In-Class Assignment: Feature Engineering, Bag of Words

DATA 5420/6420

Name: Dallin Moore

In this in-class assignment we will explore the more traditional route of textual feature engineering, a family of vector space representation models called *bag of words*. These models ignore sentence structure, word order, and word meaning and instead produce vector representations of words based on their frequencies within and amongst documents.

We will utilize a toy corpus with 'unknown' category labels, and utilize various bag of word models in combination with various unsupervised text mining techniques to try and ascertain appropriate labels for each text in the corpus.

Let's begin by importing our required libraries and packages:

```
import re                                                               # import regex
import nltk                                                             # import nltk

import pandas as pd                                                     # import pandas
import numpy as np                                                      # import numpy
import matplotlib.pyplot as plt                                         # import matplotlib
%matplotlib inline

from sklearn.feature_extraction.text import CountVectorizer             # import CountVectorizer and TfidfVectorizer from sklearn.feature_ex
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity                  # import cosine_similarity from sklearn.metrics.pairwise
from sklearn.decomposition import LatentDirichletAllocation             # import LatentDirichlectAllocation from sklearn.decomposition

from scipy.cluster.hierarchy import fcluster, dendrogram, linkage       # import fcluster, dendrogram, and linkage from scipcy.cluster.hiera

pd.options.display.max_colwidth = 200
```

```
nltk.download("stopwords")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

1)

A) Create a toy corpus & Dataframe

```
corpus = ["I am unable to log in to my account. Please help!",
          "My order has not arrived yet. When can I expect it?",
          "I was supposed to get my order in yesterday, but did not. What is the updated arrival date?",
          "I got my product but it does not fit, I would like to return it.",
          "The product I received is damaged. I would like to return it and  get a replacement.",
          "I want to get rid of my subscription, help me cancel it please.",
          "I need to cancel my subscription. How can I do that?",
          "I have forgotten my account password. How can I reset it?"]
```

```
df = pd.DataFrame(corpus, columns=['Customer Support Ticket']) # convert to dataframe
df.index.rename('Ticket Number', inplace=True) # rename index to ticket number
df.head()
```

Ticket Number	Customer Support Ticket
0	I am unable to log in to my account. Please help!
1	My order has not arrived yet. When can I expect it?
2	I was supposed to get my order in yesterday, but did not. What is the updated arrival date?
3	I got my product but it does not fit, I would like to return it.

B) Just based on a quick read-over of these texts, how many distinct topics would you propose exist amongst these set of eight sentences?

4 topics.

C) Preprocess text

As a final step in developing our corpus, we need to noramlize it. If the goal is to best extract the topic from the text based on the number and variety of words within it, what are the required preprocessing/normalizing steps? Which ones do we not need to implement, based on the given task and texts and why?

- **Decoding/Removal of HTML**: Not necessary, the text is in plain english.
- **Special Character Removal**: Necessary, if there are special characters they will not be useful.
- **Case conversion**: Necessary, capitalization does not contain important data for this set.
- **Contraction Expansion**: Necessary, comparing different strings that contain the same words w/ or w/o contractions.
- **Stemming/Lemmatization**:
- **Stopword Removal**:

```
wpt = nltk.WordPunctTokenizer()
stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
    doc = re.sub(r'[^a-zA-Z/s]', ' ', doc)
    doc = doc.lower()
    doc = doc.strip()
    tokens = wpt.tokenize(doc)
    filtered_tokens = [token for token in tokens if token not in stop_words]
    doc = ' '.join(filtered_tokens)
    return doc

normalize_corpus = np.vectorize(normalize_document)
```

```
norm_corpus = normalize_corpus(corpus) # apply normalize_corpus function to corpus
norm_corpus
```

```
array(['unable log account please help', 'order arrived yet expect',
       'supposed get order yesterday updated arrival date',
       'got product fit would like return',
       'product received damaged would like return get replacement',
       'want get rid subscription help cancel please',
       'need cancel subscription', 'forgotten account password reset'],
      dtype='<U58')
```

2) Vector Space Representation Models - Term Frequency

Now let's apply various means of representing the words and texts as vectors in a higher dimensional space. We begin with the basic bag of words model which utilizes a sparse vector representation based on term-frequency. We will produce word and document vectors within a term-document matrix, using the `CountVectorizer` function from `sklearn`.

```
cv = CountVectorizer(min_df=0., max_df=1.)                                         # set parameters of Countvectorizer, (use 1. not 1)
cv_matrix = cv.fit_transform(norm_corpus)                                          # apply countvectorizer to norm_corpus
print(cv_matrix)                                                                   # view sparse representation
```

```
      (0, 26)       1
      (0, 13)       1
      (0, 0)        1
      (0, 17)       1
      (0, 11)       1
      (1, 15)       1
      (1, 2)        1
      (1, 31)       1
      (1, 6)        1
      (2, 15)       1
      (2, 25)       1
      (2, 9)        1
      (2, 30)       1
      (2, 27)       1
      (2, 1)        1
      (2, 5)        1
      (3, 10)       1
      (3, 18)       1
      (3, 7)        1
      (3, 29)       1
      (3, 12)       1
      (3, 22)       1
      (4, 9)        1
      (4, 18)       1
      (4, 29)       1
      (4, 12)       1
      (4, 22)       1
      (4, 19)       1
      (4, 4)        1
      (4, 20)       1
      (5, 17)       1
      (5, 11)       1
      (5, 9)        1
      (5, 28)       1
      (5, 23)       1
      (5, 24)       1
      (5, 3)        1
      (6, 24)       1
      (6, 3)        1
      (6, 14)       1
      (7, 0)        1
      (7, 8)        1
      (7, 16)       1
      (7, 21)       1
```

A) What are we looking at here? What do each of these tuples represent and what is the associated value to right of each tuple?

the first number tells us what word number it is referring to, the second number refers to what number it is in alphabetic order of all of the words. The last number is a count. For this set, none of the words repeat within a single sentence.

B) Now let's convert `cv_matrix` to an array to view it in a matrix format. What does each row represent? What about each column?

```
cv_matrix = cv_matrix.toarray()
cv_matrix
```

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

the rows represent each sentence while the columns refer to the 32 words. The 1 indicates within which sentence they exist.

C) Let's confirm that and also make our matrix easier to read by applying the feature names (vocab names):

```
vocab = cv.get_feature_names_out()                                                     # assign feature names to vocab ob
tf = pd.DataFrame(cv_matrix, columns = vocab)                                                   # convert matrix to
tf
```

	account	arrival	arrived	cancel	damaged	date	expect	fit	forgotten	get	...	re
0	1	0	0	0	0	0	0	0	0	0	...	
1	0	0	1	0	0	0	1	0	0	0	...	
2	0	1	0	0	0	1	0	0	0	1	...	
3	0	0	0	0	0	0	0	1	0	0	...	
4	0	0	0	0	1	0	0	0	0	1	...	
5	0	0	0	1	0	0	0	0	0	1	...	
6	0	0	0	1	0	0	0	0	0	0	...	
7	1	0	0	0	0	0	0	0	1	0	...	

8 rows × 32 columns

```
tf.shape # 8 sentences, 32 words (features)
```

```
(8, 32)
```

D) What are the most frequently occurring words in the corpus? Do you think these are the most import words?

```
tf.mean() # because we only have 1s and 0s
```

```
account        0.250
arrival        0.125
arrived        0.125
cancel         0.250
damaged        0.125
date           0.125
expect         0.125
fit            0.125
forgotten      0.125
get            0.375
got            0.125
help           0.250
like           0.250
log            0.125
need           0.125
order          0.250
password       0.125
please         0.250
product        0.250
received       0.125
replacement    0.125
reset          0.125
return         0.250
rid            0.125
subscription   0.250
supposed       0.125
unable         0.125
updated        0.125
want           0.125
would          0.250
yesterday      0.125
```

```
yet              0.125
dtype: float64
```

'Get' is the most frequent word but it is not very important and almost a stop word. However, the words like 'account', 'return' and 'cancel' do convey more meaning.

2) Weighted Vector Space Representation -- TF-IDF Vectorizer

As we discussed in 2D, the most important words are not always the most frequent words. Term frequency-inverse document frequency (TF-IDF) provides a weighting to each word, in an attempt to balance the importance of words based on how frequent they are in specific documents, vs. all documents.

We can easily calculate the tf-idf of each word in our corpus using the `TfidfVectorizer` from `sklearn`.

```
tv = TfidfVectorizer(min_df=0., max_df=1.) # can use ngram_range=(2,2)

tv_matrix = tv.fit_transform(norm_corpus)                                                # apply tf-idf to norm_corpus
tv_matrix = tv_matrix.toarray()                                                          # convert to array

vocab = tv.get_feature_names_out()                                                       # apply vocab labels
pd.DataFrame(np.round(tv_matrix, 2), columns=vocab)                                      # convert to DF, round to 2 decimal places
```

	account	arrival	arrived	cancel	damaged	date	expect	fit	forgotten	get	...
0	0.41	0.0	0.00	0.00	0.0	0.0	0.00	0.00	0.00	0.00	...
1	0.00	0.0	0.52	0.00	0.0	0.0	0.52	0.00	0.00	0.00	...
2	0.00	0.4	0.00	0.00	0.0	0.4	0.00	0.00	0.00	0.29	...
3	0.00	0.0	0.00	0.00	0.0	0.0	0.00	0.46	0.00	0.00	...
4	0.00	0.0	0.00	0.00	0.4	0.0	0.00	0.00	0.00	0.29	...
5	0.00	0.0	0.00	0.36	0.0	0.0	0.00	0.00	0.00	0.31	...
6	0.00	0.0	0.00	0.54	0.0	0.0	0.00	0.00	0.00	0.00	...
7	0.44	0.0	0.00	0.00	0.0	0.0	0.00	0.00	0.52	0.00	...

8 rows × 32 columns

A) Examine the tfidf term-document matrix, what observations can you make about the difference between these vector representations as compared to the term-frequency based representations?

The more important words are weighted more heavily. Words like 'get' have lower values because they are more frequent and less important.

B) Document Similarity

Let's utilize our new document vectors to examine document similarity, this might provide some additional insight into the number of categories amongst our topics, and which documents fall into those categories. We can quantify document similarity by examining the distance (via `cosine_similarity`) between document vectors.

```
similarity_matrix = cosine_similarity(tv_matrix)                                                        # apply cosine_similarity
similarity_df = pd.DataFrame(similarity_matrix)                                                            # display similarity ma

similarity_df.style.background_gradient(cmap="Blues")                                  # let's make it easier to read
```

	0	1	2	3	4	5	6	7
0	1.000000	0.000000	0.000000	0.000000	0.000000	0.300168	0.000000	0.180119
1	0.000000	1.000000	0.146300	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.146300	1.000000	0.000000	0.083298	0.090774	0.000000	0.000000
3	0.000000	0.000000	0.000000	1.000000	0.509086	0.000000	0.000000	0.000000
4	0.000000	0.000000	0.083298	0.509086	1.000000	0.090002	0.000000	0.000000
5	0.300168	0.000000	0.090774	0.000000	0.090002	1.000000	0.392282	0.000000
6	0.000000	0.000000	0.000000	0.000000	0.000000	0.392282	1.000000	0.000000
7	0.180119	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000

```
df
```

	Customer Support Ticket
Ticket Number	
0	I am unable to log in to my account. Please help!
1	My order has not arrived yet. When can I expect it?
2	I was supposed to get my order in yesterday, but did not. What is the updated arrival date?
3	I got my product but it does not fit, I would like to return it.
4	The product I received is damaged. I would like to return it and get a replacement.
5	I want to get rid of my subscription, help me cancel it please.
6	I need to cancel my subscription. How can I do that?

Which two documents are most similar? Is that surprising?

Our model is working! 3 and 4 are most similar followed by 5 and 6. 3 and 4 are requesting a return while 5 and 6 are about cancelling subscriptions.

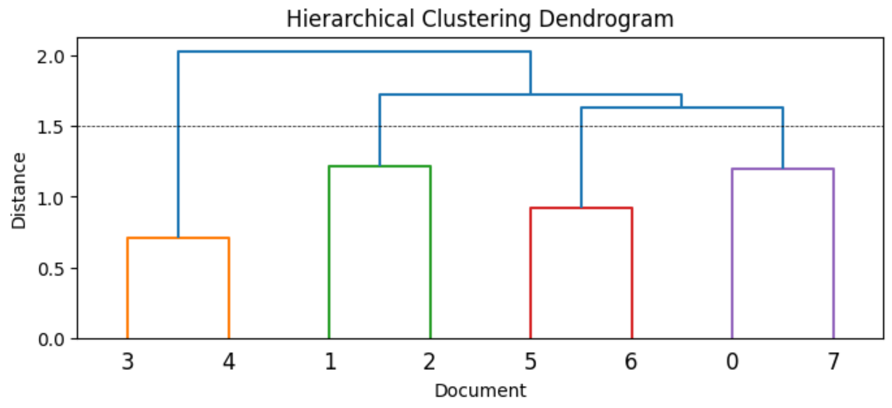
C) Document Clustering with Similarity Features

Let's take our analysis a step further now, and use the pairwise document similarities to perform an unsupervised text mining technique, document clustering. This method will evaluate the distance between documents, based on their cosine similarities to find naturally occurring clusters of documents.

We can produce a nice visualization of these clusters using a Dendrogram!

```
from matplotlib.pyplot import figure
Z = linkage(similarity_matrix, 'ward')                                                      # use ward-linkage function

plt.figure(figsize=(8,3))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Document')
plt.ylabel('Distance')
dendrogram(Z)                                                           # plot a dendrogram of Z
plt.axhline(y=1.5, c='k', ls='--', lw=0.5)
plt.show()
```



> **What can we deduce from this visualization? How does this relate to our results from the similarity matrix?**

At the bottom level, how the model paired each of the tickets is the same way that I did. 3&4 and 5&6 are together as the similarity matrix highlighted, but so are 1&2 and 0&7. They did not have the highest similarities, but when you take out 3, 4, 5, & 6, they do have the highest similarities and it's clear why they were paired on the base level.

Also, because 3&4 have such a distant connection from the 6 other tickets, they are the most distinct.

D) Cluster Labeling

> Now we have good reason to support that there are three distinct clusters, or categories of documents. Let's add our cluster labels to our original corpus dataframe. Next, we'll try to figure out some good labels for our clusters/categories of documents.

```
max_dist = 1.5

cluster_labels = fcluster(Z, max_dist, criterion = 'distance')                    # fit clusters using Z object
cluster_labels = pd.DataFrame(cluster_labels, columns=['Cluster Label'])                         # convert cluster_labels to DF
df = pd.concat([df, cluster_labels], axis = 1)                                                          # append
df
```

	Customer Support Ticket	Cluster Label
0	I am unable to log in to my account. Please help!	4
1	My order has not arrived yet. When can I expect it?	2
2	I was supposed to get my order in yesterday, but did not. What is the updated arrival date?	2
3	I got my product but it does not fit, I would like to return it.	1
4	The product I received is damaged. I would like to return it and get a replacement.	1
5	I want to get rid of my subscription, help me cancel it please.	3
6	I need to cancel my subscription. How can I do that?	3

3) Topic Modeling

The last thing we need to do is replace our mystery topic labels with some logical labels. We know we have ***four*** distinct clusters of documents, so let's ***find*** which words fall into which cluster so that we can come up with appropriate cluster labels.

Topic modeling can be performed using a number of different unsupervised algorithms; we will using Latent Dirichlet Allocation (LDA) -- which we will learn more in-depth later in the semester.

```
lda = LatentDirichletAllocation(n_components = 4, max_iter=10000, random_state=0)                # set parameters of LDA (set component
dt_matrix = lda.fit_transform(tv_matrix)                                                                 # fit LDA to count vectorizer
features = pd.DataFrame(dt_matrix, columns = ['top_1','top_2','top_3','top_4'])                                       # show c
features
```

	top_1	top_2	top_3	top_4
0	0.765821	0.077726	0.077721	0.078732
1	0.083829	0.083892	0.085409	0.746869
2	0.069118	0.069162	0.791366	0.070353
3	0.072889	0.779891	0.072925	0.074295
4	0.065972	0.071870	0.066309	0.795849
5	0.071531	0.069192	0.069517	0.789760
6	0.091937	0.091976	0.091972	0.724116
7	0.748927	0.083725	0.083721	0.083627

A) Extract the weights of words from each topic, display as list

```
tt_matrix = lda.components_

for topic_weights in tt_matrix:
    topic = [(token, weight) for token, weight in zip(vocab, topic_weights)]
    topic = sorted(topic, key = lambda x: -x[1])
    topic = [item for item in topic if item[1] > 0.6]
    print(topic)
    print()
```

```
[('account', 1.0986349631546841), ('forgotten', 0.7691634187433909), ('password', 0.7691634187433909), ('reset', 0.7691634187433909), ('

[('fit', 0.7056138907515034), ('got', 0.7056138907515034), ('like', 0.6361174070349611), ('product', 0.6361174070349611), ('return', 0.6

[('arrival', 0.65042745741183), ('date', 0.65042745741183), ('supposed', 0.65042745741183), ('updated', 0.65042745741183), ('yesterday',

[('cancel', 1.152612249460371), ('subscription', 1.152612249460371), ('need', 0.8939142959673863), ('get', 0.8506808203313126), ('arrive
```

Based on the words in these topics, how might be relabel our corpus categories?

1. account help
2. product returns
3. arrival updates
4. subscription cancellation

B) Apply category labels to dataframe

```
cluster_to_topics = {1: "returns",
                     2: "arrival updates",
                     3: "subscription cancellation",
                     4: "account help"}

df['Cluster Label'].replace(cluster_to_topics, inplace = True)
df
```

	Customer Support Ticket	Cluster Label
0	I am unable to log in to my account. Please help!	account help
1	My order has not arrived yet. When can I expect it?	arrival updates
2	I was supposed to get my order in yesterday, but did not. What is the updated arrival date?	arrival updates
3	I got my product but it does not fit, I would like to return it.	returns