DATA SOCIETY®

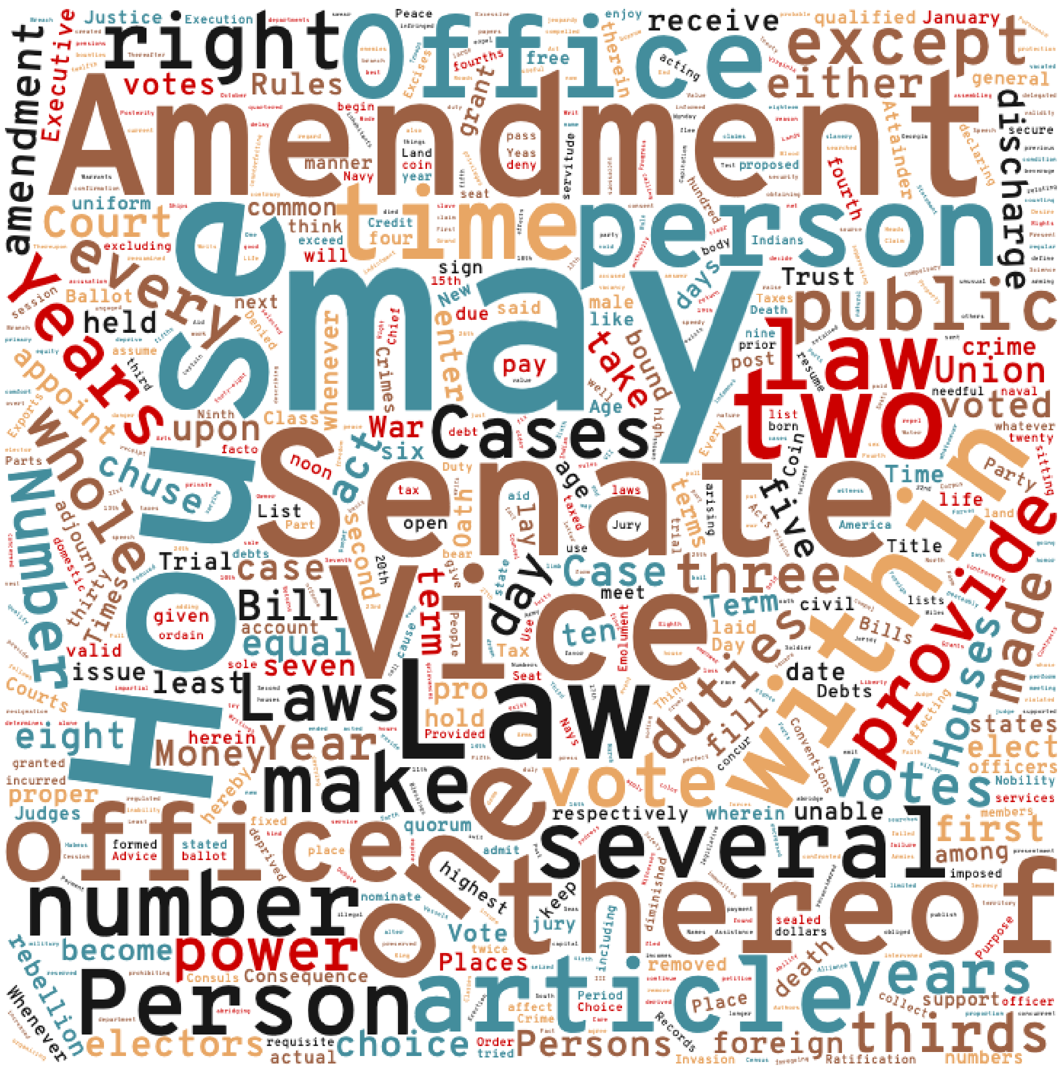
Introduction to text mining - part 2

"One should look for what is and not what he thinks should be."
-Albert Einstein.

Welcome back

Before we get started, here's a warm-up question.

What **corpus** do you think the **word cloud** below represents? **Type your guess into the chat window.**



Recap and next steps

The topics covered in the last module:

- Processing text data using bag-of-words approach
- Creating a term document matrix

In this module, we will learn about:

- Visualizing text data
- n-grams and a bit about their probabilities vs. single words (unigrams)
- Creating the term frequency-inverse document frequency (TF-IDF) matrix

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your `booz-allen-hamilton` folder

```
from pathlib import Path
# Set `home_dir` to the root directory of your computer.
home_dir = Path.home()

# Set `main_dir` to the location of your `booz-allen-hamilton` folder.
main_dir = home_dir / "Desktop" / "booz-allen-hamilton"

# Make `data_dir` from the `main_dir` and remainder of the path to data directory.
data_dir = main_dir / "data"
```

Working directory

- Set working directory to the `data_dir` variable we set

```
# Set working directory.
os.chdir(data_dir)
```

```
# Check working directory.
print(os.getcwd())
```

```
/home/[user-name]/Desktop/booz-allen-hamilton/data
```

Loading packages

```
# Helper packages.
import os
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
```

```
# Packages with tools for text processing.
import nltk
from wordcloud import WordCloud

# Packages for working with text data.
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
```

```
# Packages for getting data ready for and building a LDA model.
import gensim
from gensim import corpora, models
from pprint import pprint
```

Module completion checklist

Objective	Complete
Visualize distribution of words	
Introduction to n-grams	
Explain use cases for bag-of-words	
Summarize supervised vs. unsupervised learning	
Weight text data with term frequency inverse document frequency (TF-IDF)	

Load objects

- As a refresher, `pickle` saves objects by "flattening" them:
  - **pickle/saving:** a Python object is converted into a byte stream
  - **unpickle/loading:** the inverse operation where a byte stream is converted back into an object
- Let's **unpickle** what we need for now, the `NYT_clean`, `NYT_clean_list` and `corpus_freq_dist` objects:

```
processed_docs = pickle.load(open("NYT_clean.sav","rb"))  #<- the processed NYT snippets
```

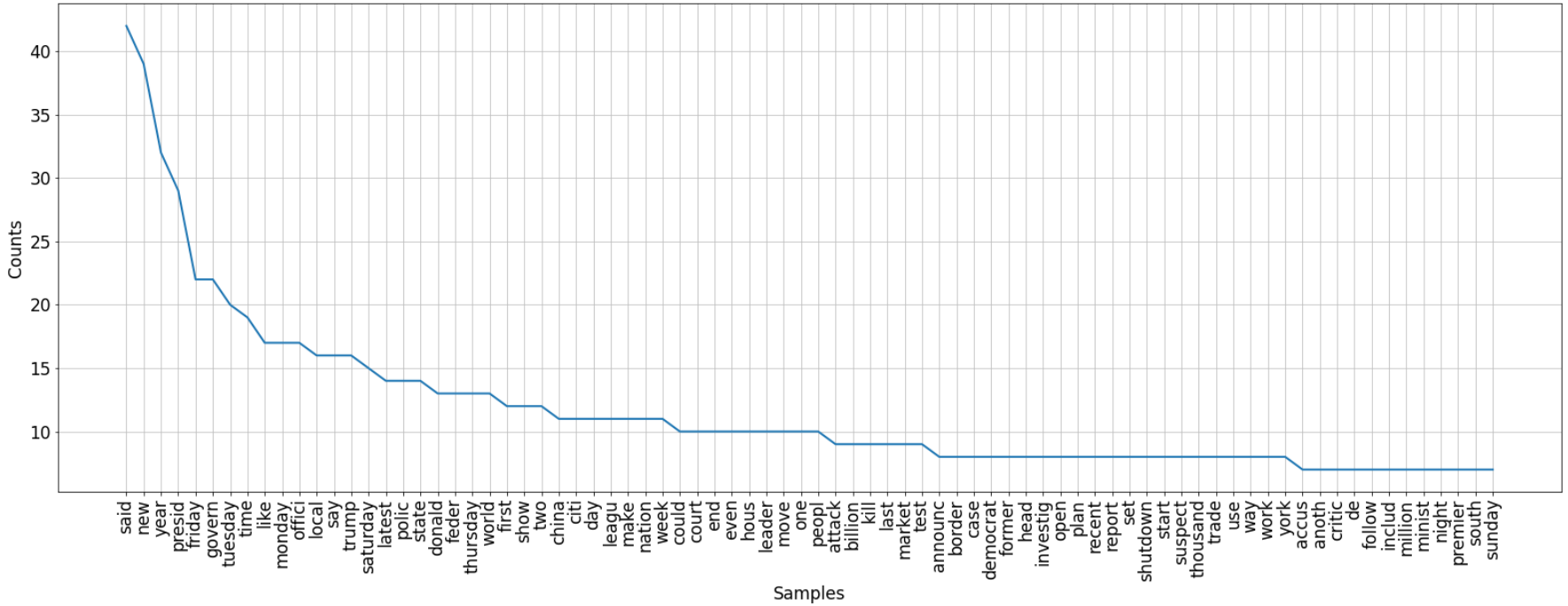
```
NYT_clean_list = pickle.load(open("NYT_clean_list.sav","rb"))  #<- the processed NYT snippets
```

```
corpus_freq_dist = pickle.load(open("corpus_freq_dist","rb"))  #<- the processed NYT snippets
```

Plot distribution of words in snippet corpus

```
# Save as a FreqDist object native to nltk.
corpus_freq_dist = nltk.FreqDist(corpus_freq_dist)
```

```
# Plot distribution for the entire corpus.
plt.figure(figsize = (50, 10))
corpus_freq_dist.plot(80)
```



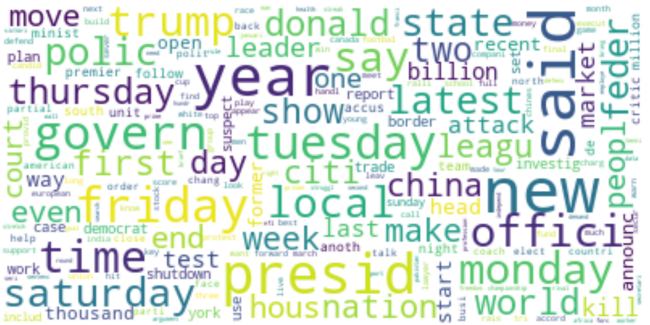
Visualizing word counts with word clouds

- A **word cloud** is a visualization of the most common words in a corpus.
- The **larger** the word, the **more frequently** it appears.

```
# Construct a word cloud from corpus.
wordcloud = WordCloud(max_font_size = 40,
                      background_color = "white",
                      collocations = False)
wordcloud = wordcloud.generate(' '.join(NYT_clean_list))
```

```
# Plot the cloud using matplotlib.
plt.figure()
plt.imshow(wordcloud, interpolation = "bilinear")
plt.axis("off")
```

```
plt.show()
```



- What words are **most common** in the text data that we just cleaned?
- Based on that, what **topics** do you think these documents focus on?

Module completion checklist

Objective	Complete
Visualize distribution of words	✔
Introduction to n-grams	
Explain use cases for bag-of-words	
Summarize supervised vs. unsupervised learning	
Weight text data with term frequency inverse document frequency (TF-IDF)	

Introducing n-grams

- An **n-gram** is a sequence of n words where n can take on any discrete value ranging from 1 to ∞
  - If $n = 1$, we call it a 1-gram or a **unigram**
  - If $n = 2$, we call it a 2-gram or a **bigram**
  - If $n = 3$, we call it a 3-gram or a **trigram**
  - If $n = 4$, we call it a 4-gram or a **quadgram**
- For example, the word `I` is a unigram, `I love` is a bigram, `I love cheese` is a trigram and `I love cheese sticks` is a quadgram

Display n-grams in a dataset

- Convert the `NYT_clean_list` to a list of words

```
def sentence_convert_to_word(lst):
    return ' '.join(lst).split()

words = sentence_convert_to_word(NYT_clean_list)
```

- The method **nltk.ngrams()** gives the list of n-grams from a given list of words

```
# Display the frequency of counts of the bigrams
(pd.Series(nltk.ngrams(words, 2) # <- set the value to 2 for bigram
          ).value_counts())[:10] # <- display the top 10 bigrams
```

```
(donald, trump)       13
(presid, donald)      13
(time, local)          9
(new, york)            8
(said, monday)         7
(new, year)            6
(premier, leagu)       6
(govern, shutdown)     5
(white, hous)          5
(said, friday)         5
dtype: int64
```

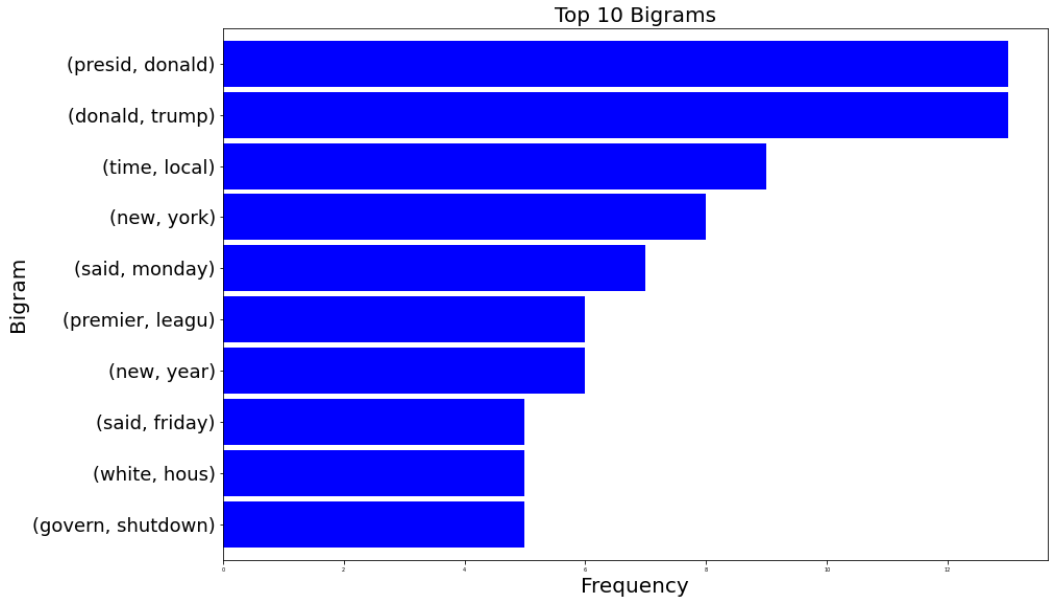
Display n-grams in a dataset (cont'd)

```
bigrams = (pd.Series(nltk.ngrams(words, 2)).value_counts())[:10].sort_values()
trigrams = (pd.Series(nltk.ngrams(words, 3)).value_counts())[:10].sort_values()
```

Visualizing n-grams using bar chart

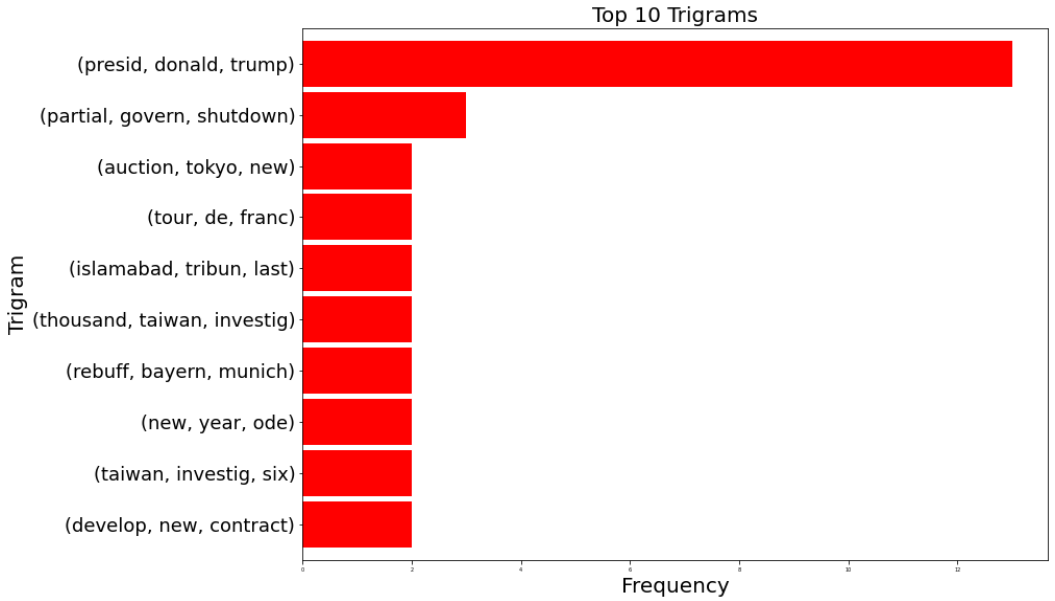
Bigrams

```
bigrams.plot.barh(color='blue', width=.9,
figsize=(14, 8))
plt.yticks(fontsize = 18)
plt.title('Top 10 Bigrams', fontsize = 20)
plt.ylabel('Bigram', fontsize = 20)
plt.xlabel('Frequency', fontsize = 20)
plt.tight_layout() #<- adjusts plot layout
plt.show()
```



Trigrams

```
trigrams.plot.barh(color='blue',
                   width=.9, figsize=(14, 8))
plt.yticks(fontsize = 18)
plt.title('Top 10 Trigrams', fontsize = 20)
plt.ylabel('Trigram', fontsize = 20)
plt.xlabel('Frequency', fontsize = 20)
plt.tight_layout() #<- adjusts plot layout
plt.show()
```



n-gram models - use cases

- Help decide which n-grams can be joined together to form single entity
  - *Ex: The words "San" and "Fransisco" can be joined together as one word, and so can "high" and "school"*
- Make next word predictions
  - *Ex: For an incomplete sentence, like "what a cute," it is more likely that the next word is going to be "baby" or "puppy" than "duck"*
- Correct spelling errors
  - *Ex: The phrase "drink mik" could be corrected to "drink milk" if you knew that the word "milk" has a higher probability of occurrence after the word "drink," and also since the overlap of letters between "mik" and "milk" is high*

n-gram probabilities

- Consider the following corpus:
  - Thank you for helping me.
  - I really like your sweater.
  - Excuse me, could you tell me what time it is?
  - I'm really sorry that I forgot to call you.
  - I really appreciate you helping me out.

- Let's say we train a bigram sentence completion model on the corpus above
- The probability of the occurrence of the word `a1` after the word `a2` would be:

$$count(a2, a1)/count(a2)$$

n-gram probabilities (cont'd)

- We will calculate the probability of the words occurring after the word `really`

$$count(\text{really like})/count(\text{really}) = 1/3 = 0.33$$
$$count(\text{really sorry})/count(\text{really}) = 1/3 = 0.33$$
$$count(\text{really appreciate})/count(\text{really}) = 1/3 = 0.33$$

- So, for the test case `I really`, the model will correctly predict the next word only 1 out of 3 times as the probability of the correct answer is just `1/3`

Understanding n-gram models

- Using the basic concept of n-gram probabilities, **n-gram models** can be created
- An n-gram model can be built and used to predict the next word i.e Nth word in a sentence based on the probability of occurrence of previous $N-1$ words
  - Say we train a bigram model, it can predict the next word in a given sentence based on $N-1$ words i.e $2-1=1$ word
  - The same logic applies to the trigram models, quadgram models and so on

Limitations of n-gram models

- Does not capture long range dependencies
- Hugely dependent on the corpus of training data, i.e. unable to predict a new instance if not present in the training corpus
- Extremely sparse

These limitations make it impossible for the algorithm to classify new instances because it is entirely illogical to the machine. Hence, n-grams is only suitable for extremely large amounts of data. Having said that, even after training the algorithm with large data, there is no guarantee that it will perform well on unseen data.

Knowledge check 1

Exercise 1

Module completion checklist

Objective	Complete
Visualize distribution of words	✔
Introduction to n-grams	✔
Explain use cases for bag-of-words	
Summarize supervised vs. unsupervised learning	
Weight text data with term frequency inverse document frequency (TF-IDF)	

"Bag-of-words" analysis: key elements

- In our last class, we got up to this point in our analysis:

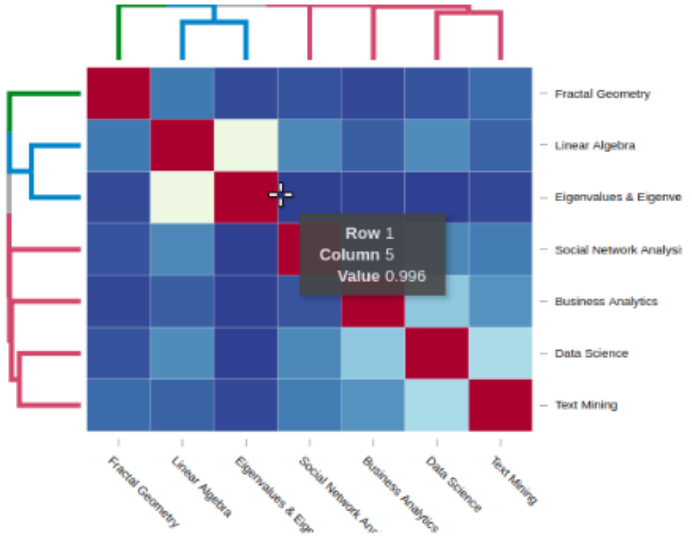
What we need	What we have learned
A corpus of documents cleaned and processed in a certain way • All words are converted to lower case • All punctuation, numbers, and special characters are removed • Stopwords are removed • Words are stemmed to their root form	✔
A Document-Term Matrix (DTM), with counts of each word recorded for each document	✔
A transformed representation of a Document-Term Matrix (i.e. weighted with TF-IDF weights)	

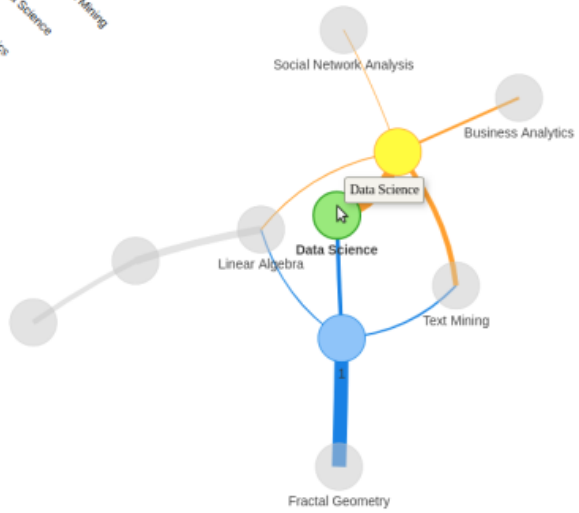
Article snippet analysis

- So far, the steps we have taken are:
  - **load** the corpus, where each 'document' is actually one article snippet
  - **clean** the text, removing punctuation, numbers, special characters and stop words
  - *stem* the words to their root forms
  - **create** a document-term matrix (DTM) with counts of each word recorded for each document
- In the last part of today's session, we will build the final, optimized matrix - a weighted **T**erm **F**requency - **I**nverse **D**ocument **F**requency (TF-IDF) by
  - **transforming** the DTM to be a **weighted TF-IDF**

"Bag-of-words" analysis: use cases

- What can be done with such a seemingly *crude* approach?
- Quite a few things, actually! They include:
  - topic modeling
  - word and document similarity query processing
  - word and document clustering
  - sentiment analysis
  - automated document summarization





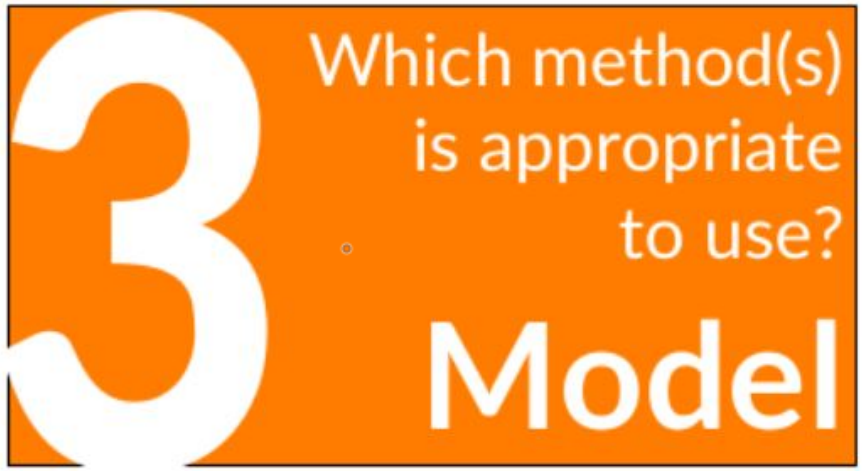
"Bag-of-words" analysis: article snippets

- We are going to dive deeper into one of these use cases: **topic modeling**
- In our next class, we are going to implement a very popular method called **Latent Dirichlet Allocation (LDA)**
- This will help us understand **broad topics** within our corpus of article snippets
- **LDA** is a great way to **explore** text data and **generate hypotheses**

Data science control cycle (DSCC)

What stage of the DSCC would this goal fit into?

DSCC: modeling



- A model by definition is a replica of a real thing
- Using a ship to imitate a train won't cut it
- Select a **model that suits your problem/data** or **simulates the real-life situation** in the closest possible way

$$\min_{w,b,\varepsilon} \frac{1}{2}\sum_{p=1}^{d} w_p^2 + \gamma \sum_{i=1}^{n} \varepsilon_i$$

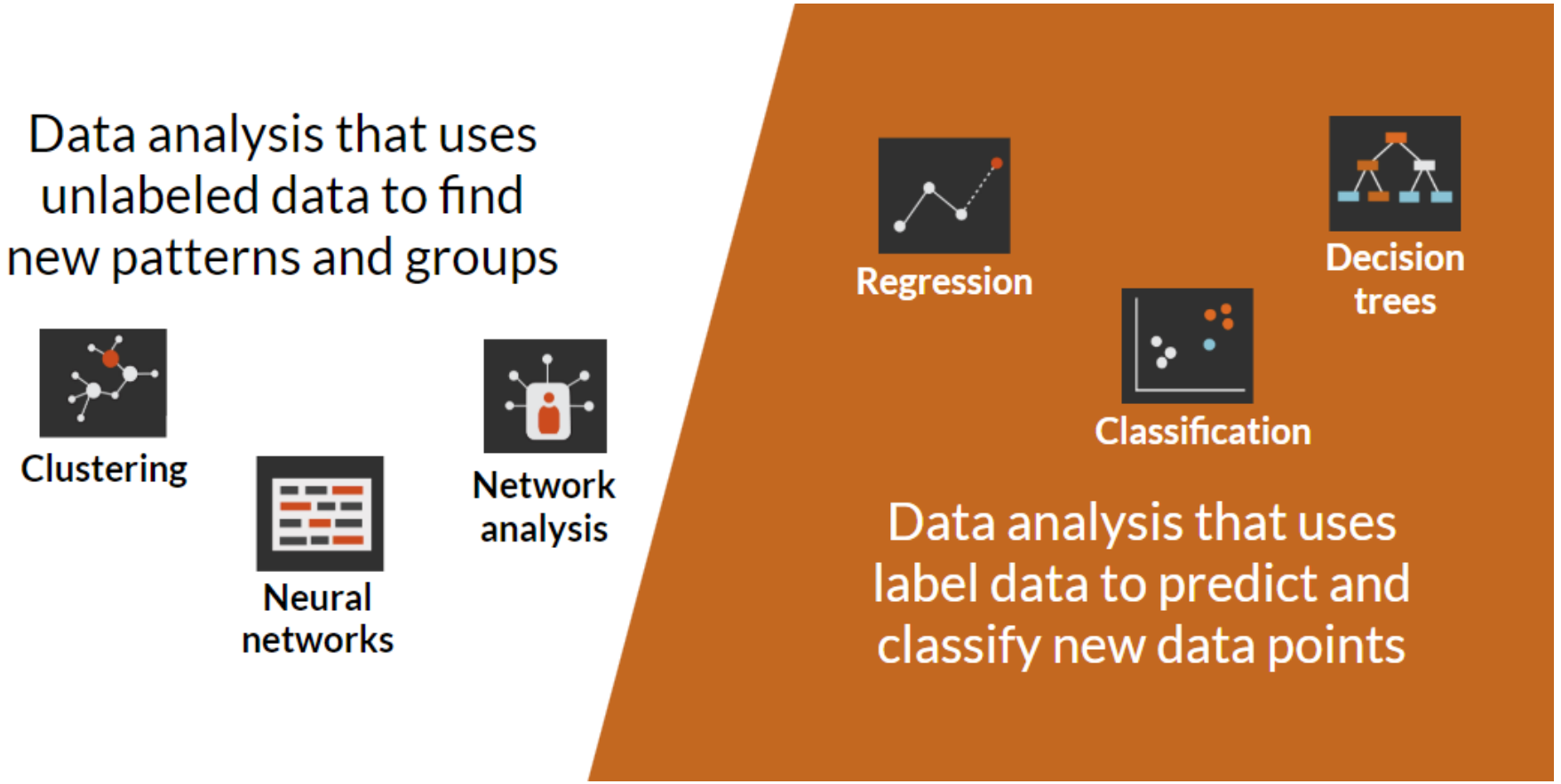
$$\text{subject to} \begin{cases} y_i \left[\sum_{p=1}^{d} w_p x_i^p + b \right] \geq 1 - \varepsilon_i, \ \forall i = 1, \ldots, n \\ \varepsilon_i \geq 0, \qquad \forall i = 1, \ldots, n. \end{cases}$$

- What kinds of **text analysis models** have you already encountered?
- For text data, the first step in model building is often to use **unsupervised learning**

Module completion checklist

Objective	Complete
Visualize distribution of words	✔
Introduction to n-grams	✔
Explain use cases for bag-of-words	✔
Summarize supervised vs. unsupervised learning	
Weight text data with term frequency inverse document frequency (TF-IDF)	

Unsupervised vs supervised



How do you see text analysis and exploration fitting into unsupervised learning?

LDA: unsupervised text analysis

- How does LDA fall into the category of unsupervised learning?
  - **Topics** are formed from **unlabeled** text
  - The basis of this algorithm is **clustering** documents into **"topics"**
  - **Clustering** is one of the best known unsupervised techniques
- We will learn about LDA in detail in our next session
- First, we need to learn how to **transform our DTM to a TF-IDF weighted matrix**
- We will continue working with the **NY Times article snippets** and the **UN agreements** data sets.

Knowledge check 2

Module completion checklist

Objective	Complete
Visualize distribution of words	✔
Introduction to n-grams	✔
Explain use cases for bag-of-words	✔
Summarize supervised vs. unsupervised learning	✔
Weight text data with term frequency inverse document frequency (TF-IDF)	

Field trip

- Visit ***databasic.io/en/wordcounter***
- Select your favorite artist under the “use a sample” menu and hit “COUNT”

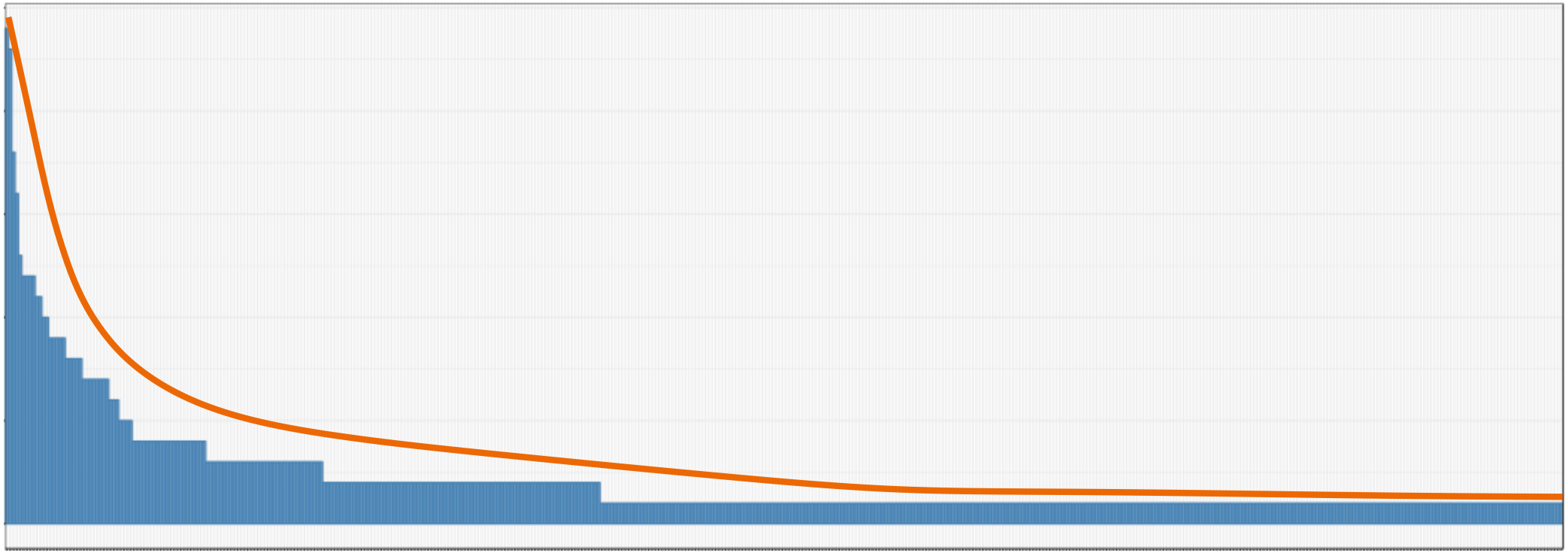
- Does the data tell a story about the selected artist?
- What additional type of analysis of this data might be interesting?

Gensim package

- `gensim` is a Python package made for topic modeling
- We will be using `gensim` to transform our DTM to a TF-IDF matrix, as well as to build the LDA model
- Click ***here*** for complete documentation

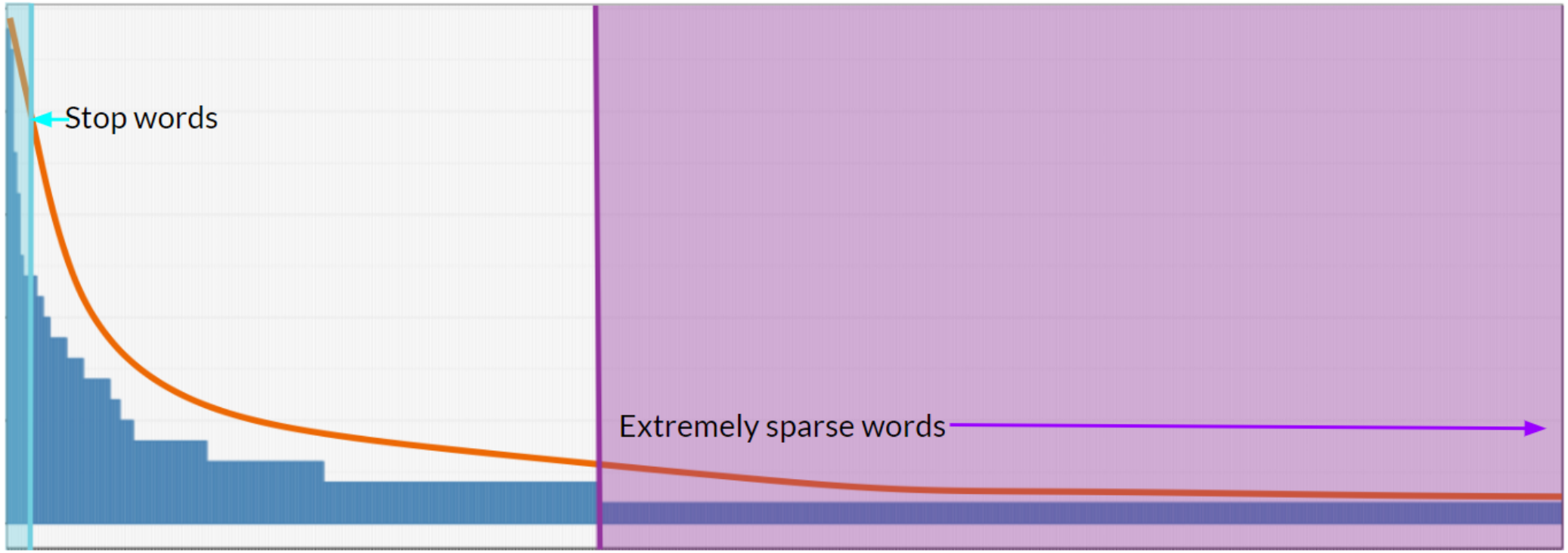
Word distribution in a language and corpus

- The distribution of words in a corpus and in a language is **highly skewed right**, which indicates that few words have very high frequencies and most words have very low counts!
- This phenomenon has implications in many areas of research and applications like information theory, natural language processing, search engines and many more



Word distribution in a language and corpus

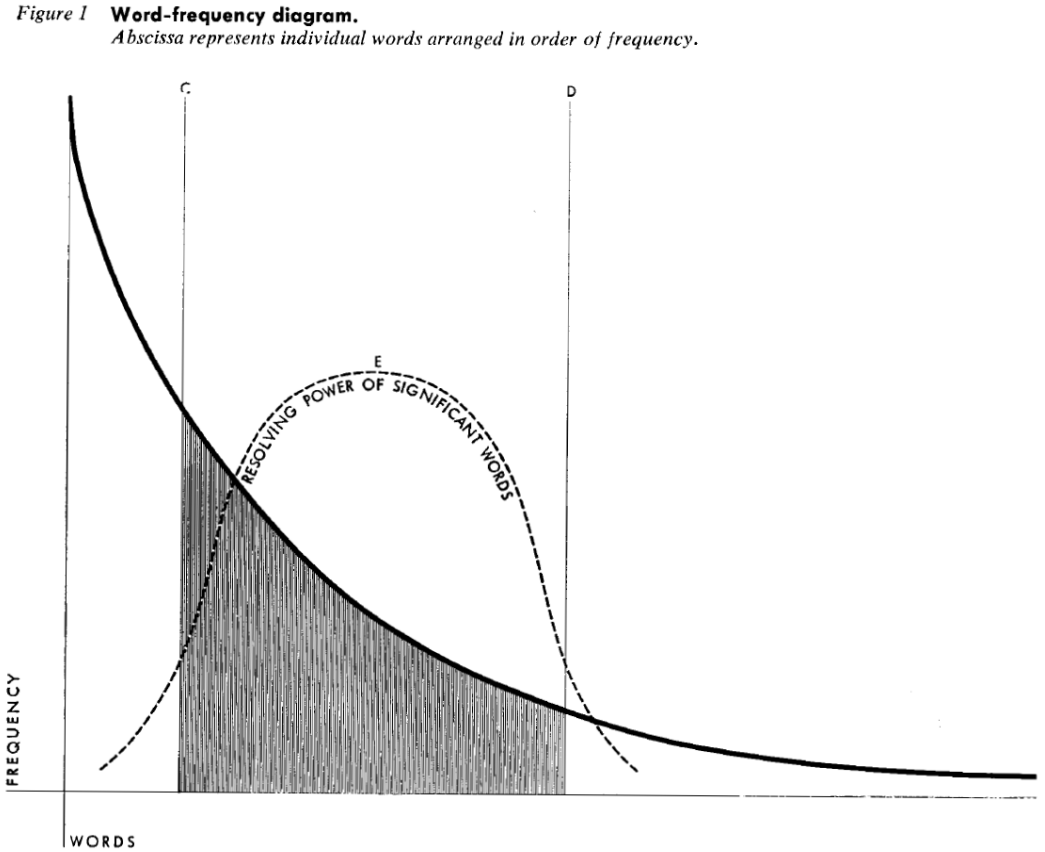
- Words that have extremely high frequencies are considered **stop words**
- Those that have extremely low frequencies are considered **extremely sparse words**



Power of significant words

- By removing *BOTH* ends of the distribution, we **reduce the dimensionality** of our data and **avoid "overfitting"** our text model

- In 1958, *H.P. Luhn* (a researcher for IBM) *in 1958* proved that the **power of significant words is approximately normally distributed** and the top of the bell-shaped curve coincides with the **mid-portion** of the word frequency distribution



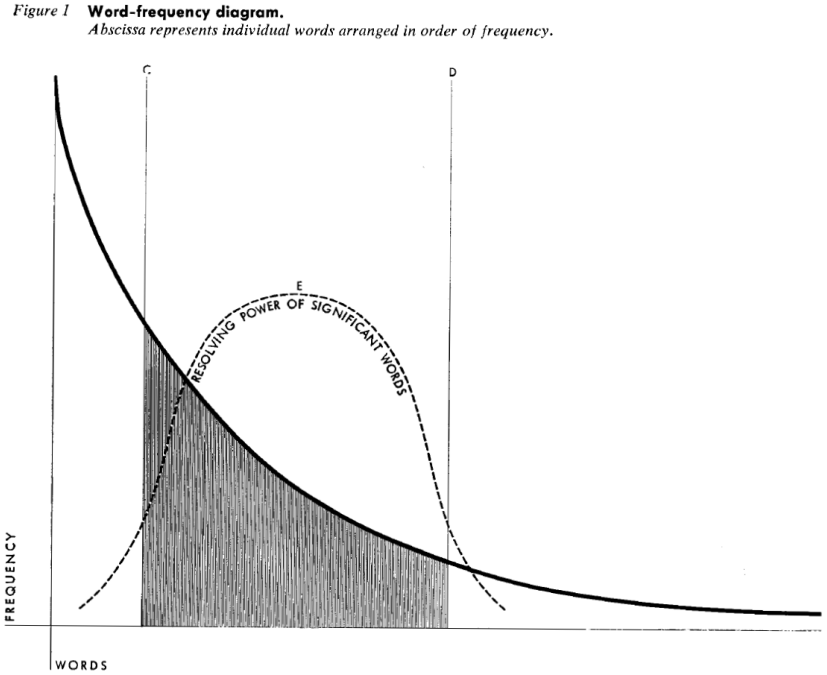
Source: H.P. Luhn, "The Automatic Creation of Literature Abstracts", *Presented at IRE National Convention, New York, March 24, 1958. Published in IBM JOURNAL APRIL 1958*

Why bother trimming words?

- **Avoid** having **too many dimensions** (in text analysis words represent variables, i.e. dimensions)
- **Speed up** computations
- **Reduce noise** and **prevent overfitting**

Ways to overcome extreme skewness

- Since the word significance is maximized towards the mid-portion of the skewed distribution, we need to **"reshape"** our data to let the words that are in that mid-portion stand out
- One of the most common ways of achieving this is **weighting** our data, which in statistics is usually known as **data transformation**



Source: H.P. Luhn, "The Automatic Creation of Literature Abstracts",*

**Presented at IRE National Convention, New York, March 24, 1958.*

Published in IBM JOURNAL APRIL 1958

Text Frequency - Inverse Document Frequency

- **TF-IDF** is a famous transformation of text data used to battle the skewness of the word distribution in a corpus and is given by the following formula:

$$TF \times IDF = \frac{F_{wd}}{N_d} \times log \frac{M}{M_w}$$

- Where:
  - Subscripts w and d stand for $word$ and $document$ respectively
  - F_{wd} is the frequency of a word in a document
  - N_d is a total number of words in a given document
  - M is the total number of documents in a corpus
  - M_w is the number of documents containing the given word in the corpus

TF-IDF effect on DTM

- Normalization of **TF** (i.e. $TF = \frac{F_{wd}}{N_d}$) eliminates the size effect of documents
  - Longer documents with more words no longer overpower smaller documents with fewer words that may contain valuable information
- **IDF** (i.e. $log\frac{M}{M_w}$) helps to adjust for the fact that some words appear more frequently in general, but carry less value in their meaning
  - These words are not specific or salient to a particular concept that would otherwise make a document or group of documents in a corpus stand out

TF-IDF effect on DTM (cont'd)

- Consider the following DTM:

DTM with raw counts

	analyze	data	fractal
Doc A	1	2	0
Doc B	0	0	1
Doc C	0	4	3

Weighted by

TF-IDF

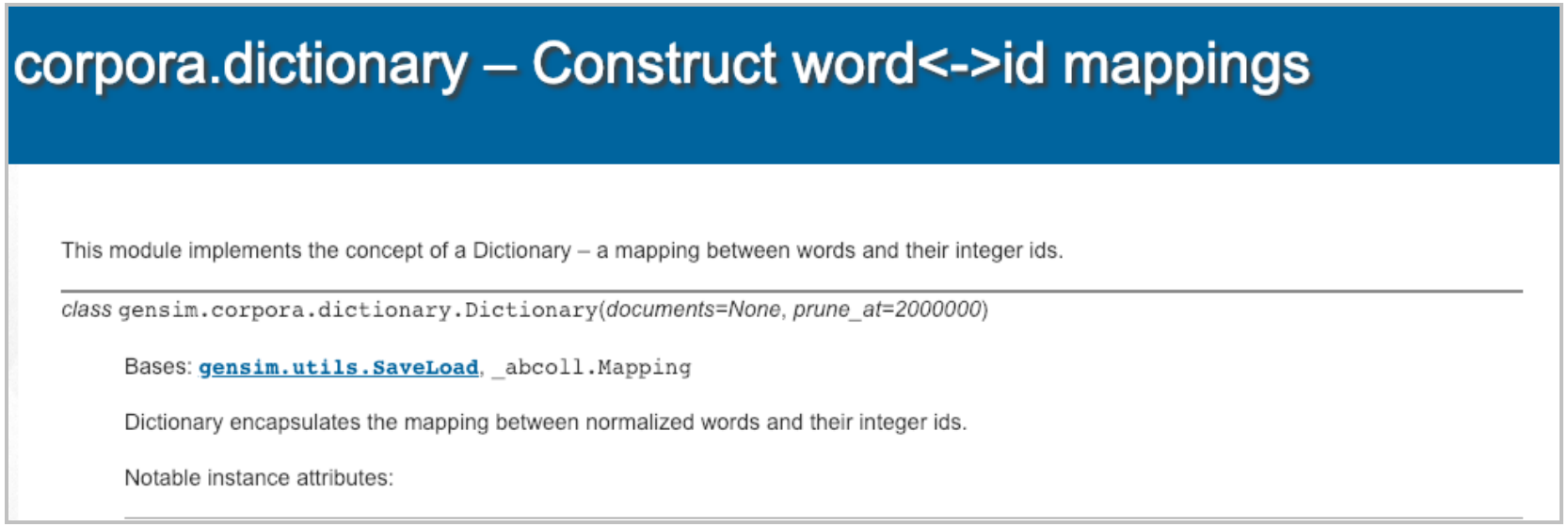
DTM with TF-IDF weights

	analyze	data	fractal
Doc A	0.528	0.390	0
Doc B	0	0	0.585
Doc C	0	0.334	0.251

- The word `data` appears in `DocA` and `DocC`, and although it appeared twice as often in `DocC`, its weight has adjusted for `DocA` to a very similar one as in `DocC`, making it of similar "value"

Create a dictionary of counts

- Remember when we created a convenience function to look at counts?
- Now let's create a **dictionary** of counts using a `gensim` function `gensim.corpora.Dictionary`
- This will give us a dictionary from `processed_docs` that contains the **number of times a given word appears** within the entire corpus



Create a dictionary of counts

- Let's create the dictionary using `gensim.corpora.Dictionary` and look at our output using a small loop

```
# Set the seed.
np.random.seed(1)

dictionary = gensim.corpora.Dictionary(processed_docs)

# The loop below iterates through the first 10 items of the dictionary and prints out the key and
value.
count = 0
for k, v in dictionary.iteritems():
    print(k, v)
    count += 1
    if count > 10:
        break
```

```
0 africa
1 attack
2 back
3 batsmen
4 claw
5 find
6 handl
7 like
8 live
9 must
10 newland
```

Create a dictionary of counts

- We can filter out words by their frequency in the dictionary
- `.filter_extremes()` will remove all values in the dictionary that are:
  - less frequent than `no_below` documents
  - more than `no_above` documents (fraction of total corpus size, not absolute number)
  - keep only first `keep_n` most frequent tokens

```
dictionary.filter_extremes(no_below = 5, no_above = 0.5, keep_n = 248)

# How many words did are left in the dictionary?
len(dictionary)
```

```
155
```

Document to bag-of-words

- Now we will use `gensim` library `doc2bow` to transform each document to a dictionary
- Each document will become a **dictionary** that has the **number of words** and has the **number of times each of those words appear**
- This is the object we will use to build our TF-IDF matrix

```
# We use a list comprehension to transform each doc within our processed_docs object.
bow_corpus = [dictionary.doc2bow(doc) for doc in processed_docs]

# Let's look at the first document.
print(bow_corpus[0])
```

```
[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 2)]
```

Document to bag-of-words

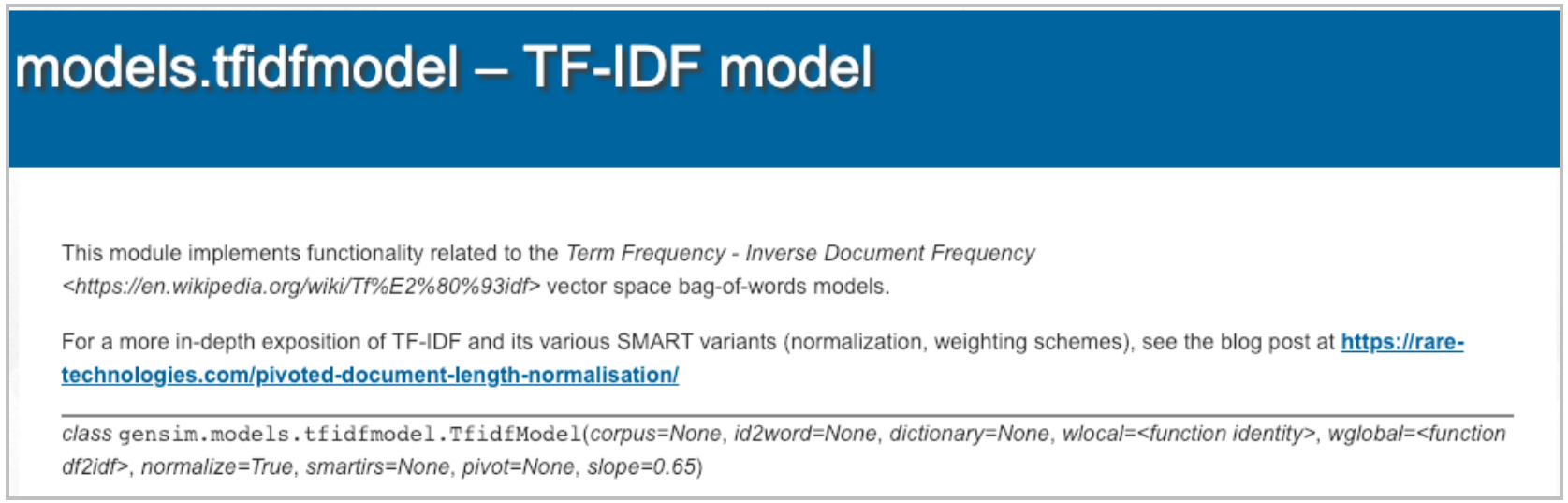
- Let's preview the bag-of-words for the first document

```
# Isolate the first document.
bow_doc_1 = bow_corpus[0]

# Iterate through each dictionary item using the index.
# Print out each actual word and how many times it appears.
for i in range(len(bow_doc_1)):
    print("Word {} (\"{}\") appears {}
time.".format(bow_doc_1[i][0],

          dictionary[bow_doc_1[i][0]],
          bow_doc_1[i][1]))
```

```
Word 0 ("attack") appears 1 time.
Word 1 ("back") appears 1 time.
Word 2 ("find") appears 1 time.
Word 3 ("handl") appears 1 time.
Word 4 ("like") appears 1 time.
Word 5 ("south") appears 1 time.
Word 6 ("start") appears 1 time.
Word 7 ("test") appears 1 time.
Word 8 ("thursday") appears 1 time.
Word 9 ("way") appears 2 time.
```

Transform counts with TfidfModel

- To transform a Document-Term matrix with `TF-IDF`, we will use `TfidfModel` from `gensim` library's `model` module for working with text



models.tfidfmodel – TF-IDF model

This module implements functionality related to the *Term Frequency - Inverse Document Frequency* <*https://en.wikipedia.org/wiki/Tf%E2%80%93idf*> vector space bag-of-words models.

For a more in-depth exposition of TF-IDF and its various SMART variants (normalization, weighting schemes), see the blog post at **https://rare-technologies.com/pivoted-document-length-normalisation/**

class `gensim.models.tfidfmodel.TfidfModel`(*corpus=None*, *id2word=None*, *dictionary=None*, *wlocal=<function identity>*, *wglobal=<function df2idf>*, *normalize=True*, *smartirs=None*, *pivot=None*, *slope=0.65*)

Transform counts with TfidfModel

- We will now activate the `TfidfModel` function and transform our `bow_corpus`
- Our output will be the **TF-IDF transformation** applied to **each document:**

$$TF \times IDF = \frac{F_{wd}}{N_d} \times log \frac{M}{M_w}$$

```
# This is the transformation.
tfidf = models.TfidfModel(bow_corpus)

# Apply the transformation to the entire corpus.
corpus_tfidf = tfidf[bow_corpus]

# Preview TF-IDF scores for the first document.
for doc in corpus_tfidf:
    pprint(doc)
    break
```

```
[(0, 0.25954540320095165),
 (1, 0.29127945804505345),
 (2, 0.3055490022471601),
 (3, 0.3055490022471601),
 (4, 0.20976922764421047),
 (5, 0.29127945804505345),
 (6, 0.268763787812797),
 (7, 0.25954540320095165),
 (8, 0.23076512567108073),
 (9, 0.5825589160901069)]
```

"Bag-of-words" analysis: key elements

What we need	What we have learned
A corpus of documents cleaned and processed in a certain way • All words a converted to lower case • All punctuation, numbers and special characters are removed • Stopwords are removed • Words are stemmed to their root form	✔
A Document-Term Matrix (DTM): with counts of each word recorded for each document	✔
A transformed representation of a Document-Term Matrix (i.e. weighted with TF-IDF weights)	✔

Knowledge check 3

Exercise 2

Module completion checklist

Objective	Complete
Visualize distribution of words	✔
Introduction to n-grams	✔
Explain use cases for bag-of-words	✔
Summarize supervised vs. unsupervised learning	✔
Weight text data with term frequency inverse document frequency (TF-IDF)	✔

Save results as a pickle

- Pickle all generated data for next steps

```
pickle.dump(bow_corpus, open('bow_corpus.sav', 'wb'))
pickle.dump(corpus_tfidf, open('corpus_tfidf.sav', 'wb'))
pickle.dump(dictionary, open('dictionary.sav', 'wb'))
```

This completes our module

Congratulations!

What's next?

- In the next module, we'll learn about:
  - Using Latent Dirichlet Allocation (LDA) for topic modeling
  - Evaluating LDA using topic coherence
  - Visualizing LDA topics with pyLDAvis for data exploration
  - Extracting document-topic information
  - Measuring document similarity using cosine similarity metric