Tenn-Share 2019 Beginner Text Mining Workshop

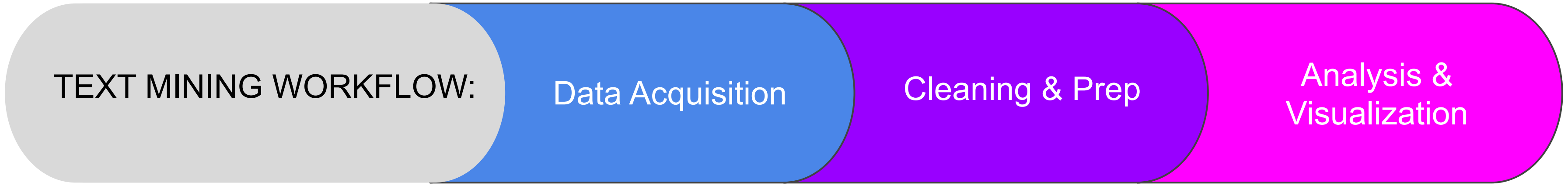
Monica Ihli: monica@utk.edu

Tools of the Trade

- ***Proprietary softwares*** (WordStat, SAS Text Miner).
    - Can enable non-programmers to perform sophisticated analytics.
    - Can still need code for acquiring and cleaning data.

- ***Vendor produced, web-based tools*** (Gale Digital Scholar Lab, HathiTrust Research Center Analytics).
    - Simple to deploy and available through existing deals.
    - Limited to the subscribed content.

- ***Programming libraries and packages*** (R, Python).
    - Learning curve; Requires mastery of highly specialized packages and libraries.
    - Can support complete workflows from acquisition and cleaning to analysis.
    - Choice is largely matter of personal preference & what library you need.


TEXT MINING WORKFLOW:
Data Acquisition
Cleaning & Prep
Analysis & Visualization

Data Acquisition

Forms of Data

- ***Machine Readable forms (less work!):***
  - XML (eXtensible Markup Language)
  - JSON (JavaScript Object Notation)
  - CSV / TSV / other delimited forms

- ***Human readable:***
  - Web pages
  - Documents formatted to be read by people

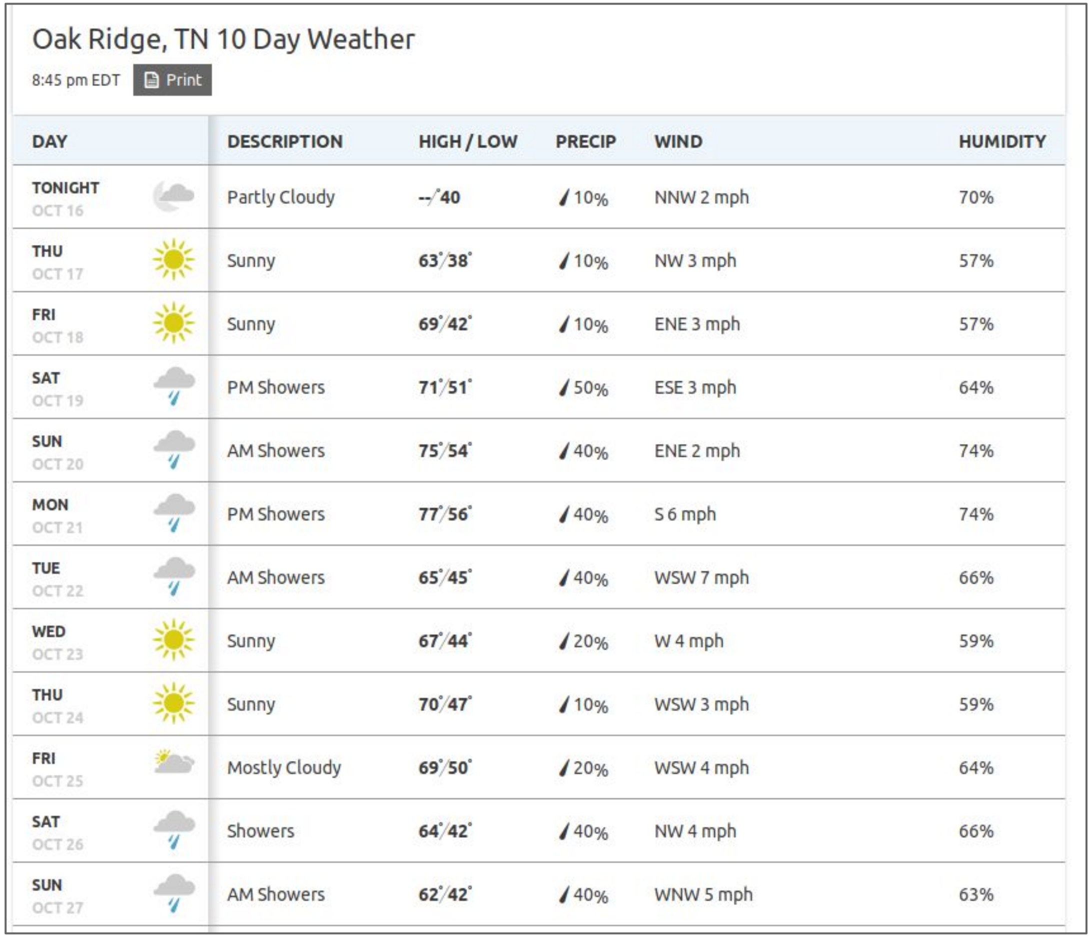
XML

```
<family>
        <people>
                <person>
                        <type>spouse</type>
                        <name>George</name>
                </person>
                <person >
                        <type>child</type>
                        <name>Grace</name>
                </person>
        </people>
        <pets>
                <pet>
                        <type>dog</type>
                        <name>Biscuit</name>
                </pet>
        </pets>
</family>
```

JSON

```
{"family": {
        "people": {
                "person": [
                        {"type": "spouse", "name": "George"},
                        {"type": "child", "name": "Grace"}]},
        "pets": {
                "pet": {"type": "dog", "name": "Biscuit"}}
}}
```

Human Readable



Oak Ridge, TN 10 Day Weather

8:45 pm EDT Print

DAY	DESCRIPTION	HIGH / LOW	PRECIP	WIND	HUMIDITY
TONIGHT OCT 16	Partly Cloudy	--/40	10%	NNW 2 mph	70%
THU OCT 17	Sunny	63°/38°	10%	NW 3 mph	57%
FRI OCT 18	Sunny	69°/42°	10%	ENE 3 mph	57%
SAT OCT 19	PM Showers	71°/51°	50%	ESE 3 mph	64%
SUN OCT 20	AM Showers	75°/54°	40%	ENE 2 mph	74%
MON OCT 21	PM Showers	77°/56°	40%	S 6 mph	74%
TUE OCT 22	AM Showers	65°/45°	40%	WSW 7 mph	66%
WED OCT 23	Sunny	67°/44°	20%	W 4 mph	59%
THU OCT 24	Sunny	70°/47°	10%	WSW 3 mph	59%
FRI OCT 25	Mostly Cloudy	69°/50°	20%	WSW 4 mph	64%
SAT OCT 26	Showers	64°/42°	40%	NW 4 mph	66%
SUN OCT 27	AM Showers	62°/42°	40%	WNW 5 mph	63%

Acquiring Data

- Downloads, exports, and other direct file transfer methods
- Web Services and Application Programming Interfaces (APIs)
- Web scraping.

Jargon: What is a Library?

- A **library** in Python refers to a collection of existing code that allows you to easily add additional functionality to your own programs.

- Usually, libraries are organized around a theme of related tasks, such as processing text (natural language toolkit) or creating plots (matplotlib).

- **Standard library** refers to all the libraries that ship with Python out of the box.

- **Third party libraries** are not part of the official Python release, but can still offer robust and widely implemented solutions (such as requests library).

- In R, the term **package** is typically used instead of library.

Web Services & APIs Jargon

- **API** - functionality for letting different software talk to each other.

- **Web Services** - an implementation of an API for transferring data over http.

- We will use “API” and “web services” interchangeably.

Examples:

- USGS Publications Warehouse:
  https://pubs.er.usgs.gov/documentation/web_service_documentation

- PubMed:
  https://www.ncbi.nlm.nih.gov/home/develop/api/

- Elsevier Scopus / Science Direct/ SciVal:
  https://dev.elsevier.com/

- Library of Congress:
  https://labs.loc.gov/lc-for-robots/

- Twitter:
  https://developer.twitter.com/en/doc

More Web Services & APIs Jargon

- A **request** starts with a **base url** (hostname + api path)
- An API can have multiple **endpoints** -- the locations from which you request different resources.
- Request is fleshed out with **parameters** that tell the server the specific information you are asking for.
- A **rate limit** or **quota** refers to how many requests you are allowed to make in a certain amount of time.
- Some services require you to request an **API Key** (secret code) to use the service (Twitter, Elsevier).

Tools for HTTP & Web Services

- Libraries/packages for working with HTTP allow you to do the same thing with code that your browser does, without any need for a user interface.

- ***Python* example**: requests library.

- ***R* example**: httr package

- A little bit of familiarity with HTTP can be helpful:
    - Status codes of the server response (such as 200 OK, 400 Bad Request) can help you check if your code is working right or not, or tell your script how to respond to different situations.
    - Headers can pass useful information back and forth between you and the server.

Python Workflow for Web Services

Connect

Establish a connection & retrieve content using a tool like ***requests*** library.

Parse

Extract data elements from the returned content using such libraries as ***lxml*** or ***xml.etree.ElementTree**** for XML, or ***json**** for JSON.

Restructure

Use containers like lists and dictionaries to organize the information you are interested in. Data can also be converted to dataframes for additional processing or saved to text files for import into something else.

Anatomy of a Web Services Request: Elsevier

https://api.elsevier.com/content/search/scopus?apiKey=my_api_key&
query=AF-ID(60024266) AND PUBYEAR = 2018

Base url: https://api.elsevier.com/content/search

Endpoint: /scopus

Parameters:

apiKey=my_api_key

query=AF-ID(60024266) AND PUBYEAR = 2018

Anatomy of a Web Services Request: Elsevier

https://api.elsevier.com/content/search/scopus?apiKey=my_api_key&
query=AF-ID(60024266) AND PUBYEAR = 2018

? separates the base url and endpoint from query parameters.

& separates multiple parameters from each other. It designates the end of one parameter and the start of another.

Anatomy of a Web Services Request: Elsevier

Multiple endpoints for the Scopus API:

- ***Search Authors:*** https://api.elsevier.com/content/search/**author**
- ***Search Institutions:*** https://api.elsevier.com/content/search/**affiliation**
- ***Search Abstracts:*** https://api.elsevier.com/content/search/**scopus**

Anatomy of a Web Services Request: PubMed OAI-PMH

https://www.ncbi.nlm.nih.gov/pmc/oai/oai.cgi?verb=ListRecords&from=2019-02-01&until=2019-10-01&set=bmcbioc&metadataPrefix=oai_dc

Base url: https://www.ncbi.nlm.nih.gov/pmc *(API I want to use)*

Endpoint: /oai/oai.cgi *(The resource communication point I'm interested in)*

Parameters:

- ***verb=***ListRecords *(From OAI-PMH standard, what I'm asking for)*
- ***from=***2019-02-01 *(Start date for interval of records I want)*
- ***until=***2019-10-01 *(End date for the records I'm interested in)*
- ***set=***bmcbioc *(Only records in this category)*
- ***metadataPrefix=***oai_dc *(The metadata format I want results in)*

Anatomy of a Web Services Request: PubMed

https://www.ncbi.nlm.nih.gov/research/bionlp/RESTful/pubmed.cgi/BioC_xml/17299597/unicode

Parameters:

- ***Format***: BioC_xml
- ***PubMed ID of Record***: 17299597
- ***Character Encoding:*** unicode

Web Scraping

- Attempts to access and transform content intended for humans into something machines can work with.

- Process still includes:
    - Establish connection & retrieve content.
    - ***Parse out the elements of interest.***
    - Restructure the parsed content into something we can use.

- **Beautiful Soup** (beautifulsoup4) is a Python library that extends common parsers for pulling data out of HTML and XML files.

- Beautiful Soup handles the messiness of HTML content and even sloppy XML.

- Disadvantage: You can’t use the same script on a different web page.

Web Scraping Pages using Selenium

- Sometimes web pages aren't fully rendered in the moment you make the request. A page with JavaScript can take time to finish processing and loading the content.

- This means that whatever you get back from that HTTP request isn't going to have the full page content.

- Solution: Use ***Selenium*** & ***chromedriver--*** Take control of a browser (Chrome) programmatically using a special driver.

- Allows page to fully load before actually retrieving its contents.

Python Summary

HTTP Tools:

requests - Let’s you make http requests programmatically and returns the resulting content.

Selenium & Chromedriver - Uses a driver to control your Chrome browser using Python. Useful when a web page takes time to load, and situations where you want to interact with the page as if you were a human user.

Python Summary

Parsing Tools:

xml.etree.ElementTree - standard library XML parser

lxml - similar to standard library ElementTree, but some more advanced functionality and xpath support.

json - standard library for converting json to Python data types for further manipulation in a script.

beautifulsoup4 - Beautiful Soup (bs4) is an XML and HTML parsing library builds added functionality on top of parsers like lxml to handle the messiness of parsing HTML and can (often) handle problematic XML without breaking.

Cleaning & Preparing Data

Data Types in General

Refers to the type of variable you are working with:

- ***Text*** such as characters or strings *(string is jargon for text)*

- ***Numeric*** types such as integers and floating point numbers (numbers with decimals)

- ***Boolean*** values of True or False

Python Container Types

List:

- holds a sequence of items
- Unlike arrays in other language, can hold different types of items.
- Items are referenced by their index

```
shopping_list = [‘milk’, ‘bread’, ‘cereal’]
```

Looks like:

```
[0][‘milk’]
[1][‘bread’]
[2][‘cereal’]
```

Python Container Types

Dictionary:

- holds a sequence of items and their labels/keys.
- Elements are referenced by their keys.
- Can be ordered or unordered.
- There is such a thing as a list of dictionaries. :D

```
person = {‘name’ : ‘Shrek’,
          ‘age’ : 108,
          ‘occupation’: ‘ogre’}
```

Looks like:

```
[‘name’][‘Shrek’]
[‘age’][108’]
[‘occupation’][‘ogre’]
```

Python Data Structures

Pandas Dataframes:

- Powerful and designed for data analysis.
- Like storing an entire table in memory.
- Can convert other things to a dataframe, like a CSV file or a list of dictionaries.
- Works well together with numpy library for serious numbers crunching.

String Operations

- Strings are the variables we care the most about.
- Data is DIRTY! It’s full of stuff we don’t want!
- String operations are the bread and butter of data cleaning.
- Many concepts of string manipulation are common across languages.

Common String Operations

Remove extra **whitespace** (tabs, newlines, spaces): |

“\t\t\t*Shrek 2*   \t\t\n” should become   “*Shrek 2*”.

Split up a string into parts based on spaces, words, punctuation, or other values:

“*This is a sentence*” becomes:

“*This*”

“*Is*”

“*a*”

“*sentence*”

Common String Operations

Join two strings together (concatenation):

“String A” plus *“ and “* plus *“String B”* becomes *“String A and String B”*

Find some words or characters within a string.

“Donkey is Shrek’s lovable sidekick”
The word “key” is found at index [3] in this string.

Common String Operations

Replace a segment of text with something else:

“Donkey is Shrek’s lovable sidekick.” becomes *“Donkey is Shrek’s lovable friend.”*

Remove a segment of text altogether:

“Who’s <br> swamp is it?<br>” becomes *“Who’s swamp is it?”*

Convert to upper or lower case:

“Hello World” becomes *“HELLO WORLD”* or *“hello world”*.

Regular Expressions

- Gives us a powerful tool for matching patterns in text, so that we can then extract subsections or perform string operations on them.

- Implemented in Python, R, php, and many other languages.

- Take some practice. Use tools like **https://regex101.com/** to help you get the hang of it.

- Example - Find the 4 digit publication year in a string:
  ```
  re.search("\d{4}", ““1999-05-01”)
  ```

Text Preprocessing Steps

Uniform Case - Converting to all lowercase usually.

Tokenizing - Breaking up text into component words, keywords, or phrases called tokens.

Stemming - Algorithms for normalizing words by chopping off the ends of words in order to try to come up with a consistent form.

Lemmatization - Algorithms for normalizing words by converting variants to a standard form.

Stopword Removal - Removing unimportant words like “a” or “and” or “the”.

Analysis & Visualization

Python Libraries for Data Processing & Analysis

- **pandas** for new data containers, data prep and transformation.
- **Numpy** for numerical processing and matrices.
- **Natural Language Toolkit** for text processing and working with human language.
- **SciKit-Learn** for machine learning based techniques.

Some R packages for Data Processing & Analysis

- Tidyverse - a collection of powerful text cleaning, manipulation, processing, and analytics packages that can be installed separately or as the whole tidyverse..

- Rweka - Machine learning algorithms

- tm - text mining framework for preparing and analyzing data in R.

Example #1: Word Frequency

- Once data is acquired (via export, web service, or web scraping), it takes critical thinking to identify the appropriate next steps for cleaning and prepping the data.

- Basic string processing techniques, and regular expressions can help you:
    - Get rid of unwanted stuff like HTML tags.
    - Replace name or word variants with a canonical form for terms outside of stopword lexicons.
    - Clear out punctuation and numbers.

Example #1: Word Frequency

- Prepping data employs all the steps we’ve talked about:
    - Uniform case.
    - Tokenizing
    - Stopword removal
    - Stemming or lemmatization.

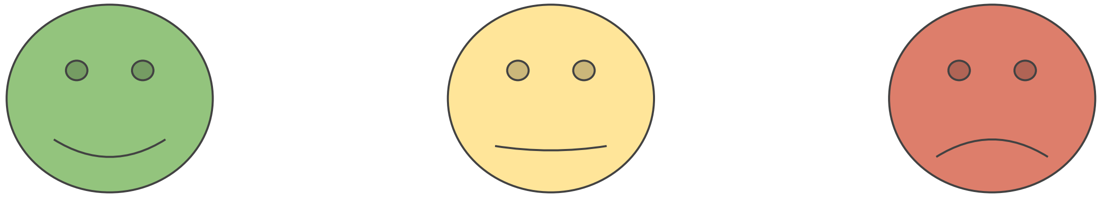
- The advantage of doing these steps rather than just dumping text straight into some word-frequency or word cloud software is that you get higher quality, more meaningful results.

Example #1: Word Frequency

- **NLTK** handles the text mining preprocessing steps and word frequency:
    - from nltk.tokenize import **word_tokenize** # *breaks text up into lists of words*
    - from nltk.corpus import **stopwords** # *removes unimportant words from text*
    - from nltk import PorterStemmer # *converts words to a stemmed form*
    - from nltk.stem import **WordNetLemmatizer** #*standardize words*
    - from nltk import **FreqDist** # *allows us to report frequency of words in text*

Example #2: Sentiment Analysis

Sentiment analysis is a process of computationally characterizing text as an expression of positive or negative attitude on behalf of the communicator.



Example #2: Sentiment Analysis

Two main approaches:

- **Lexicon based** - use a predefined dictionary of words which someone has defined as positive or negative.

- **Machine Learning Classifier**:
  - Create a training set of text data. A well-known example from Stanford is a set of movie reviews classified as good or bad reviews.
  - Use a ML algorithm to figure out what are the types of words and other features that characterize a positive versus a negative case.
  - Then apply the model that's been developed to your data.

Example #2: Sentiment Analysis

- Lexicon-based approach is easier.
- Our example will implement a sentiment analysis of Library chat logs, with a lexicon in R using tidytext package.
- We also employ the same preprocessing steps to get ready for sentiment analysis!