# PS 643 (AUG 30) Class Notes

## Papers to Read:

The global landscape of AI ethics guidelines
https://www.nature.com/articles/s42256-019-0088-2
Ethics as an Escape from Regulation by Ben Wagner
https://www.jstor.org/stable/pdf/j.ctvhrd092.18.pdf
Governance with Teeth by Vidushi Marda
https://www.article19.org/wp-content/uploads/2019/04/Governance-with-teeth_A19_April_2019.pdf
How to recognize AI snake oil by Arvind Narayanan
https://www.cs.princeton.edu/~arvindn/talks/MIT-STS-AI-snakeoil.pdf

The papers will be discussed in next week's classes.

## Handling local files:

### 1) Writing into the files

```
f = open("document.txt",'w')

f.write("a quick brown fox jumped over the lazy dog")
f.close()

#f.write will override the existing data.

#if we want to append to the existing data, open the file with 'a'

f.open("document.txt", 'a')

#if want the new data to be added in next line add \n in data where you want to start a new line.
```

### 2) Reading the files.

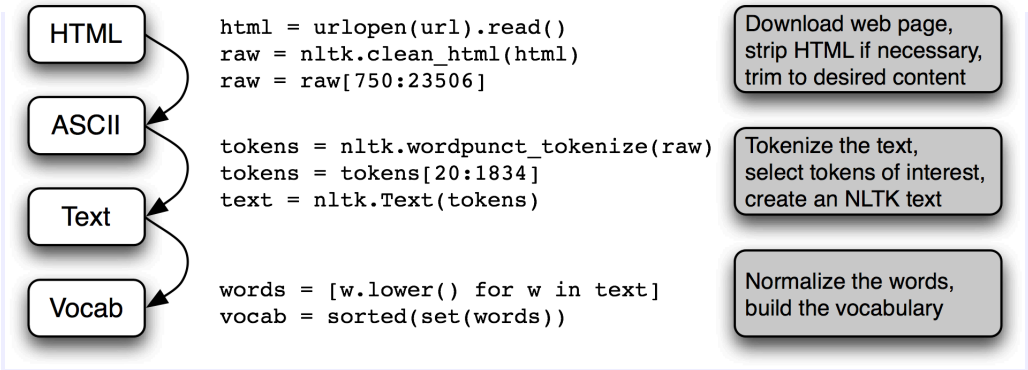
```
f.open("document.txt",'r')

for i in f:
        print(i)

f.close()

#it prints all the lines in the file
```

## NLP Pipeline




## Regular Expressions

Tool to find patterns in text

```
import re
wordlist =[w for w in wordlist if re.search("^def",w)]
wordlist=[w for w in wordlist if re.search("def$", w)]
```

^ tries to search words starting with the given expression.
$ tries to search words ending with the given expression.

| Operator | Behavior |
|---|---|
| . | Wildcard, matches any character |
| ^abc | Matches some pattern *abc* at the start of a string |
| abc$ | Matches some pattern *abc* at the end of a string |
| [abc] | Matches one of a set of characters |
| [A-Z0-9] | Matches one of a range of characters |
| ed\|ing\|s | Matches one of the specified strings (disjunction) |

| | |
|---|---|
| * | Zero or more of previous item, e.g. a*, [a-z]* (also known as *Kleene Closure*) |
| + | One or more of previous item, e.g. a+, [a-z]+ |
| ? | Zero or one of the previous item (i.e. optional), e.g. a?, [a-z]? |
| {n} | Exactly *n* repeats where n is a non-negative integer |
| {n,} | At least *n* repeats |
| {,n} | No more than *n* repeats |
| {m,n} | At least *m* and no more than *n* repeats |
| a(b\|c)+ | Parentheses that indicate the scope of the operators |

### Stemmers

What do they do:

- Remove suffixes and prefixes from words, aiming to get the core meaning.
- For example, "running", "runs", and "runner" would be reduced to "run".

Why they are used:

- Standardize words with similar meaning, improving the efficiency of NLP algorithms.
- Reduce the dimensionality of text data, making it easier to process and analyze.

### Lemmatization

The WordNet lemmatizer only removes affixes if the resulting word is in its dictionary. This additional checking process makes the lemmatizer slower than the above stemmers. Notice that it doesn't handle *lying*, but it converts *women* to *woman*.

## POS Tagging:

The process of classifying words into their **parts of speech** and labeling them accordingly is known as **part-of-speech tagging**, **POS-tagging**, or simply **tagging**.

Parts of speech are also known as **word classes** or **lexical categories**.
The collection of tags used for a particular task is known as a **tagset**.

```
text = word_tokenize("And now for something completely different")
>>> nltk.pos_tag(text)
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'),
   ('completely', 'RB'), ('different', 'JJ')]
```

# Automatic Tagging

## 1) The Default Tagger

- The simplest possible tagger assigns the same tag to each token.

## 2)The Regular Expression Tagger

- The regular expression tagger assigns tags to tokens on the basis of matching patterns.
- For instance, we might guess that any word ending in *ed* is the past participle of a verb, and any word ending with *'s* is a possessive noun. We can express these as a list of regular expressions.

## 3) The Lookup Tagger