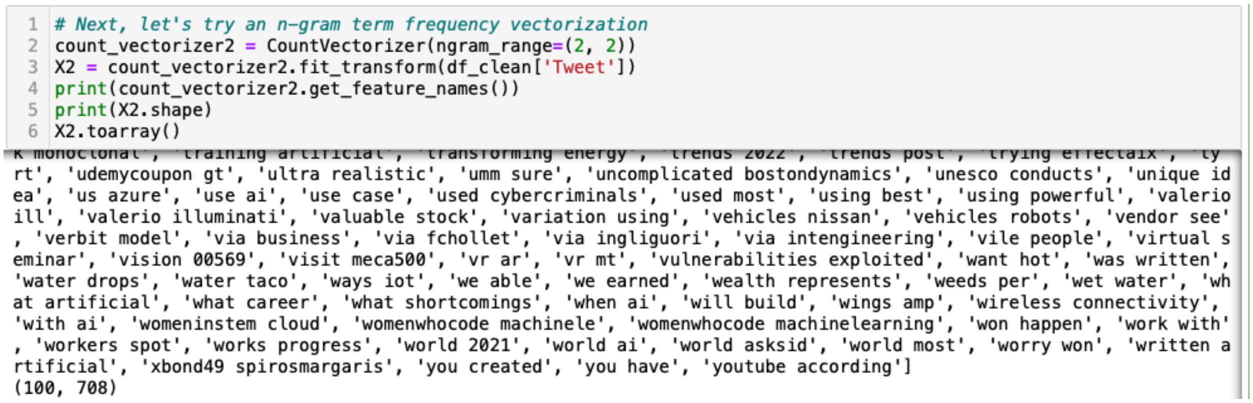
Using the unigram term frequency vectorization, we get *545* features and an overall dimensionality (100, 545).



```
# Count vectorization
# This is a basic method of counting the term frequency
# in a document
df_clean = pd.DataFrame(columns=['Tweet'])
for row in df['Tweet']:
    clean = {'Tweet': clean_tweet(row)}
    df_clean = df_clean.append(clean, ignore_index=True)

#print(df_clean)

from sklearn.feature_extraction.text import CountVectorizer
count_vectorizer = CountVectorizer()
X = count_vectorizer.fit_transform(df_clean['Tweet'])
print(count_vectorizer.get_feature_names())
# Let's visualize the vectorized document
print(X.shape)
X.toarray()
```

Fig: CountVectorizer Unigram setup and results

Likewise with bi-gram or N-gram we get 708 features and overall dimensionality (100, 708).



```
# Next, let's try an n-gram term frequency vectorization
count_vectorizer2 = CountVectorizer(ngram_range=(2, 2))
X2 = count_vectorizer2.fit_transform(df_clean['Tweet'])
print(count_vectorizer2.get_feature_names())
print(X2.shape)
X2.toarray()
```

```
k monoclonal', 'training artificial', 'transforming energy', 'trends 2022', 'trends post', 'trying effectaix', 'ty
rt', 'udemycoupon gt', 'ultra realistic', 'umm sure', 'uncomplicated bostondynamics', 'unesco conducts', 'unique id
ea', 'us azure', 'use ai', 'use case', 'used cybercriminals', 'used most', 'using best', 'using powerful', 'valerio
ill', 'valerio illuminati', 'valuable stock', 'variation using', 'vehicles nissan', 'vehicles robots', 'vendor see'
, 'verbit model', 'via business', 'via fchollet', 'via ingliguori', 'via intengineering', 'vile people', 'virtual s
eminar', 'vision 00569', 'visit meca500', 'vr ar', 'vr mt', 'vulnerabilities exploited', 'want hot', 'was written',
'water drops', 'water taco', 'ways iot', 'we able', 'we earned', 'wealth represents', 'weeds per', 'wet water', 'wh
at artificial', 'what career', 'what shortcomings', 'when ai', 'will build', 'wings amp', 'wireless connectivity',
'with ai', 'womeninstem cloud', 'womenwhocode machinele', 'womenwhocode machinelearning', 'won happen', 'work with'
, 'workers spot', 'works progress', 'world 2021', 'world ai', 'world asksid', 'world most', 'worry won', 'written a
rtificial', 'xbond49 spirosmargaris', 'you created', 'you have', 'youtube according']
(100, 708)
```

Fig: CountVectorizer Ngram setup and results

Using the toarray() function, we the following numpy array (for uni-gram):

```
array([[0, 0, 0, ..., 0, 1, 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]])
```

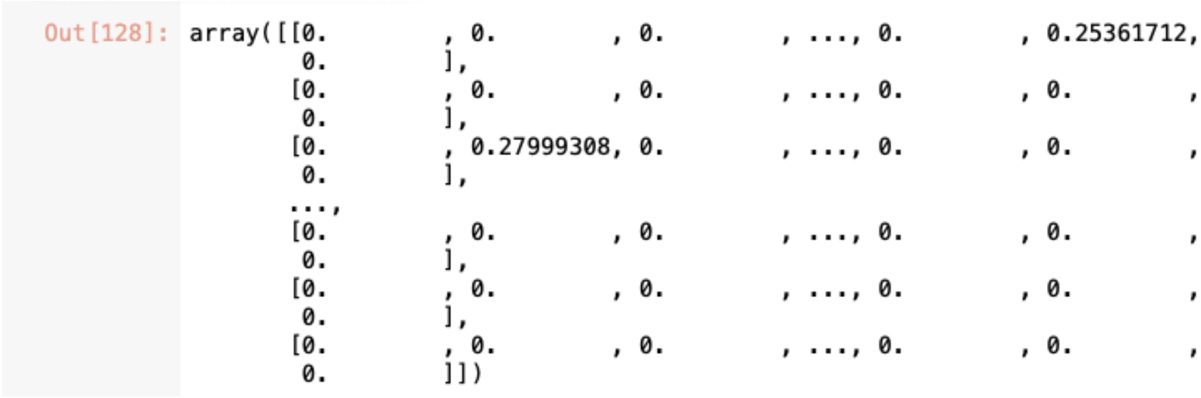
Although this may not be directly useful to interpret, this shows many cells comprise the value zero since words/features would be present in every tweet is therefore known as a Sparse Matrix.

TF-IDF

TF-IDF uses a technique related to the following formula:
W(x,y) = tf(x,y) * log(N/dfx)

where:
W(x,y) = Word x within document y
tf(x,y) = Frequency of x in y
dfx = Number of documents containing x
N = Total number of documents

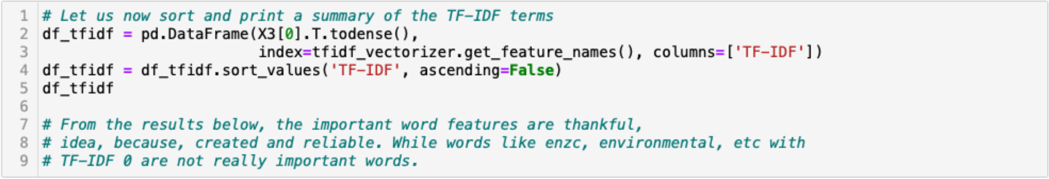
Using the sklearn library for TF-IDF, we obtain the following sparse matrix:



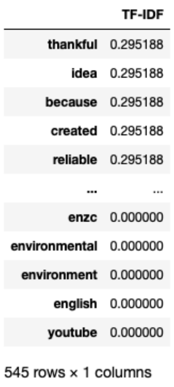
```
Out[128]: array([[0.        , 0.        , 0.        , ..., 0.        , 0.25361712,
                  0.        ],
                 [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                  0.        ],
                 [0.        , 0.27999308, 0.        , ..., 0.        , 0.        ,
                  0.        ],
                 ...,
                 [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                  0.        ],
                 [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                  0.        ],
                 [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                  0.        ]])
```

Fig: TF-IDF sparse matrix output

To visualize this better, we will transpose the numpy array to a single dimensional dataframe and sort in descending order:



```
# Let us now sort and print a summary of the TF-IDF terms
df_tfidf = pd.DataFrame(X3[0].T.todense(),
                        index=tfidf_vectorizer.get_feature_names(), columns=['TF-IDF'])
df_tfidf = df_tfidf.sort_values('TF-IDF', ascending=False)
df_tfidf

# From the results below, the important word features are thankful,
# idea, because, created and reliable. While words like enzc, environmental, etc with
# TF-IDF 0 are not really important words.
```



	TF-IDF
thankful	0.295188
idea	0.295188
because	0.295188
created	0.295188
reliable	0.295188
...	...
enzc	0.000000
environmental	0.000000
environment	0.000000
english	0.000000
youtube	0.000000

545 rows × 1 columns

Fig: Results (sorted) TF-IDF results

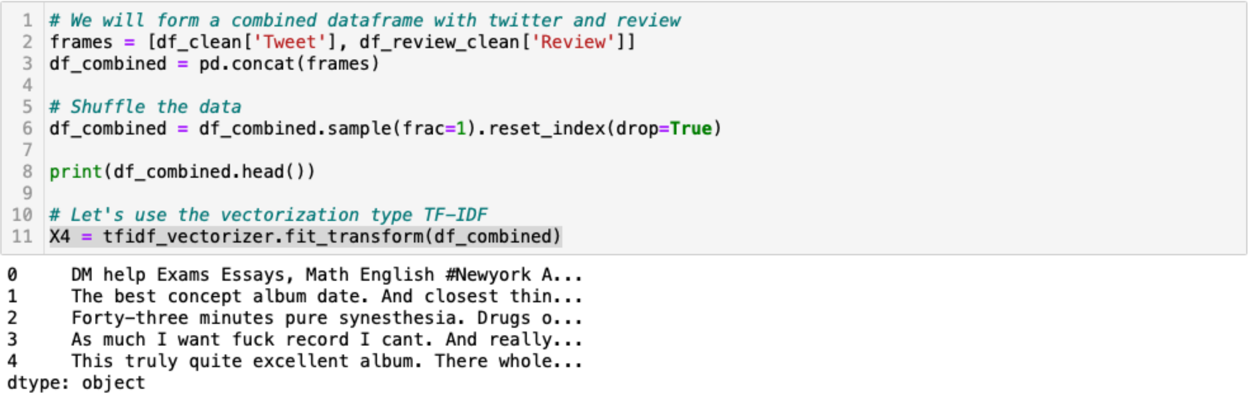
Results show important word features are thankful, idea, because, created and reliable. While words like enzc, environmental, etc. withTF-IDF 0 are not important.

Clustering using k-means

In this section we will explore the use of k-means clustering method to solve a text prediction problem. The sequence of steps is following:

1. Make a combined dataframe (randomized or shuffled) comprising data:
    a. corpus from twitter on topic artificial intelligence
    b. csv from Kaggle with reviews on an album on Pink Floyd
2. Apply the tfidf vectorizer
3. Run the kmeans clustering algorithm

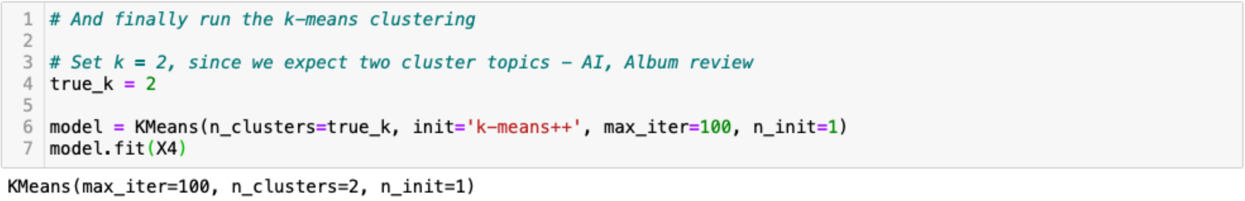
Each of the steps is illustrated through code blocks:



```
# We will form a combined dataframe with twitter and review
frames = [df_clean['Tweet'], df_review_clean['Review']]
df_combined = pd.concat(frames)

# Shuffle the data
df_combined = df_combined.sample(frac=1).reset_index(drop=True)

print(df_combined.head())

# Let's use the vectorization type TF-IDF
X4 = tfidf_vectorizer.fit_transform(df_combined)
```

```
0     DM help Exams Essays, Math English #Newyork A...
1     The best concept album date. And closest thin...
2     Forty-three minutes pure synesthesia. Drugs o...
3     As much I want fuck record I cant. And really...
4     This truly quite excellent album. There whole...
dtype: object
```

Fig: Concat of data-frames (+ shuffling it) and applying vectorization



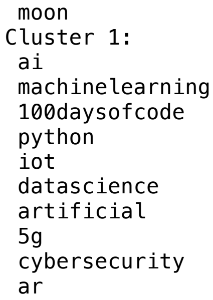
```
# And finally run the k-means clustering

# Set k = 2, since we expect two cluster topics - AI, Album review
true_k = 2

model = KMeans(n_clusters=true_k, init='k-means++', max_iter=100, n_init=1)
model.fit(X4)
```

```
KMeans(max_iter=100, n_clusters=2, n_init=1)
```

Fig: Setting up params for k-means and creating the model

Once the model is built, we will visualize the centroids and the clusters where it belongs (sample **output**):

```
Cluster 0:
 album
 the
 time
 side
 one
 dark
 it
 music
 great
```

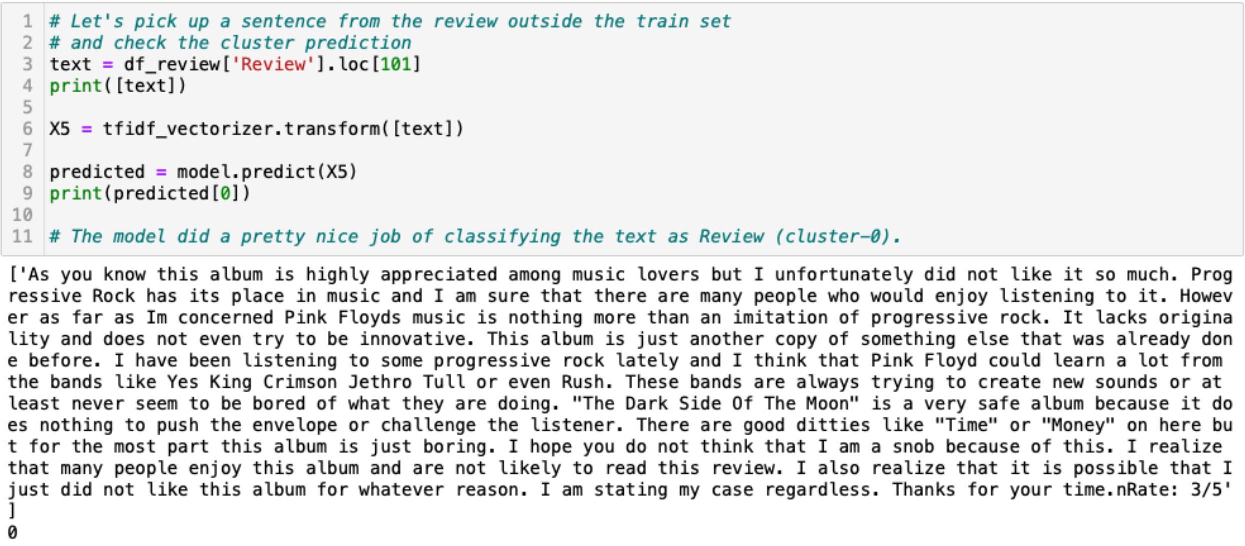


```
 moon
Cluster 1:
 ai
 machinelearning
 100daysofcode
 python
 iot
 datascience
 artificial
 5g
 cybersecurity
 ar
```

NOTE

Cluster-0 is album review and Cluster-1 is tweets related to artificial intelligence. The segregation based on the sample set of words here looks reasonable.

Finally, run the prediction on a review and see the classification result:



```python
# Let's pick up a sentence from the review outside the train set
# and check the cluster prediction
text = df_review['Review'].loc[101]
print([text])

X5 = tfidf_vectorizer.transform([text])

predicted = model.predict(X5)
print(predicted[0])

# The model did a pretty nice job of classifying the text as Review (cluster-0).
```

```
['As you know this album is highly appreciated among music lovers but I unfortunately did not like it so much. Prog
ressive Rock has its place in music and I am sure that there are many people who would enjoy listening to it. Howev
er as far as Im concerned Pink Floyds music is nothing more than an imitation of progressive rock. It lacks origina
lity and does not even try to be innovative. This album is just another copy of something else that was already don
e before. I have been listening to some progressive rock lately and I think that Pink Floyd could learn a lot from
the bands like Yes King Crimson Jethro Tull or even Rush. These bands are always trying to create new sounds or at
least never seem to be bored of what they are doing. "The Dark Side Of The Moon" is a very safe album because it do
es nothing to push the envelope or challenge the listener. There are good ditties like "Time" or "Money" on here bu
t for the most part this album is just boring. I hope you do not think that I am a snob because of this. I realize
that many people enjoy this album and are not likely to read this review. I also realize that it is possible that I
just did not like this album for whatever reason. I am stating my case regardless. Thanks for your time.nRate: 3/5'
]
0
```

Conclusion

In this homework, we explored how text can be tokenized, stemmed, or lemmatized and eventually vectorized to be applied to machine-learning use-cases.

This in summary is everything that happens under the hood in applications like search engines and information retrieval systems.

Specifically, we demonstrate cleaning (removal of stop words, redundant text) in incoming text - concise storage and lookups (stemming and lemmatization) - vectorization to translate text to feature-sets and numeric data which can then be ingested by machine learning algorithms.

Finally, we demonstrate a use-case or application where, given a sentence or text we can correctly classify it to a cluster of documents. This can be thought of as a naïve information retrieval task.