Lambton College
In Toronto

Lambton College
School of Computer Studies

AML-2304 Natural Language Processing

[Prepared By: Vahid Hadavi, PhD]

Lecture1

1. Evaluate the key concept of Natural Language Processing (NLP)

Major Topics:

- 1.1 Define Natural Language Processing
- 1.2 Describe the Challenges of NLP; extraction of meaning from text
- 1.3 Discuss the importance of Natural Language Processing

1.1 Define Natural Language Processing

Natural Language Processing

Definition

- English, Chinese, Spanish are natural languages
- Python, Java are artificial languages
- Natural language processing is any computation, manipulation of natural languages to get insights
- Natural Language Processing is a subfield of Artificial Intelligence.

Main goal

- Computer reads, deciphers and understands the human languages, and acts based on its understanding.

Natural Language Processing

History

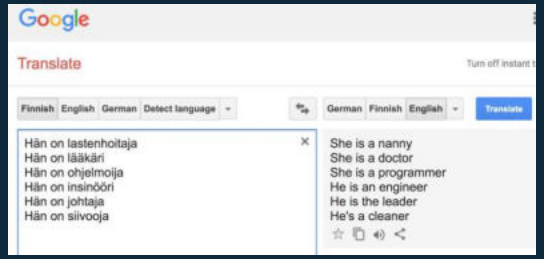
- World War 2; Idea came to existence for creating a machine that translates
- Up to the 1980s; NLP systems were based on complex hand-written mathematical rules.
- Late 1980s; a revolution in NLP with the introduction of machine learning (ML) algorithms for language processing.
- Recent years; a significant breakthrough in NLP due to the emergence of the subfield of ML called Deep Learning (DL).

NLP on Python

- In NLP, full automation is achieved using modern software libraries and packages.
- Natural Language Toolkit (NLTK), a platform used for building Python programs that work with human language data for applying in statistical NLP.
- Written by Steven Bird and Edward Loper at the University of Pennsylvania in 2001.

NLP Applications

Translation



Search engine

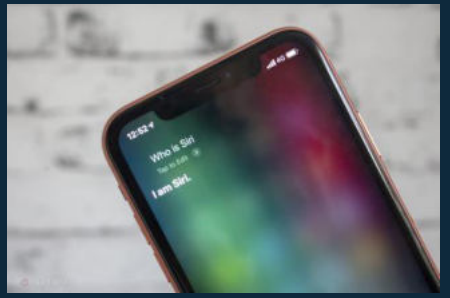
Sentiment analysis

Spam filtering

NLP Applications

Banking Fraud detection

Question answering



Information extraction

Healthcare

1.2 Describe the Challenges of NLP; extraction of meaning from text

Data-related challenges

Low resource languages

- To train computers to understand a natural language, many data resources are required:
  - A high amount of raw text from various genres; books, scientific papers, etc.
  - Lexical, syntactic and semantic resources; dictionaries, dependency tree corpora, etc.
- There are more than 7000 natural languages.
- More than 1000 languages alone in Africa with very low resources.

Dealing with large or multiple documents

- Hard to read entire books and movie scripts.
- Current models cannot represent longer contexts well.

Natural Language Understanding

Ambiguity

- Lexical ambiguity; words have different meanings:
  - "I saw a **bat"**
- Syntactic ambiguity; a sentence has multiple parse trees:
  - "I shot the man with the ice cream" → A man with the ice cream was shot.
  - "I shot the man with the ice cream" → I shot the man with my ice cream.

Intention, emotions

- Intention and emotion depend on author's personality.
- People use different styles to express the same idea.
- Irony or sarcasm may convey a meaning that is opposite to the literal one.

Natural Language Understanding

Synonymy

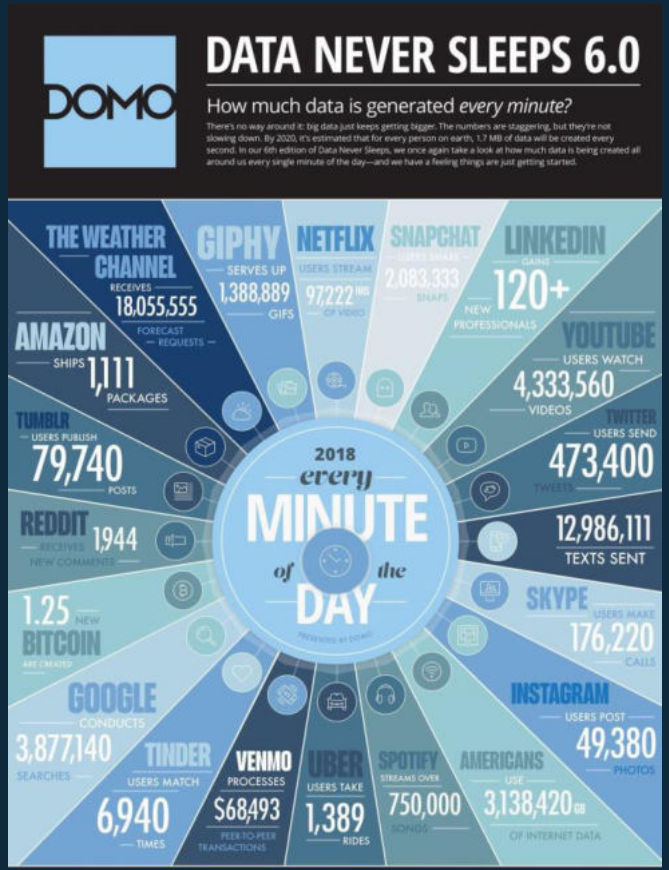
- we can express the same idea with different terms:
  - Big & Large → In some contexts they can be interchangeably used.
    → In some contexts they cannot be interchangeably used.
    .

Coreference

- coreference resolution; the process of finding all expressions that refer to the same entity in a text.
  - "Tom dropped the glass jar by accident and broke it" → What does it refer to?
- Important step for a lot of higher-level NLP tasks; document summarization, question answering

1.3 Discuss the importance of Natural Language Processing

Why NLP is important?



Why NLP is important?

Handling large volumes of text data

- With the big data technology, NLP has entered the mainstream as this approach can now be applied to handle large volumes of text data via cloud/distributed computing at an unprecedented speed.
  - Imagine you're given a sentence and tasked to identify if this sentence has **positive/negative/neutral sentiment**, manually.
  - Imagine now you're given millions of sentences and perform the sentiment analysis again. How long would that take you?

Structuring highly unstructured data source

- Describing text data as unstructured data is an understatement.

Text data is a mess

- NLP helps to structure texts and add convert them to numerical structures

Any questions so far?
Any comments?

Lambton College
In Toronto

Lambton College
School of Computer Studies

AML-2304 Natural Language Processing

[Prepared By: Vahid Hadavi, PhD]

Lecture2

2. Evaluate the application of NLP in various Canadian businesses

Major Topics:

- 2.1 Discuss the usage of NLP in Customer Service
- 2.2 Describe how NLP is used in Market Intelligence
- 2.3 Discuss the applicability of NLP for Sentiment Analysis

2.1 Discuss the usage of NLP in Customer Service

Customer Service

Definition

- The support offered to customers, both before and after they buy and use products or services, that help them have an easy and enjoyable experience.

Based on Call Center research

- 25% of customers pay more for better customer experience.
- 64% of consumers expect real-time responses at any time.
- 62% of customers consider switching to a competitor after only 1-2 bad experiences.

NLP in Customer Service

Chatbots

- Use of chatbots in customer service industry is growing; 85% of all customer interactions.
- Based on the estimations, 40% of consumers do not care whether a chatbot or a real human helps them, if they are getting the help.

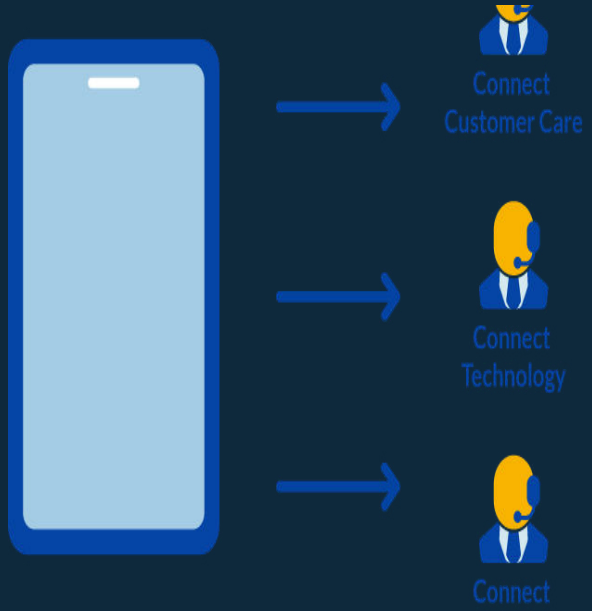
- With NLP:
  - ✓ Chatbots will be able to handle increasingly complex consumer communications.
  - ✓ Make personalization possible. Chatbots will be able to remember past conversations with users to customize questions, responses, and options.

NLP in Customer Service

IVR technology

- Interactive Voice Response
- Ask callers specific questions and route them through pre-programmed paths

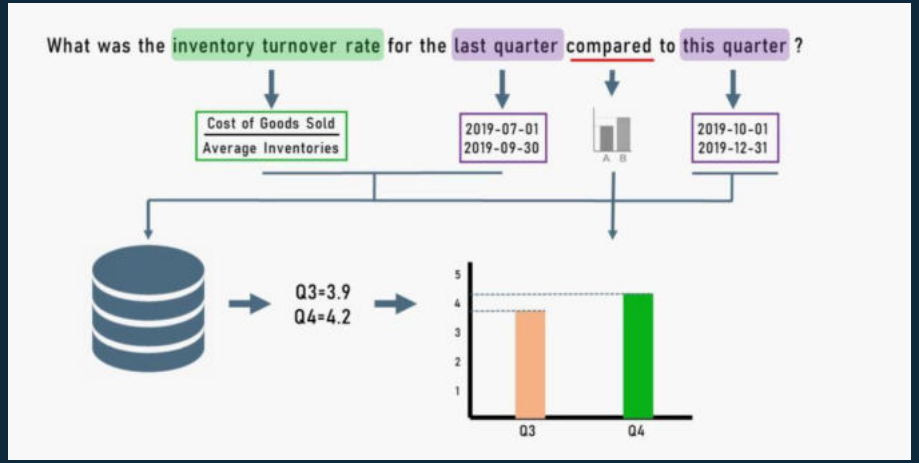
- With NLP:
  - ✓ Callers tell the system what they need in their own words.
  - ✓ No need to use certain phrases and speak in a stiff manner.
  - ✓ Possible for the system to ask open-ended questions.
  - ✓ Understands context in replies.
  - ✓ Responds to caller requests to repeat something or pause.



2.2 Describe how NLP is used in Market Intelligence

Market Intelligence with NLP

- Most business markets are much impacted and influenced by market knowledge and information exchange among various companies, stakeholders and governments.
- NLP is a great tool for monitoring market intelligence reports to extract new information that can help businesses to produce new strategies.
- It empowers more users to explore and manipulate data in search of valuable insights, the more likely those insights will lead to better business decisions.



2.3 Discuss the applicability of NLP for Sentiment Analysis

Sentiment Analysis

Definition

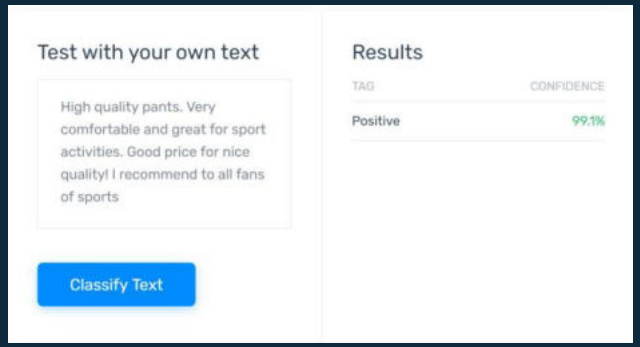
- Sentiment Analysis is the process of determining whether a piece of writing is positive, negative or neutral.

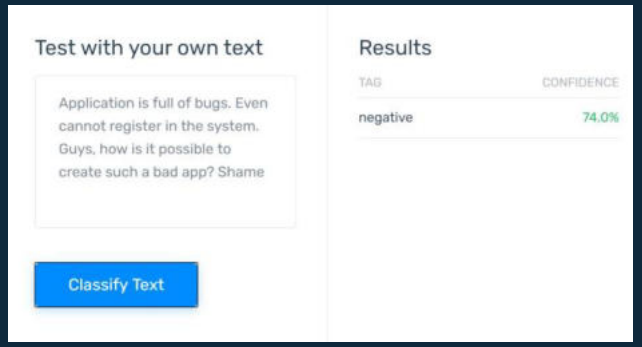
Use cases and applications

- Social media monitoring
- Brand monitoring
- Voice of customer
- Customer service
- Market research

Sentiment analysis with NLP

- One of the most popular applications of NLP.
- Key aspect of sentiment analysis is to analyze a body of text for understanding the opinion expressed by it.
- We quantify this sentiment with a positive or negative value, called polarity.





- We can analyze the sentiment of large texts, or thousands of reviews and extract keywords with high accuracy.

Any questions so far?
Any comments?

Lambton College In Toronto

Lambton College

School of Computer Studies

AML-2304 Natural Language Processing

[Prepared By: Vahid Hadavi, PhD]

Lecture3

3. Explain Text Wrangling and Cleansing

Major Topics:

- 3.1 Text Wrangling
- 3.2 Discuss Sentence Splitting
- 3.3 Define Word Tokenization
- 3.4 Explain Stemming and Lemmatization
- 3.5 Discuss Stop Words
- 3.6 Discuss Regular Expression (Regex)
- 3.7 Explain Bag-of-Words
- 3.8 Discuss TF-IDF

3.1 Text Wrangling

Text Wrangling

- Text wrangling is the pre-processing work that's done to prepare raw text data ready for training a machine learning model.
- Simply put, it's the process of cleaning your data to make it readable by your program.
- It involves:
  - Sentence splitting
  - Tokenization
  - Stemming and lemmatization
  - Stop word removal.
- To start Natural Language Toolkit (NLTK) needs to be installed.

```
import nltk
nltk.download('all')
```

3.2 Discuss Sentence Splitting

Sentence splitting

- Sentence splitting (tokenization) is the process of splitting text into individual sentences.
- sent_tokenize on NLTK is used to split text into sentences.
- It needs the text name and the language of the text (English as default) as input.

```
samplestring = 'Let's make this our sample paragraph. It even knows that the period in Mr. Jones is not the end. Try it out!'
from nltk.tokenize import sent_tokenize
tokenized_sent = sent_tokenize(samplestring)
print(tokenized_sent)
```

```
['Let's make this our sample paragraph.', 'It even knows that the period in Mr. Jones is not the end.', 'Try it out!']
```

- It recognizes that the "." in Mr. is not a period ending a sentence.

3.3 Define Word Tokenization

Word tokenizing

- A token is the smallest text unit a machine can process.
- Every chunk of text needs to be tokenized before you running natural language programs on it.
- Python split() can be used to tokenize a text.

```
msg = 'Hey everyone! The party starts in 10mins. Be there ASAP!'
print(msg.split())
```

```
['Hey', 'everyone!', 'The', 'party', 'starts', 'in', '10mins.', 'Be', 'there', 'ASAP!']
```

- Word_tokenize on NLTK can be used to tokenize a text.
- It needs the text name and the language of the text (English as default) as input.

```
from nltk.tokenize import word_tokenize
tokenized_word = word_tokenize(msg)
print(tokenized_word)
```

```
['Hey', 'everyone', '!', 'The', 'party', 'starts', 'in', '10mins', '.', 'Be', 'there', 'ASAP', '!']
```

- Their outputs are slightly different. What are the differences?

3.4 Explain Stemming and Lemmatization

Stemming

- Stemming is a text normalization technique used to extract the base form of the words by removing affixes from them.
- Stemming algorithms work by cutting off the end or the beginning of the word, considering a list of common prefixes and suffixes that can be found in an inflected word.
  - Example: Jump-*ing*, Jump-*ed*, Jump-*s* → **Jump**
- There are different algorithms in python for stemming.
- The most common stemming algorithm in English is Porter stemmer.
- It needs a word or a list of tokens as input.

```
from nltk.stem import PorterStemmer
porter = PorterStemmer()
porter.stem('running')
```

```
'run'
```

- There are other algorithms such as Snowball Stemmer that works well with other languages.

Stemming drawbacks

```
from nltk.stem import PorterStemmer
porter = PorterStemmer()
porter.stem('ate')
```

```
'ate'
```

- We expected to see “eat”.

```
from nltk.stem import PorterStemmer
porter = PorterStemmer()
print(porter.stem('university'), porter.stem('universal'))
```

```
univers univers
```

- “university” and “universal” are two different words with completely different meanings. Stemming returned a same root for both

Lemmatization

- Lemmatization is a more advanced text normalization technique.
- It considers context and part of speech to determine the lemma.
- A lemma (plural lemmas or lemmata) is the canonical form, dictionary form of a set of words.
- Python NLTK provides WordNet Lemmatizer that uses the WordNet Database to lookup lemmas of words.
- Wordnet is database like a web of synonyms or a thesaurus.
- Because lemmatization involves deriving the meaning of a word from something like a dictionary, it's very time consuming.
- You need to provide the context in which you want to lemmatize that is the parts-of-speech (POS).
- You need to provide the word and pos (contect) as the input.
- pos constants:
  - ADJ, ADJ_SAT, ADV, NOUN, VERB = 'a', 's', 'r', 'n', 'v'

Lemmatization

```
from nltk.stem import WordNetLemmatizer
lem = WordNetLemmatizer()
print(lem.lemmatize("running", pos = 'n'))
print(lem.lemmatize("running", pos = 'v'))
```

```
running
run
```

```
from nltk.stem import WordNetLemmatizer
lem = WordNetLemmatizer()
print(lem.lemmatize("better", pos = 'a'))
print(lem.lemmatize("ate", pos = 'v'))
```

```
good
eat
```

3.5 Discuss Stop Words

Stop Words

- Stop words are commonly-used word that are usually ignored because of their many occurrences.
- They do not add much meaning to a sentence.
- Most of these words are articles and prepositions, such as “the”, “a”, “in”, etc.
- These words can either end up taking too much space or eating up too much time.
- NLTK has a list of stop words in 16 different languages.

```
from nltk.corpus import stopwords
stopwords = stopwords.words('english')
```

- First 10 words in the stopwords list:

```
stopwords[:10]
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

Stop Words removal

```
text = 'A sentence is a textual unit consisting of words that are grammatically linked.'
```

- First, we tokenize the text.

```
from nltk.tokenize import word_tokenize
tokenized_word = word_tokenize(text)
tokenized_word
```

```
['A',
 'sentence',
 'is',
 'a',
 'textual',
 'unit',
 'consisting',
 'of',
 'words',
 'that',
 'are',
 'grammatically',
 'linked',
 '.']
```

Stop Words removal

- Now we filter out the stopwords.

```
from nltk.corpus import stopwords
stopwords = stopwords.words('english')
filtered_tokens = []
for token in tokenized_word:
    if token not in stopwords:
        filtered_tokens.append(token)
filtered_tokens
```

```
['sentence',
 'textual',
 'unit',
 'consisting',
 'words',
 'grammatically',
 'linked',
 '.']
```

‘a’, ‘is’, ‘that’, ‘are’ and ‘of were removed.

3.6 Discuss Regular Expression (Regex)

Regular expression (Regex)

Definition

- A fascinating programming tool available within most of the programming languages.
- A regular expression is a set of characters, or a pattern, which is used to find sub strings in a given string..
- For example, extracting all hashtags from a tweet, getting email id or phone numbers etc.
- To use it in python, you need to “import re”

Common Regex Functions

- re.search(patterns, text)
  - Detects whether the given regular expression pattern is present in the given text.
- re.match(patterns, text)
  - It returns a match object on success, None on failure.
- re.sub(Pattern, Substitute, text)
  - It replaces the Pattern in the text with the Substitute.
- re.findall(Pattern, text)
  - It returns a list of all the matches.

Regular expression

```
import re
tweet = 'Hello guys, this is my first #tweet. Check this @joy! #nlp #machine #learning'
```

```
re.findall('#', tweet)
```

```
['#', '#', '#', '#']
```

```
re.sub('#', '_', tweet)
```

```
'Hello guys, this is my first _tweet. Check this @joy! _nlp _machine _learning'
```

Special Sequences

- A special sequence is a \ followed by one of the characters in the list below, and has a special meaning.

Character	Description	Example
\A	Returns a match if the specified characters are at the beginning of the string	"\AThe"
\b	Returns a match where the specified characters are at the beginning or at the end of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\bain" r"ain\b"
\B	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\Bain" r"ain\B"
\d	Returns a match where the string contains digits (numbers from 0-9)	"\d"
\D	Returns a match where the string DOES NOT contain digits	"\D"
\s	Returns a match where the string contains a white space character	"\s"
\S	Returns a match where the string DOES NOT contain a white space character	"\S"
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	"\w"
\W	Returns a match where the string DOES NOT contain any word characters	"\W"
\Z	Returns a match if the specified characters are at the end of the string	"Spain\Z"

Meta Characters

- Metacharacters are characters with a special meaning.

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"world$"
*	Zero or more occurrences	"aix*"
+	One or more occurrences	"aix+"
{}	Exactly the specified number of occurrences	"al{2}"
|	Either or	"falls|stays"
()	Capture and group	

Meta Characters

- A set is a set of characters inside a pair of square brackets [] with a special meaning.

Set	Description
[arn]	Returns a match where one of the specified characters (`a` , `r` , or `n`) are present
[a-n]	Returns a match for any lower case character, alphabetically between `a` and `n`
[^arn]	Returns a match for any character EXCEPT `a` , `r` , and `n`
[0123]	Returns a match where any of the specified digits (`0` , `1` , `2` , or `3`) are present
[0-9]	Returns a match for any digit between `0` and `9`
[0-5][0-9]	Returns a match for any two-digit numbers from `00` and `59`
[a-zA-Z]	Returns a match for any character alphabetically between `a` and `z` , lower case OR upper case
[+]	In sets, `+` , `*` , `.` , `|` , `()` , `$` , `{}` has no special meaning, so `[+]` means: return a match for any `+` character in the string

Examples

#1

```
text = 'hello 12 hi 89. Howdy 34'
```

```
pattern = '\d+'
result = re.findall(pattern, string)
print(result)
```

```
['12', '89', '34']
```

#2

```
text = 'abc 12 de 23 f45 6'
```

```
pattern = '\s+'
replace = ''
new_text = re.sub(pattern, replace, string)
print(new_string)
```

```
abc12de23f456
```

Examples

#3

```
text = '#UNSG @ NY Society for Ethical Culture bit.ly/2gyVelr @UN @UN_Women '
```

```
tokens = text.split()
[w for w in tokens if  re.search('@[a-zA-Z]+',w)]
```

```
['@UN', '@UN_Women']
```

#4

```
time_sentences = "Monday: The doctor's appointment is at 2:45pm.Tuesday: The dentist's appointment is at 11:30 am"
```

```
re.findall(r'(\d?\d):(\d\d)', time_sentences)
```

```
[('2', '45'), ('11', '30')]
```

3.7 Explain Bag-of-Words

Bag-of-Words

- Machine learning algorithms cannot work with raw text directly; the text must be converted into numbers. Specifically, vectors of numbers.
- This is called feature extraction or feature encoding.
- Bag-of-Words is a popular and simple method of feature extraction with text.
- A bag-of-words is a representation of text that describes the occurrence of words within a document.
- It involves two things:
  - A vocabulary of known words.
    Constructing a document corpus which consists of all the unique words in the whole text.
  - A measure of the presence of known words.
    Counting the number of occurrences for each unique word.

Example

- Consider the below two:
  - John likes to watch movies. Mary likes movies too.
  - John also likes to watch football games.
- They can be represented by following:

  {"John":1, "likes":2, "to":1, "watch":1, "movies":2, "Mary":1, "too":1}

  {"John":1, "also":1, "likes":1, "to":1, "watch":1, "football":1, "games":1}
- Or the following:

  [1, 2, 1, 1, 2, 1, 1, 0, 0, 0]

  [1, 1, 1, 1, 0, 0, 0, 1, 1, 1]

Bag-of-Words

- We use scikit-learn's CountVectorizer to implement bag-of-words.

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
```

```
text_data = np.array(['I love Brazil. Brazil!',
                      'Sweden is best',
                      'Germany beats both'])
```

```
count = CountVectorizer()
bag_of_words = count.fit_transform(text_data)
```

- The output is a sparse matrix

```
bag_of_words

<3x8 sparse matrix of type '<class 'numpy.int64'>'
    with 8 stored elements in Compressed Sparse Row format>
```

Bag-of-Words

- We can use toarray to view a matrix of word counts for each observation.

```
bag_of_words.toarray()

array([[0, 0, 0, 2, 0, 0, 1, 0],
       [0, 1, 0, 0, 0, 1, 0, 1],
       [1, 0, 1, 0, 1, 0, 0, 0]], dtype=int64)
```

- We can use the vocabulary_ method to view the word associated with each feature

```
count.get_feature_names()

['beats', 'best', 'both', 'brazil', 'germany', 'is', 'love', 'sweden']
```

3.8 Discuss TF-IDF

TF-IDF

- A problem with scoring word frequency (bag-of-words) is that highly frequent words start to dominate in the document.
- Highly frequent words may not contain as much “informational content” to the model as rarer but perhaps domain specific words.
- One approach is to rescale the frequency of words by how often they appear in all documents.
- the scores for frequent words like “the” that are also frequent across all documents are penalized.
- Term Frequency (TF) is a scoring of the frequency of the word in the current document.
- Inverse Document Frequency (IDF) is a scoring of how rare the word is across documents

TF-IDF

- We use scikit-learn's TfidfVectorizer to implement bag-of-words.

```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
text_data = np.array(['I love Brazil. Brazil!',
                      'Sweden is best',
                      'Germany beats both'])
```

```
tfidf = TfidfVectorizer()
feature_matrix = tfidf.fit_transform(text_data)
```

- The output is a sparse matrix

TF-IDF

- We can use .toarray to see the feature matrix.

```
array([[ 0.        ,  0.        ,  0.        ,  0.89442719,  0.        ,
         0.        ,  0.4472136 ,  0.        ],
       [ 0.        ,  0.57735027,  0.        ,  0.        ,  0.        ,
         0.57735027,  0.        ,  0.57735027],
       [ 0.57735027,  0.        ,  0.57735027,  0.        ,  0.57735027,
         0.        ,  0.        ,  0.        ]])
```

- We can use the vocabulary_ method to view the word associated with each feature

```
tfidf.vocabulary_

{'beats': 0,
 'best': 1,
 'both': 2,
 'brazil': 3,
 'germany': 4,
 'is': 5,
 'love': 6,
 'sweden': 7}
```

Any questions so far?
Any comments?

Lambton College In Toronto

Lambton College

School of Computer Studies

AML-2304 Natural Language Processing

[Prepared By: Vahid Hadavi, PhD]

Lecture4

4. Demonstrate Text Preprocessing: Replacing and Correcting Words

Major Topics:

- 4.1 Illustrate Text conversion to Lowercase
- 4.2 Apply Number Removal, Punctuation Removal, and Whitespace Removal
- 4.3 Use Parts of Speech Tagging (POS)
- 4.4 Practice Named Entity Recognition
- 4.5 Show Collocation Extraction and Synonyms
- 4.6 Demonstrate use of Word Lengthening to detect sentiment
- 4.7 Apply Spell Correction

4.1 Illustrate Text conversion to Lowercase

Text conversion to Lowercase

- It may be necessary for to convert a text into lowercase.
- We can use lower() function provided by Python on our text to convert it into lowercase.

```
myString = "The 5 countries include China, United States,
Indonesia, India,and Brazil."
```

```
str = myString.lower()
print(str)
```

```
the 5 countries include china, united states, indonesia, india,and brazil.
```

4.2 Apply Number Removal, Punctuation Removal, and Whitespace Removal

Number removal

- You may not want to work with numbers in your analysis.
- Number removal can be done in Python using regular expressions.

```
import re
```

```
myString = 'Box A has 4 red and 6 white balls, while Box B has 3 red and 5 blue balls.'
```

```
output = re.sub(r'\d+', '', myString)
print(output)
```

```
Box A has  red and  white balls, while Box B has  red and  blue balls.
```

Punctuation removal

- We may want to remove punctuations from our text for easy processing.
- Examples of such symbols include #, $, %, *, & (), +, -, ., /, :, ;, <=>, ?, @, [, \,], ^, _, `, {, |, }, ~,].

```
import string
```

```
myString = "This &is [a] string? {with} many. punctuation.? marks!!!!"
```

```
text.translate(str.maketrans('', '', string.punctuation))
```

White space removal

- We may want to work with text without leading and ending spaces.
- You can do away with these from your text by calling the strip() method.

```
myString = " \t a sample string\t "
```

```
str = myString.strip()
print(str)
```

```
a sample string
```

4.3 Use Parts of Speech Tagging (POS)

POS

- The goal of POS is to assign the various parts of a speech to every word of the provided text.
- This is normally done based on the definition and the context.
- There are various tools that provide us with POS taggers, including NLTK, TextBlob, etc.
- In this lecture, we will use TextBlob.
- It needs to be installed (pip for windows).

```
pip3 install textblob
```

```
from textblob import TextBlob
import nltk
```

```
nltk.download('averaged_perceptron_tagger')
```

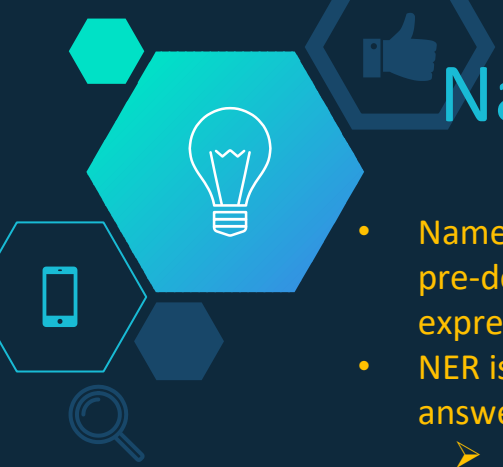
POS

```
myString = "Parts of speech: an article, to run, fascinating, quickly, and, of"
```

```
output = TextBlob(myString)
print(output.tags)
```

```
[('Parts', 'NNS'), ('of', 'IN'), ('speech', 'NN'), ('an', 'DT'), ('article', 'NN
'), ('to', 'TO'), ('run', 'VB'), ('fascinating', 'VBG'), ('quickly', 'RB'), ('an
d', 'CC'), ('of', 'IN')]
```

4.4 Practice Named Entity Recognition



Named Entity Recognition

- Named Entity Recognition (NER) seeks to locate and classify named entities in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.
- NER is used in many fields in Natural Language Processing (NLP), and it can help answering many real-world questions:
  - Which companies were mentioned in the news article?
  - Were specified products mentioned in complaints or reviews?
  - Does the tweet contain the name of a person? Does the tweet contain this person's location?

Named Entity Recognition

```
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

```
myString = "He worked for Microsoft and attended a conference in Italy"
```

```
(S
  He/PRP
  worked/VBD
  for/IN
  (ORGANIZATION Microsoft/NNP)
  and/CC
  attended/VBD
  a/DT
  conference/NN
  in/IN
  (GPE Italy/NNP))
```

4.5 Show Collocation Extraction and Synonyms

Collocation

- Collocations refer to words that occur together more frequently that it would happen by chance.
- Examples of collocations include “free time”, “break the rules”, “by the way”, “keep in mind”, “get ready”.
- Idiom and Collocation Extractor (ICE) can be used to identify collocations.

```
input=[“he and Chazz duel with all keys on the line.”]
from ICE import CollocationExtractor
extractor = CollocationExtractor.with_collocation_pipeline(“T1” ,
bing_key = “Temp”,pos_check = False)
print(extractor.get_collocations_of_length(input, length = 3))
```

```
[“on the line”]
```

Synonym and antonym

- We will use wordnet, which is an NLTK corpus reader, an English lexical database to find synonyms and antonyms.
- It can be defined as a semantically oriented English dictionary.
- If a word is not defined in wordnet, it might not be identified as a synonym or antonym.
- We use synsets() on wordnet to find synonyms for a word.

```
from nltk.corpus import wordnet
```

```
syn = wordnet.synsets("cat")
print(syn)
```

Synonym and antonym

```
from nltk.corpus import wordnet
        synonyms = []
        antonyms = []

        for syn in wordnet.synsets("active"):
                for l in syn.lemmas():
                        synonyms.append(l.name())
                        if l.antonyms():
                                antonyms.append(l.antonyms()[0].name())

        print(set(synonyms))
        print(set(antonyms))
```

{'dynamic', 'fighting', 'combat-ready', 'active_voice', 'active_agent', 'participating', 'alive', 'active'} -- Synonym

{'stative', 'passive', 'quiet', 'passive_voice', 'extinct', 'dormant', 'inactive'} -- Antonym

4.6 Demonstrate use of Word Lengthening to detect sentiment

Fixing word lengthening

- Word lengthening refers to the process of repeating characters.
- In English, words can only have a maximum of two characters repeated.
- Any additional characters should be done away with; otherwise, we may be dealing with misleading information.
- We can use the regular expressions' library to help us remove any repeated characters from our text

```
def remove_lengthening(text):
    patt = re.compile(r"(.)\1{2,}")
    return patt.sub(r"\1\1", text)
```

```
print(remove_lengthening( "commmmmmmitttteeee" ))
```

```
committee
```

4.7 Apply Spell Correction

Spell checker

- This is the process by which the spelling of a word is corrected.
- Spell correction algorithms work based on min-edit functions since brute forcing one's may take too much time.
- For the mid-edit functions to work effectively, word lengthening should be used first.
- NLTK comes with no spell correction module, but there are numerous other libraries that we can use to perform this task.
- We use suggest() on pattern.en package to find correct spellings.

Spell checker

```
from pattern.en import suggest
```

```
print(suggest("Whitle"))
print(suggest("acttibe"))
```

```
[('within', 0.769041769041769), ('tikhon', 0.20147420147420148), ('pigeon', 0.009828009828009828), ('patron', 0.009828009828009
828), ('piston', 0.004914004914004914), ('pithy', 0.002457002457002457), ('pathos', 0.002457002457002457)]
[('active', 0.96), ('attire', 0.04)]
```

Any questions so far?
Any comments?