

Natural Language Processing (NLP) for Digital Forensics

Mia Mohammad Imran

Hala Ali

Dr. Irfan Ahmed

Dr. Kostadin Damevski

Outline

- Digital Forensics Overview [5 minutes]
- Natural Language Processing (NLP) Overview [55 minutes]
- Enron Case Study with Python [80 minutes]
- Break [20 minutes]
- Authorship Attribution [80 minutes]

Digital Forensics Overview

- **Textual forensics**, also known as forensic linguistics or forensic text analysis, is a field that involves the analysis and interpretation of written or textual evidence for investigative or legal purposes.
- **Digital evidence** refers to any electronic data that can be used in legal proceedings.
- **Types Textual Digital Evidence:**
 - Emails.
 - Chat Logs.
 - Social Media Posts.
 - Text Documents.
 - Web Pages.
 - Instant Messaging Conversations.



Get the Notebook!

Go to the link and download the dataset

<https://shorturl.at/cEHM5>

NLP Overview

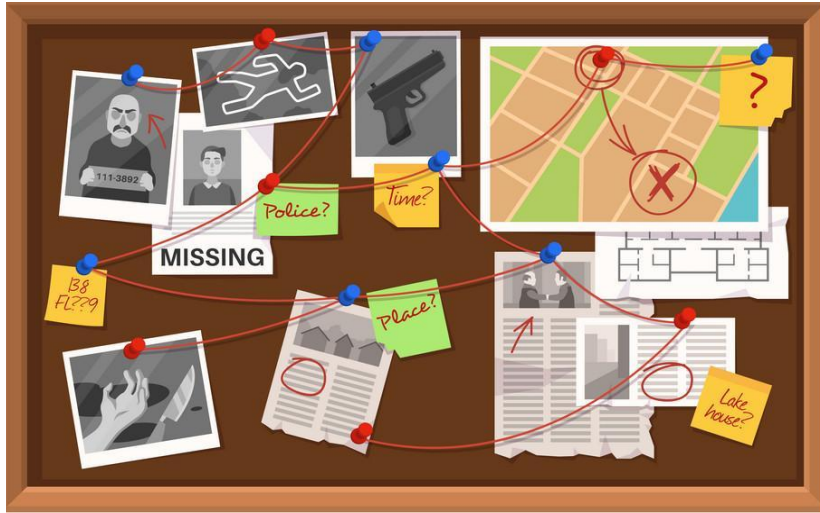
- **Natural Language Processing (NLP)** is a branch of artificial intelligence that focuses on the interaction between computers and human language.
- **NLP** enables machines to understand, interpret, and generate human language, leading to various applications.



NLP Overview: Applications

- **Information Extraction:** Extract structured information from unstructured text.
- **Information Retrieval (IR):** Locate and retrieve relevant documents or information that match a user's information needs.
- **Named Entity Recognition (NER):** Identify and extract named entities from text, such as names, organizations, locations, and dates.
- **Text Classification:** Categorize text documents into predefined categories or labels.
- **Topic Modeling:** Identify underlying topics and themes within a collection of text documents.
- **Text Summarization:** Generate concise summaries from longer texts.
- **Machine Translation:** Translate text from one language to another automatically.

NLP Overview: Example of Information Extraction

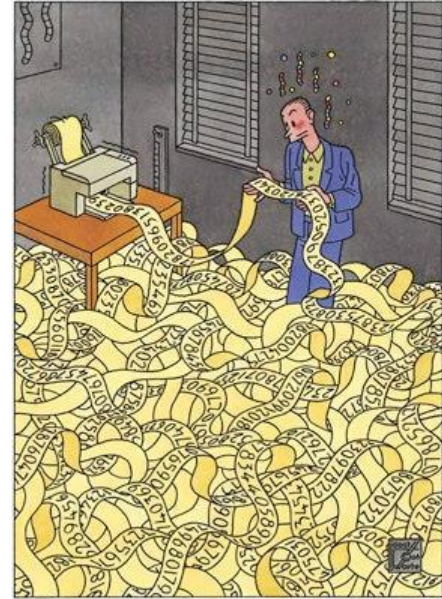


“We have received a report that a 4-year-old child named John has gone missing at approximately 6 pm in Monroe Park.”

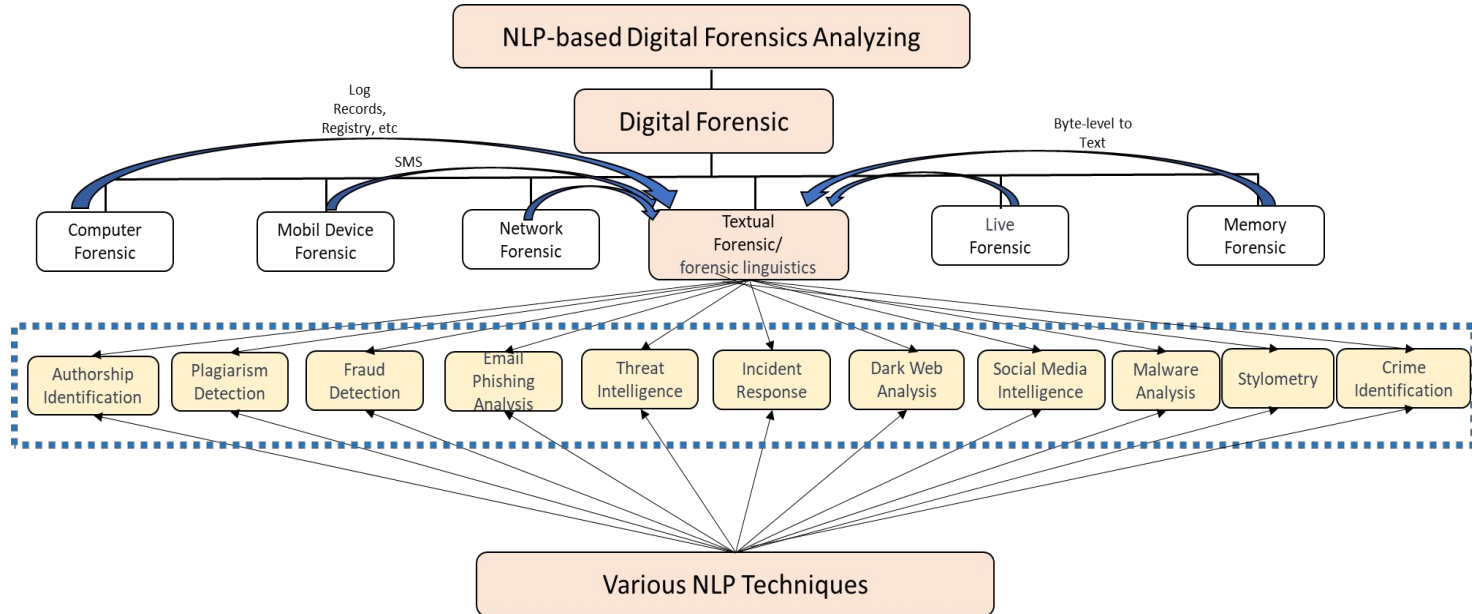
- Person: John
- Place: Monroe park
- Time: 6 pm
- Age: 12

NLP Overview: Role of NLP in Digital Forensics

- In the field of Digital Forensics, the analysis of large volumes of textual data poses significant challenges.
- NLP plays a vital role in automating analysis and making it faster and easier.
- Automated analysis using NLP significantly reduces the time and effort required for manual analysis in digital forensics.



NLP Overview: Role of NLP in Digital Forensics



NLP Overview: Techniques

- **Text Pre-Processing:**
 - Involves cleaning and preparing text data for further analysis.
 - Tasks include removing punctuation, converting to lowercase, and handling special characters.
- **Syntactical Analysis:**
 - Analyzes the grammatical structure and syntax of sentences.
 - Involves tasks like parsing, identifying sentence boundaries, and analyzing the syntactic relationships between words.
- **Lexical Analysis:**
 - Focuses on analyzing individual words or tokens in a text.
 - Includes tasks like stemming and lemmatization (reducing words to their base form).
- **Similarity Measures:**
 - Quantify the similarity or relatedness between words, phrases, or documents.
 - Methods include cosine similarity, Jaccard similarity, or Word2Vec-based similarity.

NLP Overview: Techniques

- **Word Sense Disambiguation:**
 - Resolves the ambiguity of word meanings based on context.
 - Determines the correct sense of a word in a particular context, considering multiple possible interpretations.
- **Part-of-Speech (POS) Tagging:**
 - Assigns grammatical tags to words in a sentence.
 - Tags indicate the words' function, such as noun, verb, adjective, or adverb.
- **N-gram:**
 - Refers to a contiguous sequence of N items, such as words or characters in a given text.
 - It captures local context and sequential dependencies by considering N consecutive items.
- **Language Modeling:**
 - Creates statistical models that predict the probability of the next word or sequence of words.
 - Language models learn from large datasets to generate coherent and contextually appropriate text.

NLP Basics: Text Preprocessing

- Library:

- **NLTK** (Natural Language Toolkit) is a popular Python library for NLP. It provides a wide range of tools and resources for NLP.

- To use NLTK for lowercasing, you can follow these steps:

- Install NLTK:

```
pip install nltk
```

- Import the NLTK library: In your Python script or notebook, import the NLTK library using the following line of code:

```
import nltk
```

- Download NLTK Resources: NLTK provides various resources such as corpora, models, and lexicons for different NLP tasks:

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

NLP Basics: Text Preprocessing

- Lowercasing:
 - Converting all uppercase characters in a text to their corresponding lowercase form.
 - Perform lowercasing: To convert text to lowercase, using python build-in `lower()` method. For example:

```
lowercase_text = text.lower()
```

NLP Basics: Text Preprocessing

- Tokenization:
 - Is the process of breaking a text into individual units, known as tokens. These tokens can be words, sentences, or even subword units.
 - To tokenize text using NLTK, you can use the `word_tokenize()` or `sent_tokenize()` functions.

```
from nltk.tokenize import word_tokenize

tokens = word_tokenize(text)
print("Tokenized text: ", tokens)
```

NLP Basics: Text Preprocessing

- Stop Words Removal:
 - Stopwords are common words that are often considered insignificant and do not carry much meaning in the context such as “the,” “and,” “a”, etc.
 - To download the stopwords resource, run the following command:

```
nltk.download('stopwords')

from nltk.corpus import stopwords

stop_words = set(stopwords.words('english'))
filtered_tokens = [token for token in tokens if token.lower() not
in stop_words]
```

NLP Basics: Text Preprocessing

- Part of Speech (POS) Tagging:
 - POS refers to the grammatical category or syntactic role that a word plays in a sentence.
 - POS tagging is the process of assigning a specific POS tag to each word in a given text, based on its syntactic function.
 - The POS tags used in NLTK follow standard conventions, such as the **Penn Treebank** POS tags, which are widely used in the field of NLP.
 - The `pos_tag` function from NLTK uses the default tagset defined by the Penn Treebank tagset.

```
from nltk import pos_tag
nltk.download('maxent_treebank_pos_tagger')
pos_tags = pos_tag(filtered_tokens)
```


NLP Basics: Penn Treebank Tagger

Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun

Tag	Description
PRP\$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Whdeterminer
WP	Whpronoun
WP\$	Possessive whpronoun
WRB	Whadverb

NLP Basics: Text Preprocessing

- Stemming and Lemmatization:
 - Stemming and lemmatization are techniques used to reduce words to their base or root form.
 - Stemming involves removing prefixes or suffixes from words, while lemmatization aims to bring words to their base form by considering their part of speech.

Stemming: "running" -> "run"

Lemmatization: "running" -> "run"

NLP Basics: Text Preprocessing

NLTK provides APIs for various Stemming Algorithms:

- Porter stemming.
- Lancaster stemming.

```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
stemmed_tokens = [stemmer.stem(token) for token in
filtered_tokens]
```

```
from nltk.stem import LancasterStemmer
```

NLTK provides APIs for various Lemmatizing options

- WordNetLemmatizer.
- TreebankWordNetLemmatizer.

You can do lemmatizing with or without POS Tagging!

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(token) for token in
filtered_tokens]
```

```
from nltk.stem import TreebankWordNetLemmatizer
```

NLP Basics: Text Preprocessing Demo in Notebook!

- Lowercasing.
- Tokenization.
- POS Tagging.
- Lemmatization and Stemming.

NLP Basics: Named Entity Recognition (NER)

- **NER** is the process of identifying and classifying named entities in text data, such as people, organizations, and locations.

Named Entity Recognition

Name Date Designation Subject

John McCarthy who was born on September 4, 1927 was an American computer scientist and cognitive scientist. He was one of the founders of the discipline of artificial intelligence. He co-authored the document that coined the term "Artificial intelligence" (AI), developed the programming language family Lisp, significantly influenced the design of the language ALGOL

NLP Basics: Named Entity Recognition (NER)

- Library:

- **SpaCy:** an open-source library for NLP in Python. It can be used for NER tasks.
- To use SpaCy for NER task, you can follow these steps:

- Install SpaCy:

```
pip install spacy
```

- Import SpaCy and Load the Model:

```
import spacy
# Load the English language model
nlp = spacy.load("en_core_web_sm")
```

NLP Basics: Named Entity Recognition (NER)

```
import spacy

# Load the English language model
nlp = spacy.load("en_core_web_sm")

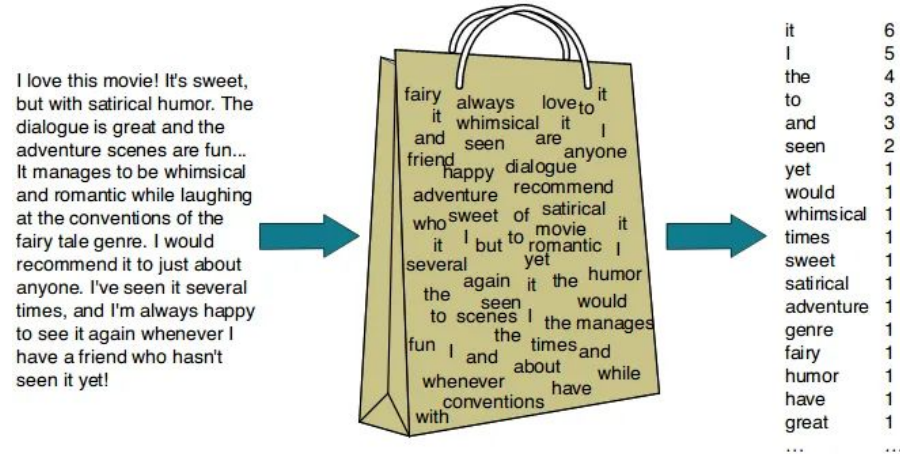
# Process the text
doc = nlp("Kenneth Lay was the founder and chairman of Enron Corporation. He was
indicted on multiple charges related to the scandal, including securities fraud and
insider trading.")

# Print the named entities
for ent in doc.ents:
    if ent.label_ == "PERSON":
        print(ent.text, ent.label_)
```

Can you change the code to show only the persons?

NLP Basics: Bag of Word (BoW)

- **BoW** is a simple yet powerful technique used in NLP to represent text data as a collection of word frequencies.
- **BoW** disregards the word order and only considers the presence or absence of words in a document.
- Steps:
 - Vocabulary construction: Creating a unique set of words in the corpus.
 - Document vectorization: Representing each document as a vector.
- Library:
 - **scikit-learn**, also known as **sklearn**, is a popular machine learning library for Python.



NLP Basics: Bag of Word (BoW)

```
from sklearn.feature_extraction.text import CountVectorizer
documents = [ "I love Digital Forensics." ,
              "Digital Forensics is challenging." ]

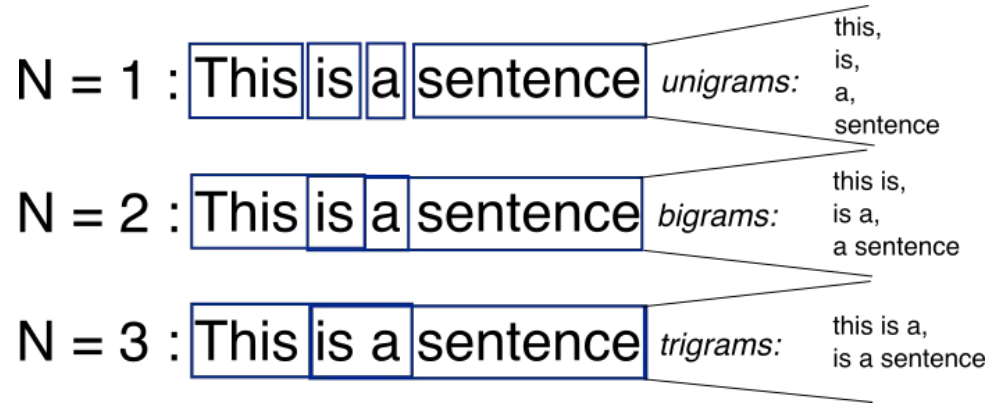
# Create an instance of CountVectorizer
vectorizer = CountVectorizer()
# Convert a collection of text documents into a numerical
representation known as the "bag of words"
bow_vectors = vectorizer.fit_transform(documents)
# Print the vocabulary (unique words)
print(vectorizer.get_feature_names_out())
# Print the BoW vectors for the documents
print(bow_vectors.toarray())
```

NLP Basics: N-Gram

- Refers to contiguous sequences of N items from a given text, where an item can be a character, word, or even larger units like phrases. The "N" in N-gram represents the number of items in the sequence.

- **Application:**

- Text Generation.
- Information Retrieval.
- Spell Checking and Correction.
- Machine Translation.



NLP Basics: N-Gram

- **Scikit-learn** library provides N-gram API!

```
from nltk.tokenize import word_tokenize
from nltk import ngrams
from collections import Counter

text = "....."
tokens = nltk.word_tokenize(text)
# Calculate N-grams
n_grams = ngrams(tokens, 2)
# Calculate the frequency of N-grams
frequency = Counter(n_grams)
for ngram, count in frequency.most_common():
    print(' '.join(ngram), count)
```

Can you change the code to get the frequency of trigram?

NLP Basics: TF-IDF

- **TF-IDF (Term Frequency-Inverse Document Frequency)** is a numerical statistic used in information retrieval and text mining to measure the importance of a term within a document or a collection of documents.
- **Term Frequency (TF)**: measures how frequently a term appears in a document.
- **Inverse Document Frequency (IDF)**: measures the rarity or uniqueness of a term in a collection of documents.
- Application:
 - Document Similarity.
 - Information Retrieval.
 - Keyword Extraction.
 - Text Classification.

$$w_{i,j} = tf_{i,j} \times \log \left(\frac{N}{df_i} \right)$$

tf_{ij} = number of occurrences of i in j
 df_i = number of documents containing i
 N = total number of documents

NLP Basics: TF-IDF

- **Scikit-learn** library provides TF-IDF vectorizer APIs!

```
from sklearn.feature_extraction.text import TfidfVectorizer
documents = ["I love Digital Forensics." ,
            "Digital Forensics is challenging." ]

# Create an instance of TfidfVectorizer
vectorizer = TfidfVectorizer()

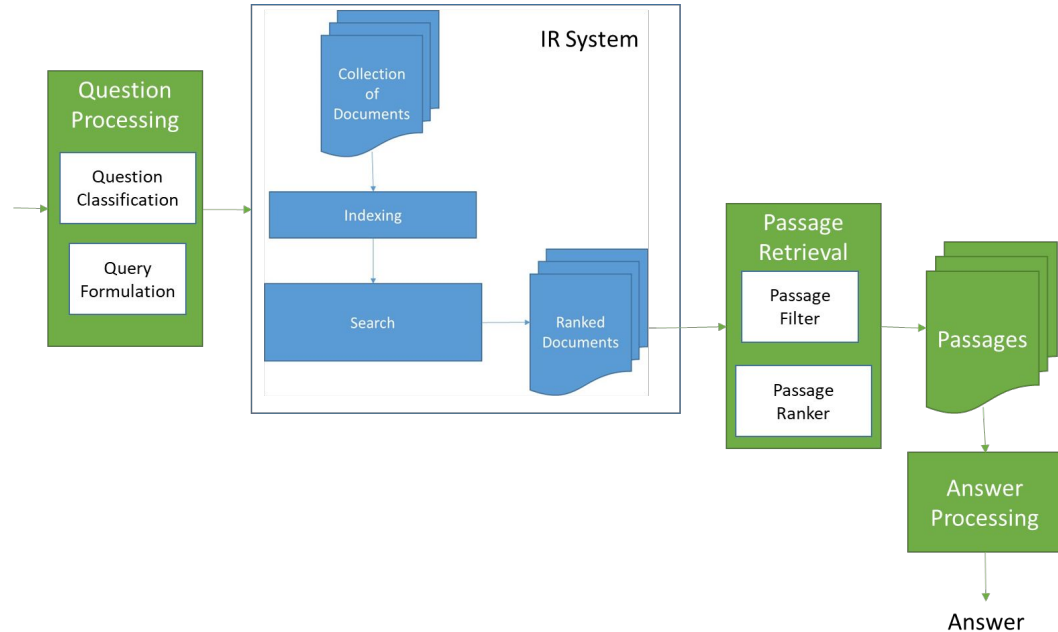
# Fit the vectorizer to the documents and transform the documents into TF-IDF
vectors

tfidf_vectors = vectorizer.fit_transform(documents)

# Print the TF-IDF vectors for the documents
print(tfidf_vectors.toarray())
```

NLP Applications: Information Retrieval (IR)

- Process of finding relevant information from a collection of unstructured or semi-structured data.
- **Used in:**
 - Search Engines.
 - Recommendation Systems.
 - Question-Answering Systems.



NLP Applications: IR Use Case

Finding the most frequent words in a text.

```
text = "....."  
# Step 1: Convert the text to lowercase  
text = text.lower()  
# Step 2: Tokenize the text into words  
words = text.split()  
# Step 3: Create a dictionary to store the frequency of each word  
word_freq = {}  
# Step 4: Loop through each word and update its frequency  
for word in words:  
    if word in word_freq:  
        word_freq[word] += 1  
    else:  
        word_freq[word] = 1  
# Step 5: Sort the words by their frequency in descending order  
sorted_words = sorted(word_freq.items(), key=lambda x:  
x[1], reverse=True)  
# Step 6: Print the top 5 most frequent words  
for word, freq in sorted_words[:5]:  
    print(f"{word}: {freq}")
```

NLP Applications: IR Use Case

Can you do it after applying Stop Word removal technique?

```
text = "..."  
# Step 1: Convert the text to lowercase  
text = text.lower()  
# Step 2.1: Tokenize the text into words  
words = text.split()  
# Step 2.2: Removing stop words  
stop_words=set(stopwords.words('english'))  
filtered_tokens = [token for token in words if  
token.lower() not in stop_words]  
# Step 3: Create a dictionary to store the frequency of each word  
word_freq = {}  
# Step 4: Loop through each word and update its frequency in the dictionary  
.....  
# Step 5: Sort the words by their frequency in descending order  
.....
```


NLP Applications: IR Use Case

Can you do it after applying Stemming technique?

```
text = "..."  
# Step 1: Convert the text to lowercase  
text = text.lower()  
# Step 2.1: Tokenize the text into words  
words = text.split()  
# Step 2.2: Removing stop words and stemming  
stop_words = set(stopwords.words('english'))  
filtered_tokens = [token for token in words if token.lower() not in  
stop_words]  
stemmer = PorterStemmer()  
stemmed_tokens = [stemmer.stem(token) for token in filtered_tokens]  
  
# Step 3: Create a dictionary to store the frequency of each word  
word_freq = {}  
# Step 4: Loop through each word and update its frequency in the dictionary  
.....  
# Step 5: Sort the words by their frequency in descending order  
.....
```

NLP Applications: IR Use Case

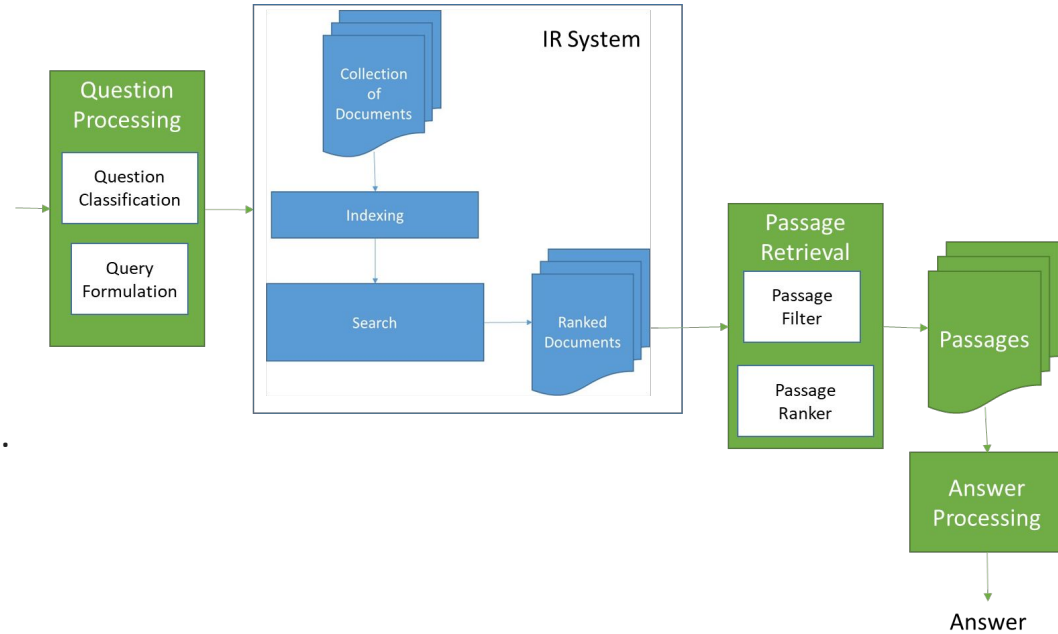
Finding out the sentences in a given text that contain a specific keyword.

```
text = "....."
# Step 1: Convert the text to lowercase
text = text.lower()
# Step 2: Split the text into sentences
sentences = text.split(". ")
# Step 3: Define the keyword to search for
keyword = "Energy"
# Step 4: Loop through each sentence and check if it contains the keyword
matching_sentences = []
for sentence in sentences:
    if keyword.lower() in sentence:
        matching_sentences.append(sentence)
# Step 5: Print the sentences that contain the keyword
print(f"Sentences containing the keyword ' {keyword}':")
for sentence in matching_sentences:
    print(sentence.strip() + ".")
```

NLP Applications: Query Searching in Multiple Documents

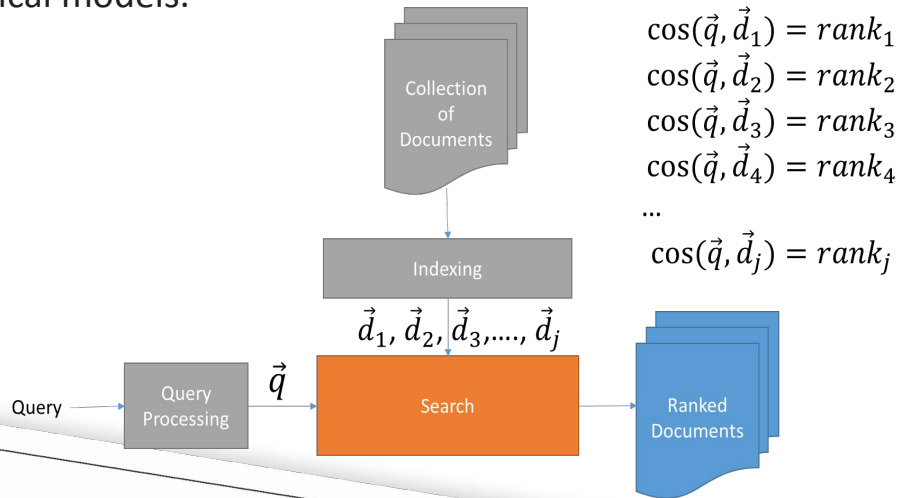
Finding the best matched documents in a collection of documents.

1. Document Collection.
2. User Queries.
3. Indexing: to facilitate efficient searching, documents in the collection are indexed.
4. Retrieval Models: retrieve relevant documents based on the user queries.
5. Ranking and Scoring: the documents are ranked or scored to determine their relevance to the query.

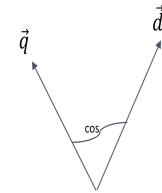


NLP Applications: Query Searching in Multiple Documents

- Boolean Model: Matches documents based on exact term matches.
- Vector Space Model: Represents documents and queries as vectors, calculating their similarity using measures like cosine similarity.
- Probabilistic Models: Estimate the probability of a document being relevant to a query based on statistical models.



Cosine similarity



$$\text{Cosine}(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| |\vec{y}|}$$

NLP Applications: Text Classification

- Process of assigning text data to predefined categories.
- **Used in:**
 - Fake News detection
 - Spam Email detection
 - Sentiment detection
 - Fraud text detection
 - Movie review
- **Techniques:** rule-based approaches, machine learning models, and deep learning models.

NLP Applications: Naive Bayes Classifier

- Naive Bayes Algorithm which is based on Bayes' theorem and assumes independence between features. Choose the best class (\hat{S}), given feature vector (f):

$$\hat{s} = \operatorname{argmax}_{s \in S} P(s|\vec{f})$$

- NLTK provides Naive Bayes API!

```
#Determining the Naive Bayes classifier, a popular algorithm for text classification.  
from nltk.classify import NaiveBayesClassifier
```

NLP Applications: Movie Reviews Classification using Naive Bayes

- Download the **movie_reviews** dataset provided by NLTK. It consists of movie reviews along with their corresponding sentiment labels.

```
nltk.download('movie_reviews')
```

- Preprocess the text and prepare data for classification: Extract the positive and negative reviews from the dataset. The `fileids` parameter is used to specify the file(s) to retrieve from the **movie_reviews** dataset. In this case, the `fileids` argument is set to `[f]`, where `f` represents a specific file id.

```
from nltk.corpus import movie_reviews
neg_reviews = [(preprocess(movie_reviews.raw(fileids=[f])), 'negative') for f in
movie_reviews.fileids('neg')]
pos_reviews = [(preprocess(movie_reviews.raw(fileids=[f])), 'positive') for f in
movie_reviews.fileids('pos')]
reviews = neg_reviews + pos_reviews
```

NLP Applications: Movie Reviews Classification using Naive Bayes

- Split dataset into training and testing sets

```
split dataset into training and testing sets
split = int(len(reviews)*0.8)
train_set = reviews[:split]
test_set = reviews[split:]
```

- Train classifier on training data

```
classifier =
NaiveBayesClassifier.train(train_set)
```

- Evaluate classifier on testing data

```
accuracy = nltk_accuracy(classifier, test_set)
```

- Print the Accuracy

```
print("Accuracy:", accuracy)
```


Questions?!

Enron Case Study: Data Source

- Enron Dataset: a large collection of emails and related data from the Enron Corporation, an energy company that filed for bankruptcy in 2001.
- It contains a vast amount of unstructured textual data, including emails, attachments, and metadata, from Enron employees across different organizational levels and departments.
- We have already **downloaded** this dataset at the beginning!

Enron Case Study: Email Sample

```
Date: Wed, 6 Jun 2001 08:32:00 -0700 (PDT)
From: wendy.conwell@enron.com
To: rhonda.denton@enron.com
Subject: RE: MEGA Deals to Assigned
Cc: melissa.murphy@enron.com, carol.clair@enron.com, tracy.ngo@enron.com,
    christian.yoder@enron.com
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Bcc: melissa.murphy@enron.com, carol.clair@enron.com, tracy.ngo@enron.com,
    christian.yoder@enron.com
X-From: Wendy Conwell
X-To: Rhonda L Denton
X-cc: Melissa Ann Murphy, Carol St Clair, Tracy Ngo, Christian Yoder
X-bcc:
X-Folder: \Carol_StClair_June2001\Notes Folders\Mega_morgan assignment
X-Origin: STCLAIR-C
X-FileName: cstclai.nsf

Ooops, meant to say assigning All transactions with "EPMI" to "Morgan" NOT
"ENA"!!! See what happens when I rush around like a chicken with her head
cut off!
```

```
-----Original Message-----
From: Conwell, Wendy
Sent: Wednesday, June 06, 2001 3:31 PM
To: Denton, Rhonda
Cc: Murphy, Melissa Ann; St Clair, Carol; Ngo, Tracy; Yoder, Christian
Subject: RE: MEGA Deals to Assigned

To Rhonda,

My bad...It looks like MEGA is assigning ALL of its physical power positions
with EPMI to ENA. The attached spreadsheet below has the options in the
lower section, which should really be part of the "to be assigned" group.

To All,

Sorry for the confusion,

Wendy
```

Enron Case Study: Retrieving Emails

- Retrieving email one by one by reading the email files.

```
# Reading the files one by one
email_rows = []

for root, dirs, files in os.walk('maildir'):
    for file in files:
        with open(os.path.join(root, file), "r", encoding='utf-8', errors='ignore')
as email_content:
    contents = email_content.read()
    # Each row has two columns: file path and email message
    email_rows.append([root+ '/' +file, contents])
```

Enron Case Study: Retrieving Emails

- Manipulating default python array is difficult!
- So we use **Pandas!**
 - Data manipulation library.
 - Provides data structures and functions for efficiently working with structured data.
 - Pandas introduces `DataFrame` which provides two-dimensional axis (row and columns).

```
import pandas as pd

emails_df = pd.DataFrame(email_rows, columns =
['file', 'message'])
print(emails_df.shape)
```

Enron Case Study: Retrieving Emails

- Calling the `emails_df.head()` method to display the first few rows of the DataFrame. By default, it shows the top 5 rows.

	file	message
0	maildir/lucci-p/internet/21.	Message-ID: <2739116.1075855367072.JavaMail.ev...
1	maildir/lucci-p/internet/31.	Message-ID: <8009352.1075855367319.JavaMail.ev...
2	maildir/lucci-p/internet/35.	Message-ID: <28084092.1075855367409.JavaMail.e...
3	maildir/lucci-p/internet/15.	Message-ID: <25476725.1075855366933.JavaMail.e...
4	maildir/lucci-p/internet/42.	Message-ID: <25042578.1075855367568.JavaMail.e...



Enron Case Study: Retrieving Emails

- `emails_df` is python object. We need to convert it to readable emails.
- We use `email` library for this purpose.
- API for converting to email is `message_from_string`.

```
import email  
  
messages = list(map(email.message_from_string, emails_df[ 'message' ]))
```

Enron Case Study: Email Contents

- How Does the Email Look after Converting?

```
print(messages[1])
```

```
Message-ID:
<2230250.1075863609465.JavaMail.evans@thyme>
Date: Tue, 5 Feb 2002 11:48:27 -0800 (PST)
From: sean.crandall@enron.com
To: karen.buckley@enron.com
Subject: RE: URGENT URGENT URGENT - DRUG TEST
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-From: Crandall, Sean
</O=ENRON/OU=NA/CN=RECIPIENTS/CN=SCRANDA>
X-To: Buckley, Karen
</O=ENRON/OU=NA/CN=RECIPIENTS/CN=Kbuckley>
X-cc:
X-bcc:
X-Folder: \ExMerge - Crandall, Sean\Sent Items
X-Origin: CRANDELL-S
X-FileName:
```

I completed mine on Monday the 4th.

```
-----Original Message-----
From: Buckley, Karen
Sent: Monday, February 04, 2002 4:08 PM
Subject: URGENT URGENT URGENT - DRUG TEST
Importance: High
Sensitivity: Private
```

If you have not already taken your drug test , please do so by Tuesday, 5th February. This process needs to be completed and results fed back to UBS prior to your start date. You cannot commence employment with UBS until they have this data which takes approximately 5 days from date of test.

If for any reason you cannot take this test by tomorrow, please advise. Also, if you have completed your test please let me know so I can update the UBS records.

Regards,

Karen.

Enron Case Study: Email Fields

- What are the email fields (keys)?

```
keys = messages[0].keys()
```

```
['Message-ID',  
'Date',  
'From',  
'To',  
'Subject',  
'Mime-Version',  
'Content-Type',  
'Content-Transfer-Encoding',  
'X-From',  
'X-To',  
'X-cc',  
'X-bcc',  
'X-Folder',  
'X-Origin',  
'X-FileName']
```

Enron Case Study: Email Fields as Features for Digital Forensics

- **From:** The sender of the email.
- **To:** The recipient(s) of the email.
- **CC** and **BCC:** Additional recipients of the email.
- **Subject:** provides insights into the topic or purpose of the email.
- **Date** and **Time:** helps establish timelines, patterns, or time-sensitive information related to the email communication.
- **Message-ID:** A unique identifier assigned to each email to track and link related email conversations or identify potential email chains.
- **Attachments:** Files or documents attached to the email.
- **Email Body:** The main content of the email.
- **Headers:** Additional metadata and routing information.

By extracting and analyzing these email fields as features, digital forensic investigators can gain valuable insights into email communications, identify potential patterns, detect anomalies, and uncover evidence that may be relevant to their investigations.

Enron Case Study: Email Fields as Features for Digital Forensics

- We will make each keys a **pandas** DataFrame column in `emails_df`.

```
for key in keys:  
    emails_df[key] = [doc[key] for doc in messages]
```

- However the keys did not contain the body! This is called **payload**. We will parse them separately.

```
def get_content (msg, max_word_len=30):  
    return msg.get_payload()  
  
emails_df['content'] = list(map(get_content, messages))
```

Enron Case Study: Email Fields as Features for Digital Forensics

- Sometimes there are multiple email addresses from in 'From' and 'To' columns. So we will separate them too.

```
def split_email_addresses (line):  
    if line:  
        addrs = line.split(',')  
        addrs = frozenset(map(lambda x: x.strip(), addrs))  
    else:  
        addrs = None  
    return addrs  
  
emails_df['From'] = emails_df['From'].map(split_email_addresses)  
emails_df['To'] = emails_df['To'].map(split_email_addresses)
```

Enron Case Study: Email Fields as Features for Digital Forensics

- Some keys are not important. We will remove them.
 - 'file', 'Mime-Version', 'Content-Type', and 'Content-Transfer-Encoding' from the DataFrame.

```
emails_df = emails_df.set_index( 'Message-ID' ).drop([ 'file' ,  
            'Mime-Version' , 'Content-Type' , 'Content-Transfer-Encoding' ], axis=1)
```

- Next we will preprocess Subject and content column using the preprocess concepts from NLP basics!
 - Lowercasing, tokenization, stopword removal, and lemmatization.

```
emails_df[ 'Subject' ] = emails_df[ 'Subject' ].apply( text_preprocessing )  
emails_df[ 'content' ] = emails_df[ 'content' ].apply( text_preprocessing )
```

Enron Case Study: Manual Task

- For a given email, can you find individual names, organization names, and specific dates mentioned in the content?

"PHC Statement draft To: Parties Who Participated in the A.01-09-024 (SoCalGas and SDG&E BCAP)
Meet and Confer: Attached is a proposed PHC Statement as required by ALJ Brown in her Ruling of November 5, 2001. Per that Ruling, this Statement must be filed within 10 days of our meeting (which by my calculation would be the 26th of November). I would REALLY like to get it filed on the 21st if possible, before the Thanksgiving weekend. I tried to keep the Statement simple, while reflecting the consensus views articulated at the meeting. I tried not to go into too much detail on the description of some of the outstanding CPUC proceedings that may affect this case, but wanted to give the judge some flavor for each proceeding. I ask that each of you let me know by noon next Tuesday (the 20th) if you want to sign on. I will incorporate any non-substantive edits you may have. For substantive edits (hopefully there will be few if any) I will do my best to incorporate. However, let's remember that none of us want to be de-positioned by what is said in this document so argumentative statements will be ignored. If only nonsubstantive edits are received, I will file on behalf of all participating parties without further review. If substantive changes are made, a second review will be provided. I hope we can keep this from becoming an unwieldy exercise. I ask for your cooperation and appreciate your input. If you want to provide your individual PHC Statement, fine.
Just let me know so I can take your client's name off this one.
Thanks again."

Enron Case Study: Manual Task

Difficult !!!!!

That's why NLP is used to make the text analysis easier and faster.

```
import spacy

# Load the English language model
nlp = spacy.load("en_core_web_sm")

# Process the text
doc = nlp(email_text)

# Print the named entities
for ent in doc.ents:
    if ent.label_ == "PERSON" or ent.label_ == "ORG" or ent.label_ == "DATE":
        print(ent.text, ent.label_)
```

Enron Case Study: Information Retrieval

- Can you find how many emails mentioned the keyword "PHC Statement"?

```
counter = 0
keyword = 'PHC Statement'
first = False
for email in emails_df['content_new']:
    if keyword in email:
        counter = counter + 1
        if first is False:
            print(email)
            first = True
# if keyword is in the content, increase the counter
print(counter)
```


Enron Case Study: Visualization

- **Library: Matplotlib:** is a popular data visualization library in Python. It provides a wide range of functions and tools for creating various types of plots and visualizations, such as line plots, scatter plots, bar plots, and histograms.

```
import matplotlib.pyplot as plt
```

- Who sent most emails? Finding the top 10 email address.

```
sub_df = emails_df[['From', 'content', 'Date']].dropna()
display(sub_df.shape)
grouped_by_people = sub_df.groupby('From').agg({'Date': 'count'})
grouped_by_people.rename(columns={'Date': 'Count'}, inplace=True)
grouped_by_people.sort_values('Count', ascending=False).head()
top_10_frequent = grouped_by_people['Count'].sort_values(ascending = False)[:10]
print(top_10_frequent)
```

Enron Case Study: Visualization

- Can you find who received most emails? Find the top 10 email address?

```
sub_df = emails_df[['To', 'content', 'Date']].dropna()
display(sub_df.shape)
grouped_by_people = sub_df.groupby('To').agg({'Date': 'count'})
grouped_by_people.rename(columns={'Date': 'Count'}, inplace=True)
grouped_by_people.sort_values('Count', ascending=False).head()
top_10_frequent = grouped_by_people['Count'].sort_values(ascending = False)[:10]
print(top_10_frequent)
```

Enron Case Study: Clustering

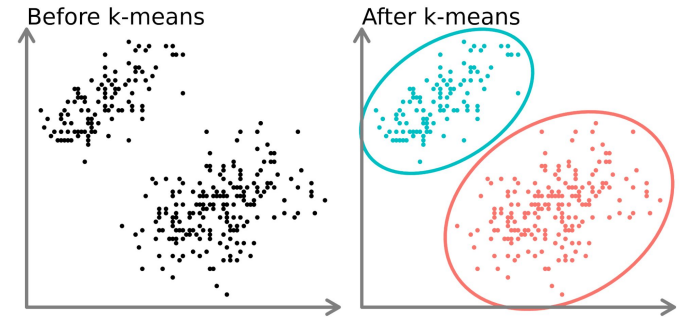
- Clustering is a technique used to group similar documents or pieces of text together based on their content.
- Clustering helps discover inherent patterns, similarities, and relationships within a collection of documents without prior knowledge of their labels or categories.
- Principal Component Analysis (PCA): Dimensionality reduction technique, often used to transform high-dimensional data into a lower-dimensional space while preserving the most important information.
- Different clustering techniques:
 - K-means
 - Gaussian Mixture Models

Enron Case Study: K-means Clustering

- K-means clustering is a type of unsupervised learning method.
- The goal of this algorithm is to find groups in the data, whereas the no. of groups is represented by the variable K.
- Library:
 - **scikit-learn**, also known as sklearn, is a popular machine learning library for Python. It provides a wide range of tools and algorithms for various machine learning tasks, including classification, regression, clustering, dimensionality reduction, and model selection.
 - Provides API form **K-means** as well as **PCA**.

```
from sklearn.cluster import KMeans
from sklearn.decomposition import
```

PCA



Enron Case Study: K-means Clustering

1. Representing the text in a numerical vectors, using TF-IDF. TF-IDF stands for Term Frequency-Inverse Document Frequency.

```
vectorizer = TfidfVectorizer()  
tfidf_matrix = vectorizer.fit_transform(documents)
```

2. Defining the PCA algorithm. PCA is widely used for feature extractions, noise filtering, etc.

```
pca = PCA(n_components = 2)
```

3. Computing the principal components of the data and transforming the data to the new lower-dimensional space.

```
pca_matrix = pca.fit_transform(tfidf_matrix.toarray())
```

Enron Case Study: K-means Clustering

4. Define the number of clusters.

```
k = 3
```

5. Applying K-means clustering with number of iterations.

```
kmeans = KMeans(n_clusters= k, max_iter=600, algorithm = 'auto')  
fitted = kmeans.fit(pca_matrix)
```

6. Get the cluster labels for each document.

```
labels = kmeans.predict(pca_matrix)
```

Enron Case Study: K-means Clustering Exercise

- Exercise-1: Run the model by increasing the number of clusters and changing the number of components in PCA. Observe how the plots changes.
- Exercise-2: Clustering using Gaussian Mixture Models (GMM).

```
# Perform Gaussian Mixture Model (GMM) clustering  
gmm = GaussianMixture(n_components=k, random_state=42)
```

Enron Case Study: Topic Modeling

- **Topic modeling** is a technique used in NLP and ML to identify latent topics or themes present in a collection of documents.
- It allows for the discovery of hidden patterns and structures within textual data without prior knowledge of the specific topics.
- Topic modeling assumes that each document in a collection is a mixture of different latent topics, and each topic is characterized by a distribution of words.
- Topic modeling is often based on probabilistic models, such as **Latent Dirichlet Allocation (LDA)**. These models generate a set of topics and associated word distributions, considering the probability of a document belonging to a particular topic and the probability of a word being generated by a particular topic.

Enron Case Study: Topic Modeling Library

- **gensim**: is a Python library for topic modeling. It is designed to handle large-scale text corpora efficiently and offers a simple and intuitive API for various NLP tasks.
- **gensim** offers methods for computing document similarity based on different similarity measures, such as cosine similarity.
- The "corpora" module in Gensim focuses on building and working with corpora, which are collections of texts or documents used for analysis and modeling.

```
import gensim
from gensim import corpora
```

Enron Case Study: Topic Modeling Steps

1. Tokenizing the text

```
# Tokenize the documents
tokenized_documents = [doc.split() for doc in emails_df_sampled['content']]
```

2. Creating a Gensim Dictionary: to map between words and their unique numeric IDs.

```
dictionary = corpora.Dictionary(tokenized_documents)
```

3. Extracting the whole words in the text with their frequencies. doc2bow () converts the text into a bag-of-words representation.

```
corpus = [dictionary.doc2bow(doc) for doc in tokenized_documents]
```

Enron Case Study: Topic Modeling Steps

4. Define the number of topics that the **LDA** model will attempt to identify.
5. Applying LDA model on the the document-term matrix in the form of a bag-of-words representation. `id2word`: The `id2word` parameter is a dictionary that maps unique IDs to words in the corpus.

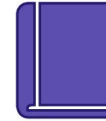
```
num_topics = 3  
lda_model = gensim.models.LdaModel(corpus=corpus, id2word=dictionary, num_topics=num_topics)
```

Questions?!

Authorship Attribution

- The process of determining the author of a given text based on linguistic patterns and stylometric analysis.
- Involves examining patterns, writing style, and content.
- Different from Author Identification:
 - Authorship Attribution aims to identify a specific author among a set of potential authors, often referred to as the candidate authors or suspects.
 - Author Identification aims to verify the authorship claim of a particular text by matching it to the writing style of a known author.

Unknown Documents



Who is the author?

Author 1



Author 2



Author 3



Author 4



Authorship Attribution: Applications

- **Forensic Linguistics:**
 - Helps in legal and criminal investigations by identifying anonymous or disputed texts.
 - Provides valuable evidence in cases involving threats, ransom notes, or suspicious communications.
- **Plagiarism Detection:**
 - Assists in academia and content creation by detecting instances of plagiarism.
 - Helps maintain academic integrity and originality in research, writing, and publishing.
- **Literary Studies:**
 - Allows for the analysis and understanding of the works of different authors and literary movements.
 - Sheds light on an author's unique style, thematic preferences, and influences.

Authorship Attribution: Steps Using Machine Learning

- Data collection: gather a dataset of texts written by different authors.
- Data preprocessing: preprocess the text data to make it suitable for analysis.
- Feature extraction and selection: commonly used features:
 - Stylometric Features: Capture stylistic aspects of writing, including word and sentence lengths, vocabulary richness, punctuation usage, and syntactic patterns.
 - N-gram Features: Represent the frequency of occurrence of N-grams (sequences of n consecutive words) in the text.
 - TF-IDF Features: Measure the importance of words or terms in a document relative to a corpus.

Authorship Attribution: Steps Using Machine Learning

- **Data split:** Divide the dataset into training and testing sets.
 - The training set will be used to train the machine learning model, while the testing set will be used to evaluate its performance on unseen data.
- **Model selection and training:** Select an appropriate machine learning algorithm and train the model using the training dataset.
 - Commonly used ML models are: Decision Tree, Naive Bayes, SVM, k-NN, etc.
- **Model evaluation:** Using the trained model to make predictions or decisions on new, unseen data.
 - Commonly used evaluation metrics are: accuracy, precision, recall, and F1-score.

Authorship Attribution: Data Collection

- Data source: Publicly available dataset crawled from Twitter.
 - Open `Authorship Attribution.ipynb` from google drive folder.
- Data format: CSV.
- Library: Importing data using **Pandas**.
 - provides data structures and functions.
 - efficiently working with structured data.

```
import pandas as pd
# Load the dataset
data = pd.read_csv('authorship.csv', sep=',')
```

Authorship Attribution: Data Sample

- Contains 18.6k tweets from 8 different persons.

	Name	Tweet description
0	Barack Obama	We're seeing the fastest rise in sea-levels in...
1	Barack Obama	RT @WhiteHouse: Watch @POTUS and @FLOTUS share...
2	Barack Obama	See what editorial boards are saying about Sen...
3	Barack Obama	President Obama is at the @CopernicusCtr in Ch...
4	Barack Obama	LIVE: President Obama is answering questions f...
...
18675	Sebastian Ruder	@EmilStenstrom Are you using Firefox? Someone ...
18676	Sebastian Ruder	@KieranSnyder The following are some interesti...
18677	Sebastian Ruder	The initial work on casting NLP tasks as QA (h...
18678	Sebastian Ruder	New NLP News - Paperclip maximizer, Generative...
18679	Sebastian Ruder	RT @jeremyjordan: How to Write a 📄 Research Pa...

18680 rows × 2 columns

Authorship Attribution: Data Preprocessing

- Same as we discussed in NLP basics!
 - Preprocess the text data (remove punctuation, lowercase, tokenize, etc).

```
# Remove punctuation and convert to lowercase
data['processed_text'] = data['Tweet description'].apply(lambda x:
x.translate(str.maketrans('', '', string.punctuation)).lower())
# Tokenize the text
data['tokenized_text'] = data['processed_text'].apply(word_tokenize)
# Remove stop words
stop_words = set(stopwords.words('english'))
data['filtered_text'] = data['tokenized_text'].apply(lambda x: [word for word in x if word
not in stop_words])
```

Authorship Attribution Feature: Stylometric Features

- One of the simplest methods for authorship attribution is analyzing stylometric features.
- Stylometric Features involve analyzing various stylistic aspects of writing, such as word length, sentence length, and vocabulary richness.
- These features help capture the unique writing style of authors.

```
# Calculate stylometric features
data['avg_word_length'] = data['tokenized_text'].apply(lambda x: sum(len(word) for word in x)
/ len(x))
data['avg_sentence_length'] = data['tokenized_text'].apply(lambda x: len(x) /
(x.count('.')+1))
data['vocabulary_richness'] = data['tokenized_text'].apply(lambda x: len(set(x)) / len(x))
```

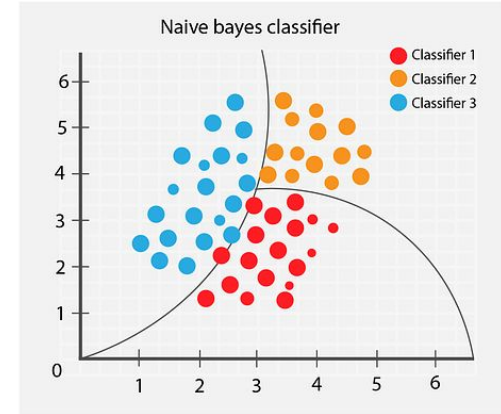
Authorship Attribution: Data Split

- Before training a model, we need to split the data into training and testing sets. This allows us to evaluate the model's performance on unseen data.
- Usually data set are divided into 80%-20% between training and test set.
- Used library: **scikit-learn** (sklearn).
 - Provides an API (`train_test_split`).

```
from sklearn.model_selection import train_test_split
# Prepare the data
X = data[['avg_word_length', 'avg_sentence_length', 'vocabulary_richness']]
y = data['Name']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2, random_state=42)
```

Authorship Attribution Classifier: Naive Bayes

- Based on Bayes' theorem and assumes independence between features.
- Known for its simplicity, and effectiveness in text classification tasks.
- Widely used in various NLP tasks, including spam detection, sentiment analysis, and authorship attribution.
- For authorship attribution task, it calculates the probability of a given text belonging to a particular author based on the frequencies of features present in the text.



$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

using Bayesian probability terminology, the above equation can be written as

$$\text{Posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

Authorship Attribution Classifier: Naive Bayes

- **sklearn** provides high-level API for Naive Bayes.
- Naive Bayes classifiers can be trained using different variants, such as Multinomial Naive Bayes, Bernoulli Naive Bayes, and Gaussian Naive Bayes, depending on the nature of the features.

```
from sklearn.naive_bayes import
MultinomialNB

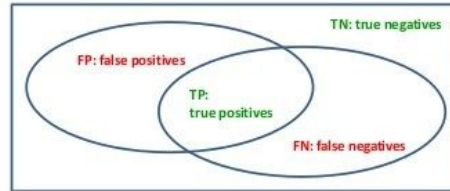
# Train the classifier
model = MultinomialNB()
model.fit(X_train, y_train)
from sklearn.naive_bayes import BernoulliNB

from sklearn.naive_bayes import GaussianNB
```

Authorship Attribution: Evaluation Metrics

Definitions

Accuracy, Precision, Recall, and F-measure



Accuracy: How many predictions are correct?

Accuracy:

$$acc = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision:

$$p = \frac{TP}{TP + FP}$$

Recall:

$$r = \frac{TP}{TP + FN}$$

F-measure:

$$F = \frac{1}{\frac{1}{2} \left(\frac{1}{p} + \frac{1}{r} \right)} = \frac{2pr}{p+r}$$

Precision: Of all positive predictions, how many are really positive?

Recall: Of all real positive cases, how many are predicted positive?

F1-score = Harmonic mean of precision and recall.

Authorship Attribution: Evaluation Metrics

- **sklearn** provides high-level API for Evaluation metrics.

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_true, y_pred)
```

```
from sklearn.metrics import precision_score
precision = precision_score(y_true, y_pred)
```

```
from sklearn.metrics import recall_score
recall = recall_score(y_true, y_pred)
```

```
from sklearn.metrics import f1_score
f1 = f1_score(y_true, y_pred)
```

Authorship Attribution Case Study: Demo on Notebook!

- **Demo:** Using Stylometrics features.
- Naive Bayes (`MultinomialNB`) as classifier.
- Accuracy and F1-score as evaluation metrics.

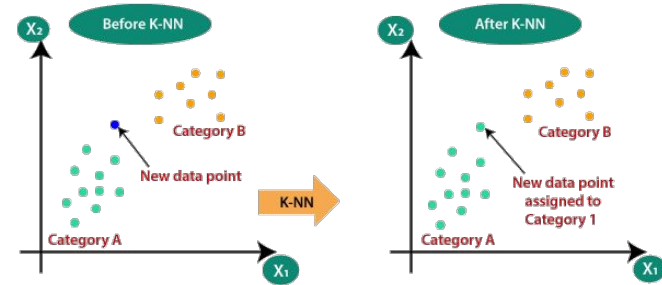
- **Exercise:** `BernoulliNB` and `GaussianNB`.
- Observe the performance differences.

Authorship Attribution: More Classifiers!

- k-Nearest Neighbours (k-NN).
- Decision Tree.
- Support Vector Machine (SVM).

Authorship Attribution Classifier: k-Nearest Neighbours

- k-NN is a simple and intuitive algorithm that classifies new data points based on their similarity to the labeled data points in the training set.
- Select the k nearest neighbors (data points) from the training set based on the calculated distances.
- k is a predefined parameter that determines the number of neighbors to consider.
- **sklearn** provides high level API.



```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors= 3)
model.fit(X_train, y_train)
```

Authorship Attribution Case Study: Exercise on Notebook!

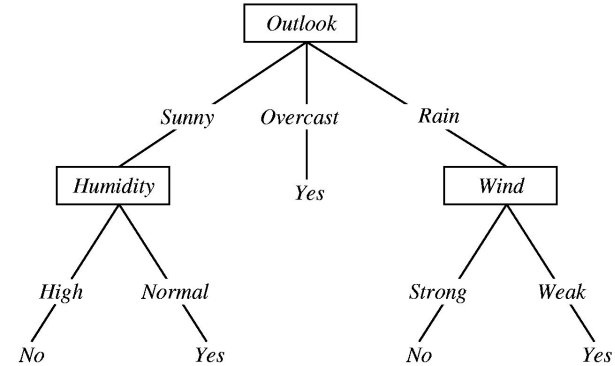
- Using Stylometrics features.
- k-NN as classifier.
- Accuracy and F1-score as evaluation metrics.

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors= 3)
model.fit(X_train, y_train)
```

Authorship Attribution Classifier: Decision Tree

- Decision tree is a supervised machine learning algorithm, often used in various text classification tasks.
- It is a flowchart-like structure where each internal node represents a feature or attribute, each branch represents a decision rule, and each leaf node represents the outcome.
 - Makes predictions based on a sequence of logical rules learned from the training data.
- **sklearn** provides high level API.

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```



Simple Decision Tree

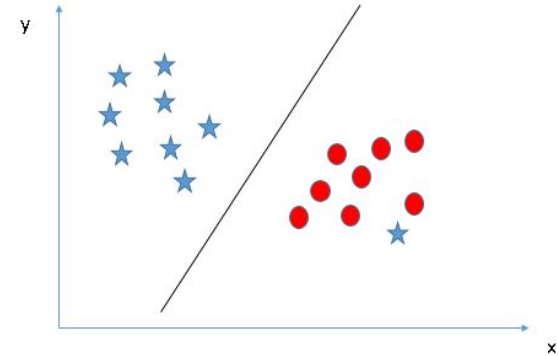
Authorship Attribution Case Study: Exercise on Notebook!

- Using Stylometrics features.
- Decision Tree as classifier.
- Accuracy and F1-score as evaluation metrics.

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

Authorship Attribution Classifier: Support Vector Machine

- In SVM, the goal is to find a hyperplane that separates the data points belonging to different classes with the maximum margin.
- **sklearn** provides multiple high level APIs: **LinearSVC** and **SVC**.
- **LinearSVC** is an implementation of SVM that uses a linear kernel.
- **SVC** is a more general implementation of SVM that supports various kernel functions. It can handle both linearly separable and non-linearly separable data by transforming the feature space using kernel functions.



```
from sklearn.svm import  
LinearSVC  
model = LinearSVC()  
model.fit(X_train, y_train)
```

```
from sklearn.svm import SVC  
model = SVC()  
model.fit(X_train, y_train)
```


Authorship Attribution Case Study: Exercise on Notebook!

- Using Stylometrics features.
- SVM as classifier.
- Accuracy and F1-score as evaluation metrics.

```
from sklearn.svm import  
LinearSVC  
model = LinearSVC()  
model.fit(X_train, y_train)
```

```
from sklearn.svm import SVC  
model = SVC()  
model.fit(X_train, y_train)
```

Authorship Attribution: What about Other Features?

- Bag of Words.
- N-grams.
- TF-IDF.

```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
# Extract Bag-of-Words (BoW) features
bow_vectorizer = CountVectorizer()
bow_features = bow_vectorizer.fit_transform(data[ 'processed_text' ])
```

```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
# Extracting bi-grams and tri-grams
vectorizer = CountVectorizer(ngram_range=( 2, 3))
ngram_matrix = vectorizer.fit_transform(data[ 'processed_text' ])
```

```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
# Extract TF-IDF features
tfidf_vectorizer = TfidfVectorizer()
tfidf_features = tfidf_vectorizer.fit_transform(data[ 'processed_text' ])
```

Authorship Attribution: Bag of Words as Feature

- Using **sklearn** API to extract features.

```
from sklearn.feature_extraction.text import CountVectorizer

bow_vectorizer = CountVectorizer()
bow_features = bow_vectorizer.fit_transform(data[ 'processed_text' ])
```

Authorship Attribution Case Study: Demo on Notebook!

- **Demo:** Using BoW as features for Naive Bayes
- Observe the performance differences from Stylometric features.

```
from sklearn.feature_extraction.text import CountVectorizer

bow_vectorizer = CountVectorizer()
bow_features = bow_vectorizer.fit_transform(data[ 'processed_text' ])
```

```
from sklearn.naive_bayes import
MultinomialNB

# Train the classifier
model = MultinomialNB()
model.fit(X_train, y_train)
```

Authorship Attribution: N-grams as Feature

- Using **sklearn** API to extract features.

```
from sklearn.feature_extraction.text import CountVectorizer

# Example: Extracting bi-grams and tri-grams
vectorizer = CountVectorizer(ngram_range=( 2, 3))

ngram_matrix = vectorizer.fit_transform(data[ 'processed_text' ])
```

```
# Example: Extracting uni-gram and bi-grams
vectorizer = CountVectorizer(ngram_range=( 1, 2))

ngram_matrix = vectorizer.fit_transform(data[ 'processed_text' ])
```

Authorship Attribution Case Study: Demo on Notebook!

- **Demo:** Using N-grams (bi-gram and tri-gram) as features for all classifiers.
- Observe the performance discrepancy among the models.

- **Exercise:** Using uni-gram and bi-gram as features for Naive Bayes.
- Observe the performance from using bi-gram and tri-grams.

Authorship Attribution Case Study: Demo on Notebook!

- **Demo:** Using N-grams (bi-gram and tri-gram) as features for all classifiers.
- Observe the performance discrepancy among the models.

- **Exercise:** Using uni-gram and bi-gram as features for Naive Bayes.
- Observe the performance from using bi-gram and tri-grams.

Model	Accuracy	F1-score
Naive Bayes	0.68	0.68
Decision Tree	0.51	0.52
SVM (LinearSVC)	0.65	0.66
KNN	0.14	0.06

Authorship Attribution: TF-IDF as Feature

- Using **sklearn** API to extract features.

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Extract TF-IDF features
tfidf_vectorizer = TfidfVectorizer()
tfidf_features = tfidf_vectorizer.fit_transform(data[ 'processed_text' ])
```


Authorship Attribution Case Study: Demo on Notebook!

- Using TF-IDF as features for all classifiers.
- Observe the differences from N-grams.
- **Exercise:** Implement k-NN with TF-IDF and check its performance. How it differs from N-grams?

Authorship Attribution Case Study: Demo on Notebook!

- Using TF-IDF as features for all classifiers.
- Observe the differences from N-grams.
- **Exercise:** Implement k-NN with TF-IDF and check its performance. How it differs from N-grams?

Model	Accuracy	F1-score
Naive Bayes	0.80	0.80
SVM (LinearSVC)	0.85	0.85
KNN	0.75	0.75

Authorship Attribution: What about combining features?

- You can combine features!
 - N-grams.
 - TF-IDF.
- You can stack features.
- Used library for stacking:
hstack API from **scipy**.
 - SciPy is a free and open-source Python library used for scientific computing and technical computing.

```
from scipy.sparse import hstack

# Extract TF-IDF features
tfidf_vectorizer = TfidfVectorizer()
tfidf_features =
tfidf_vectorizer.fit_transform(data[ 'processed_text' ]
)

# Extract N-gram features
ngram_vectorizer = CountVectorizer(ngram_range=( 2,
3))
ngram_features =
ngram_vectorizer.fit_transform(data[ 'processed_text' ]
)

# Combine TF-IDF and N-gram features
features = hstack([tfidf_features, ngram_features])
```

Authorship Attribution Case Study: Demo on Notebook!

- **Demo:** Using both TF-IDF and N-grams as features for all classifiers.
- **Exercise:** Can you add Bag-of-Words on top of TF-IDF and N-grams?

Model	Accuracy	F1-score
Naive Bayes	0.76	0.75
SVM (LinearSVC)	0.77	0.78
KNN	0.14	0.05

TF-IDF and N-grams as features

Authorship Attribution: Advances Topics - LLM

- Large Language Model (LLM) fine-tuning is a powerful technique for enhancing Authorship Attribution using pre-trained models like BERT, RoBERTa, GPT.
- BERT is a popular LLM that has been pre-trained on a large corpus of text data to learn contextualized word representations.
 - Wikipedia, English literature.
- We can fine-tune BERT to a specific task, such as Authorship Attribution, NER, etc.
- LLMs are memory heavy. Can't run them on colab.
 - Usually requires GPU.





```
Epoch 1/10, Loss: 435.2440082579851
Accuracy: 0.8458244111349036
F1-score: 0.8440268575238626
Epoch 2/10, Loss: 153.98593225330114
Accuracy: 0.8808886509635975
F1-score: 0.8812815071533733
Epoch 3/10, Loss: 75.55887137725949
Accuracy: 0.887847965738758
F1-score: 0.887818932523332
Epoch 4/10, Loss: 36.09384918073192
Accuracy: 0.8830299785867237
F1-score: 0.8823622164335424
Epoch 5/10, Loss: 25.028166307136416
Accuracy: 0.8835653104925053
F1-score: 0.8838460168886778
Epoch 6/10, Loss: 16.322425733786076
Accuracy: 0.8894539614561028
F1-score: 0.8897736085412024
Epoch 7/10, Loss: 12.012017809203826
Accuracy: 0.8760706638115632
F1-score: 0.8757705878092785
Epoch 8/10, Loss: 12.176514446386136
Accuracy: 0.8816916488222698
F1-score: 0.88070882250017
Epoch 9/10, Loss: 11.584088880859781
Accuracy: 0.889186295503212
F1-score: 0.8892709230058843
Epoch 10/10, Loss: 9.22819634329062
Accuracy: 0.8921306209850107
F1-score: 0.8917678738852298
```

Better than any ML models!





Authorship Attribution: Advances Topics - ChatGPT!

- You can prompt to ChatGPT and similar prompt-based OpenSource LLMs!

M I will provide you some texts. Your task is to determine whether it's written by Jane Austen, Charles Dickens or Neither.


 Of course! I'll do my best to help you determine whether the texts you provide were written by Jane Austen, Charles Dickens, or neither. Please go ahead and provide me with the texts you would like me to analyze.   

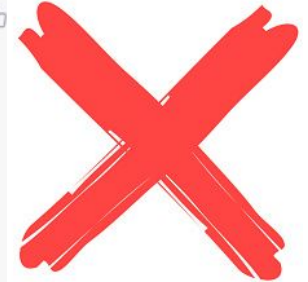
M Text: "They are much to be pitied who have not been given a taste for nature early in life."

 Based on the provided text, it is more likely to be written by Jane Austen. While Charles Dickens also wrote about human nature and social issues, this particular sentence does not exhibit the distinct style and themes typically associated with his works. Jane Austen, on the other hand, often emphasized the importance of nature and the beauty of the natural world in her novels. The sentiment expressed in the text aligns with Austen's emphasis on the value of appreciating nature. However, without additional context, it is still possible that the text could be from another author or source.   

Authorship Attribution: Advances Topics - ChatGPT!

M Text: "Family not only needs to consist of merely those whom we share blood, but also of those whom we'd give blood."

 Based on the provided text, it is neither written by Jane Austen nor Charles Dickens. The sentiment expressed in the text does not align with the typical themes and writing styles of either Austen or Dickens. The language used in the sentence is more modern and does not reflect the linguistic patterns and vocabulary of the 19th-century authors. This sentence appears to be a contemporary statement emphasizing the importance of chosen or extended family, rather than a passage from the works of Austen or Dickens.



Thank You!
Questions?

Survey Link:

<https://forms.gle/BisfsWMgUP1MiEmr5>