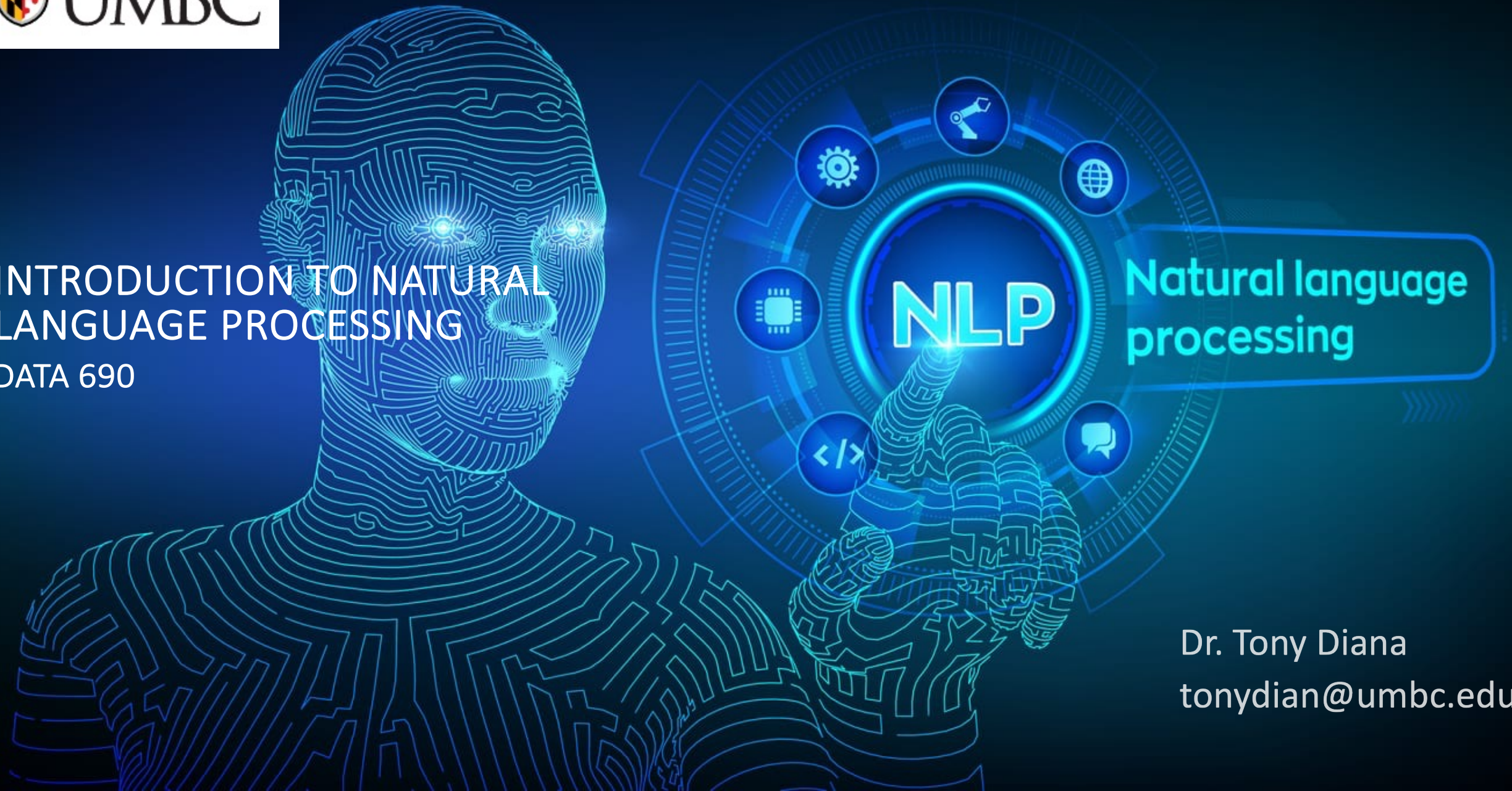
UMBC

INTRODUCTION TO NATURAL LANGUAGE PROCESSING

DATA 690



Dr. Tony Diana
tonydian@umbc.edu

UMBC

Table of Contents

UMBC

1. INTRODUCTION TO NATURAL LANGUAGE PROCESSING AND APPLICATIONS

UMBC

What is Natural Language Processing?

- It is one of the fastest growing areas in Machine Learning
- **Natural language** is developed by humans and evolved through natural use and communication, rather than constructed and created **artificially**
- The basic objective of NLP is to transform **unstructured text** data into **knowledge** (structured data) and use that knowledge to improve
  - **Interactions** among **humans** (*sentiment analysis*), and, also
  - **Exchanges** between **humans** and **machines** (*chatbots*)
- NLP spans multiple disciplines. Here are some of them:
  - **Machine Learning**: Almost any NLP process is built upon a Machine Learning system
  - **Language and Grammar**: Understanding how a language and general grammar rules work
  - **General Artificial Intelligence**: Knowledge about **rule-based** and **knowledge-based** systems
  - **Statistics**: Building language models and measuring the accuracy of predictive models
  - **Algebra**: Matrix operations (i.e., embedding, Count Vectorizer, Tf-Idf, Bag of Words)
- Cole Howard (2019:11) define NLP as follows

  > *"Natural language processing* is an area of research in computer science and artificial intelligence (AI) concerned with processing natural languages such as English or Mandarin. This processing generally involves translating natural language into data (numbers) that a computer can use to learn about the world. And this understanding of the world is sometimes used to generate natural language text that reflects that understanding"
- For Zhang and Teng (2021:3), "NLP refers to the study of automatically processing or synthesing human languages"

UMBC

What Makes a Language Natural?

- It is important to understand what makes a language natural
- The language is '**natural**' because it is the way humans communicate
- **"Natural Language Processing** is a field of **artificial intelligence** that gives the machines the ability to read, understand and derive meaning from human languages. It is a discipline that focuses on the **interactions** between **data science and human language**, and is scaling to countless industries," (Medium, 2/25/2019)
- Language can be communicated through **speech**, **writing**, and **signs**
- Text data is notoriously unstructured
- A **Natural Language Processing** system is often referred to as a ***pipeline*** because it usually involves several stages of processing where natural language flows from one end and the processed output flows out to the other
- Text data belongs to a specific language following specific syntax and semantics
- The study of language includes to key elements
  - *The philosophy of language* deals with the nature of meaning in a language, the use of language, language cognition, and the relationship between language and reality
  - *Linguistics* is the scientific study of language, including form and syntax of language, meaning, semantics characterized by the use of language and context of use

UMBC

Why is Natural Language Processing Increasingly Popular?

- NLP is popular because it helps in a variety of sectors:
  - **Identification of diseases**, their origins, outcomes, and treatments through the interpretation of clinical trial reports and other reports whose access is permitted by law (Amazon Comprehend Medical)
  - **Improving customer service and community engagement** by analyzing concerns, emotions and sentiments of targeted groups (sentiment analysis)
  - **Cognitive assistance** can serve as an adviser in a variety of tasks such as job seeking and recommendations (IBM Watson Cognitive Assistant)
  - **Spam filtering** and **speech recognition**
  - **Fake news identification** to avoid the problem of trolls in social platforms (i.e., research from the NLP Group at MIT)
  - **Rumor identification** that helps financial traders in their buy or sell strategy (‘Buy the rumor, sell the news’)
  - **Talent recruitment** through the analysis of resumes and portfolios
  - **Identification of fraud** in reports collected by banks and insurance companies

UMBC

What is Linguistics?

- **Linguistics** includes these key concepts:
  - **Morphology** is the study of **morphemes**, which is the smallest unit of language that has distinctive meaning
  - **Phonetics** is the study of the **acoustic properties of sounds** produced by the human vocal tract during speech
  - **Phonology** is the study of **sound patterns** as interpreted in the human mind and used for distinguishing between different **phonemes** to find out which ones are significant
  - **Syntax** is the study of sentences, phrases, words, and their structures
  - **Semantics** involves the study of **meaning** in language and can be further subdivided into **lexical** and **compositional** semantics
- **Linguistics** also includes the study of
  - **Lexicon** is the study of properties of words and phrases used in a language and how they build the vocabulary of the language
  - **Pragmatics** is the study of how both linguistics and non-linguistic factors like content and scenario may affect the meaning of an expression of a message or an utterance
  - **Discourse analysis** is the analysis of languages and exchange of information in the form of sentences across conversations among human beings
  - **Stylistics** is the study of language with a focus on the style of writing, including the tone, accent, dialogue, grammar, and type of voice
  - **Semiotics** is the study of signs, symbols, and sign processes and how they communicate meaning
- **Language syntax** and **semantics** are the most important concepts that represent the foundations to NLP

UMBC

What is Semantics?

- It is the study of meaning. It is an important branch of Natural Language Processing
- **Lexical semantics** focuses on relationships between lexical units and how they are correlated to the structure and syntax of the language
- A **lexicon** is a complete vocabulary of the lexical units. Here are some of the components:
  - **Lemmas** are the base form for a set of words known as **lexeme** in this context
  - **Wordforms** are inflected forms of a lemma. For instance, the **lexeme** [ate, eating, eats], which contains the **wordforms**, and their **lemma** is the word eat
  - **Homonyms** are words that share the same spelling or pronunciation but have different meanings i.e., 'pen'
  - **Homographs** are words that have the same written form or spelling but have different meanings i.e., 'bat'
  - **Homophones** are words that have the same pronunciation but different meanings. They can have the same or different spelling i.e., 'aisle' and 'isle'
  - **Heteronyms** are words that have the same written form or spelling but different pronunciations and meanings i.e., 'lead' can mean 'direct' but it can also be a type of 'metal'
  - **Heterographs** are words that have the same pronunciation but different meanings and spellings i.e., 'one' and 'won'
  - **Polysemes** are words that have the same written form or spelling and different but very relatable meaning i.e., the 'bank' of a river and the 'bank' as the location where money is transacted
  - **Capitonyms** are words that have the same written form or spelling but have different meanings when capitalized ('March' v. 'march')

UMBC

What is Semantics? (Cont.)

- **Hyponyms** are words that are usually a subclass of another word i.e., ’daisy’ and ‘rose’
- **Hypernyms** are words that act as the superclass to hyponyms i.e., ‘fruit’ v. ‘mango’ and ‘orange’)
- **Synonyms** are words that have a different spellings and pronunciations but the same meaning in some or all contexts i.e., ‘beautiful’ and ‘pretty’
- **Antonyms** are pairs of words that define a binary opposite relationship i.e., ‘beautiful’ v. ‘ugly’
- **Holonyms** are entities that contain a specific entity of our interest i.e., ‘forest’ is a holonym for ‘tree’
- **Meronyms** are semantic relationships that relate a term or entity as a part or constituent of another term or entity i.e., ‘the upper branches and leaves of a tree or other plant

UMBC

What are the Challenges of NLP?

- **NLP** deals with extremely diverse and dynamic languages that continually evolve
- Here are some of the challenges:
  - **Lack of structure**. Textual data are unstructured, and we cannot apply mathematical models directly to them
  - **Splitting text into sentences.** Not all sentences start with a capital letter and end with a period
  - **Ambiguity**. Language can be figurative, ambiguous, and/or sarcastic:
    - How is the computer supposed to know you mean 'smart' when you use the word 'sharp'?
    - How can the computer interpret the following sentence accurately: *I saw Mary on top of the hill with a telescope*? Did you see her with a telescope, or did you notice she had a telescope in her possession?
  - **Emotions**. They are hard to define, identify, and measure.
    - How do you interpret the following sentence: 'The best I can say about this product is that it was definitely interesting.' Is it sarcastic or a true statement?
  - **Meaning conflation deficiency.** It is the inability to discriminate among different meanings of a word. For instance, 'bat' can be an animal and an object. This creates a challenge when transforming words into vectors (word embedding)
  - **Co-reference**. It is the problem of finding all expressions that refer to the same entity in a text.
  - **Many forms of natural language**. Natural language includes speeches, writing, and signs (think emoticons)

- For NLP, it is possible to use **supervised** (**Logistic Regression, SVM**) and **unsupervised** models (**Non-Negative Matrix Factorization, PCA**), as well as **Neural Networks** (**Recurrent Neural Networks, LSTM, Seq2Sec, BERT, and transformers**) to measure the accuracy of model, generate and summarize text, as well as to predict outcomes

UMBC

Some of the Main Applications of Natural Language Processing

- **Sentiment Analysis** is the analysis and classification of attitudes into categories
- **Machine Translation** is the technique that helps in providing syntactic, grammatical, and semantically correct translation between any two pairs of languages
- **Speech Recognition System** is complex because it is limited by the context. The **Turing test** is designed to evaluate if it is impossible to say which of the answers given was provided by the human
- **Question/Answering System** is based on the principle of information retrieval following a question that would be given by humans
- **Chatbots** are computer programs designed to simulate conversation with human user
- **Contextual Recognition and Resolution** pertain to syntactic and semantic-based reasoning. This includes word sense disambiguation. **Coreference resolution** is another problem in linguistics and refers to the issue when two or more terms in a text refer to the same entity
- **Text Summarization** consists in taking a corpus of text documents and reducing the content appropriately to create a summary that retains the key points of the collection. There are two types: extraction-based and abstraction-based summary. Text summarization provides **generic** and **query-based** summarization
- **Text Categorization** refers to identifying to which category or class a specific document should be placed based on the contents of the document. It is the most popular application of NLP that takes advantage of supervised and unsupervised models
- **Text Generation** leverages knowledge in computational linguistics and artificial intelligence to automatically generate natural language texts

Every Day's NLP for Practitioners

- **Information Retrieval** (Google finds relevant and similar results)
- **Information Extraction** (Gmail structures events from emails)
- **Machine Translation** (Google Translate translates language from one language to another)
- **Text Simplification** (Rewordify simplifies the meaning of sentences)
- **Sentiment Analysis** (Hater News gives us the sentiment of the user)
- **Text Summarization** (Smmry or Reddit's autotldr gives a summary of sentences)
- **Spam Filter** (Gmail filters spam emails separately)
- **Auto-Predict** (Google Search predicts user search results)
- **Auto-Correct** (Google Keyboard and Grammarly correct words otherwise spelled wrong)
- **Speech Recognition** (Google WebSpeech or Vocalware)
- **Question Answering** (IBM Watson's answers to a query)
- **Natural Language Generation** (Generation of text from image or video data)
- **Natural Language Understanding** (chatbots)

UMBC

Other Practical Applications of Natural Language Processing

Search	Web	Documents	Autocomplete
Editing	Spelling	Grammar	Style
Dialog	Chatbot	Assistant	Scheduling
Writing	Index	Concordance	Table of contents
Email	Spam filter	Classification	Prioritization
Text mining	Summarization	Knowledge extraction	Medical diagnoses
Law	Legal inference	Precedent search	Subpoena classification
News	Event detection	Fact checking	Headline composition
Attribution	Plagiarism detection	Literary forensics	Style coaching
Sentiment analysis	Community morale monitoring	Product review triage	Customer care
Behavior prediction	Finance	Election forecasting	Marketing
Creative writing	Movie scripts	Poetry	Song lyrics

Cole Howard (2019)

UMBC

Most Popular Libraries in NLP

- **NLTK** is the most well-known Python Natural Language Processing library (http://www.nltk.org)
- **scikit-learn** has classes to help create bags of word, vectorize words, and determine Term Frequency, Inverse Data Frequency) and model topics with Latent Dirichlet Allocation and Non-Negative Matrix Factorization
- **Gensim** (unsupervised topic modeling, document indexing, retrieval by similarity, and other natural language processing functionalities, using modern statistical machine learning)
- **SpaCy**
- **Langchain**
- **openAI**
- **TextBlob**
- **Stanford CoreNLP**
- **AllenNLP**
- **Fair**
- **Regular Expressions or re** (very useful to clean a text and prepare it for processing)
- **Text Rank** (information retrieval, text summarization)

- For those who use **R**, the most used libraries are quanteda, topicmodels, QDAP (Quantitative Discourse Analysis Package), tm (Text Mining), sentimentR, SentimentAnalysis, koRpus, maxent (maximum entropy models), lsa (latent semantic analysis), tidytext, and tidyverse, among others

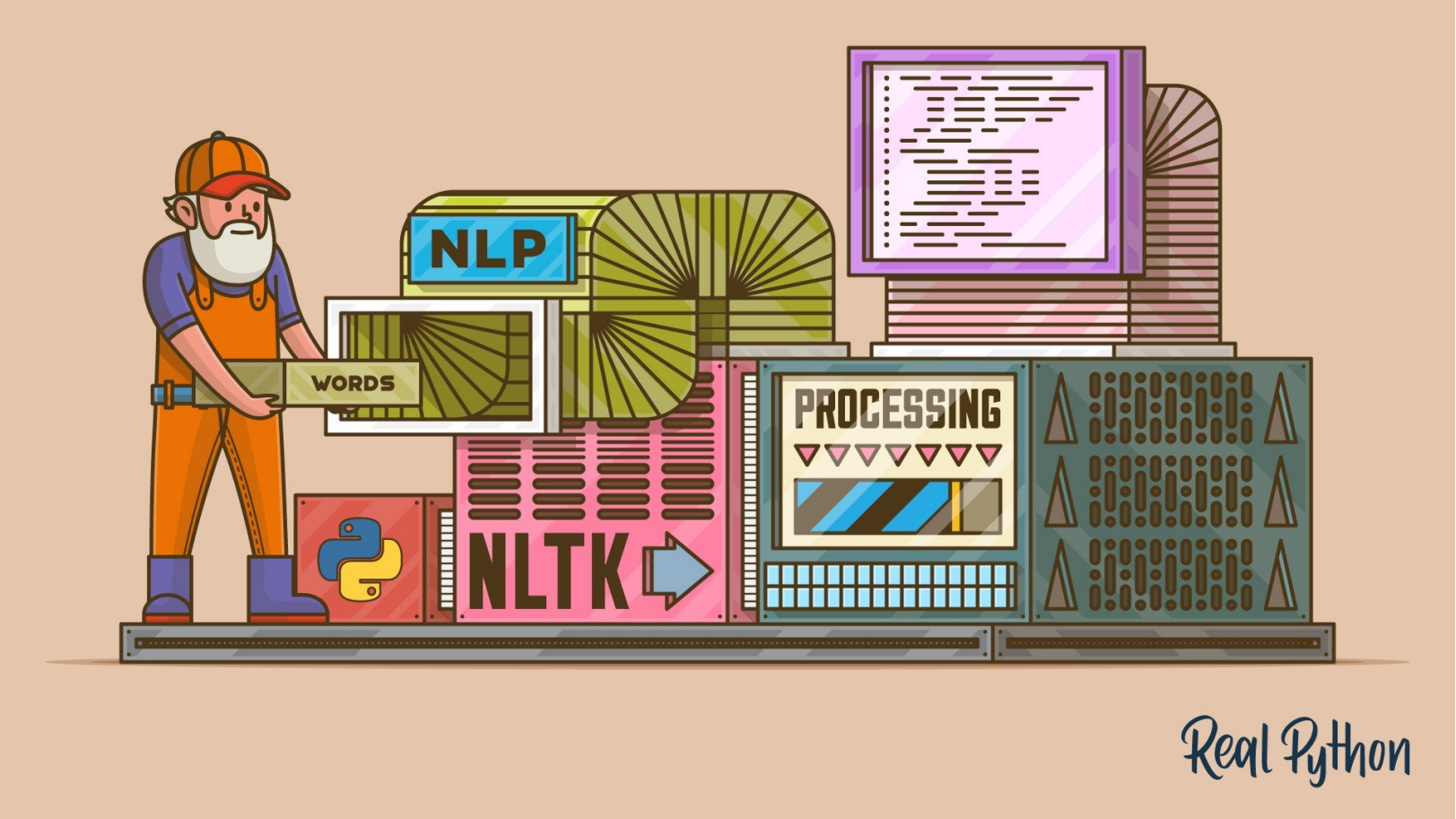
- For those who use **Julia**, the most popular libraries are TextAnalysis, and WordTokenizers, among others

Natural Language Processing and the Machine Learning Process

These are the important concepts that you learned in DATA 601 and DATA 602 that apply to NLP

- **Data Preparation** consists in pre-processing the data before extracting features and training
- **Feature Extraction** is the process of extracting useful features from raw data used to train machine learning models
- **Features** are various useful attributes of the data (age, weight, etc.)
- **Training Data** is a set of data points used to train a model
- **Testing/Validation Data** is a set of data points on which a pre-trained model is tested and evaluated to see how well it performs
- A **Model** is built on a combination of data/features and a machine learning algorithm that could be supervised or unsupervised
- **Accuracy** represents how well the model predicts a label. It also refers to detailed evaluation metrics such as precision, recall, and F1 score

APPLICATIONS OF NATURAL LANGUAGE PROCESSING



UMBC

2. TEXT MINING AND TEXT ANALYTICS

UMBC

Text Mining, NLP, and Text Analytics

- **Text mining** is a process of **exploring** sizeable textual data and find patterns
  - Finding **frequency counts of words**, **length of sentences**, **presence/absence of specific words** is known as **text mining**
  - Text mining is **preprocessed data** for **text analytics**

- **Natural Language Processing** is one of the components of text mining
  - NLP helps identify **sentiments**, find **entities** in the sentence, and categorize blogs/articles/posts

- **Text analytics** is the automated process of translating large volumes of **unstructured text** into **quantitative data** to **uncover insights, trends, and patterns**. It is the methodology and process followed to derive quality and actionable information and insights from textual data
  - In **text analytics**, statistical and machine learning algorithms are primarily used to
    - **Classify, cluster,** and **summarize information,** and
    - Perform **sentiment analysis**, **entity extraction** and **recognition**, **similarity analysis,** and **relation modeling**

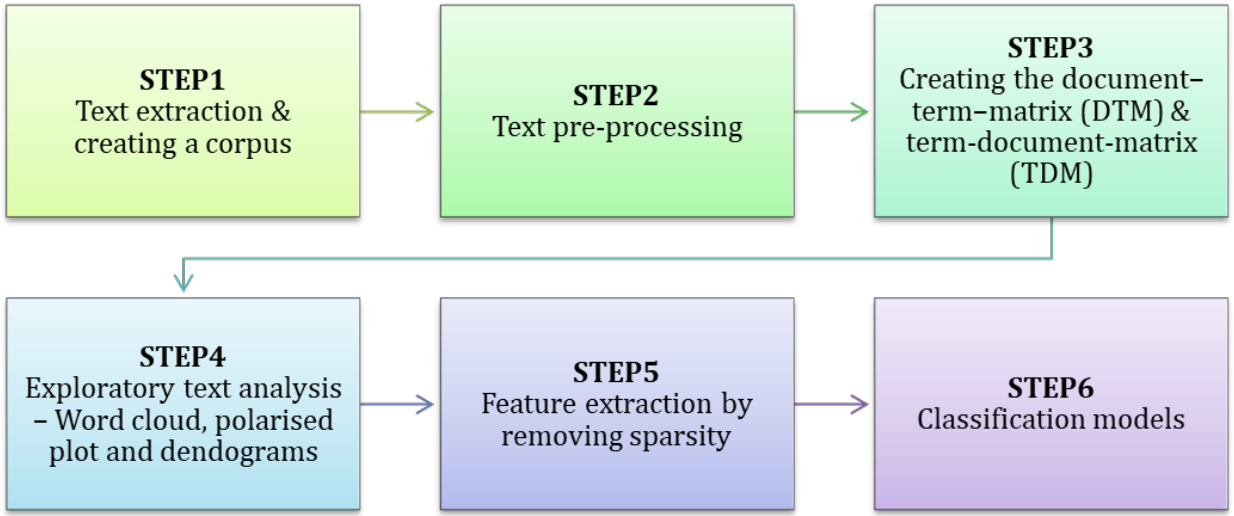
- **Text analytics** is designed to have a better understanding of the text content
- **Text analytics** resorts to the **frequency distribution** to identify the occurrence of words, count words by genre, plot and tabulate a distribution. With the NLTK library, it is also possible to compute the conditional frequency of a distribution ('cfdist' and 'cfdist.plot()')
- Below is an example of how to compute the distribution frequency:

```
from nltk.book import *
print("\n\n\n")
freqDist = FreqDist(text1)
print(freqDist)

# The class FreqDist  works like a dictionary where the keys are the words in the text and the values are the count
# associated with that word. For example, if you want to see how many words "man" are in the text, you can
# type

print(freqDist["man"])

print(type(words))
```

What is Text Mining?

- It is the process of **examining large collections of text** and **converting unstructured text data into structured data** for further analysis like **visualization** and **model building**
- Customer reviews represent a source of the "Voice of the customer", and they could offer tremendous insight into what customers like and dislike about a product or service
- For the e-commerce business, customer reviews are very critical, since existing reviews heavily influence buying decisions of new customers in the absence of the actual look and feel of the product to be purchased

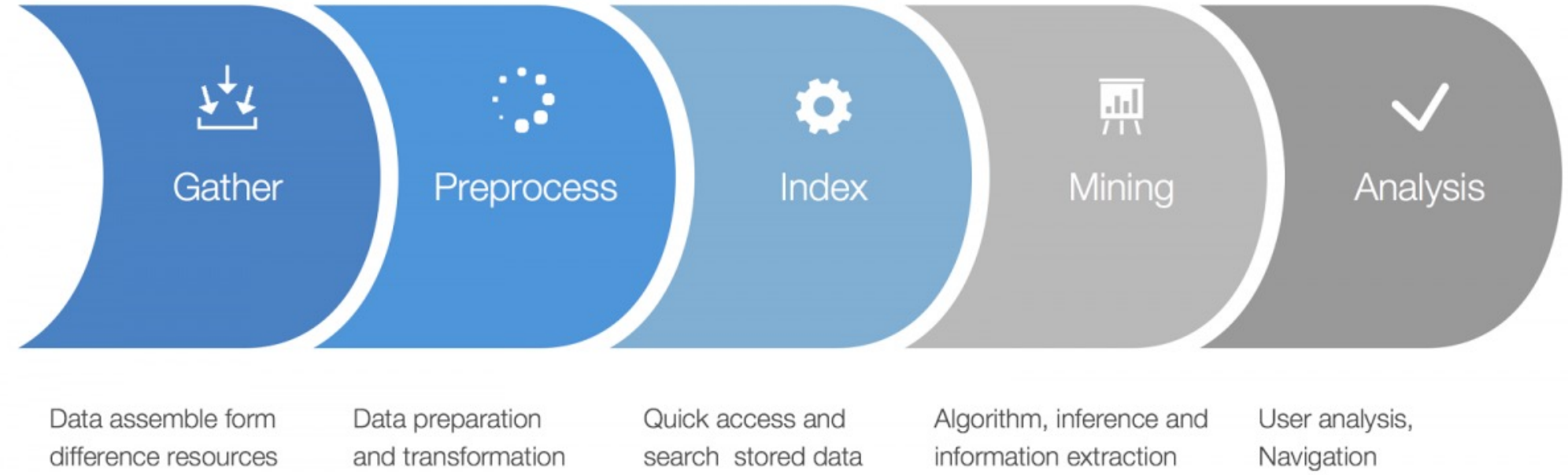
What are the Key Steps in Text Mining?



UMBC

The Text Mining Process

1. **Collect text(s)** into a document called **'corpus'**
2. **Analyze** the corpus
3. **Feature Generation**: Create a bag of words
4. **Feature Selection**: Count words and derive some statistics
5. **Text/Data Mining**: Use **classification** (supervised learning) or **clustering** (unsupervised learning)
6. **Analyze** and **communicate** the results



What are the Two Techniques in Text Mining?

- **Exploratory Analysis of Text Data.** For instance, customer reviews can serve as a basis for further recommendations
- **Classification Models** use review text data—as the independent variable—to predict whether a customer may recommend a product or not (target variable)

Two Main Approaches in Text Mining

- **Semantic Parsing:** The word sequence, word usage (i.e., noun or verb), or the hierarchical word structure, among others, matters
  - It is the task of **translating natural language** into a **formal meaning representation** upon which a machine can act
  - It is the **process of mapping a natural-language sentence** into a **formal representation of its meaning**
- **Bag of words (BOW):** All the words are analyzed as a single token and order does not matter
  - The bag of word approach does not care about the order of the words, or how many times a word occurs
  - All that matters is whether the word is present in a list or a 'bag' of words. It can be defined as follows:

```
def bag_of_words(words):
    return dict([(word, True) for word in words])
```

Text Mining: Web Scraping

- The incredible amount of data on the Internet is a rich resource for any field of research or personal interest
- The Python libraries ***request*** and ***BeautifulSoup*** are powerful tools to accomplish that task
- **Web scraping** is simply the process of gathering information from the Internet
- The words 'web scraping' usually refer to a process that involves automation
- Keep in mind that some websites do not like it when automatic scrapers gather their data, while others do not mind
- It is a good idea to do some research to make sure you are not violating any 'Terms of Service' before you start a large-scale project
- **Automated web scraping** can be a solution to speed up the data collection process: Once you write the codes, they can be re-used to get information repeatedly. Manual web scraping can take a lot of time and repetition
- The major challenge associated with **web scrapping** is the huge amount of **unstructured** information, its **variety**, and **durability**
- Because the Internet is dynamic, the automatic web scrapers usually require continuous maintenance and updates
- It is important to set up continuous integration to run scraping tests periodically, which ensures the main script is still valid and operational
- Some website providers offer **Application Programming Interfaces (APIs)** that allow access to data in a predefined manner such as ***WebHarvy*** (https://www.webharvy.com), ***parsehub*** (https://www.parsehub.com)
- With APIs, you can avoid parsing HTML and, instead, access the data directly using formats like JSON and XML
- HTML is primarily a way to visually present content to users

UMBC

Text Mining: Different Methods of Web Scraping

- **Human copy-and-paste:** Tedious but effective because of targeting and scraping the intended type of information
- **Text pattern matching:** It is the process of checking a given **sequence** of tokens to identify elements of patterns
- **HTTP programming:** Static and dynamic web pages can be retrieved by posting HTTP requests to the remote web server using socket programming
- **HTML parsing:** It relies on a wrapper as a program that detects templates, extracts its content, and translates it into a relational form. Jsoup is an HTML Java parser. It involves taking in HTML code and extracting relevant information like the title of the page, paragraphs in the page, headings in the page, links, bold text etc.
- **Document Object Model (DOM) parsing:** The browser controls can parse web pages into a DOM tree, which serves to retrieve parts of the pages with languages such as Xpath. The **DOM** represents a **document** with a logical tree

UMBC

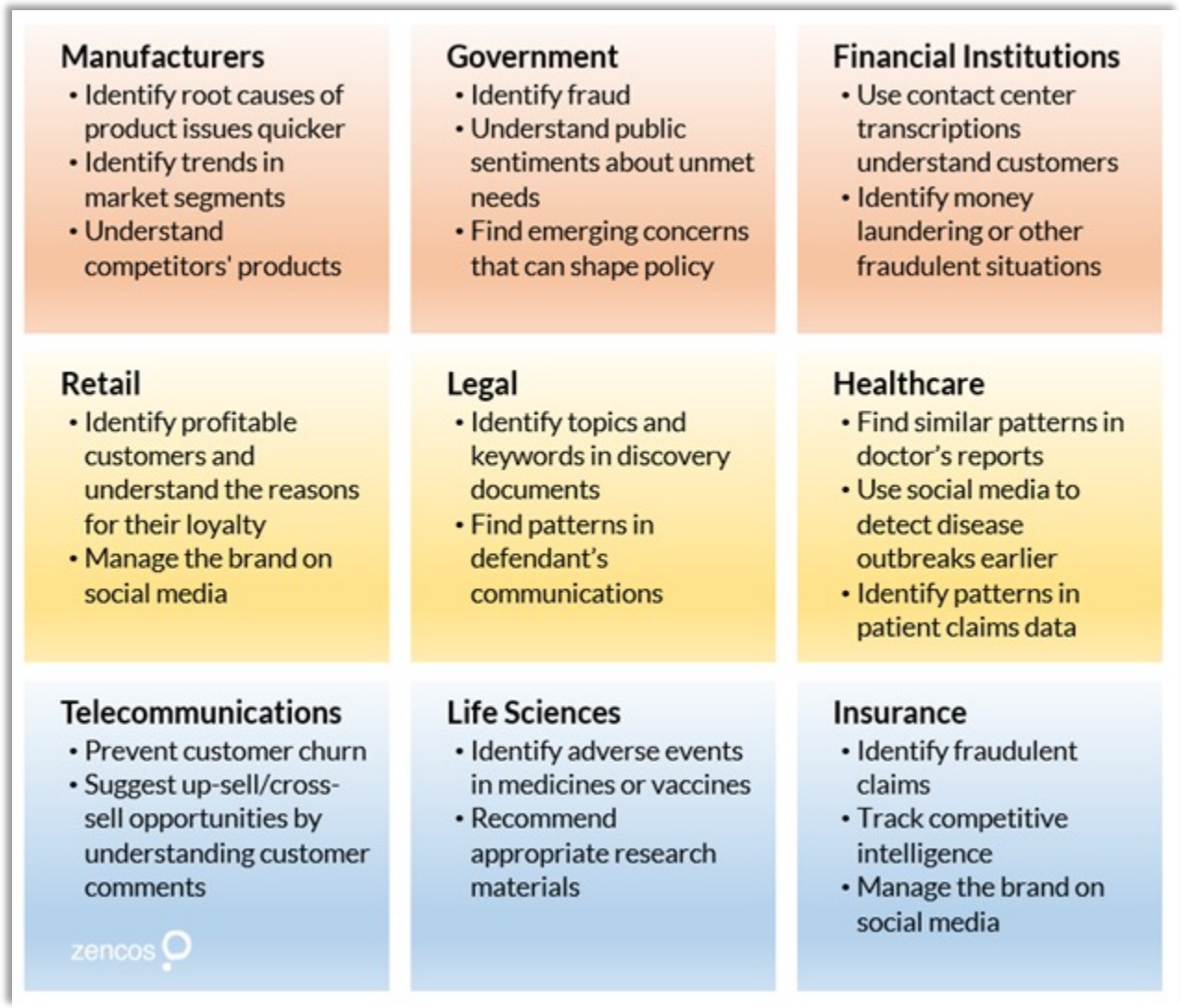
Text Mining: Different Methods of Web Scraping (Cont.)

- **Vertical aggregation:** Google, Yahoo, or Bing search services, commonly referred to as verticals, include highly-specialized search engines that focus on a particular type of media (e.g., images, video), search engines that focus on a particular type of search task (e.g., news, local), and applications that return a specific type of information (e.g., weather, stock quotes). Aggregated search is the task of providing integrated access to all these different vertical search services, and to the core Web search engine, within a common search interface—a single query box and a unified presentation of results
- **Semantic annotation recognizing:** Semantic annotation or tagging is the process of attaching additional information to various concepts (e.g., people, things, places, organizations, etc.) in a specific text or any other content. Unlike classic text annotations, which are for the reader's reference, semantic annotations are used by machines
- **Computer vision web-page analysis:** Computer vision describes the ability of machines to process and understand visual data. Computer vision helps identify web pages quickly, making it possible to strategically pull product information, images, videos, articles and other data without having to sort through unnecessary information

- **JSON (JavaScript Object Notation)** is a lightweight data-interchange format
- It is easy for humans to read and write and easy for machines to parse and generate
- It is based on a subset of the **JavaScript** Programming Language Standard ECMA-262 3rd Edition - December 1999
- JSON is a text format that is completely **language-independent** but uses conventions familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others
- These properties make **JSON** an ideal data-interchange language
- **JSON** is built on two structures:
  - **A collection of name/value pairs**. In various languages, this is realized as an *object*, *record*, *structure*, dictionary, hash table, keyed list, or associative array
  - **An ordered list of values**: In most languages, this is realized as an *array*, vector, list, or sequence
- Since the **JSON** format is text only, it can easily be sent to and from a server, and used as a data format by any programming language
- JavaScript has a built-in function to convert a string, written in **JSON** format, into native JavaScript objects, which is the 'JSON.parse()' function
- Data from a server in **JSON** format can be used like any other JavaScript object
- This site provides comparisons of **JSON** with other languages: https://json.org/example.html

UMBC

Text Analytics: Use Cases



Data Analytics: Quantifying the Content of a Corpus (Cont.)

One important function in FreqDist class is the .keys() function. if you run your code now, you can see that it returns you the class dict_keys , in other words, you get a list of all the words in your text

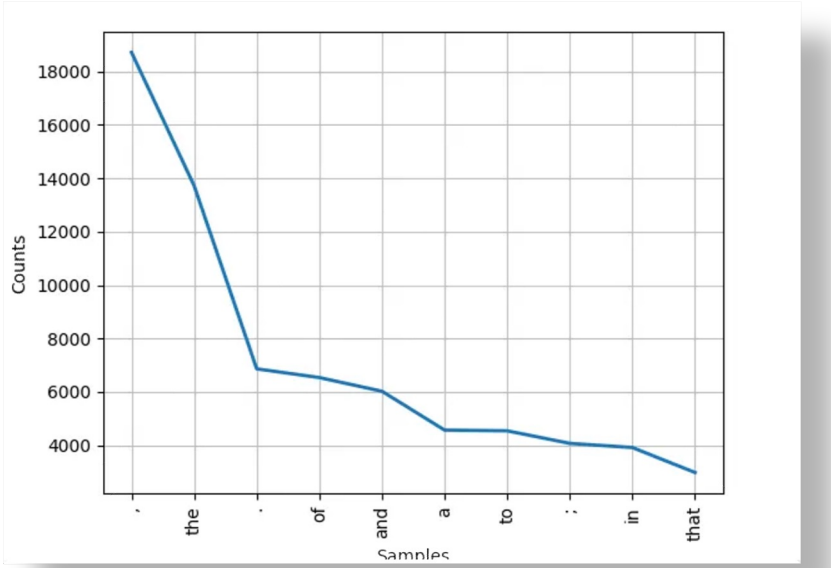
words = freqDist.keys()

To see how many words there are in the text, you can write:

print(len(words))

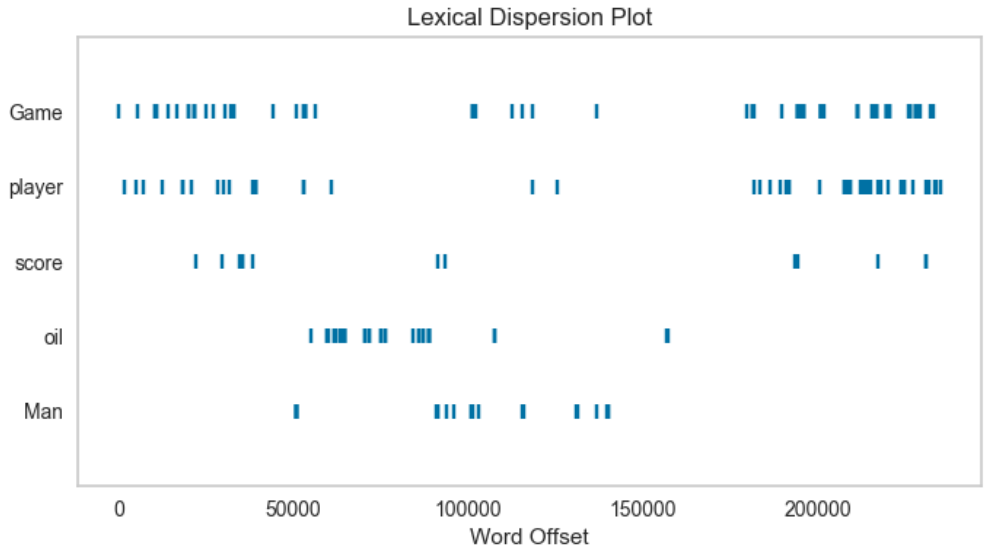
Another useful function is plot() . You can determine the number of words you want to show such as

freqDist.plot(10)



Data Analytics: Quantifying the Content of a Corpus (cont.)

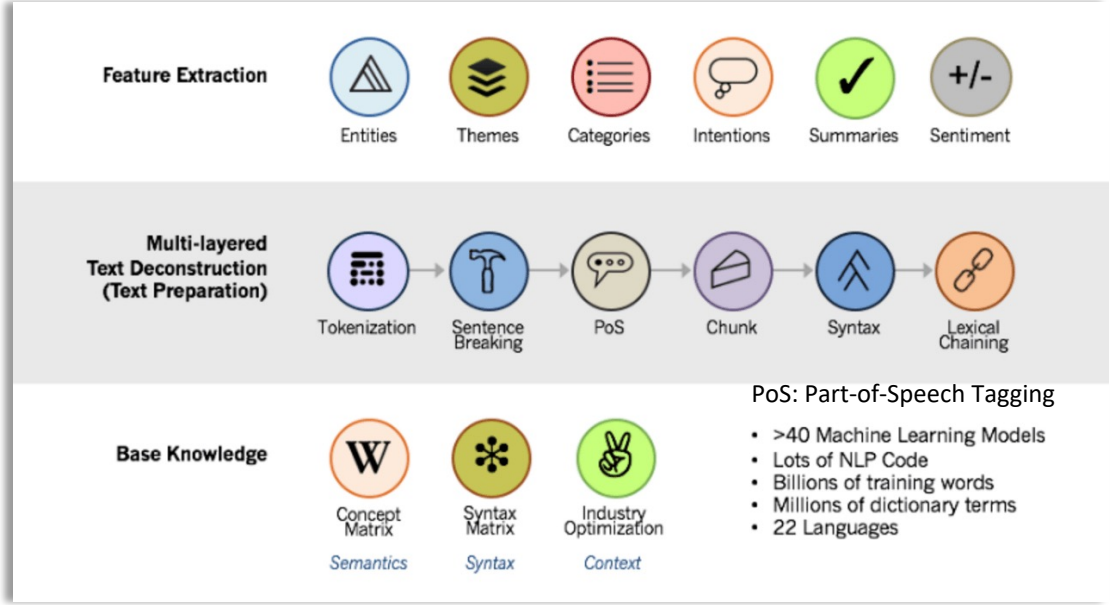
- The 'Yellowbrick' library in Python allows to visualize the word dispersion in a **corpus** or text under study
- Below is a **lexical dispersion plot** that shows how many times the word 'game' appears in a corpus from beginning to end and at what density

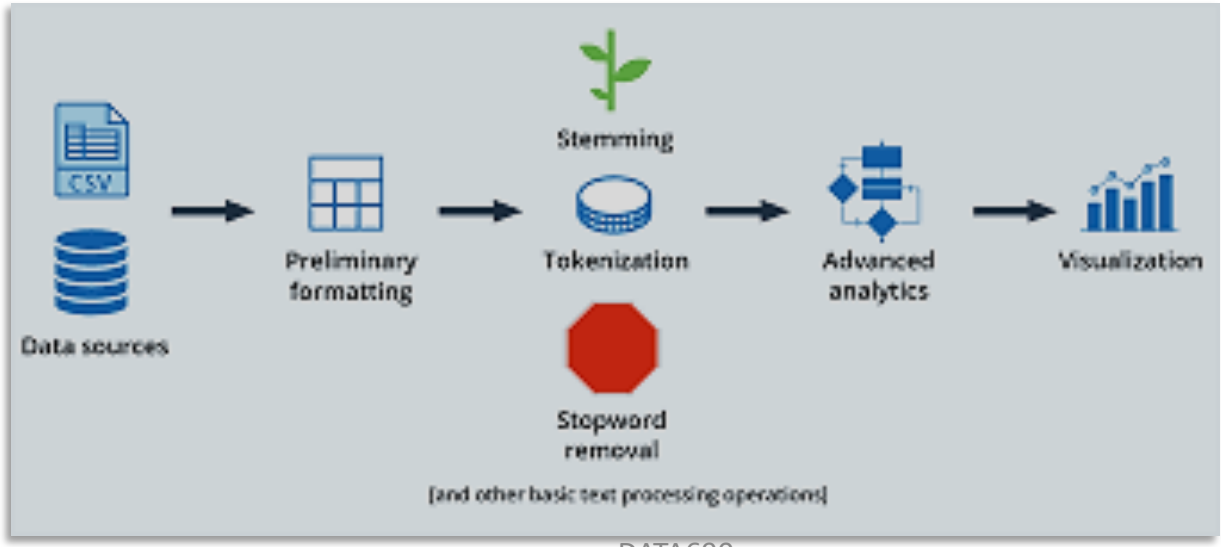


UMBC

3. PREPARING TEXT BEFORE ANALYSIS

Preparing Text Before Analysis



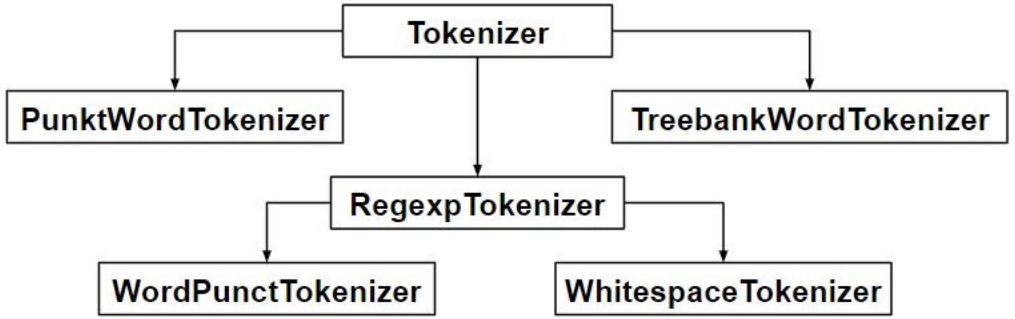


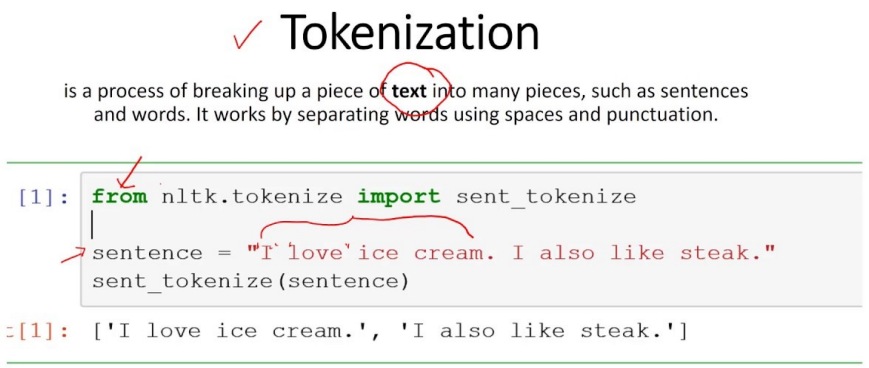
Source: Lexalytics

Preparing Text Before Analysis (Cont.)

Text Normalization or Text Wrangling

- **Text wrangling** is the process that consists of a series of steps to wrangle, clean, and standardize textual data into a form that can be used for further analysis
- **Tokenization** is a process of breaking down a text paragraph into smaller chunks such as a word or sentence. Tokenization occurs before or after removing unnecessary characters and symbols from the data
- **Word tokenization** is important to cleaning and normalizing text where operations like stemming and lemmatization work on each individual word based on its respective stems and lemma. NLTK provides various user interfaces to carry out tokenization such as 'word_tokenize' and 'TreebankWordTokenizer'
- **Sentence Tokenization** breaks text paragraph into sentences. Basic techniques include looking for specific delimiters between sentences, such as a period (.), or a new line character (\n), and sometimes a semi-colon (;). NLTK provides two tokenizers for sentences: 'sent_tokenize' and 'PunktSentenceTokenizer'





Preparing Text Before Analysis (Cont.)

Text Normalization

- **Clean text** by eliminating html tags with a library called '**beautifulsoup**' when scraping data from the Web
- **Eliminate stopwords** considered as noise in the text. Text may contain stopwords such as 'is,' 'am,' 'are,' 'this,' 'a,' 'an,' or 'the' that do not add any information on text sentiment or meaning
- **Remove special characters**. They may be symbols or punctuation characters. These characters do not add much significance in feature extraction and interpretation. They can be for instance http tags, JavaScript, etc.
- **Lower case conversion** enables a text to be converted only to lower case words
- **Correcting words** that are not correctly spelt ensures no important information is lost after tokenization
- **Correct repeating characters** in a word using a regex pattern and use a substitution to remove the characters one by one i.e., finallyyyyyy!
- **Stemming** is a process of linguistic normalization, which **reduces words to their root** or chops off the derivational affixes
  - **Morphemes** as the smallest independent units in a language consist of stems (or wordforms) and affixes (prefixes, suffixes) that create new word and change the meaning of the stems
  - **Inflection** is the process of attaching a suffix to a stem
  - For example, 'connection,' 'connected,' and 'connecting' reduce to a common word 'connect'

```
1  from nltk.stem import PorterStemmer
2  from nltk.tokenize import sent_tokenize,
   word_tokenize
3  sentence="Hello Guru99, You have to
   build a very good site and I love
   visiting your site."
4  words = word_tokenize(sentence)
5  ps = PorterStemmer()
6  for w in words:
7          rootWord=ps.stem(w)
8          print(rootWord)
```

There are two main stemmers: PorterStemmer and LancasterStemmer

Preparing Text Before Analysis (Cont.)

Text Normalization

- **Lemmatization** reduces words to their **base word** or **root word,** which is linguistically correct **lemmas**
  - The **root stem** may not always be a lexicographically correct word: it may not be in the dictionary
  - The **root word** known as the lemma will be in the dictionary
  - NLTK has a lemmatization module that uses WordNet including the word's syntax, semantics, and context
  - It transforms root words with the use of vocabulary and morphological analysis
  - A **stemmer** works on an individual word without knowledge of the context
    - For example, "better" has "good" as its lemma
- **Part-of-Speech** (POS) **Tagging** serves to identify the grammatical group of a given word. It is the process of classifying and labeling POS tags for words called parts of speech tagging
  - **POS tagging** is important to determine prominent words, word sense disambiguation, and grammar analysis
  - **POS** are specific lexical categories to which words are assigned based on their syntactic context and role
  - There are mainly three entities (nouns, verbs, adjectives) that occur most frequently in natural language
  - **POS tagging** looks for relationships within the sentence and assigns a corresponding tag to the word such noun, pronoun, adjective…

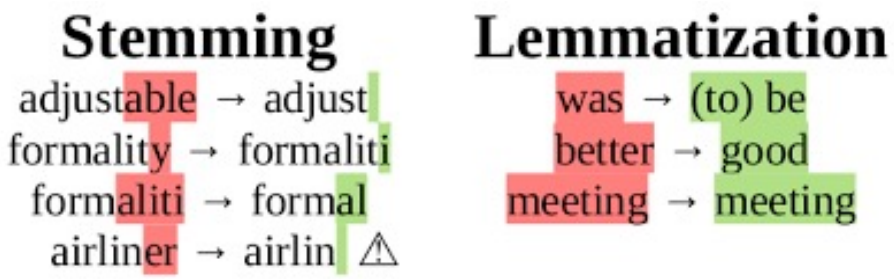
- These text treatments (cleaning and normalization) make it possible to evaluate the **sentiment** in a text

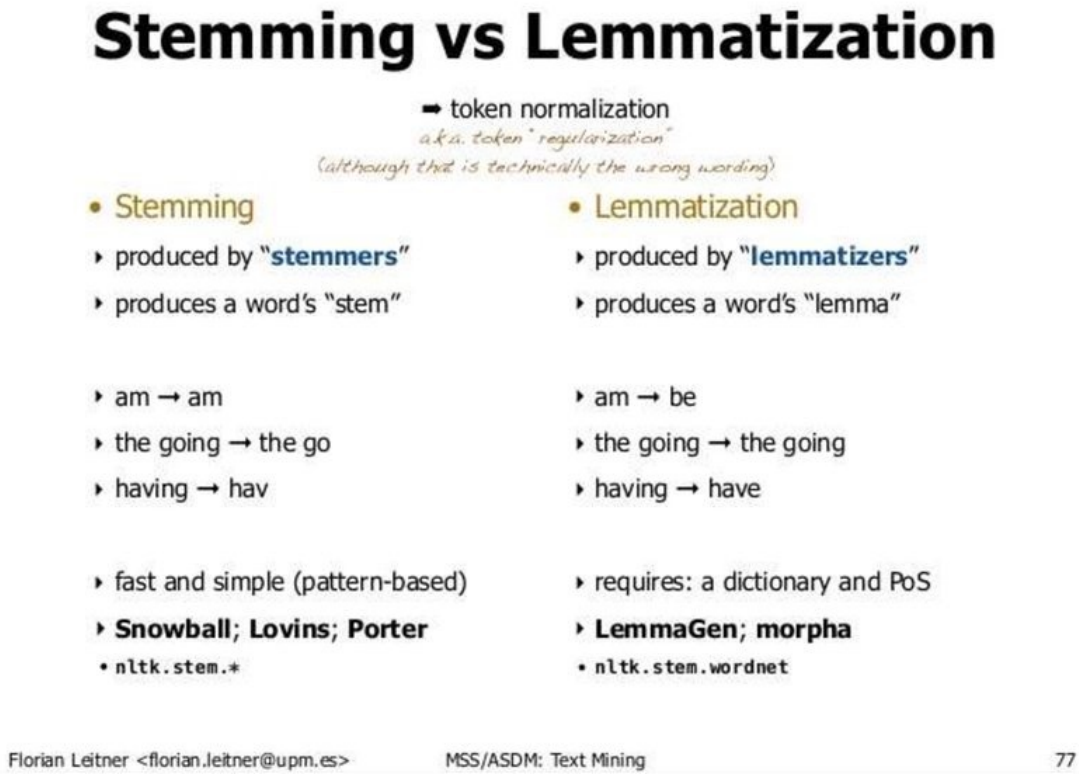
Preparing Text Before Analysis (Cont.)

Text Normalization

Difference between Stemming and Lemmatization

Stemming	Lemmatization
Word Representation may not have any meaning	Word Representation has meaning
Takes Less Time	Takes More time than Stemming
Use stemming when the meaning of the word is not important for analysis. Example – Spam Detection	Use lemmatization when the meaning of the word is important for analysis. Example – Question Answering Application





There are several types of stemmers:

- PorterStemmer
- LancasterStemmer
- RegexpStemmer
- SnowballStemmer

UMBC

Preparing Text Before Analysis (Cont.)

```
import re

# sample review from the IMDB dataset.
review = "<b>A touching movie!!</b> It is full of emotions and wonderful acting.<br> I could have sat through it a second time."

cleaned_review = re.sub(re.compile('<.*?>'), '', review) #removing html tags
cleaned_review =  re.sub('[^A-Za-z0-9]+', ' ', cleaned_review) #taking only words

print(cleaned_review)
```

```
A touching movie It is full of emotions and wonderful acting I could have sat through it a second time
```

```
cleaned_review = cleaned_review.lower()

print(cleaned_review)
```

```
a touching movie it is full of emotions and wonderful acting i could have sat through it a second time
```

UMBC

Preparing Text Before Analysis (Cont.)

```
# import nltk
# nltk.download('punkt')

from nltk.tokenize import word_tokenize

tokens = nltk.word_tokenize(cleaned_review)

print(cleaned_review)
print(tokens)
```

```
a touching movie it is full of emotions and wonderful acting i could have sat through it a second time
['a', 'touching', 'movie', 'it', 'is', 'full', 'of', 'emotions', 'and', 'wonderful', 'acting', 'i', 'could', 'have', 'sat', 'th
rough', 'it', 'a', 'second', 'time']
```

```
# nltk.download('stopwords') # you will have to download the set of stop words the first time
from nltk.corpus import stopwords

stop_words = stopwords.words('english')

filtered_review = [word for word in tokens if word not in stop_words] #removing stop words.

print(filtered_review)
```

```
['touching', 'movie', 'full', 'emotions', 'wonderful', 'acting', 'could', 'sat', 'second', 'time']
```

UMBC

Preparing Text Before Analysis (Cont.)

```
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

stemmed_review = [stemmer.stem(word) for word in filtered_review]

print(stemmed_review)
```

```
['touch', 'movi', 'full', 'emot', 'wonder', 'act', 'could', 'sat', 'second', 'time']
```

```
# nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

lemm_review = [lemmatizer.lemmatize(word) for word in filtered_review]

print(lemm_review)
```

```
['touching', 'movie', 'full', 'emotion', 'wonderful', 'acting', 'could', 'sat', 'second', 'time']
```

Preparing Text Before Analysis (Cont.)

Resolving the Issue of Sentence Detection

- One of the most important and basic problems in processing text is **to identify sentence boundaries** in a paragraph of text in order to provide context and support meaning
- Identifying sentence boundaries helps break paragraphs into its constituents which can be further processed upon necessity (like **POS tagging** , **stemming**, etc.)
- Basic boundary detection focuses on identifying dots, semi-colons, exclamations, question marks to break the text stream
- This may lead to errors because semantically full stops may not mark a true end of a sentence. Instead, they may refer to abbreviations, names, titles and more, as illustrated below:

  U.S. Corporate
  Mr. Kolmogrov
  Kidding!
  Christopher K. Columbus
  For e.g.
  "Are you kidding?" asked the Senator.
  aftermath@gmail.cs.iastate.edu

- The above samples are not sentence boundaries. Therefore, a simple full stop detection would not work here

Preparing Text Before Analysis (Cont.)

Resolving the Issue of Sentence Detection

- The research article which was implemented for this problem is available at http://nlp.stanford.edu/courses/cs224n/2005/agarwal_herndon_shneider_final.pdf
- We start by training a **Maximum Entropy Classifier with several features based on local context**
  - **Maximum Entropy Classifier** works best for these classes of problems when we try to classify context as a feature
  - The problem at hand is a binary classification where we ask the classifier whether a token is a "sentence boundary"
  - If the classifier says it is true, then we have a sentence boundary. Otherwise, we do not
- When considering paragraph as a stream of text, for each token, we have the previous and next tokens to provide context information
- In the research article referenced above, using **Part-of-Speech tags** data do not improve accuracy. Therefore, POS tags are not considered as features
- When features represent true or false **EOS** (End of Sentence) along with their context information, the **Maximum Entropy Classifier** uses this information to determine a process model for generating training samples
- Finally, we use the identified distribution to evaluate whether incoming tokens represent sentence boundaries

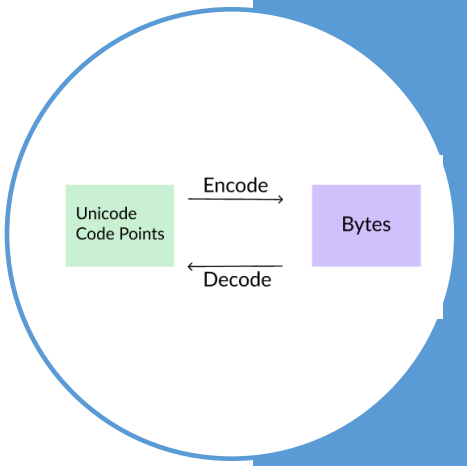
Resolving the Issue of Sentence Detection: How to Use Python to Determine the Sentence Boundary?

```
import en_core_web_sm
nlp = en_core_web_sm.load()
import spacy
import re
from spacy.language import Language
boundary = re. compile ('^ [0-9]$')
@Language.component ("component" )
def custom_seg (doc) :
prev = doc[0].text
length = len (doc)
for index, token in enumerate (doc) :
if (token.text == '.' and boundary.match (prev) and index!=(length - 1)):
doc[index+1].sent_start = False
prev = token.text
return doc
nlp.add_pipe ("component", before='parser')
doc = nlp ('My name is Jonas E. Smith. Please turn to p. 55.')
print (list (doc.sents))
```

UMBC

Encoding and Decoding

- A **Unicode** string is a sequence of code points, which are numbers from 0 through 0x10FFFF (1,114,111 decimal)
- This sequence of code points needs to be represented in memory as a set of **code units** and **code units** are then mapped to 8-bit bytes
- The rules for translating a Unicode string into a sequence of bytes are called a **character encoding**, or just an **encoding**
- **UTF** stands for “Unicode Transformation Format”
- **UTF-8** is one of the most used **encodings** and **Python** often defaults to using it
- The '**8**' means that **8**-bit values are used in the **encoding**
- **UTF-8** uses **1, 2, 3 or 4 bytes** to encode every code point
  - It is backwards compatible with ASCII
  - All English characters just need 1 byte — which is quite efficient. We only need more bytes if we are sending non-English characters
- There are also **UTF**-16 and **UTF**-32 **encodings**, but they are less frequently used than **UTF-8**
- We need the ‘encode’ method to convert Unicode code points to bytes. This will happen typically during writing string data to a CSV or JSON file for example
  We also need ’decode’ method to convert bytes to Unicode code points. This will typically happen during reading data from a file into strings.



UMBC

4. PARSING AND PART-OF-SPEECH TAGGING

Parsing

Parsing is the task of assigning structure to a sentence. It serves to

- Analyze a sentence in terms of grammatical constituents, identifying the parts of speech, syntactic relations, etc.
- Describe a word in a sentence grammatically, identifying the part of speech, inflectional form, syntactic function, etc.

Parsing is the process of determining the **syntactic structure** of a text by analyzing its **constituent** words based on an underlying **grammar**

There are three challenges in parsing

- **Ambiguity**. A grammar is said to ambiguous if, for any string generated by it, it produces more than one parse tree. Ambiguity depends on
  - **Lexical ambiguity** (noun v. verb such as 'book' vs. 'to book'),
  - **Attachment ambiguity** (constituent can attach in multiple places i.e., I shot an elephant in my pyjamas), and
  - **Coordination ambiguity** (different constituents can be conjoined such as old men and women)
- **Repeated Substructures.** Efficient parsing techniques require storage of shared substructure. Dynamic programming is used to avoid repeated work by tabulating solutions to subproblems
- **Recursion** is the repeated sequential use of a particular type of linguistic element or grammatical structure
  - The fact that English permits more than one adjective in a sequence is an example of a more general feature of languages that linguists call **recursion**
  - **Recursion** is often used to create expressions that modify or change the meaning of one of the elements of the sentence. Recursion refers to the ability of a language model or algorithm to process nested structures within language, where a phrase or clause can be embedded within another phrase or clause of the same type

UMBC

Parsing (Cont.)

- **Parsing** is the method of analyzing a sentence to determine its structure according to a grammar. It can be accomplished
  - Top down, or
  - Bottom up
- A **grammar** is defined as a set of rules that can generate strings. The set of all strings that can be generated from a grammar is called the **language of the grammar**
- **Parsing** raises the membership problem such that grammar G generates a language L(G). Then, is a given string a member of L(G)?
- In a **context-free grammar (CFG)**, G=(V, X, R, S) and we have a string w where
  - *V* is a finite set of variables or non-terminal symbols,
  - *X* is a finite set of terminal symbols,
  - *R* is a finite set of rules,
  - *S* is the start symbol, a distinct element of *V*, and
  - *V* and *X* are assumed to be disjoint sets.
- We need to convert **context-free grammar (CFG)** into **Chomsky Normal Form (CNF)**
- A **context-free grammar (CFG)** is in **Chomsky Normal Form (CNF)** if all production rules satisfy one of the following conditions:
  - A non-terminal generating a terminal (e.g., X-->x)
  - A non-terminal generating two non-terminals (e.g., X-->YZ)
  - Start symbol generating ε. (e.g., S--> ε)

Parsing (Cont.)

- In formal language theory, a **context-free grammar (CFG)** is a set of rules used to generate patterns of strings in a formal language
- It is called "**context-free**" because the rules can be applied regardless of the context of a symbol within a string.
- Here are the key components of a CFG:
  1. **Variables** (non-terminal symbols): These represent abstract structures or categories in the language, like nouns, verbs, or phrases. They are typically denoted by uppercase letters (e.g., S, NP, VP).
  2. **Terminal symbols**: These are the actual words or symbols that make up the language, like "cat," "run," or "+". They are typically denoted by lowercase letters or characters
  3. **Start symbol**: This is a special variable that represents the root of all possible strings in the language. It's usually denoted by S
  4. **Production rules**: These are the core of a CFG. They define how variables can be expanded into strings of terminals and/or other variables. They are written in the form of A → β, where A is a variable and β is a string of terminals and/or variables
- Here is a simple CFG to generate English sentences:
  - S → NP VP (A sentence can consist of a noun phrase followed by a verb phrase)
  - NP → Det Noun (A noun phrase can consist of a determiner followed by a noun)
- Cannot capture all aspects of natural language: CFGs cannot capture all the nuances and complexities of natural language, such as long-range dependencies and semantic relationships
- Ambiguity: Some CFGs can produce ambiguous sentences, meaning they have multiple possible interpretations
- Expressiveness: CFGs cannot capture some types of language features (cross-serial dependencies and agreement patterns)

Parsing (Cont.)

- **The Cocke–Younger–Kasami-Algorithm (CYK or CKY)** is a highly efficient parsing algorithm for context-free grammars
- It helps resolve the word-problem for context-free grammars, given in Chomsky normal form (CNF)
- It is used to check if a particular string can be derived from the language generated by a given grammar
- It is also called the membership algorithm as it tells whether the given string is a member of the given grammar or not

- In a CYK algorithm, the structure of grammar should be in Chomsky normal form
- The CYK algorithm uses the dynamic programming or table filling algorithm
- The grammar will be in CNF if each rule has one of the following forms:
  - $A \rightarrow BC$ (at most two variables on the right-hand side)
  - $A \rightarrow$ a (a single terminal on the right-hand side)
  - $S \rightarrow \emptyset$ (null string)
- If the given Grammar is not in the CNF form, convert it to the CNF form before applying the CYK Algorithm

See: https://www.tutorialspoint.com/explain-about-cyk-algorithm-for-context-free-grammar

Probabilistic Context-Free Grammars (PCFGs)

- Probabilistic grammar (PG) is a type of phrase structure grammar that assigns probabilities to the symbols and rules
- Probabilities are numerical values that indicate how likely a word or phrase is to occur in a given context
  - For example, a rule like S -> NP VP[0.8] | VP NP[0.2] means that a sentence (S) can be composed of either a noun phrase (NP) followed by a verb phrase (VP) with 80% probability, or a verb phrase (VP) followed by a noun phrase (NP) with 20% probability
- PG can capture some aspects of natural language that other types of phrase structure grammar cannot, such as ambiguity, pragmatics, and probabilistic preferences
- However, PG is also more complex and less efficient to implement and parse, and it requires large amounts of data to estimate the probabilities accurately

Probabilistic Context-free Grammar (PCFG) (Cont…)

- Figure shows a sample PCFG for a miniature grammar with only three nouns and three verbs.
- Note that the probabilities of all of the expansions of a nonterminal sum to 1.
- Obviously in any real grammar there are a great many more rules for each nonterminal and hence the probabilities of any particular rule are much smaller.

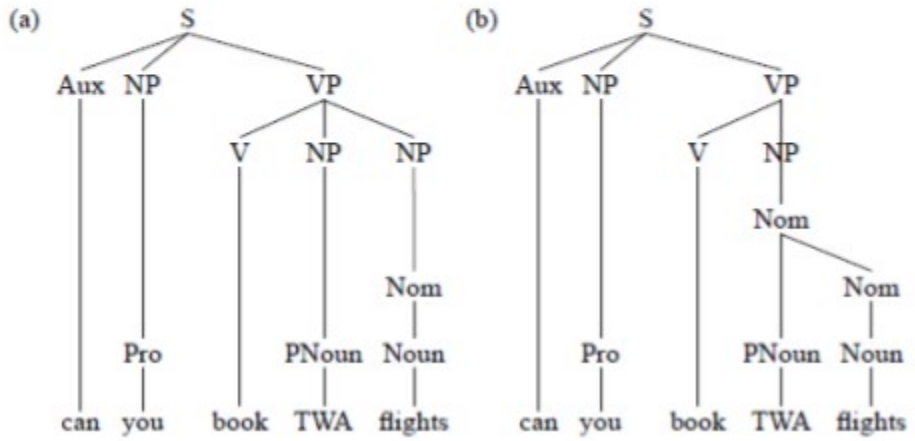
$S \rightarrow NP\ VP$	[.80]	$Det \rightarrow that$ [.05] | the [.80] | a	[.15]
$S \rightarrow Aux\ NP\ VP$	[.15]	$Noun \rightarrow book$	[.10]
$S \rightarrow VP$	[.05]	$Noun \rightarrow flights$	[.50]
$NP \rightarrow Det\ Nom$	[.20]	$Noun \rightarrow meal$	[.40]
$NP \rightarrow Proper\text{-}Noun$	[.35]	$Verb \rightarrow book$	[.30]
$NP \rightarrow Nom$	[.05]	$Verb \rightarrow include$	[.30]
$NP \rightarrow Pronoun$	[.40]	$Verb \rightarrow want$	[.40]
$Nom \rightarrow Noun$	[.75]	$Aux \rightarrow can$	[.40]
$Nom \rightarrow Noun\ Nom$	[.20]	$Aux \rightarrow does$	[.30]
$Nom \rightarrow Proper\text{-}Noun\ Nom$	[.05]	$Aux \rightarrow do$	[.30]
$VP \rightarrow Verb$	[.55]	$Proper\text{-}Noun \rightarrow TWA$	[.40]
$VP \rightarrow Verb\ NP$	[.40]	$Proper\text{-}Noun \rightarrow Denver$	[.40]
$VP \rightarrow Verb\ NP\ NP$	[.05]	$Pronoun \rightarrow you$ [.40] | I [.60]	

Probabilistic Context-free Grammar (PCFG) (Cont…)

- How are these probabilities used?
- A PCFG can be used to estimate a number of useful probabilities concerning a sentence and its parse-tree(s).
- For example a PCFG assigns a probability to each parse-tree *T* (i.e. each derivation) of a sentence *S*.
- This attribute is useful in **disambiguation**.

Probabilistic Context-free Grammar (PCFG) (Cont…)

- For example, consider the two parses of the sentence "Can you book TWA flights" (one meaning 'Can you book flights on behalf of TWA', and the other meaning 'Can you book flights run by TWA') shown in Figure



Probabilistic Context-free Grammar (PCFG) (Cont…)

- The probability of a particular parse *T* is defined as the product of the probabilities of all the rules *r* used to expand each node *n* in the parse tree:

$$P(T,S) = \prod_{n \in T} P(r(n))$$

- The resulting probability P(T,S) is both the joint probability of the parse and the sentence, and also the probability of the parse P(T).
- How can this be true?
- First, by definition of joint probability:

$$P(T,S) = P(T)P(T|S)$$

- But since a parse tree includes all the words of the sentence, P(S|T)is 1.
- Thus:

$$P(T,S) = P(T).P(S|T) = P(T)$$

Source: https://www.slideshare.net/shkulathilake/nlpkashkparsing-with-contextfree-grammar

UMBC

Parsing: Shallow Parsing

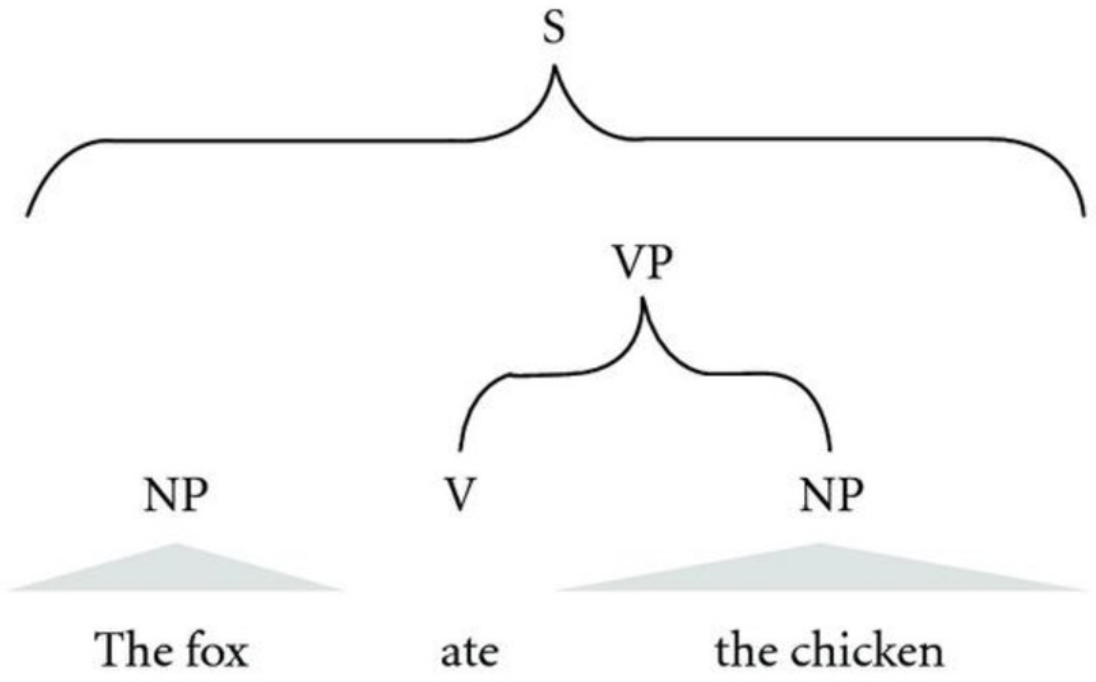
- **Shallow parsing** is known as **light parsing** or **chunking**
- It is a technique designed to **analyze the structure of a sentence** to break it down into its smallest constituents, which may be word tokens, and **group them together** into higher-level phrases
- The purpose of chunking is to obtain semantically meaningful phrases and observe relations among them
- Here is an example of shallow parsing:

```
import nltk
sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),("dog", "NN"), ("barked", "VBD"), ("at", "IN"),  ("the", "DT"),
("cat", "NN")]

grammar = "NP: {<DT>?<JJ>*<NN>}"

cp = nltk.RegexpParser(grammar)
result = cp.parse(sentence)
print(result)
```

Output:
(S (NP the/DT little/JJ yellow/JJ dog/NN) barked/VBD at/IN (NP the/DT cat/NN))

Parsing: Shallow Parsing

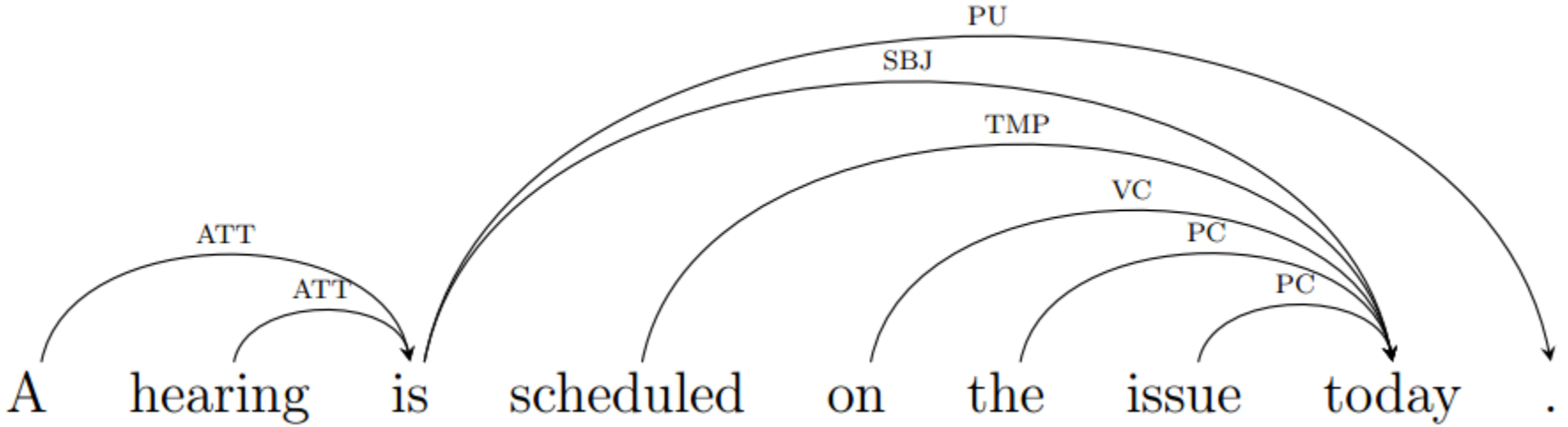
- The focus of **shallow parsing** is to identify phrases and chunks, which results in the tree below:



- **Chinking** is the reverse of chunking: we specify which specific token we do not want to be part of any chunk and then form the necessary chunk to exclude the tokens

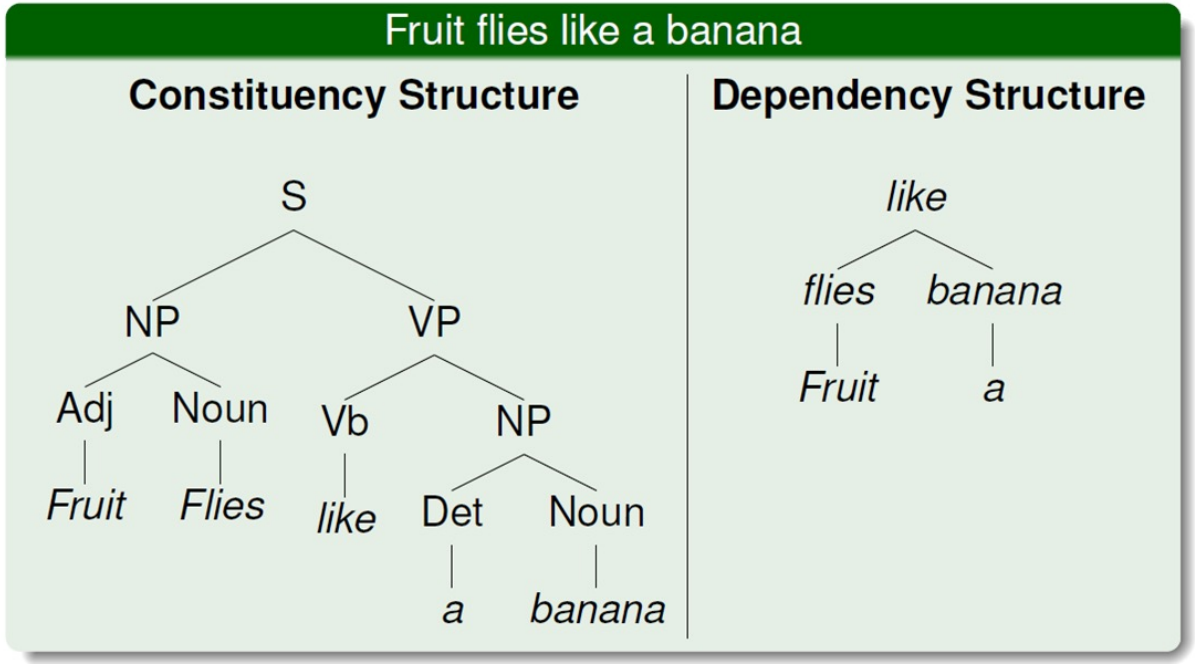
Parsing: Dependency Parsing

- With **dependency parsing**, we try to use dependency-based grammars to analyze and infer both structure and semantic dependencies and relationships between tokens in a sentence
- All words except one in a sentence have relationship or dependency on other words in the same sentence
- The word with no dependency is called the **root** of the sentence
- All the other words are directly or indirectly linked to the root verb using links, which are dependencies
- Below is an example:



Parsing: Constituency Parsing

- **Constituency-based** grammar helps specify how we can break down a sentence into various constituents
- We keep on breaking down a sentence until we reach tokens
- Each word belongs to a lexical category in the case and forms of the head words of different phrases
- Phrases are formed based on rules called phrase structure rules
- Phrase structure rules address syntax and rules that govern the hierarchy and ordering of the various constituents in the sentences
- In **constituency-based** parsing, there are several types of parsing algorithms:
  - Recursive descent parsing
  - Shift reduce parsing
  - Chart parsing
  - Bottom-up parsing
  - Top-down parsing
  - Context-free Grammar parsing



Part-of-Speech Tagging

- The **process of classifying words into their parts of speech** and **labeling them accordingly** is known as **part-of-speech tagging** or **POS-tagging**, or simply **tagging**
- **POS-tagging** refers to the **process of tagging words within sentences into their respective parts of speech** and then finally **labeling them**
- In other words, **POS** is the process of tagging words in a textual input with their appropriate part of speech
- We extract **POS** from tokens composing a sentence, so that we can filter out the **POS** that are of interest and analyze them
- If we look at the sentence "The sky is blue," we get four tokens: "The," "sky," "is," and "blue" using tokenization
- When we use a **POS tagger**, we tag parts of speech to each word/token
- This will look as follows:

  [('The', 'DT'), ('sky', 'NN'), ('is', 'VBZ'), ('blue', 'JJ')]
  DT = determiner
  NN = noun, common, singular or mass
  VBZ = verb, present tense, 3rd person singular
  JJ = Adjective

- A part of speech is a category of words which have similar grammatical properties or usage
- Parts of speech are also known as **word classes** or **lexical categories**
- The part of speech explains how a word is used in a sentence

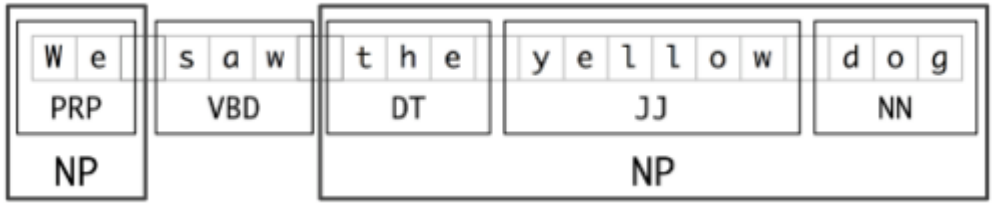
Part-of-Speech Tagging (Cont.)

Commonly Listed Parts of Speech in English

Noun - The name of a person, place, thing, or idea
Verb - The action or being
Adjective - This modifies or describes a noun or a pronoun
Adverb - This modifies or describes a verb, adjective, or another adverb
Pronoun - The word to be used in place of a noun
Preposition - The word placed before a noun or pronoun to form a phrase modifying another word in the sentence
Conjunction - This joins words, phrases, or clauses
Interjection - A word used to express emotion

Part-of-Speech Tagging (Cont.)

- There are eight main **parts of speech**: nouns, pronouns, adjectives, verbs, adverbs, prepositions, conjunctions, and interjections
- The collection of tags used for a specific task is known as a **tagset**
- A part-of-speech tagger, or POS-tagger, processes a sequence of words and attaches a part of speech tag to each word
- **Chunking** is used for entity detection in sentence



- **Chunking** works on top of **POS tagging**, it uses **POS-tags** as input and provides **chunks** as output
- In simple words it is a **generalization of tagging** in which a **contiguous sequence of words** is assigned a **single tag**
- It is also known as shallow parsing. The resulted group of words is called "chunks“
- In **shallow parsing**, there is maximum of one level between roots and leaves, while deep parsing comprises of more than one level
- **Shallow parsing** is also called **light parsing** or **chunking**

Part-of-Speech Tagging (Cont.)

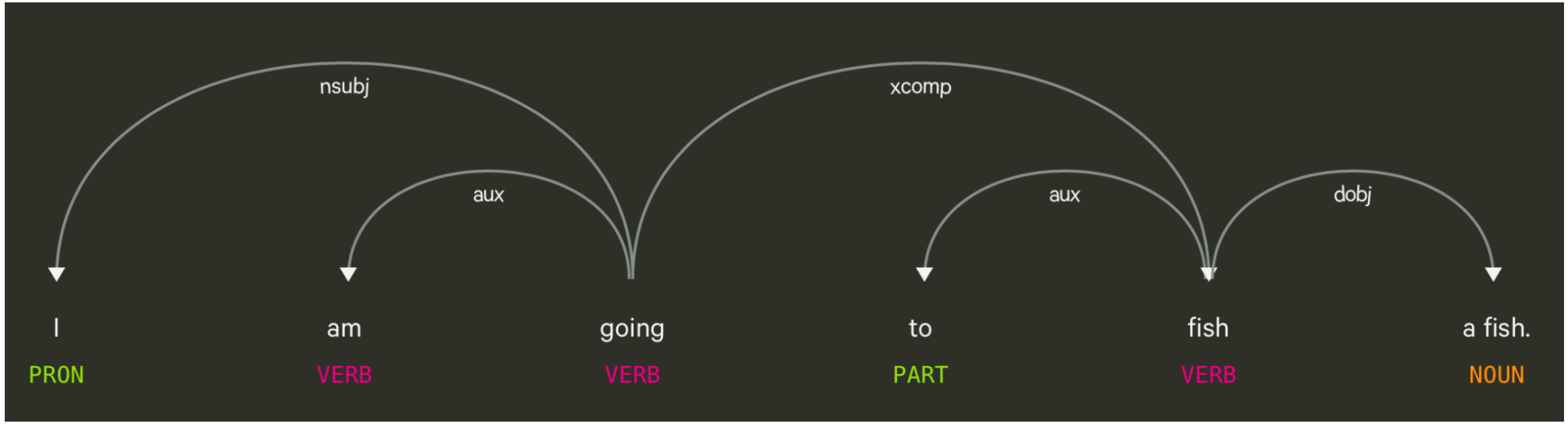
- **POS-tagging** is not always a straightforward task and words have different POS-tags depending on the context
- A simple example is the word **refuse:**
  - If it used as a **verb** it means to decline an offer, and
  - When used as a **noun** it is used to refer to something you throw away or rubbish
- It is important for us to be able to identify which meaning of the word is being referred to, and the POS-tag can help us here
- The context is crucial: it is not possible to tag a word with its part of speech unless it is in a sentence or phrase
- The first few probabilistic models used to train a **POS-tagger** were based on **Hidden Markov Models** to predict the tag
- **Hidden Markov Models** tend to be used whenever there are existing sequences: the context of a word can be used to predict what the POS-tag might be
  - For example, once you've seen an article such as **the**, perhaps the next word is a noun 40% of the time, an adjective 35%, and a number 25%
  - Knowing this, a program can decide that **refuse** in **the refuse** is far more likely to be a noun than a verb, therefore solving the problem discussed so far

Part-of-Speech Tagging (Cont.)

- Apart from statistical models, there are also **rule-based POS-taggers**
- They use **predefined rules** to perform the tagging or learn these rules from the corpus
- In his 1998 paper titled '**A Simple Rule-Based Part of Speech Tagger,'** Eric Brill describes a popular **rule-based POS-taggers**
- There are other more naive methods such as using a regular expressions to evaluate parts of speech or simply storing the most likely tag for a word and tag all future occurrences with the same tag
- **Part of speech tagging** has since moved on to statistical learning or deep learning
- State-of-the-art results have been reached with neural networks on multiple datasets
- ACL web maintains a list of this on their website: https://aclweb.org/aclwiki/POS_Tagging_(State_of_the_art)
- A perceptron used for POS-tagging works by learning the probability of the tag of the word based on various features, or information
  - These can include the tag of the previous word or the last few letters of the word
  - By positively rewarding correct classification and punishing incorrect classification, this model learns weights which it uses to predict the tag of the new word
  - Most supervised machine learning algorithms function on similar principles, and these are the algorithms that perform well in **POS-tagging** tasks

Part-of-Speech Tagging (Cont.)

- **POS-tagging** is used for **Dependency Parsing**
- As the name suggests, dependency parsing is the process of identifying dependencies, or relationships between words in a sentence or phrase
- Identifying the part of speech of each word is an important part of generating such a dependency tree
- If we use the nifty spaCy **displacy** module in the case of '**I am going to fish a fish'**, below is the dependency tree:



Part-of-Speech Tagging (Cont.)

```
# Printing Tagsets in NLTK
import nltk
nltk.download('tagsets’)
print(nltk.help.upenn_tagset())

# Printing Part Of Speech tags
import nltk
# nltk.download('averaged_perceptron_tagger’)
Sentence = nltk.word_tokenize("The grass is always green on the other side")
print(nltk.pos_tag(Sentence))

# Using WordNetLemmatizer()
import nltk
nltk.stem.WordNetLemmatizer().lemmatize('loving’)
nltk.stem.WordNetLemmatizer().lemmatize('loving', 'v’)
```

Part-of-Speech Tagging (Cont.)

- There are two main Python libraries used to carry out POS tagging: **NLTK** and **spaCy**

# With NLTK
import nltk
text = nltk.word_tokenize("And now for something completely different")
nltk.pos_tag(text)
Out: [('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'), ('completely', 'RB'), ('different', 'JJ')]

With spaCy

```
import en_core_web_sm
nlp = en_core_web_sm.load()
import spacy
sent_0 = nlp(u'John and I went to the park.')
sent_1 = nlp(u'If Jeff was asked to take out the garbage, he would refuse.')
sent_2 = nlp(u'Brian was in charge of the refuse treatment center.')
sent_3 = nlp(u'Marie took out her rather suspicious and fishy cat to go fish for fish.')
for token in sent_0:
print(token.text, token.pos_, token.tag_)
```

Out: John PROPN NNP and CCONJ CC I PRON PRP went VERB VBD to ADP IN the DET DT park NOUN NN . PUNCT .

Part-of-Speech Tagging (Cont.)

```
# Chunking and Tree creation

import nltk
sentence = "The grass is always green on the other side"
regex = ('''NP: {<DT>?<JJ>*<NN>} # NP''')
chunkParser = nltk.RegexpParser(regex)
taggedWordList = nltk.pos_tag(nltk.word_tokenize(sentence))
tree = chunkParser.parse(taggedWordList)
tree.draw()
```

Part-of-Speech Tagging (Cont.)

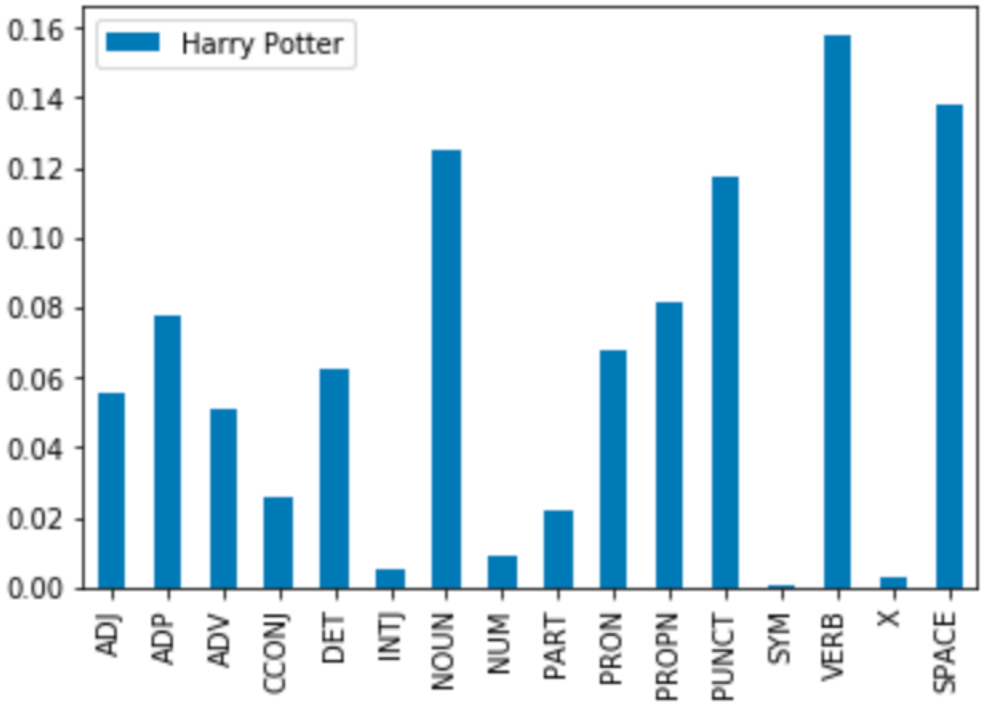
- You can also train your own **POS-tagger**:

```
TRAIN_DATA = [
    ("Facebook has been accused for leaking personal data of users.", {'entities': [(0, 8, 'ORG')]}),
    ("Tinder uses sophisticated algorithms to find the perfect match.", {'entities': [(0, 6, "ORG")]})]

nlp = spacy.blank('en')
optimizer = nlp.begin_training()
for i in range(20):
   random.shuffle(TRAIN_DATA)
   for text, annotations in TRAIN_DATA:
      nlp.update([text], [annotations], sgd=optimizer)
nlp.to_disk('/model')
```

Part-of-Speech Tagging (Cont.)

- To count the occurrences of each kind of **POS**
- This can be accomplished with the following code snippet, where we find out the number of occurrences of these words in the 1st Harry Potter book:

```
import pandas as pd

harry_potter = open("HP1.txt").read()
hp = nlp(harry_potter)
hpSents = list(hp.sents)
hpSentenceLengths = [len(sent) for sent in hpSents]
[sent for sent in hpSents if len(sent) == max(hpSentenceLengths)]
hpPOS = pd.Series(hp.count_by(spacy.attrs.POS))/len(hp)

tagDict = {w.pos: w.pos_ for w in hp}
hpPOS = pd.Series(hp.count_by(spacy.attrs.POS))/len(hp)
df = pd.DataFrame([hpPOS], index=['Harry Potter'])
df.columns = [tagDict[column] for column in df.columns]
df.T.plot(kind='bar')
```



Hidden Markov Models and the Viterbi Algorithm

- It is possible to find POS tags with Markov models
- A **Markov model** assumes that future states depend only on the current state, not on the events that occurred before it
- **Markov models** are used to model **the probabilities of different states** and **the rates of transitioning among them**
- Part-of-speech tagging is a **fully-supervised learning task**, because we have a **corpus of words** labeled with the correct part-of-speech tag. However, many applications do not have labeled data
- A **Markov chain** is useful when we need to compute a **probability for a sequence of observable events**. In many cases, however, the events we are interested in are hidden: We do not observe them directly
- A **hidden Markov model** (HMM) allows us to talk about **both observed events** (like words that we see in the input) and **hidden events** (like **part-of-speech tags**) that consider causal factors in our probabilistic model
- A **first-order hidden Markov model** instantiates two simplifying assumptions:
  - First, as with a first-order Markov chain, the probability of a specific state depends only on the previous state:
    - *Markov Assumption*: $P(q_i \mid q_1 \ldots q_{i-1}) = P(q_i \mid q_{i-1})$
  - Second, the probability of an output observation o_i depends only on the state that produced the observation q_i and not on any other states or any other observations:
    - *Output Independence*: $P(o_i \mid q_1 \ldots q_i, \ldots, q_T, o_1, \ldots, o_i, \ldots, o_T) = P(o_i \mid q_i)$
- The **Viterbi algorithm** is used to calculate the best path to a node and to find the path to each node with the lowest negative log probability

Hidden Markov Models and the Viterbi Algorithm

- Rabiner (1989) introduced the idea that **HMM** should be characterized by three fundamental problems:
  - Likelihood
  - Decoding
  - Learning
- For any model, such as an **HMM**, that contains hidden variables, the task of determining which sequence of variables is the underlying source of some sequence of decoding observations is called the **decoding task**
- This can be summarized as
  - Given as input an **HMM** $\lambda = (A,B)$ and a sequence of observations $O = o_1, o_2, ..., o_T$, find the most probable sequence of states $Q = q_1, q_2, q_3, ..., q_T$
- The **Viterbi algorithm** has three steps: (1) Initialization, (2) induction, and (3) termination and path readout
- We compute two functions $\delta_i(j)$, which gives us the probability of being in state j (= tag j) at word i, and $\psi_{i+i}(j)$, which gives us the most likely state (or tag) at word i given that we are in state j at word i + 1
- The most common **decoding algorithms** for **HMM**s is the **Viterbi algorithm**
- Like the forward algorithm, the **Viterbi** algorithm is a kind of **dynamic programming** that makes uses of a **dynamic programming trellis.** A **trellis** is a diagram representation
- The **Viterbi algorithm** is used to calculate the **best path to a node** and to **find the path to each node with the lowest negative log probability**

UMBC

Hidden Markov Models and the Viterbi Algorithm

Viterbi algorithm

Commonly we want to find the most likely complete path, that is:

$$\arg\max_X P(X|O,\mu)$$

To do this, it is sufficient to maximize for a fixed O:

$$\arg\max_X P(X,O|\mu)$$

VITERBIALGORITHM An efficient trellis algorithm for computing this path is the *Viterbi algorithm*. Define:

$$\delta_j(t) = \max_{X_1 \cdots X_{t-1}} P(X_1 \ldots X_{t-1}, o_1 \cdots o_{t-1}, X_t = j|\mu)$$

This variable stores for each point in the trellis the probability of the most probable path that leads to that node. The corresponding variable $\psi_j(t)$ then records the node of the incoming arc that led to this most probable path. Using dynamic programming, we calculate the most probable path through the whole trellis as follows:

1. Initialization

$$\delta_j(1) = \pi_j, \quad 1 \le j \le N$$

2. Induction

$$\delta_j(t+1) = \max_{1 \le i \le N} \delta_i(t) a_{ij} b_{ijo_t}, \quad 1 \le j \le N$$

Store backtrace

$$\psi_j(t+1) = \arg\max_{1 \le i \le N} \delta_i(t) a_{ij} b_{ijo_t}, \quad 1 \le j \le N$$

3. Termination and path readout (by backtracking). The most likely state sequence is worked out from the right backwards:

$$\hat{X}_{T+1} = \arg\max_{1 \le i \le N} \delta_i(T+1)$$

$$\hat{X}_t = \psi_{\hat{X}_{t+1}}(t+1)$$

$$P(\hat{X}) = \max_{1 \le i \le N} \delta_i(T+1)$$

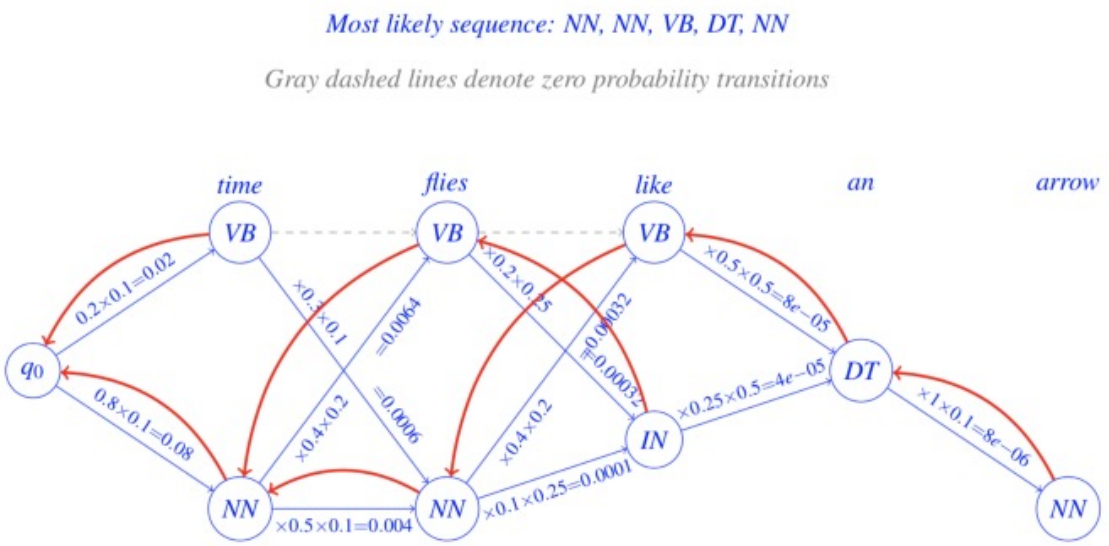
Part-of-Speech Tagging (Cont.)

The **Viterbi algorithm** is a dynamic programming algorithm used to find the most likely sequence of hidden states—called the Viterbi path—that results in a sequence of observed events, especially in the context of hidden Markov models (HMMs)

- *Speech recognition*: Decoding the most likely sequence of words from a speech signal.
- *Natural language processing* (NLP): Part-of-speech tagging, named entity recognition, and other tasks that involve inferring hidden states from observed text.
- *Bioinformatics*: Identifying genes or other features in DNA sequences.
- *Computational linguistics*: Parsing sentences and understanding their structure.
- *Error correction in digital communication*: Correcting errors in transmitted data.
- *Machine translation*: Finding the most likely translation of a sentence given a statistical machine translation model.
- *Gene prediction*: Predicting the location of genes in DNA sequences.
- *Path planning in robotics*: Finding the shortest or most likely path for a robot to take.

Hidden Markov Models and the Viterbi Algorithm

- **Hidden Markov models** (HMMs) are a **way of relating a sequence of observations to a sequence of hidden classes** or **hidden states that explain the observations**
- The **process of discovering the sequence of hidden states**, given the **sequence of observations**, is known as **decoding or inference**
- The **Viterbi algorithm** is commonly used for decoding
- The parameters of an HMM are the **A transition probability matrix** and the **B observation likelihood matrix**. Both can be trained with the **Baum-Welch** or forward-backward algorithm



Viterbi Algorithm

- Similar to computing the forward probabilities, but instead of summing over transitions from incoming states, compute the maximum
- Forward:

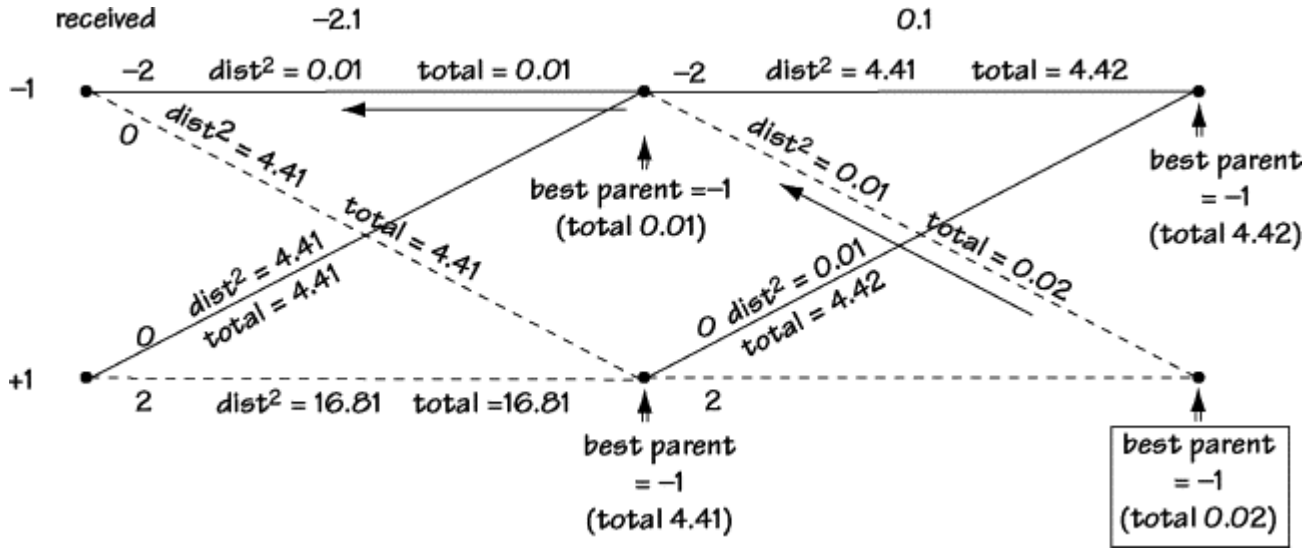
$$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i)\, a_{ij}\right] b_j(o_t)$$

- Viterbi Recursion:

$$\delta_t(j) = \left[\max_{1 \le i \le N} \delta_{t-1}(i)\, a_{ij}\right] b_j(o_t)$$

Hidden Markov Models and the Viterbi Algorithm

- The goal of the **Viterbi algorithm** is to determine the best path through the **trellis**
- For each node, the best parent is found by choosing the parent with the smallest total distance
- When we get to the last nodes, we look for the node with the smallest total and we backtrack through the trellis as marked by the arrows
- These arrows tell us that the output of the trellis should be (-1 1) (based on the branch lines of the best path)



Conditional Random Field (CRF)

- **Conditional Random Fields** is a class of **discriminative models** designed to predict tasks where contextual information or state of the neighbors affect the current prediction
- **Conditional Random Fields,** as **discriminative classifier,** model the decision boundary between the different classes. **Generative models**, on the other hand, model how data were generated, which can be used to make classifications once the model has been trained
- CRFs find their applications in **named-entity recognition** (NER), **part-of-speech tagging** (POS), gene prediction, noise reduction, and object detection problems, among others
- CRF is built upon **Markov Random Fields**. They use contextual information from previous labels, thus increasing the amount of information the model has to make a good prediction
- We model the **conditional distribution** as $\hat{y} = \text{argmax}_y P(y|x)$
- In CRFs, our input data is sequential: we must consider the previous context when making predictions on a data point
- To model this behavior, we will use **Feature Functions**, that will have multiple input values, which are going to be:
  - The set of input vectors, X
  - The position i of the data point we are predicting
  - The label of data point i-1 in X
  - The label of data point i in X

Conditional Random Field

- The feature function is defined as f(X, L, L_{i-1}, L_i)
- The purpose of the feature function is to express some characteristics in the sequence that the data point represents
- For instance, if we are using CRFs for POS tagging, then
  - f (X, i, L{i - 1}, L{i}) = **1** if L{i - 1} is a Noun, and L{i} is a Verb
    - **0** otherwise.
  - Similarly, f (X, i, L{i - 1}, L{i}) = **1** if L{i - 1} is a Verb and L{i} is an Adverb
    - **0** otherwise.
- Each feature function is based on both the label of the previous word and the current word. It is either **0** or **1**
- To build the conditional field, we next assign each feature function a set of weights (lambda values), which the algorithm is going to learn such that

$$P(y, X, \lambda) = \frac{1}{Z(X)} exp\{\sum_{i=1}^{n} \sum_{j} \lambda_j f_i(X, i, y_{i-1}, y_i)\}$$

$$\text{Where: } Z(x) = \sum_{y' \in y} \sum_{i=1}^{n} \sum_{j} \lambda_j f_i(X, i, y'_{i-1}, y'_i)$$

Conditional Random Field

- To estimate the parameters (lambda), we use Maximum Likelihood Estimation
- First, we take the negative log of the distribution to derive the partial derivative such that

$$L(y, X, \lambda) = -log\{\prod_{k=1}^{m} P(y^k|x^k, \lambda)\}$$

$$= -\sum_{k=1}^{m} log[\frac{1}{Z(x_m)} exp\{\sum_{i=1}^{n} \sum_{j} \lambda_j f_j(X^m, i, y_{i-1}^k, y_i^k)]$$

- To apply Maximum Likelihood on the Negative Log function, we will take the *argmin* (because minimizing the negative will yield the maximum). To find the minimum, we can take the partial derivative with respect to lambda such that

$$\frac{\partial L(X, y, \lambda)}{\partial \lambda} = \frac{-1}{m} \sum_{k=1}^{m} F_j(y^k, x^k) + \sum_{k=1}^{m} p(y|x^k, \lambda) F_j(y, x^k)$$

$$\text{Where: } F_j(y, x) = \sum_{i=1}^{n} f_i(X, i, y_{i-1}, y_i)$$

Conditional Random Field

- We use the partial derivative as a step in gradient descent. Gradient descent updates parameter values iteratively, with a small step, until the values converge
- The final gradient descent update equation for CRF is

$$\lambda = \lambda + \alpha[\sum_{k=1}^{m} F_j(y^k, x^k) + \sum_{k=1}^{m} p(y|x^k, \lambda) F_j(y, x^k)]$$

- As a summary, we use **Conditional Random Fields** by
  - First defining the **feature functions** needed,
  - Initializing the weights to **random values**, and then
  - Applying **Gradient Descent** iteratively until the **parameter values** (in this case, lambda) **converge**
- We can see that CRFs are like the **Logistic Regression** because they use the conditional probability distribution
- However, we apply **feature functions** as the **sequential** inputs
- **Hidden Markov Models** are **generative** and provide outputs by modeling the **joint probability distribution**
- **Conditional Random Fields** are **discriminative** and model the **conditional probability distribution**
- **Hidden Markov Models** are a very specific case of **Conditional Random Fields**, with **constant transition probabilities** instead
- **HMMs**, **Naive Bayes**, **Logistic Regression**, and **CRFs** are all related

Conditional Random Field

- We can use **CRF**s to learn how to distinguish which word of a sentence correspond to which POS
- Another application is **Named-Entity recognition**, or extracting proper nouns from sentences
- **Conditional Random Fields** can be used to predict any sequence in which multiple variables depend on each other

Appendix

Chomsky's Classification

- Type 0: Unrestricted Grammar

Format : x → y , where x and y can be Non-Terminals and/or Terminals. There is no restriction at all.

- Type 1: Context Sensitive Grammar

Format : xAy → xzy , where x, y and z can be anything and A is a Non-Terminal in context xAy.

- Type 2: Context Free Grammar

Format : A→ x, where A is Non-Terminal and x can be a sequence of Terminals or Non-Terminals.

- Type 3: Regular Grammar

Format 1 : A→ bt , where A and b are Non-Terminals and t is a Terminal.

or

Format 2 : A→ t , where A is a Non-Terminal and t is a Terminal.

UMBC

5. TEXT SUMMARIZATION

Text Summarization

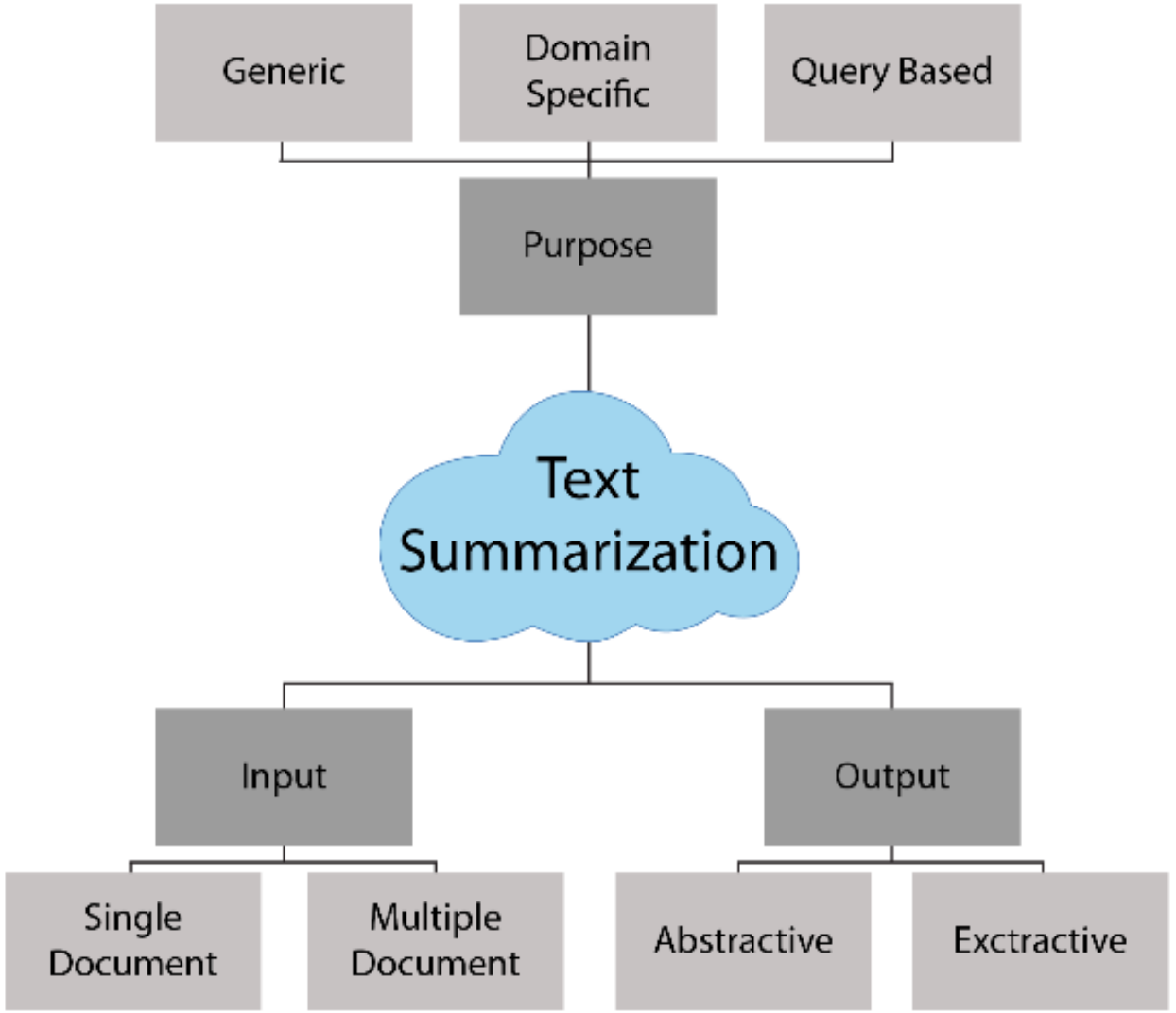
- "**Text summarization** is the process of distilling the most important information from a source (or sources) to produce an abridged version for a particular user (or users) and task (or tasks)," Mani and Maybury (1999)
- **Automated text summarization** is the process of using natural language processing (NLP) tools to produce concise versions of a text that preserve all the key information present in the original content
- Some of these tools such as **sumy** and **NLTK** are frameworks that contain algorithms for text summarization. They also have easy-to-use interfaces
- **Automated text summarization** provides benefits related to:
  - **Sampling**: An automated summary can be a sample of a document or article ahead of the article's actual release. This allows the reader to decide whether to read the full article or not
  - **Searching**: Summaries can assist in searches. They can be shown in search results for users to select
  - **Indexing:** Summaries can be used in search indexes, acting as a compressed representation of the original document. This also helps reduce the amount of storage required
  - **Reading time**: If a summary is well constructed, it can be used instead of the actual document. This can result in a much shorter reading time and is acceptable if the reader gets the gist of the original document
  - **Answering questions:** Chatbots can provide automated summaries as responses to user questions
- Google's *Smart Reply*, which suggests short replies to emails in smartphone, is an application of text summarization

Text Summarization (Cont.)

• **Generic Summarization**: Text summarizers can operate on any text input of sufficient length, and they can perform adequately on sources from many different domains

• **Domain-Specific Summarization**: Text summarization can be specific to a domain, such as finance, shopping, medicine, and travel

• **Query-Based Summarization**: They are centered around providing a response to user input, either in a search bar or from a chatbot



Text Summarization (Cont.)

Extractive Text Summarization

- **Extractive text summarization** examines the original text and then extracts sentences or phrases from it. It best conveys the ideas contained in the text without losing too much of the meaning
- Importance can depend on how much a sentence contributes to the overall meaning of the text document
- After ranking, the summary can be created by selecting the **highest ranked sentence**
- There are different criteria to rank the sentence importance for the extraction such as:
  - **Word frequency**. In this approach, sentences are ranked by how frequently the words in the sentence appear in the document
  - **Sentence similarity**. This measures the similarity of sentences to a given sentence. This approach is used in the TextRank algorithm
  - **Clustering centricity**. The sentences in the document are clustered into a group. Then, the sentences at the center of each cluster are selected for the summary. As a result, selected sentences are central to the meaning of the document

Text Summarization (Cont.)

Abstractive Text Summarization

- This method creates an abstract summary, which is a piece of text that does not use the same words as the original document, but instead just has the same meaning
- **Abstracts** are essentially re-written documents based on the original text, but in a more concise form
- Because it is not easy to create abstracts, **abstractive text summarization** usually resorts to neural network models. Here are some types of abstractive text summarization:
  - **Sequence to Sequence**: The input text is a sequence of input words or characters, while the summary can be considered the output sequence
    - We translate one sequence into another using **Recurrent Neural Networks** to translate from one language to another, or from questions to answers. The aim of the sequence-to-sequence approach is to produce a **smaller output sequence from the input**
- **Encoder/Decoder:** An alternative to sequence-to-sequence translation is an encoder-decoder network:
  - An **encoder** takes input word sequences and produces an internal representation
  - A **decoder** takes the input representation and produces an output sequence of words
- The encoder-decoder architecture is used by Google and other major companies in the NLP field

Text Summarization (Cont.)

Illustration of Extractive v. Abstractive Summary

- Recap. There are two fundamental approaches to text summarization: **extractive** and **abstractive**
  - The former extracts words and word phrases from the original text to create a summary
  - The latter learns an internal language representation to generate more human-like summaries, paraphrasing the intent of the original text

Extractive Summaries

Open title with a straight-sets victory over American Donald Young on Monday, joining Olympic Champion Andy Murray in round two.

An increase in police activity ISIS has been operating for years; now its actions in Iraq are prompting the US to target its fighters with airstrikes there are to threaten more such strikes in Syria.

Abstractive Summaries

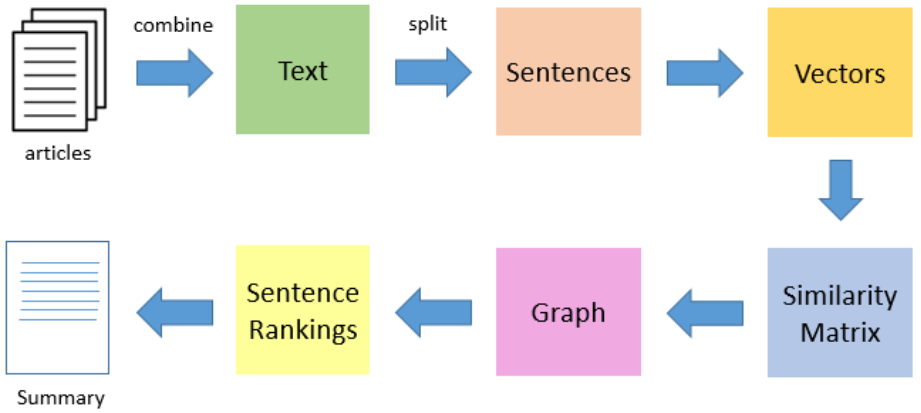
Andy Murray will be growing since the world cup 2010

Airstrikes says it is one of the U.S. military military program

Text Summarization (Cont.)

TextRank

- **TextRank** is used for extractive text summarization
- It is based on **PageRank** as the first search-ordering algorithm used by Google to order search results
- It works on the principle of **ranking pages** based on the **total number of other pages** referring to a given page
  With **TextRank**, the text units (most likely sentences) are ranked by similarity to a given sentence
- The **TextRank** algorithm works as follows:
  1. Reads and extracts text from documents
  2. Splits text into sentences
  3. Converts sentences into vectors
  4. Converts each word in a sentence into a vector
  5. Finds the vector for the entire sentence – one approach would be averaging the word vectors
  6. Calculates a similarity matrix among sentences. This is a matrix that measures how similar sentences are to each other
  7. Creates a graph from the similarity matrix
  8. Ranks the sentences by graph importance
  9. Selects the top sentences by graph importance



Text Summarization: Unsupervised Extractive Algorithms

There are Three Main Categories of Unsupervised Algorithms

1. **Term frequency:** Sentences that resemble document term frequency get a higher score. There are two important algorithms in that category:
   - In **Term Frequency**, frequencies of terms appearing in a sentence are added up to calculate a score. A regularization is done based on sentence length
   - **Sum Basic** works in a similar way. However, when the sentence with the top score is selected, the frequency of all the words appearing in the selected sentence is reduced to induce more diversity in the sentences selected
2. **Latent variable:** The sentence term matrix is factorized to a lower dimensional space. For each independent latent variable in the lower dimensional space, sentences are sorted. Finally, for each latent variable, the top sentence (i.e., the sentence that is most representative of the latent concept) is chosen
   - In **Latent Semantic Indexing (LSI)**, **matrix factorization** relies on **Singular Value Decomposition**
   - **Non-negative matrix factorization(NMF)** is comparable to **LSI**, except that the **matrix factorization technique** is different. The results of **NMF** are easier to interpret. Identified latent concepts are more in line with actual underlying concepts in the document

Text Summarization: Unsupervised Extractive Algorithms (Cont.)

There are Three Main Categories (Cont.)

3. **Graphical:** from the sentence-term matrix, a sentence-to-sentence similarity matrix is created. With the similarity matrix, a Page-Rank-like algorithm called **Text Rank** is run. After the run, each sentence gets a score
   - In **Text Rank**, **sentence-term matrix** is used to compute **cosine similarity between sentences**. The similarity matrix is used to construct a graph where **sentences are nodes**. Then, the **Text Rank** algorithm is run on the graph
   - **Embedding Text Rank** is like **TextRank.** However, instead of using raw sentence term vector, sentence embedding vector calculates a similarity matrix. Sentence embedding vector averages embeddings of all the words in the sentence

All the mentioned algorithms follow these steps:

- Split document into sentences
- Pre-process each sentence including tokenization, stopword removal, and lemmatization
- Assign algorithmic specific score to each sentence
- Sort the sentences by descending order of the score and retain top n, where n is a configurable parameter.
- Sort the sentences from the last step in the same order as they appear in the original document

Extractive summarization techniques may include some of the following methods:

- Get word embedding vectors through **Word2Vec** or **Glove**
- Get sentence embedding vectors by averaging **word embeddings**
- Get clusters using **KNN** based on sentence embeddings
- Find one or two sentences close to each cluster center

Text Summarization: Unsupervised Extractive Algorithms (Cont.)

Diversification

- Diversity is the inverse of similarity
- Ideally, we want the sentences in the **summary** to represent the **underlying concepts** and not have any redundancy
- Multiple sentences should not convey the same information
- In other words, sentences should be as **diverse** as possible
- **Sum Basic**, **Latent Semantic Indexing** and **Non-Negative Matrix Factorization** have redundancy reduction built into the algorithms
- For the remaining algorithms, **explicit diversification** is necessary. A technique called **Maximal Marginal Relevance (MMR)** is used for these other algorithms, which requires diversification as an extra step. The steps are as follows:
  - Get sentences sorted in descending score from the core algorithm
  - To select a sentence, find diversity with respect to already selected sentence. This could be minimum, maximum, or average diversity with already-selected sentences
  - A weighted score is computed based on the original score and the diversity score
  - The sentence with the highest weighted score is selected next

Text Summarization: Unsupervised Extractive Algorithms (Cont.)

Optimization and Tuning

- It is difficult to determine the appropriate algorithm and know how to tune its parameters
- Unlike supervised learning, there are no labeled data
- One solution is to get user feedback as ratings and use **Multi-Armed Bandit** learning to find the optimum solution
- In **Reinforcement Learning, Multi-Armed Bandit** learning refers to the **balance** between **exploration** and **exploitation**
    - It pertains to a problem in which a fixed limited set of resources must be allocated between competing (alternative) choices in a way that maximizes their expected gain, when each choice's properties are only partially known at the time of allocation, and may become better understood as time passes
- Optimization depends on a continuously running learning engine and striking a balance between exploring new methods and capitalizing on the outcomes already generated

UMBC

6. GENERATING TEXT

Generating Text: Markov Chains

Text Generation

- **Text Generation** takes advantage of the statistical properties of the text
- **Text Generation** is a type of Language Modelling problem
  - Language Modelling is the core problem for several NLP tasks such as **speech to text**, **conversational system**, and **text summarization**
- A trained language model learns the **likelihood of occurrence of a word** based on the previous sequence of words used in the text. Language models can be operated at **character level**, **n-gram level**, **sentence level,** or even **paragraph level**

Markov Chains

- A **Markov chain** is a *mathematical system* **for transitioning between states based on probabilities**
- **A Markov Chain** is a **stochastic process** that models a finite **set of states**, with fixed **conditional probabilities of jumping** from a given state to another
- No matter how a system arrives at its present state, **the possible future states are all predetermined**
- The **move to the next state** is based on **a set of probabilities**
- **Markov chains** operate in timesteps:
  - After each timestep, the system selects a **new state**
  - The **new state** depends on the **current state** and the **probabilities of the future states**

Generating Text: Markov Chains (Cont.)

Markov Chains

- In NLP, the **statistical patterns in text sources** are used to create **probability values** for the **state transitions**
- One statistical property is the **probability of a given word** depending on another
- **Markov chains** are not generally **reliable predictors of events in the near term**, since most processes in the real world are more complex than Markov chains allow
- **Markov chains** are, however, used to examine **the long-run behavior of a series of events** that are related to one another by **fixed probabilities**
- To generate a simulation based on a certain text:
  - Count every word used in a text
  - Then, for every word, store the words that are used next. This creates the distribution of words in that text *conditional on* the preceding word

Generating Text: Markov Chains (cont.)

- In order to generate text with Markov Chains, we need to define a few things:
  - What are our states going to be?
  - What probabilities will we assign to transition from one state to another?
- We could design a character-based model for text generation, where we define our state as the last *n* characters we have identified and try to predict the next one

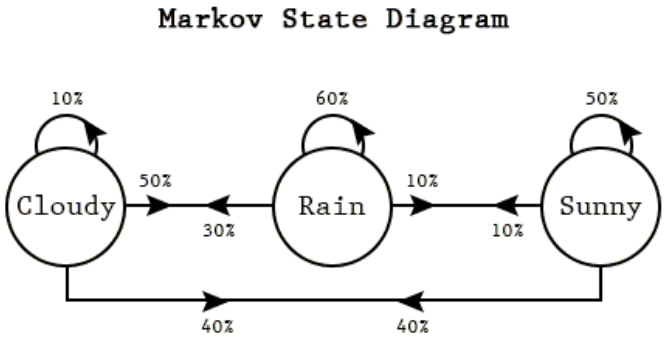


Figure 2

Transition Matrix

	C	R	S
C	0.1	0.5	0.4
R	0.3	0.6	0.1
S	0.4	0.1	0.5

Figure 3

Selected Neural Network Architectures

- **Multilayer Perceptron:** A **multilayer perceptron (MLP)** has three or more layers. It utilizes a nonlinear activation function (mainly **hyperbolic tangent** or **logistic function**) to classify data that is not linearly separable
  - Each node in a layer connects to each node in the following layer making the network fully connected. For example, multilayer perceptrons are used in speech recognition and machine translation
- **Convolutional Neural Network**: A **convolutional neural network (CNN)** contains one or more convolutional, pooling, or fully connected layers and it uses a variation of multilayer perceptrons discussed above
  - Convolutional layers handle a convolution operation to the input and pass the result to the next layer
  - This operation allows the network to get deeper with much fewer parameters
  - Convolutional neural networks show outstanding results in image and speech applications
- **Recursive Neural Network**: A **recursive neural network (RNN)** is a type of deep neural network that applies the same set of weights recursively over a structure to make a structured prediction over variable-size input structures, or a scalar prediction about it, by proceeding through a given structure in topological order
  - In the simplest architecture, nonlinearity such as tanh and a weight matrix shared across the whole network are used to combine nodes into parents

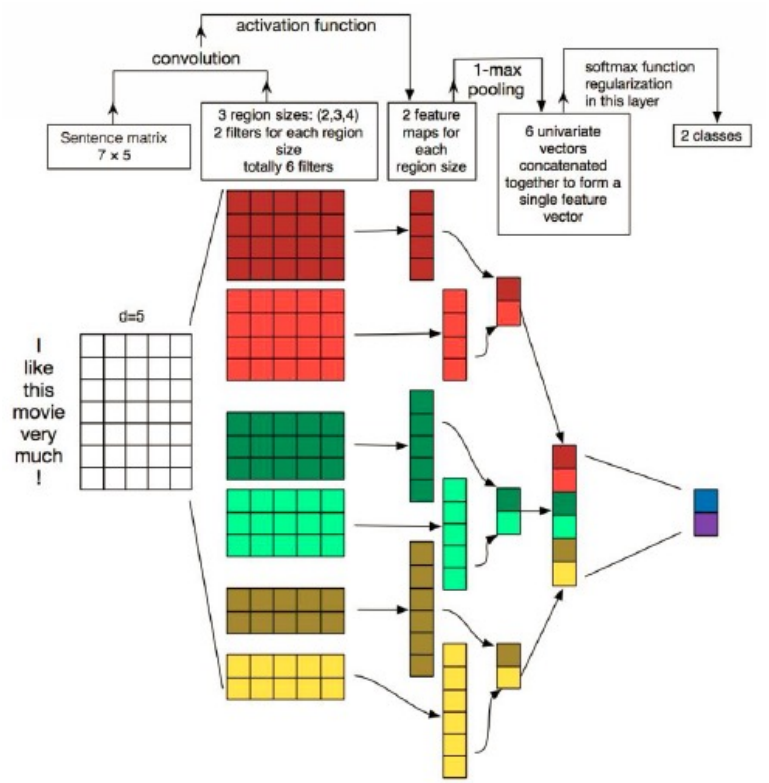
Generating Text: Neural Networks (Cont.)

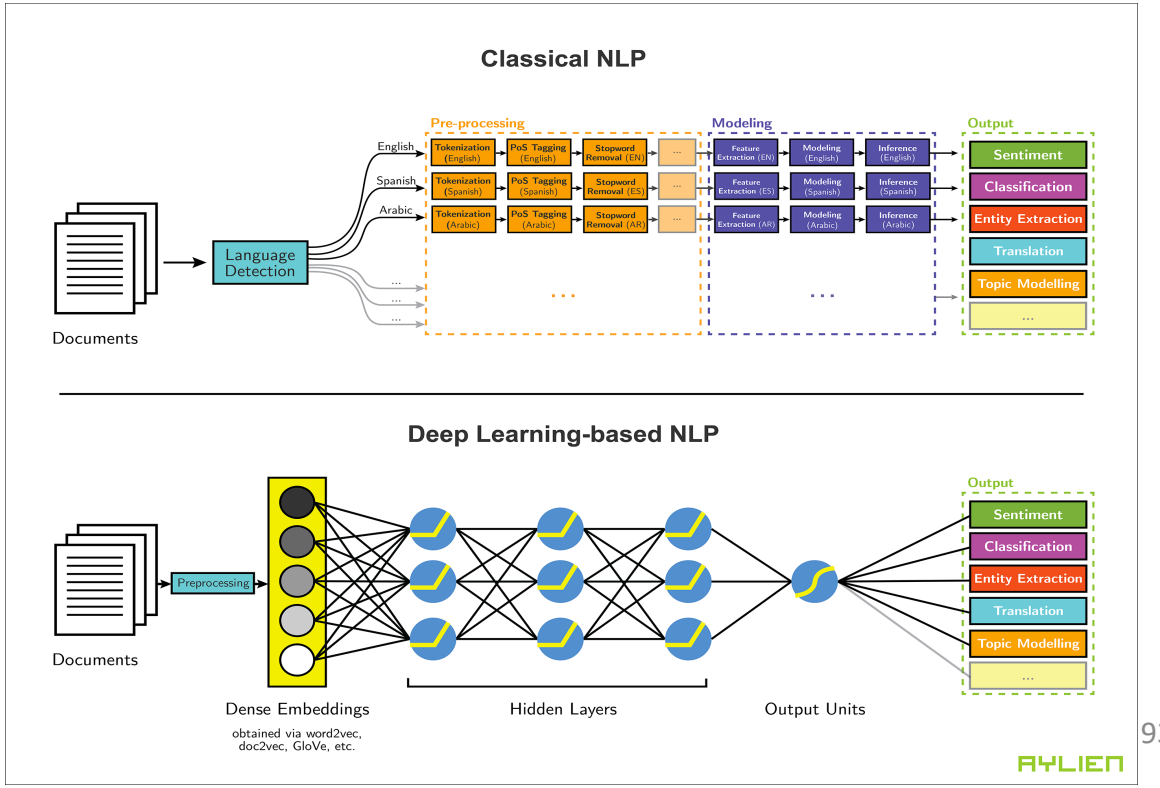
Selected Neural Network Architectures (Cont.)

- **Recurrent Neural Network**: A **recurrent neural network (RNN)**, unlike a feedforward neural network, is a variant of a recursive artificial neural network in which **connections between neurons make a directed cycle**
  - It means that outputs depend, not only on the present inputs, but also on the previous step's neuron state. This memory lets users solve NLP problems like connected handwriting recognition or speech recognition
- **Long Short-Term Memory**: **Long Short-Term Memory (LSTM)** is a specific recurrent neural network (RNN) architecture that was designed to model **temporal sequences** and their long-range dependencies more accurately than conventional RNNs
  - LSTM does not use activation function within its recurrent components; the stored values are not modified; and the gradient does not tend to vanish during training
  - Usually, LSTM units are implemented in "blocks" with several units
  - These blocks have three or four 'gates' (for example, **input modulation gate**, **input gate, forget gate, output gate**) that control information flow based on the logistic function
- **Sequence-to-Sequence Model (Seq2Seq)**: Usually, a sequence-to-sequence model consists of two recurrent neural networks, that is, an **encoder** that processes the input and a **decoder** that produces the output
  - Encoder and decoder can use the same or different sets of parameters
  - Sequence-to-Sequence models are mainly used in question answering systems, chatbots, and machine translation
  - Such multi-layer cells have been successfully used in sequence-to-sequence models for translation

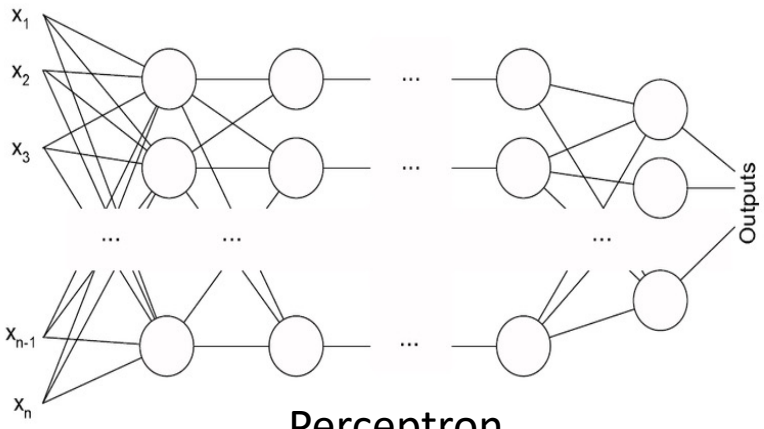
Selected Neural Network Architectures (Cont.)

- **Shallow Neural Networks**: Besides deep neural networks, shallow models are also popular and useful tools
  - For example, **word2vec** is a group of shallow two-layer models that are used to create **word embeddings**
  - **word2vec** takes a large corpus of text as its input and produces a vector space
  - Every word in the corpus is transformed into a corresponding vector in this space
  - As a result, words sharing a common context in the corpus are located close to one another in the vector space

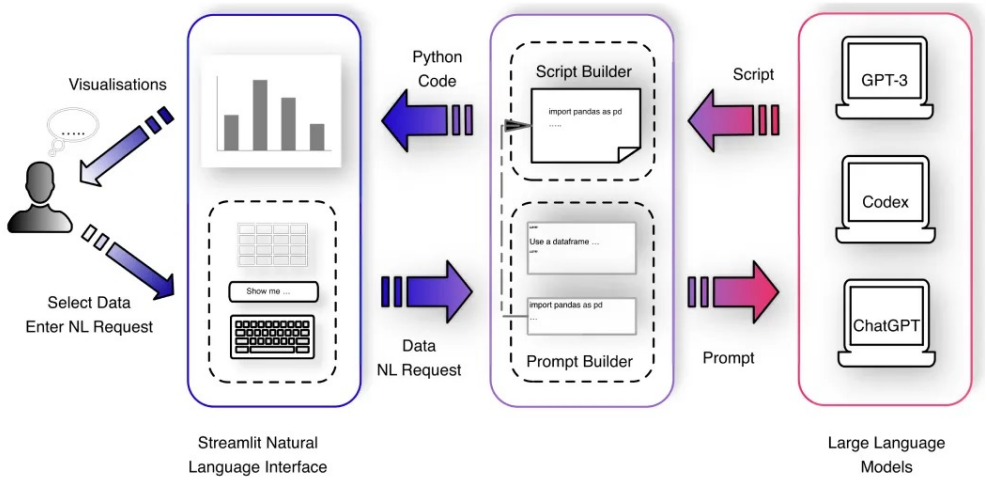




Generating Text with Neural Networks: Selected Architectures

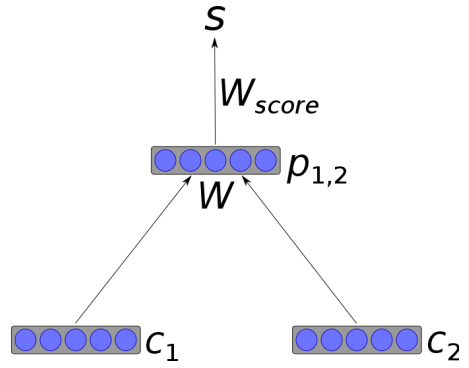


Perceptron

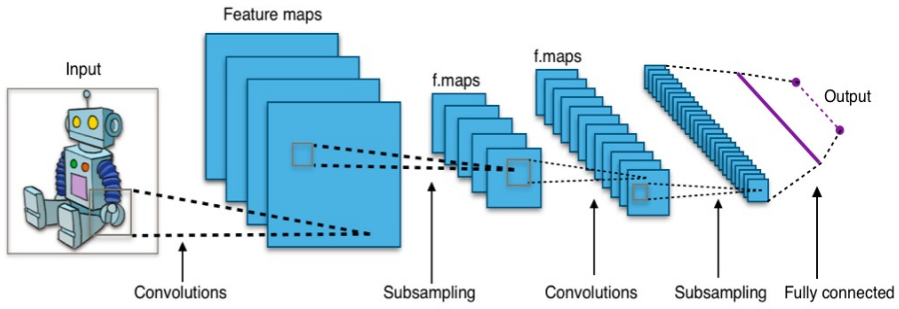


The Chat2VIS architecture for converting free-form query text into visualisations using the large language models GPT-3, Codex, and ChatGPT

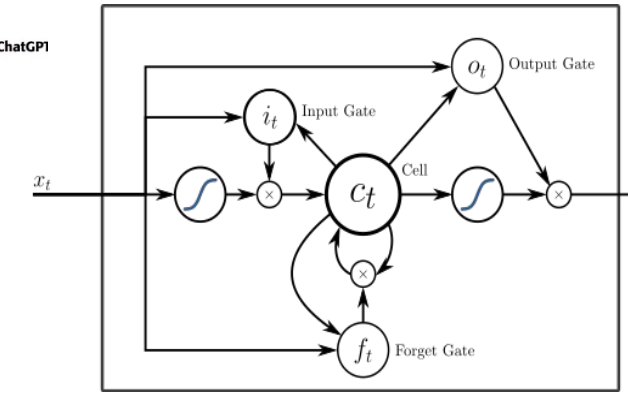
LLM



Recursive NN



CNN



LSTM

Generating Text: Generative AI Models

- **Generative AI** is a subfield of AI and DL that focuses on **generating new content**, such as images, text, music, and video, by using algorithms and models that have been trained on existing data using ML techniques
- **Generative AI** models can be trained on vast amounts of data and then they can generate new examples from scratch using patterns in that data
  - This **generative process** is different from **discriminative models**, which are trained to predict the class or label of a given example
- One of the greatest applications of **generative AI** is its capability to **produce new content** in natural language such as new text, articles, poetry, and product descriptions
- A language model such as **GPT-3**, developed by **OpenAI**, can be trained on large amounts of text data and then used to generate new, coherent, and grammatically correct text in different languages (both in terms of input and output), as well as extracting relevant features from text such as keywords, topics, or full summaries
- A great milestone was achieved in 2017 when the Transformer architecture was introduced by Google researchers in the paper 'Attention Is All You Need.' It was revolutionary in the field of language generation since it allowed for parallel processing while retaining memory about the context of language, outperforming the previous attempts of language models founded on **RNNs** or **Long Short-Term Memory (LSTM)** frameworks
- **Transformers** were the foundations for massive language models called **Bidirectional Encoder Representations from Transformers (BERT)**, introduced by Google in 2018, and they soon become the baseline in NLP experiments
- Transformers are also the foundations of all the **Generative Pre-Trained (GPT)** models introduced by **OpenAI**, including **GPT-3/4**, the models behind **ChatGPT**

Generating Text: Generative AI Models (Cont.)

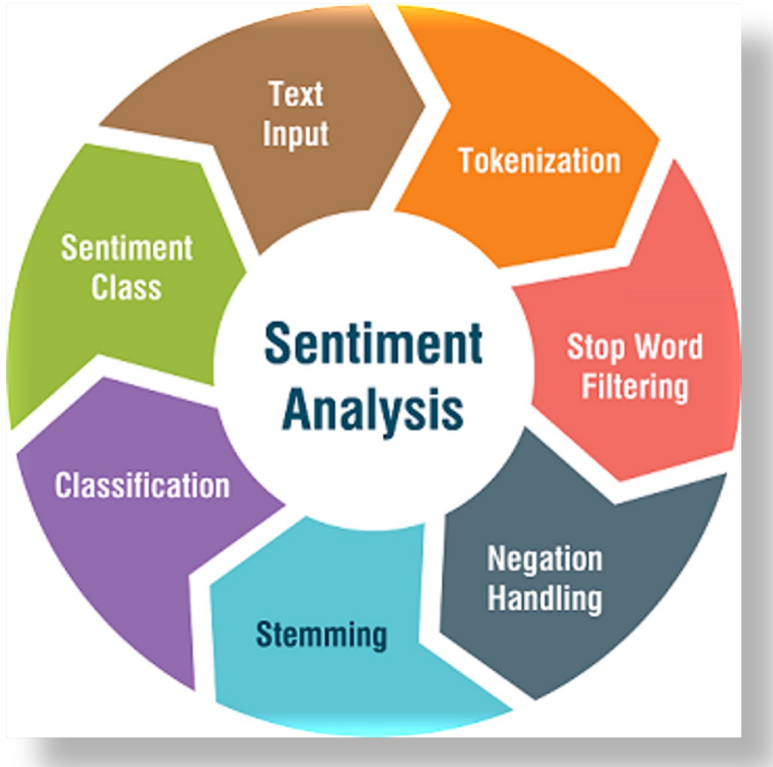
- Although there was a significant amount of research and achievements in those years, it was not until the second half of 2022 that the general attention of the public shifted toward the field of generative AI
- 2022 was the year when powerful AI models and tools became widespread among the general public:
  - Diffusion-based image services (**MidJourney**, **DALL-E 2**, and **Stable Diffusion**),
  - **OpenAI**'s **ChatGPT**,
  - Text-to-video (**Make-a-Video** and **Imagen Video**), and
  - Text-to-3D (**DreamFusion**, **Magic3D**, and **Get3D**) tools were all made available to individual users
- **Prompts** are the only way users can control the output generated by those models
- **OpenAI** models, and hence also **ChatGPT**, come in a pre-trained format. They have been trained on a huge amount of data and have had their (billions of) parameters configured accordingly
- However, **fine-tuning** is the process of adapting a pre-trained model to a new task
- With **zero-shot learning**, the model is asked to perform a task for which it has not seen any training examples. There is also **one-shot** and **few-shot learning**
- Avoid information overload, open-ended questions, and lack of constraints. So, you need to use the concepts of:
  - **Chain of Thought** -> The method divides intricate problems into smaller, manageable steps
  - **Active prompting** -> querying the LLM with a few CoT examples and generating *k* possible answers for a set of training questions
  - **Reason and Act** (**ReAct**): This approach is based on human intelligence's ability to seamlessly combine task-oriented actions with verbal reasoning

UMBC

7. SENTIMENT ANALYSIS

Sentiment Analysis Process

- **Sentiment Analysis** is the **interpretation** and **classification of emotions** (positive, negative and neutral) within text data using **text analytics**
- **Sentiment analysis** requires complex text treatment to
  - Identify **patterns in opinions** and **behaviors** through comments in Facebook posts or tweets, for instance
  - Match **identified patterns** with **sentiment classes (i.e., happiness, anger…)**
  - Derive the **degree of subjectivity** (degree of emotion) and **polarity** or **orientation** (positive, neutral, or negative)



UMBC

Sentiment Analysis (Cont.)

- **Sentiment analysis**, also called **opinion mining**, is defined as

  "The **field of study** that **analyzes** people's opinions, sentiments, evaluations, attitudes, and emotions from **written language**"[1]

- **Sentiment analysis** is applied in almost every type of **business** and **social domain** because
  - **Sentiments** are **central** to almost all **human activities**
  - **Sentiments** influence **people's behaviors**
  - **Sentiments** can be **indicators** of **positive (opportunities) and negative (threats) risks**
- People's **beliefs** and **perceptions of reality**, and the **choices** they make, are largely conditioned on how others see and evaluate the world
- For this reason, when people need to make a decision, they often seek out the opinions of others
- One of the key functions of sentiment analysis is to identify **influencers: those who drive the discussion, influence beliefs** and **perceptions**
- Today, social media generate a large amount of **information** at a great velocity, which is readily available on the Internet for mining and analysis
- Sentiment analysis is a tool to **categorize** sentiments and **opinions** and to provide some **insight** on what is **influencing** behaviors
- Sentiment analysis has mostly replaced '**traditional' research methodologies** such as **surveys, interviews,** and **panels,** which are **more costly, slower to administer,** and subject to **respondents' biases**

[1] Liu, Bing. *Sentiment Analysis and Opinion Mining.* Synthesis Lectures on Human Language Technologies, May 2012, Volume 5, No. 1, pp. 1-167.

Sentiment Analysis (Cont.)

- **Sentiment analysis** can be framed as a direct application of document classification, assuming reliable labels can be obtained
- In the simplest case, **sentiment analysis** is a two or three-class problem, with sentiments of POSITIVE, NEGATIVE, and possibly NEUTRAL
- For example, marketers are interested to know how people respond to advertisements, services, and products (Hu and Liu, 2004); social scientists are interested in how emotions are affected by phenomena such as the weather (Hannak et al., 2012), and how both opinions and emotions spread over social networks (Coviello et al., 2014; Miller et al., 2011). In the field of digital humanities, literary scholars track plot structures through the flow of sentiment across a novel (Jockers, 2015)

Sentiment analysis can often be conducted at three levels:

- **Document Level**. Document Level sentiment analysis classifies the entire document into either positive or negative
- **Sentence Level**. Sentence level classification classifies the sentence into positive, negative or neutral category
- **Entity and Aspect Level**. Entity and Aspect level also known as feature level sentimental analysis gives the summary about which feature of a product does user like or dislike

Different approaches or methods can be used for sentiment classification such as

- **Supervised learning**,
- **Unsupervised learning**, and
- **Semi-supervised learning**

Sentiment Analysis (Cont.)

There are two main approaches to Sentiment Analysis:

- **Lexicon-based**: You count the number of positive and negative words in given text. The larger count will determine the sentiment of the text
  - **Unsupervised sentiment analysis** models use curated databases including detailed information about subjective words, phrases including sentiments, mood, polarity, subjectivity, among others
  - A **lexicon** is a dictionary or vocabulary of words specifically aligned to sentiment analysis
  - The **lexicon** includes words associated with positive and negative sentiments, polarity (magnitude of negative or positive scores), POS tags, subjectivity classifiers (strong, weak, neutral), mood, and modality, among others
  - The sentiment of a text is computed by comparing the content with the presence of specific words from the lexicon and then looking at other factors such as presence of negation parameters, surrounding words, overall context, phrases, and aggregate sentiment score
  - **Lexicon-based sentiment analysis** avoids machine learning altogether, and classifies documents by counting words against positive and negative sentiment word lists (Taboada et al., 2011)
  - **Lexicon-based classification** is less effective for short documents, such as single-sentence reviews or social media posts. In these documents, linguistic issues like **negation** and **irrealis** (Polanyi and Zaenen, 2006)—events that are hypothetical or otherwise non-factual— can make bag-of-words classification ineffective
  - There are several popular **lexicons**: Bing Liu's lexicon, Multi-Purpose Question Answering (MPQA) subjectivity lexicon, Pattern lexicon, TextBlob lexicon, AFINN lexicon, SentiWordNet lexicon, and VADER (Valence Aware Dictionary and sEntiment Reasoner) lexicon

Sentiment Analysis (Cont.)

Other Sentiment Analysis Approaches

- **Machine-learning-based approach**: This approach consists in developing a classification model, which is trained using the pre-labeled dataset of ‘positive’, ‘negative’, and ‘neutral’
  - The role of machine learning, AI in Natural Language Processing (NLP) and text analytics is to improve, accelerate, and automate the underlying text analytics functions and NLP features that turn an unstructured text into useable data and insights
  - We will take a closer look at machine learning approaches when we deal with text classification
- The key libraries to conduct sentiment analysis are
  - TextBlob
  - NLTK’s VADER sentiment
  - Hugging Face libraries

Concepts Related to Sentiment Analysis

- **Subjectivity.** Closely related to sentiment analysis is **subjectivity detection**, which requires identifying the parts of a text that express subjective opinions, as well as other non-factual content such as speculation and hypotheticals (Riloff and Wiebe, 2003)
  - This can be achieved by treating each sentence as a separate document, and then applying a bag-of-words classifier
- **Stance classification.** In debates, each participant takes a side. For example, advocating for or against proposals like adopting a vegetarian lifestyle or mandating free college education
  - The problem of **stance classification** is to identify the author's position from the text of the argument. In some cases, there is training data available for each position, so that standard document classification techniques can be employed. In other cases, it suffices to classify each document as to whether it is in support or opposition of the argument advanced by a previous document (Anand et al., 2011)
  - In the most challenging cases, there are no labeled data for any of the stances, so the only possibility is group documents that advocate the same position (Somasundaran and Wiebe, 2009). This is a form of **unsupervised learning**
- **Targeted sentiment analysis.** The expression of sentiment is often more nuanced than a simple binary label. Consider the following examples:
    1. The vodka was good, but the meat was rotten
    2. Go to Heaven for the climate, Hell for the company –Mark Twain
  - These statements display a mixed overall sentiment: positive towards some entities (e.g., the vodka), negative towards others (e.g., the meat)

Sentiment Analysis (Cont.)

- **Other Concepts Related to Sentiment Analysis**

- **Targeted sentiment analysis** seeks to identify the writer's sentiment towards specific entities (Jiang et al., 2011). This requires identifying the entities in the text and linking them to specific sentiment words — much more than we can do with the classification-based approaches discussed thus far
- **Aspect-based opinion mining** seeks to identify the sentiment of the author of a review towards predefined aspects such as 'Price' and 'Service', or, in the case of bullet b in the previous page , 'Climate' and 'Company' (Hu and Liu, 2004)
- **Emotion classification** While sentiment analysis is framed in terms of positive and negative categories, psychologists generally regard **emotion** as more multifaceted. For example, Ekman (1992) argues that there are six basic emotions—happiness, surprise, fear, sadness, anger, and contempt — and that they are universal across human cultures

Alternatives to Sentiment Analysis

- **Regression.** A more challenging version of sentiment analysis is to determine not just the **class** of a document, but its **rating** on a numerical scale (Pang and Lee, 2005)
  - If the scale is continuous, it is most natural to apply **regression**, identifying a set of weights that minimize the squared error of a predictor $\hat{y} = \theta x + b$, where b is an offset. This approach is called **linear regression**, and sometimes **least squares**, because the regression coefficients are determined by minimizing the squared error, $(y - \hat{y})^2$
  - If the weights are regularized using a penalty $\lambda||\theta||_2^2$, then it is **ridge regression**. Unlike logistic regression, both linear regression and ridge regression can be solved in closed form as a system of linear equations
- **Ordinal ranking.** In many problems, the labels are ordered but discrete: for example, product reviews are often integers on a scale of 1 to 5, and grades are on a scale of A to F. Such problems can be solved by discretizing the score θx into "ranks"

Sentiment Analysis (Cont.)

Analyzing Causality in Sentiment Analysis

- It is important to determine the root cause(s) of positive and negative sentiment in a text
- There are two considerations:
  - We try to understand, interpret, and explain the mechanics behind the predictions made by the classification model
  - We apply topic modeling and extract key topics from positive and negative sentiment comments
- It is important to understand why 'positive' was accurately predicted as having a positive sentiment and vice versa
- Besides using prediction probability, we can use the 'skater' framework to interpret the results
- Preethi, Uma, and Kumar (2015) introduced a generalized prediction model based on temporal sentiment analysis of tweets to identify the causal relation between the events which can be used to predict the event sentiment and duration between the event
  - They evaluated the performance of the classifier with measures such as **precision** and **recall**
  - The accuracy of causal rule prediction was assessed using parameters such as **Mean Absolute Error (MAE)** and the **Root Mean Squared error (RMSE)**

UMBC

Sentiment Analysis Example: Step by Step
IMDB Dataset

```
import numpy as np
import pandas as pd
df= pd.read_csv('/content/imdb-reviews.csv')
```

```
# sample positive movie review
df[df['label']==1].sample(n=1)['review'].iloc[0]
```

```
"I grew up watching, and loving this cartoon every year. I didn't think they would be able to take a half hour (20 min!) cartoo
n and make it a movie. They did it. With FLYING COLOURS! Fabulous, funny, heart warming, effective movie!"
```

Sentiment Analysis (Cont.)

```
import re
# import nltk
# nltk.download('punkt') # At first you have to download these nltk packages.
# nltk.download('stopwords')
# nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

stop_words = stopwords.words('english') # defining stop_words
stop_words.remove('not') # removing not from the stop_words list as it contains value in negative movies
lemmatizer = WordNetLemmatizer()

def data_preprocessing(review):

  # data cleaning
  review = re.sub(re.compile('<.*?>'), '', review) #removing html tags
  review =  re.sub('[^A-Za-z0-9]+', ' ', review) #taking only words

  # lowercase
  review = review.lower()

  # tokenization
  tokens = nltk.word_tokenize(review) # converts review to tokens

  # stop_words removal
  review = [word for word in tokens if word not in stop_words] #removing stop words

  # lemmatization
  review = [lemmatizer.lemmatize(word) for word in review]

  # join words in preprocessed review
  review = ' '.join(review)

  return review
```

UMBC

Sentiment Analysis (Cont.)

```
df['preprocessed_review'] = df['review'].apply(lambda review: data_preprocessing(review))
df.head()
```

	review	label	preprocessed_review
0	Bromwell High is a cartoon comedy. It ran at t...	1	bromwell high cartoon comedy ran time program ...
1	Homelessness (or Houselessness as George Carli...	1	homelessness houselessness george carlin state...
2	Brilliant over-acting by Lesley Ann Warren. Be...	1	brilliant acting lesley ann warren best dramat...
3	This is easily the most underrated film inn th...	1	easily underrated film inn brook cannon sure f...
4	This is not the typical Mel Brooks film. It wa...	1	not typical mel brook film much le slapstick m...

Sentiment Analysis (Cont.)

```
from sklearn.model_selection import train_test_split

data = df.copy()
y = data['label'].values
data.drop(['label'], axis=1, inplace=True)

X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.3, stratify=y)

print("Train data:",  X_train.shape, y_train.shape)
print("Test data:",  X_test.shape, y_test.shape)
```

```
Train data: (35000, 2) (35000,)
Test data: (15000, 2) (15000,)
```

```
from sklearn.feature_extraction.text import CountVectorizer

vect = CountVectorizer(min_df=10)

X_train_review_bow = vect.fit_transform(X_train['preprocessed_review'])
X_test_review_bow = vect.transform(X_test['preprocessed_review'])

print('X_train_review_bow shape: ', X_train_review_bow.shape)
print('X_test_review_bow shape: ', X_test_review_bow.shape)
```

```
X_train_review_bow shape:  (35000, 19488)
X_test_review_bow shape:  (15000, 19488)
```

UMBC

Sentiment Analysis (Cont.)

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(min_df=10)

X_train_review_tfidf = vectorizer.fit_transform(X_train['preprocessed_review'])
X_test_review_tfidf = vectorizer.transform(X_test['preprocessed_review'])

print('X_train_review_tfidf shape: ', X_train_review_tfidf.shape)
print('X_test_review_tfidf shape: ', X_test_review_tfidf.shape)
```

```
X_train_review_tfidf shape:  (35000, 19488)
X_test_review_tfidf shape:  (15000, 19488)
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

clf = MultinomialNB()
clf.fit(X_train_review_bow, y_train)

y_pred = clf.predict(X_test_review_bow) #prediction from model
print('Test Accuracy: ', accuracy_score(y_test, y_pred))
```

```
Test Accuracy:  0.8468
```

Sentiment Analysis (Cont.)

```
clf = MultinomialNB(alpha=1)
clf.fit(X_train_review_tfidf, y_train)

y_pred = clf.predict(X_test_review_tfidf)
print('Test Accuracy: ', accuracy_score(y_test, y_pred))
```

```
Test Accuracy:  0.8538
```

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(penalty='l1')
clf.fit(X_train_review_tfidf, y_train)

y_pred = clf.predict(X_test_review_tfidf)
print('Test Accuracy: ', accuracy_score(y_test, y_pred))
```

```
Test Accuracy:  0.8807333333333334
```

Sentiment Analysis (Cont.)

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5, 3))
sns.heatmap(cm, annot=True, fmt='d')
plt.title('Test Confusion Matrix')
plt.show()
```

- Out of 7,500 negative reviews, 6,515 were correctly classified as negative and 985 were incorrectly classified as positive
- Out of 7,500 positive reviews, 6,696 were correctly classified as positive and 804 were incorrectly classified as negative

UMBC

8. FEATURE ENGINEERING, WORD EMBEDDING, AND CONCEPT OF SIMILARITY

Bag of Words (BOW) and Continuous Bag of Words (CBOW)

- We have a set of texts and their respective labels. However, we cannot readily use these texts in our model
- We need to convert words into digits
- The **Bag-of-Words** model (BOW) makes it easier to extract features from text
- **BOW** converts text into a **matrix of words** that are part of a document
- **CBOW** is the **Continuous BOW**, a model architecture that tries to predict the current target word (the center word) based on the source context word (surrounding words). The model tries to predict the target_word based on the context_window_word
- The **BOW** model determines whether given words occurred or not in the document and presents it in a matrix called **Document-Term-Matrix (DTM)**
- The **DTM matrix** can use a single word, or a combination of two or more words, which is called respectively a **bigram** or **n-gram** model
- The **DTM matrix** can be generated with scikit-learn's **CountVectorizer**

Feature Engineering for Text Representation (Cont.)

- **Word2Vec** was created in 2013 by Google. It is a predictive **Deep Learning** model to create dense vector representations of words that capture **contextual** and **semantic** similarity
- They are **unsupervised models** that can take in massive textual corpora, create a vocabulary of possible words, and generate dense word embeddings for each word in the vector space representing that vocabulary
- There are two different model architecture that can be leveraged by **Word2Vec**

  - **CBOW** is a model architecture designed to predict the center word based on the surrounding words
    - The **CBOW** model tries to predict the target word based on the context window word
  - The **Skip-Gram model** tries to predict the source context words (surrounding words) given a target word (the center word). The model tries to predict the context window words based on the target word
    - The **Skip-Gram** architecture is modeled as a deep learning classification model using the target word as the input and context words as the outputs
    - We feed pairs of (X,Y) with X as the input and Y as the label. We use [(target, context), 1] pairs as positive input samples
    - **Target** refers to a word of interest and **context** is a context word occurring near the target word. The positive label 1 indicates it is a contextually relevant pair
    - We also feed [(target, random, 0] pairs as negative input samples. In this case, **random** means the word has no context or association with the target word. A zero label means there is no contextual relevant pair

Feature Engineering for Text Representation (Cont.)

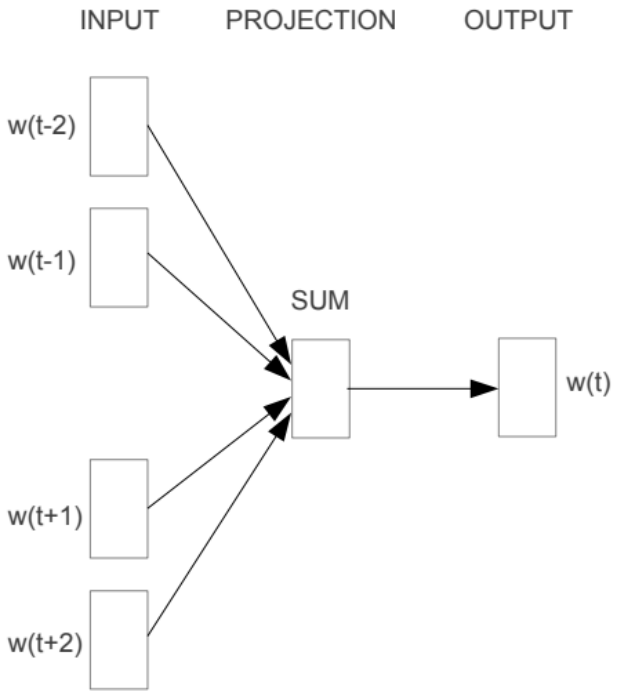
- **GloVe** is an unsupervised learning model that can be used to obtain dense word vectors like **Word2Vec**
- The first task of **GloVe** is to create a word-context co-occurrence matrix. It includes (word, context) pairs such that each element in the matrix represents how often a word occurs with the context (which can be a sequence of words)
- The idea is to apply matrix factorization to approximate the matrix. Therefore,

Word Context (WC) = **Word Feature (WF)** X **Feature Context (FC)**

- We reconstruct WC from WF and FC by multiplying them. WC and FC are initialized using random weights. The multiplication of WF by FC provides an approximation of WC. We repeat the process using Stochastic Gradient Descent to minimize errors
- WF provides the **word embeddings** for each word where F is a specific number of dimensions
- **Word2Vec** starts local individual examples of word co-occurrence pairs
- **GloVe** starts with global aggregated co-occurrence statistics across all words in the corpus
- **GloVe** is quite similar to **Word2Vec.** However, unlike **Word2Vec**, **GloVe** takes advantage of the global co-occurrences of words rather than just the local context, which makes it more a powerful algorithm

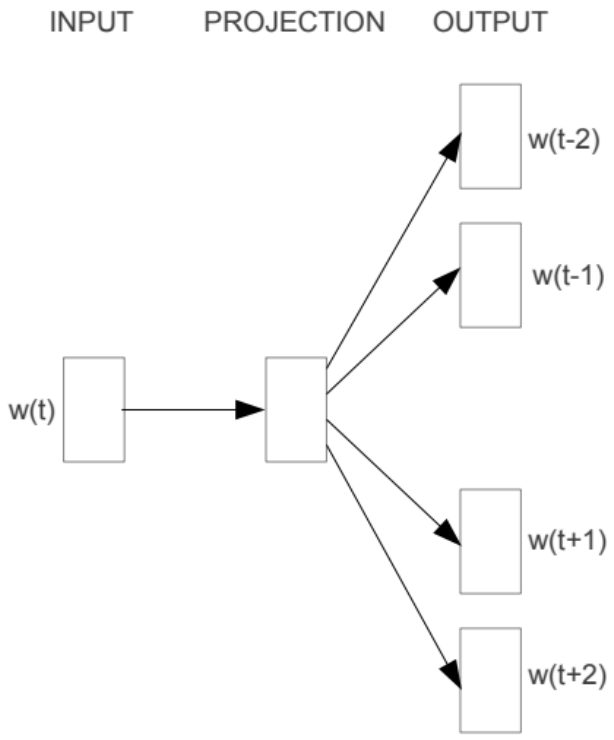
UMBC

Feature Engineering for Text Representation (Cont.)



CBOW

The **CBOW** model architecture tries to predict the current target word (the *center word*) based on the source context words (*surrounding words*)



Skip-gram

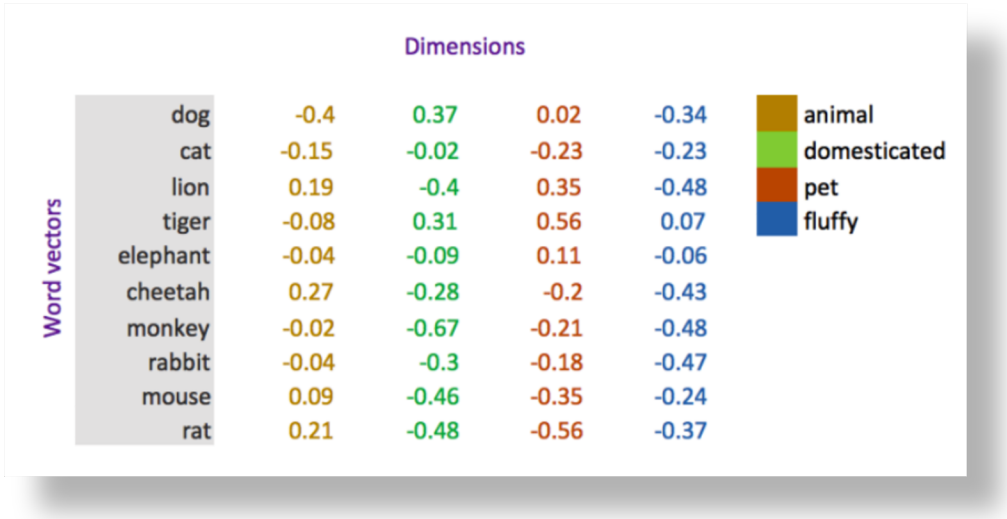
The **Skip-gram** model architecture tries to achieve the reverse of what the CBOW model does. It tries to predict the source context words (*surrounding words*) given a target word (the *center word*)

Feature Engineering for Text Representation (Cont.)

- **FastText** is a framework for learning word representations and performing robust, fast, and accurate text classifications
- It was developed by Facebook in 2016
- In general, predictive models such as **Word2Vec** consider each word a distinct entity and generate a dense embedding for the word
- The issue with **Word2Vec** is that it ignores the morphological structure of each word and considers a word a single entity
- The **FastText** model considers each word a **Bag of Character n-grams**, called a **sub word** model by Mikolov et al. who created the model
- Special boundary symbols are added at the beginning and end of words, which makes it possible to identify suffixes and prefixes
- The letter w is included in the sets n-grams to learn a representation of each word. For instance, if n = 3 and the word 'where', w includes 'wh', 'whe', 'her', 'ere', and 're' and the special sequence 'where'. The word 'her' is different from the tri-gram 'her' and the word 'where'
- Mikolov et al. recommended extracting all the n-grams for n >= 3 and n <= 6
- A word can be represented as the **sum of the vector representations of its n-grams** or the **average of the embedding of these n-grams**
- There is a better chance with this method for rare words to get a better representation since their character-based n-grams should occur across other words of the corpus

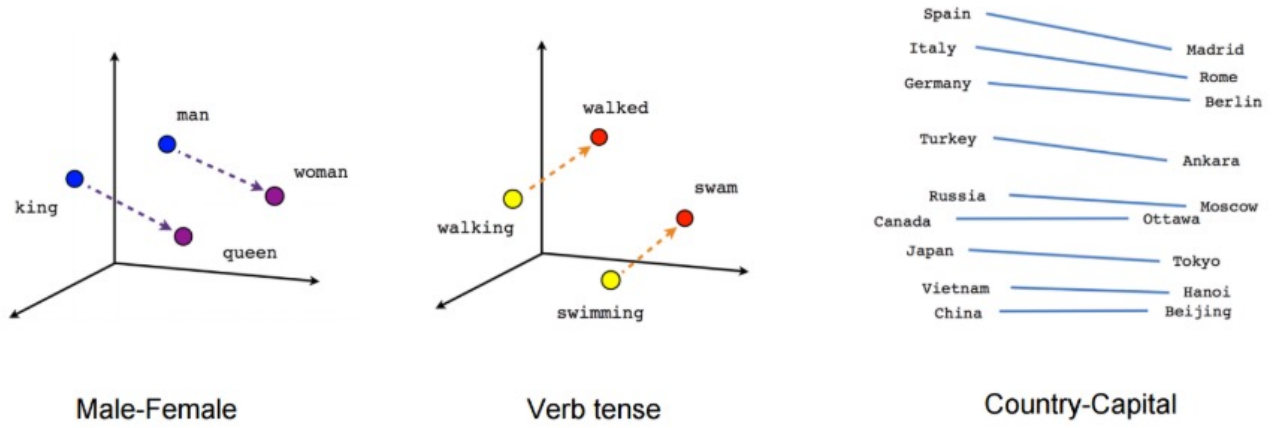
Feature Engineering: Word Embedding

- **Word embeddings** are representations of *words as vectors*, learned by exploiting vast amounts of text
- Each word is mapped to one vector and the vector values are learned in a way that resembles an artificial neural network
- Each word is represented by a real-valued vector with often tens or hundreds of dimensions
- A **word vector is a row of real valued numbers** where each number is **a dimension of the word's meaning** and **where semantically similar words have similar vector,** i.e., Queen and Princess would be closer vectors
- The numbers in the word vector represent the **word's distributed weight across dimensions**. The **semantics** of the word are embedded across these **dimensions** of the vector. Below is a simplified example across 4 dimensions:



Feature Engineering: Word Embedding (Cont.)

- The hypothetical vector values represent the abstract 'meaning' of a word
- Vectors make mathematical operators possible
- Vectors can be used as inputs to an artificial neural network



Male-Female | Verb tense | Country-Capital

Source: Google

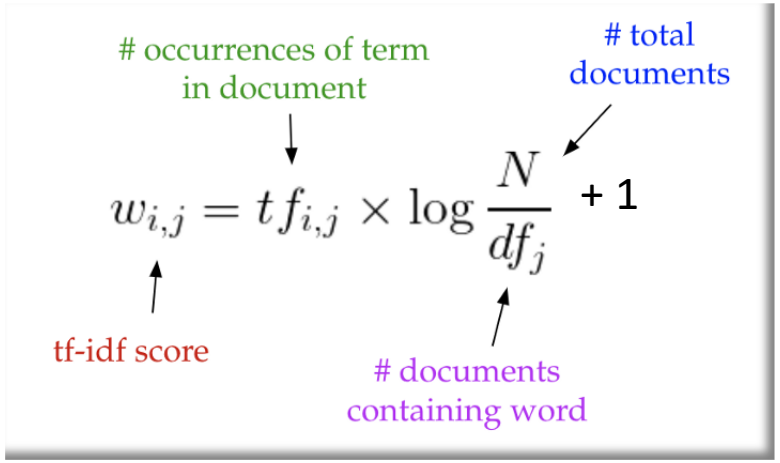
- The **word embedding algorithm** takes as its input a large corpus of text and produces these vector spaces, typically of several hundred dimensions
- A neural language model is **trained on a large corpus (body of text)**
- The output of the network is used to have each unique word assigned to a corresponding vector
- The most popular word embedding algorithms are Google 's **Word2Vec,** Stanford 's **GloVe** or Facebook 's **FastText**

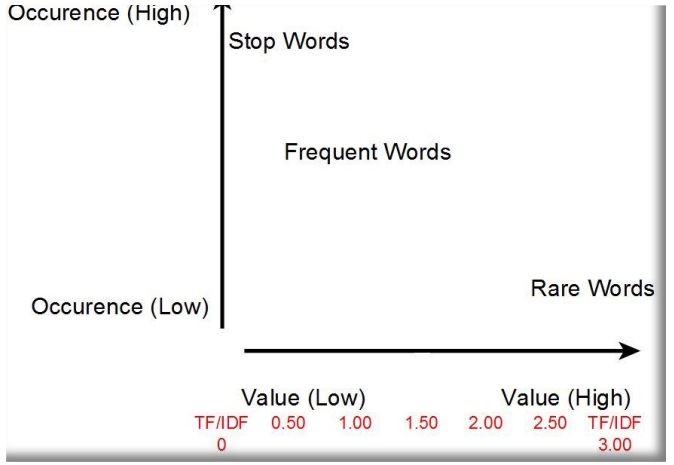
Feature Engineering: Term Frequency and Inverse Document Frequency

Key Metrics

- With **Term Frequency** (**TF**), you **count the number of words in each document**
  - **Term Frequency** provides **more weight to longer documents**
  - **Term frequency** is basically **the output of the BOW model**
- **Inverse Document Frequency** (**IDF**) measures the **importance of a given word in a document**
  - **IDF** is the **logarithmically-scaled inverse ratio of the number of documents** that contain the word to the total number of documents such that **IDF = log (1 + n/1 + df(d, t)) + 1** with n as number of documents and df(d,t) as the document frequency of the term t. Therefore, **IDF = 1/DF**
- According to **Zipf's law**, ”For a given corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table“
  - If t_1 is the most common term in the collection, t_2 is the next most common, and so on, then the collection frequency cf_i of the ith most common term is proportional to 1/i: $cf_i \propto 1/i$. So, if the most frequent term occurs cf1 times, then the second most frequent term has half as many occurrences, the third most frequent term a third as many occurrences, and so on

```
def computeTFIDF(TF_scores, IDF_scores):
    TFIDF_scores = []
    for j in IDF_scores:
        for i in TF_scores:
            if j['key'] == i['key'] and j['doc_id'] == i['doc_id']:
                temp = {'doc_id' : j['doc_id'],
                        'TFIDF_score' : j['IDF_score']*i['TF_score'],
                       'key' : i['key']}
        TFIDF_scores.append(temp)
    return TFIDF_scores
```





Feature Engineering: Term Frequency and Inverse Document Frequency (Cont.)

- **TF-IDF** is a rough way of **approximating how users value the relevance of a text match**
- **TF-IDF** measures the **relative concentration of a term in a specific text**

Advantages:

- **TF-IDF** is easy to compute
- Some basic metrics allow to extract the most descriptive terms in a document
- It is easy to compute the similarity between two documents

Disadvantages:

- **TF-IDF** is based on the **Bag-of-Words (BOW)** model. Therefore, it **does not capture position in text**, **semantics**, **co-occurrences** in different documents, etc.
- For this reason, **TF-IDF** is only useful as a **lexical level feature**
- **TF-IDF** cannot capture **semantics** (as compared to **topic models**, or **word embeddings**)

- **BM25** (Best Match) improves on **TF-IDF.** Term frequency in **BM25** dampens the impact of term frequency even further than traditional **TF-IDF.** The impact of term frequency is always increasing, but asymptotically approaches a value **((k + 1) * TF) / (k + TF)** where k is usually 1.2)
- It is also possible to consider the length of a document through the formula **((k + 1) * TF) / (k * (1.0 - b + b * L) + TF)** with L as the length and b as constant (usually .75)
- So, **Okapi BM25 = IDF * ((k + 1) * TF) / (k * (1.0 - b + b * (|d|/avgDl)) + TF)** with document length as **|d|/avgDl**

Feature Engineering: Computing Similarity

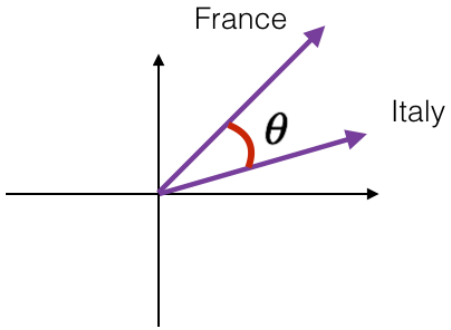
- **Feature engineering** is a method for **extracting new features from existing features**
- These new features are extracted as they tend to effectively explain variability in data
- One application of feature engineering could be to calculate how similar different 'pieces' of a text are
- There are various ways of calculating the similarity between two 'pieces' of texts
- The most popular methods are **cosine similarity** and **Jaccard similarity**
- **Cosine similarity**: The cosine similarity between two texts is the **cosine of the angle between their vector representations**
  - **BOW** and **TF-IDF** matrices can be regarded as **vector** representations of texts
  - The **cosine similarity** measures the **cosine of the angle** between **two vectors** projected in a multi-dimensional space
  - The two vectors are **arrays** containing the **word counts** of two documents. Below is the formula:

$$Cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \, \|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2} \sqrt{\sum_1^n b_i^2}}$$

where, $\vec{a} \cdot \vec{b} = \sum_1^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$ is the dot product of the two vectors.

  - **cosΘ = -1** indicates **strongly opposite vectors**, **cosΘ = 0** means **independent (orthogonal) vectors** and **cosΘ = 1** means **positive co-linear vectors**. **Intermediate values** are used to assess the **degree of similarity**
- **Jaccard similarity**: This is the **ratio** of the **number of terms common between two text documents** to the **total number of unique terms present in those texts**

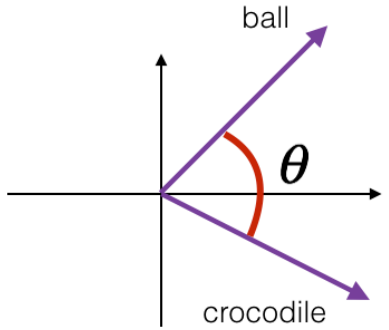
Feature Engineering: Computing Similarity (Cont.)



France and Italy are quite similar

θ is close to 0°

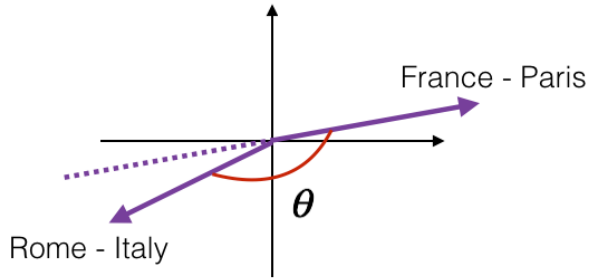
$\cos(\theta) \approx 1$



ball and crocodile are not similar

θ is close to 90°

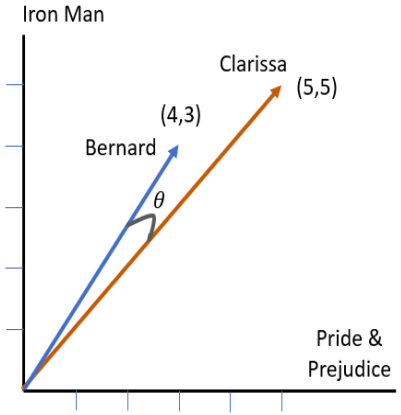
$\cos(\theta) \approx 0$



the two vectors are similar but opposite
the first one encodes (city - country)
while the second one encodes (country - city)

θ is close to 180°

$\cos(\theta) \approx -1$



Calculating:

$b.c = \sum_{i=1}^{n} b_i c_i = (4 \times 5) + (3 \times 5) = 35$

$\|b\| = \sqrt{4^2 + 3^2} = 5$

$\|c\| = \sqrt{5^2 + 5^2} = 5\sqrt{2}$

$similarity = \frac{35}{5 \times 5\sqrt{2}} \sim 0.989$

Cosine Similarity Improvement

- **Soft Cosine Measure** allows to assess the similarity between two documents in a meaningful way, even when they have no words in common. It relies on synonymy to measure the similarity.
- The angle between two basis vectors is derived from the angle between the word2vec embeddings of the corresponding words
- More accurate similarity measurement for documents with different word choice but similar meaning
- Can handle synonyms, paraphrases, and other variations in word usage
- Useful for tasks like document retrieval, question answering, and plagiarism detection

UMBC

9. TEXT SIMILARITY AND CLUSTERING

- **Supervised models** are used to classify or categorize text documents into pre-assumed document
- Topic models and document summarization are examples of **unsupervised techniques**
- To be effective, supervised models require considerable amounts of training data and it is not easy to build a database when data are unstructured. As a result, we need to take the time to label the data, build a model and use it to classify the data
- One way to accomplish this task is to get the computer to do it by looking at the context of text documents, analyzing their similarity using specific measures, and clustering similar documents together
- With **text similarity** and **clustering**, we are trying to segment and categorize documents into separate categories by making the machine learn about the text documents, their features, similarities, and differences among them

Important Concepts

- **Information retrieval** is the process of retrieving or fetching relevant sources of information from a corpus of entities that hold some information based on requests. Search engines are tools for information retrieval
  - The relevance of text can depend on **keywords** or using similarity measures to see the **similarity rank** or **score** of the documents with respect to a query
  - This is different from string matching because the words in a search string can have different order, context, and semantics in the collection of documents

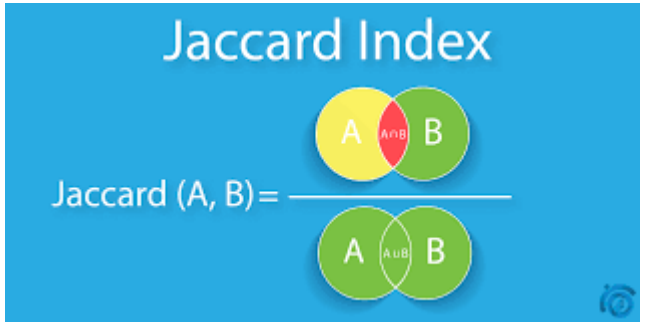
Important Concepts

- **Feature Engineering** or **feature extraction** includes methods such as **Bag of Words**, **TF-IDF**, and **word vectorization** models, which makes it possible to convert texts into vectors to apply mathematical or machine learning techniques more easily

- **Similarity Measures** are frequently used in text similarity and clustering. The purpose of such measures is to determine the degree of proximity of entities
  - There are two factors to determine the degree of similarity between entities:
    - **Inherent properties** or **features** of the entities
    - **Measure formula** and **properties**
  - All distance measures of similarity are not distance metrics of similarity
  - There are several types of similarity:
    - **Lexical similarity** involves observing the contents of the text documents with regard to syntax, structure, and content and measuring their similarity based on these parameters
    - **Semantic similarity** involves trying to find out the semantics, meaning and context of documents and then trying to see how close they are to each other
    - **Term similarity** measures similarity between individual tokens or words
    - **Document similarity** measure similarity between entire text documents

Using Similarity Measures

- **Jaccard Similarity**: Jaccard Similarity (coefficient), a term coined by Paul Jaccard, measures similarities between sets. It is defined as the size of the intersection divided by the size of the union of two sets



- **Cosine Similarity**: Cosine similarity is a metric used to measure how similar the documents are irrespective of their size
  - Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space
  - The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance (due to the size of the document), chances are they may still be oriented closer together. The smaller the angle, higher the cosine similarity
- **Soft Cosine Measure (SCM)** is a method that allows the evaluation of similarity between two documents in a meaningful way, even when they have no words in common. It uses a measure of similarity between words, which can be derived using word2vec vector embeddings of words

Using Distance Metrics

- **Hamming Distance**: It is a popular distance metric used in the information and communication theory.
  - It is the **distance measured between two strings** under the **assumption** that **they are of equal length**
  - It is also defined as the number of positions that have different characters or symbols between two strings of equal length
  - If two terms u and v are of length n, then the Hamming distance can be defined as hd(u, v) = $\sum_{i=1}^{n}(u_i \neq v_i)$. You can normalize the distance by dividing by n

Hamming distance: examples

- "karolin" and "kathrin" is 3.
- "karolin" and "kerstin" is 3.
- 1011101 and 1001001 is 2.
- 2173896 and 2233796 is 3.

- For binary strings a and b the Hamming distance is equal to the number of ones in **a XOR b**.

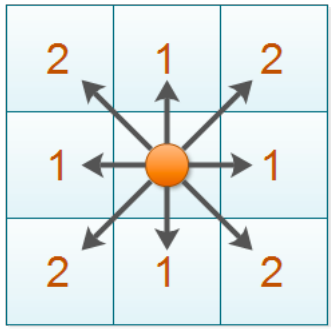
```
def hamming_distance(u, v, norm=false)
 if u.shape!= v.shape:
     raise ValueError('The vectors must have equal lengths.')
 return (u!=v).sum() if not norm else (u != v).mean()
```

- Hamming distance can be used to spell check and detect errors in coding

Using Distance Metrics

- **Manhattan Distance**: It is similar to the **Hamming distance**. However, instead of counting the number of mismatching we **subtract the difference** between **each pair of characters** at each position of the two strings
  - It is also known as the *city block distance*, *L1 norm*, *taxicab metric*
  - It is the distance between two points in a grid based on strictly horizontal or vertical paths instead of the diagonal distance conventionally calculated by the Euclidean distance metric
  - It can be defined as md(u, v) = $||u - v||_1 = \sum_{i=1}^{n} |u_i - v_i|$
  - The assumption of the two terms u and v having equal length from Hamming distance holds true
  - The distance can be normalized by dividing by n

Manhattan Distance

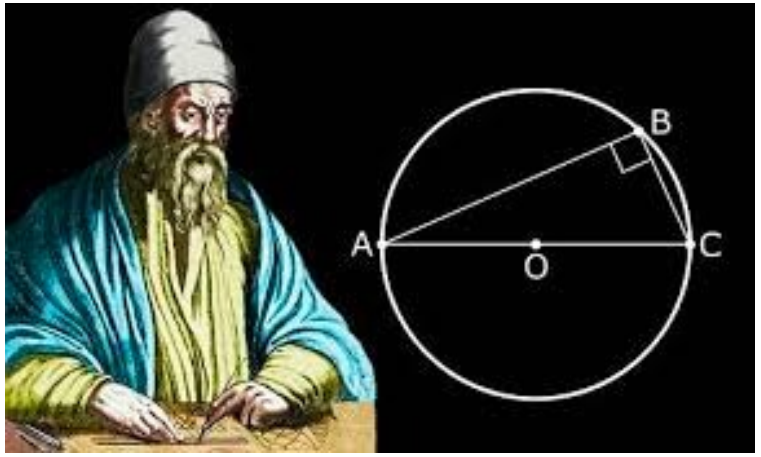


$$|x_1 - x_2| + |y_1 - y_2|$$

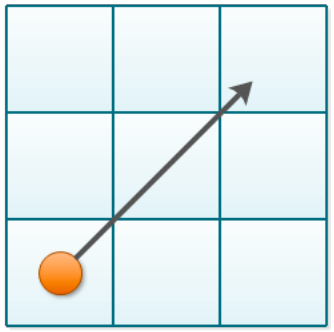
```
def manhattan_distance(u, v, norm=false)
  if u.shape != v.shape:
      raise ValueError('The vectors must have equal lengths.')
  return abs(u - v).sum() if not norm else abs(u - v).mean()
```

Using Distance Metrics

- **Euclidean Distance**: It is known as the *Euclidean norm*, *L2 norm*, *L2 distance*
  - It is defined as the **shortest straight line between two terms**
  - The formula is ed(u, v)=||u – v||$_2$ = $\sqrt{\sum_{i=1}^{n}(u_i - v_i)^2}$



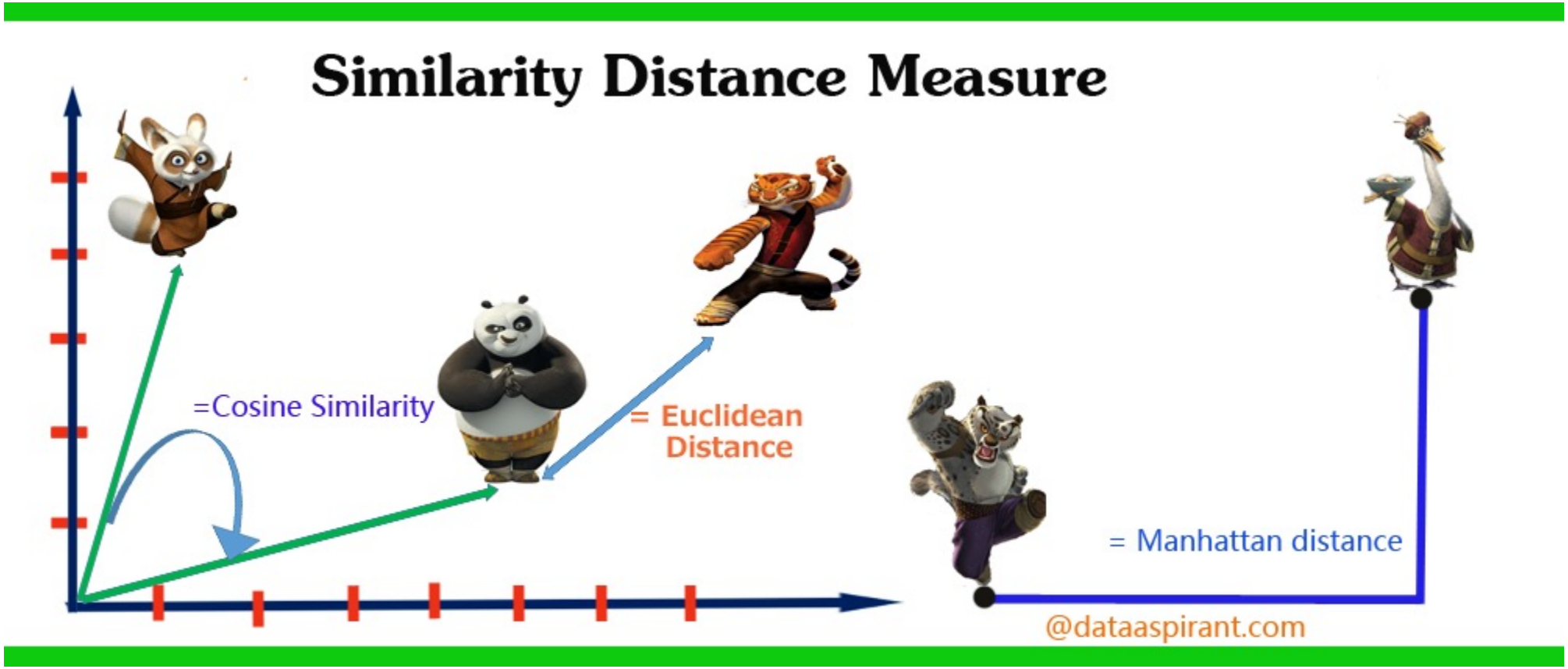
Euclidean Distance



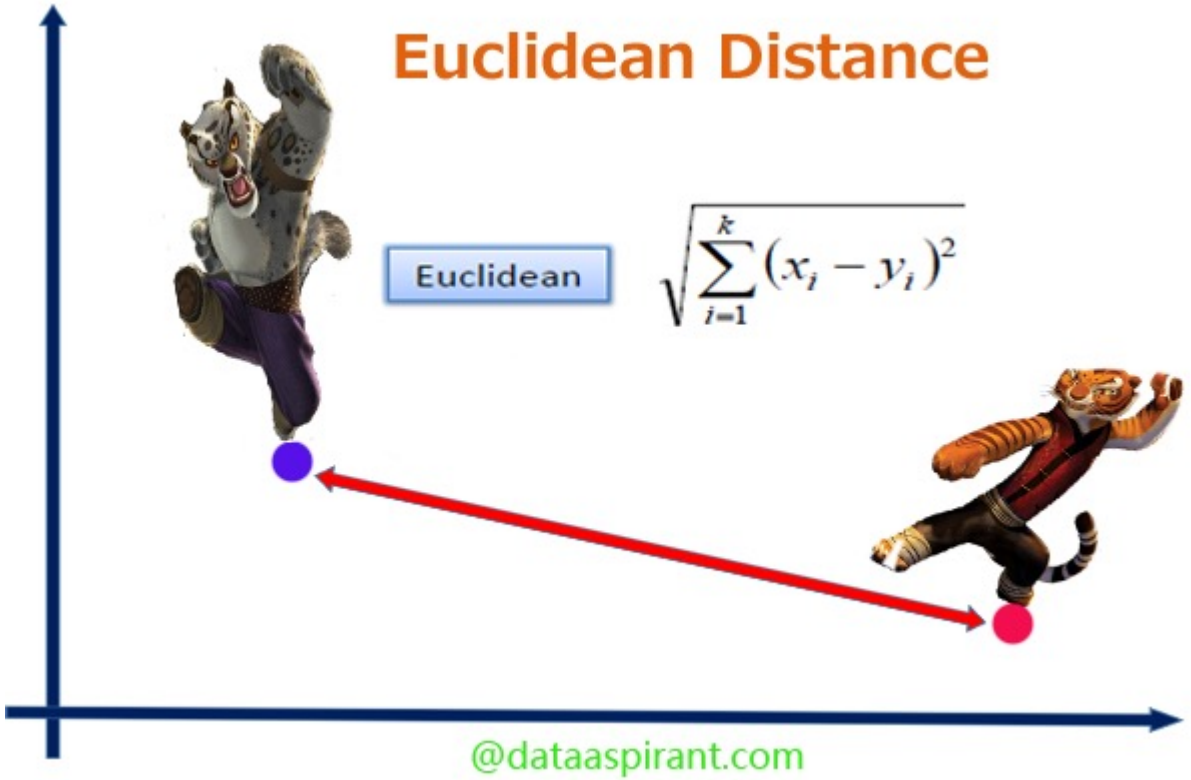
$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

```
def euclidean_distance(u, v, norm=false)
  if u.shape != v.shape:
      raise ValueError('The vectors must have equal lengths.')
  return np.sqrt(np.sum(np.square(u – v)))
```

Using Distance Metrics: Illustration of Similarity Distance Measures



Using Distance Metrics: Illustration of Similarity Distance Measures



- **Euclidean Distance** : Calculated as the square root of the sum of the squared differences between a point a and point b across all input attributes i
- **Hamming Distance**: Calculate the distance between binary vectors
- **Manhattan Distance**: Calculate the distance between real vectors using the sum of their absolute difference. Also called City Block Distance
- **Minkowski Distance**: Generalization of Euclidean and Manhattan distance

Comparison of Distance Metrics

Edit Distance

Minimum-edit (Levenstein) distance — the minimum number of insertions/deletions/substitutions needed to transform string A into B.

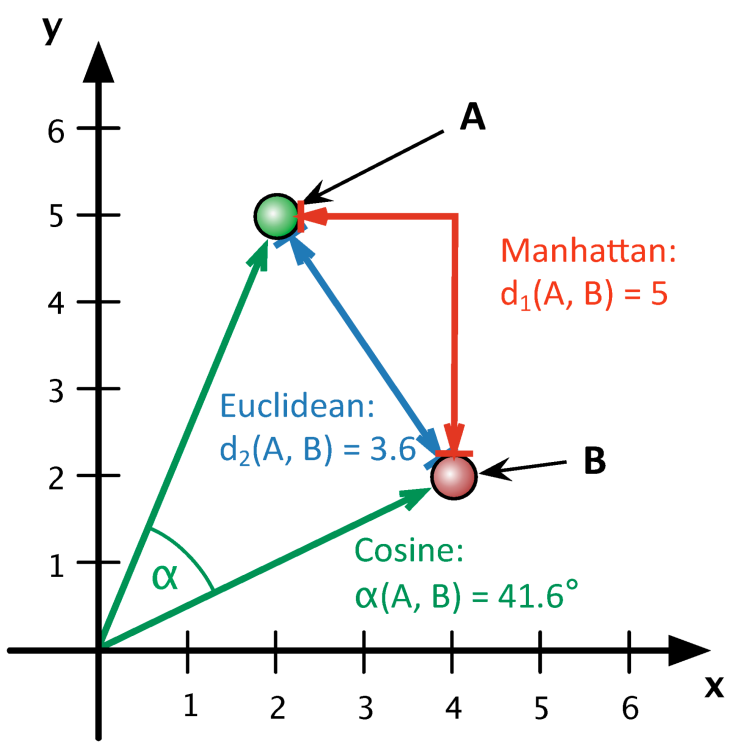
Other distance metrics:
* the Damerau-Levenstein distance adds another operation: transposition
* the longest common subsequence (LCS) metric allows only insertion and deletion, not substitution
* the Hamming distance allows only substitution, hence, it only applies to strings of the same length

Source: Dyomkin (2016)
https://www.slideshare.net/vseloved/crash-course-in-natural-language-processing-2016

Hamming and Euclidean

- Hamming (Block Distance, L1 distance, city block distance and Manhattan distance) : In information theory, the Hamming distance between two strings of **equal length** is the **sum of the differences of their corresponding components**.

- Euclidean distance or L2 distance : is the square root of the sum of squared differences between corresponding elements of the two vectors.



Other Distance Metrics

Minkowsky:

$$D(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{m} |x_i - y_i|^r\right)^{1/r}$$

Euclidean:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city-block:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i|$$

Camberra:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|}$$

Chebychev:

$$D(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1}^{m} |x_i - y_i|$$

Quadratic:

$$D(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}-\boldsymbol{y})^T Q(\boldsymbol{x}-\boldsymbol{y}) = \sum_{j=1}^{m}\left(\sum_{i=1}^{m}(x_i - y_i)q_{ji}\right)(x_j - y_j)$$

Q is a problem-specific positive definite $m \times m$ weight matrix

Mahalanobis:

$$D(\boldsymbol{x},\boldsymbol{y}) = [\det V]^{1/m}(\boldsymbol{x}-\boldsymbol{y})^{\mathrm{T}} V^{-1}(\boldsymbol{x}-\boldsymbol{y})$$

V is the covariance matrix of $A_1..A_m$, and A_j is the vector of values for attribute j occuring in the training set instances $1..n$.

Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{i=1}^{m}(x_i - \overline{x_i})(y_i - \overline{y_i})}{\sqrt{\sum_{i=1}^{m}(x_i - \overline{x_i})^2 \sum_{i=1}^{m}(y_i - \overline{y_i})^2}}$$

$\overline{x_i} = \overline{y_i}$ and is the average value for attribute i occuring in the training set.

Chi-square:

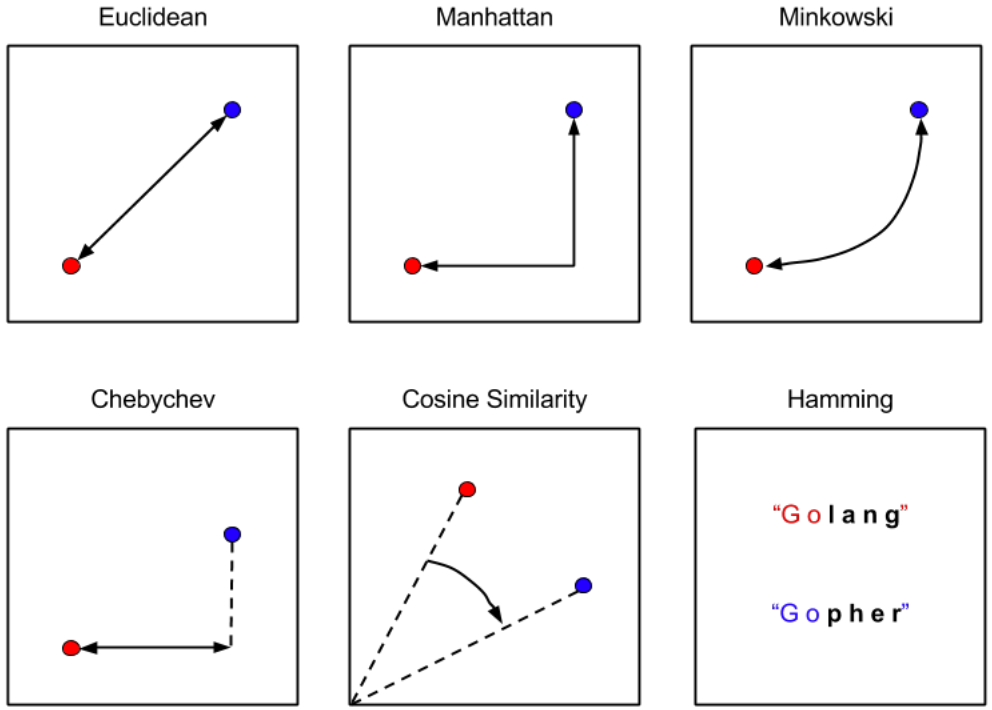
$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{1}{sum_i}\left(\frac{x_i}{size_{\boldsymbol{x}}} - \frac{y_i}{size_{\boldsymbol{y}}}\right)^2$$

sum_i is the sum of all values for attribute i occuring in the training set, and $size_x$ is the sum of all values in the vector $\boldsymbol{x}$.

Kendall's Rank Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{2}{n(n-1)}\sum_{i=1}^{m}\sum_{j=1}^{i-1}\mathrm{sign}(x_i - x_j)\mathrm{sign}(y_i - y_j)$$

sign(x)=-1, 0 or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.

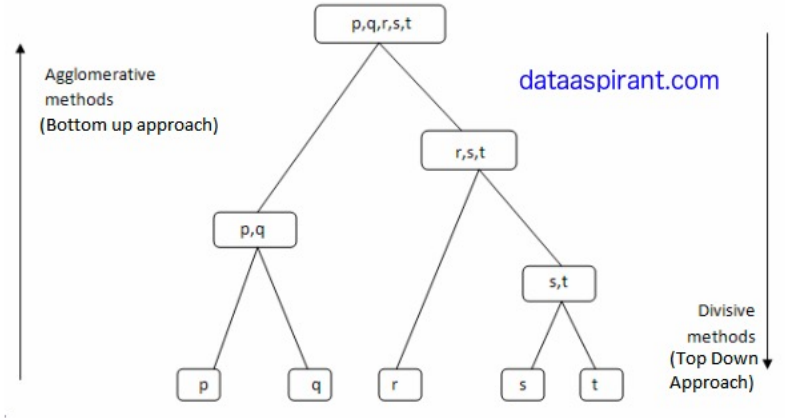


Clustering

- It is a growing area of interest in NLP and text analytics that applies **unsupervised models** and **techniques**
- We start with a corpus of documents, and we seek to separate them into various groups based on some properties, attributes, and features of the documents
- **Document classification** needs pre-labeled training data to build a model and then help categorize documents
- **Document clustering** uses unsupervised algorithms to group the documents into various clusters

Popular Clustering Algorithms

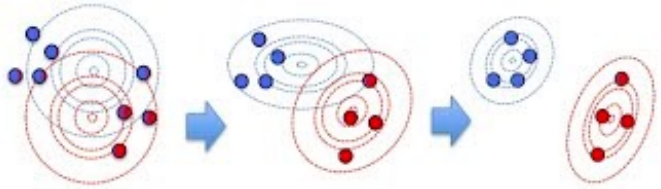
1. **Hierarchical Clustering.** They are also known as connectivity-based methods. This means that similar objects will be closer related objects in the vector space than unrelated objects, which will be farther away
   - Clusters are connected based on distance and they are usually visualized by **dendrograms**
   - Hierarchical models are divided into **agglomerative** and **divisive** clustering models



2. **Centroid-Based.** Each cluster has a central representative member that represents each cluster and has the features that distinguish that particular cluster from the rest
   - We need to set the number of clusters k in advance, which may lead to a local minima or not
   - Examples are **K-Means** and **K-Nearest Neighbors**

3. **Distribution-Based.** Data points are clustered based on probability distributions. Objects that have similar distributions are clustered into the same group
   - **Gaussian mixture models (GMM)** use algorithms such as Expectation-Maximization algorithms to build clusters
   - Feature and attribute correlations and dependencies can also be captured through these models, but they are prone to overfitting

Gaussian Mixture Model



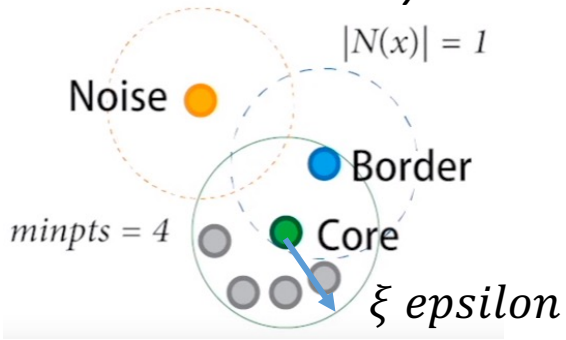
- Data with D attributes, from Gaussian sources $c_1 \dots c_k$
  - how typical is $\mathbf{x}_i$ under source **c**: $P(\vec{x}_i \mid c) = \frac{1}{\sqrt{2\pi|\Sigma_c|}} \exp\left\{-\frac{1}{2}(\vec{x}_i - \vec{\mu}_c)^T \Sigma_c^{-1} (\vec{x}_i - \vec{\mu}_c)\right\}$ $\sum_a \sum_b (x_{ia} - \mu_{ca})\left[\Sigma_c^{-1}\right]_{ab} (x_{ib} - \mu_{cb})$
  - how likely that $\mathbf{x}_i$ came from **c**: $P(c \mid \vec{x}_i) = \frac{P(\vec{x}_i \mid c)P(c)}{\sum_{c=1}^{k} P(\vec{x}_i \mid c)P(c)}$
  - how important is $\mathbf{x}_i$ for source **c**: $w_{ic} = P(c \mid \vec{x}_i) / \left(P(c \mid \vec{x}_1) + \dots + P(c \mid \vec{x}_n)\right)$
  - mean of attribute **a** in items assigned to **c**: $\mu_{ca} = w_{c1} x_{1a} + \dots + w_{cn} x_{na}$
  - covariance of **a** and **b** in items from **c**: $\Sigma_{cab} = \sum_{i=1}^{n} w_{ci} (x_{ia} - \mu_{ca})(x_{ib} - \mu_{cb})$
  - prior: how many items assigned to **c**: $P(c) = \frac{1}{n}\left(P(c \mid \vec{x}_1) + \dots + P(c \mid \vec{x}_n)\right)$

Popular Clustering Algorithms

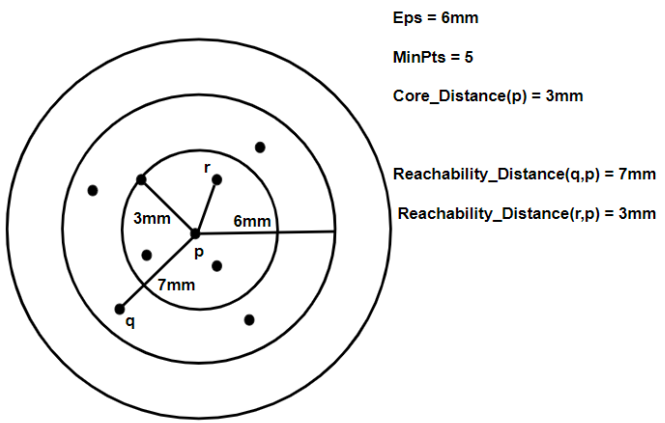
4. **Density-Based.** Data points are clustered together at areas of high density compared with other points

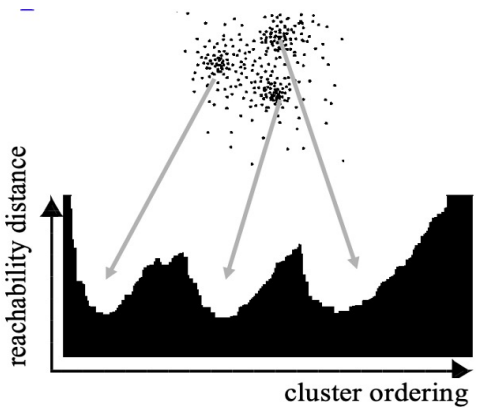
The sparse areas are treated as noise and are used as border points to separate clusters

- **DBSCAN** (**Density-based spatial clustering of applications with noise):** Given a set of points in a space, it groups together points that are closely distanced (those that have many nearby neighbors), marking as outliers points that lie alone in low-density regions (that is, whose nearest neighbors are too far away). **DBSCAN** requires a minimum number of points and epsilon
- **OPTICS (Ordering points to identify the clustering structure)** does not segregate a given set of data into clusters. It merely produces a **Reachability Distance** plot
  - Clustering depends upon the interpretation to cluster the points accordingly**.** It addresses one of DBSCAN's major weaknesses: the problem of detecting meaningful clusters in data of varying density
  - It requires epsilon and minimum points
  - **Reachability Distance** is defined with respect to another data point **q**. The Reachability Distance between a point **p and q** is the **maximum** of the **Core Distance of p** and the **Distance between p and q**. *Note that The Reachability Distance is not defined if q is*



DBSCAN v. **OPTICS**





5. Affinity-Propagation-Based

- With k-means, we need to define the number of clusters. What happens if there are more or fewer clusters/
- There are ways of checking the cluster quality and seeing what the value of the optimum k is
- We can use the *elbow method* or the *silhouette coefficient*
- **Affinity propagation (AP)** tries to build clusters based on the inherent properties of the data without any pre-assumption about the number of clusters
- AP creates **clusters** from a set of data points by passing messages between pairs of data points until convergence is achieved. The data set is represented by a few *examplars* that act as representatives for samples
- The **examplars** are like centroids. Messaging between pairs of data keeps on going until one of the points is suitable to be representative of other points. When convergence is reached, the final *examplars* are representative of each cluster.

```
!pip install -q distance
import numpy as np
from sklearn.cluster import AffinityPropagation
import distance
#Replace this line
words ="The price of gas has decreased compared with last
year despite a busy summer season and a strong demand for
travel".split(" ")
words = np.asarray(words)
#So that indexing with a list will work
lev_similarity = -1*np.array([[distance.levenshtein(w1,w2)
for w1 in words] for w2 in words])
affprop = AffinityPropagation(affinity="precomputed",
damping=0.5)
affprop.fit(lev_similarity.astype('float32'))
for cluster_id in np.unique(affprop.labels_):
exemplar =
words[affprop.cluster_centers_indices_[cluster_id]]
cluster =
np.unique(words[np.nonzero(affprop.labels_==cluster_id)])
cluster_str = ", ".join(cluster)
print(" - *%s:* %s" % (exemplar, cluster_str))
```

UMBC

10. TEXT CLASSIFICATION

Text Classification

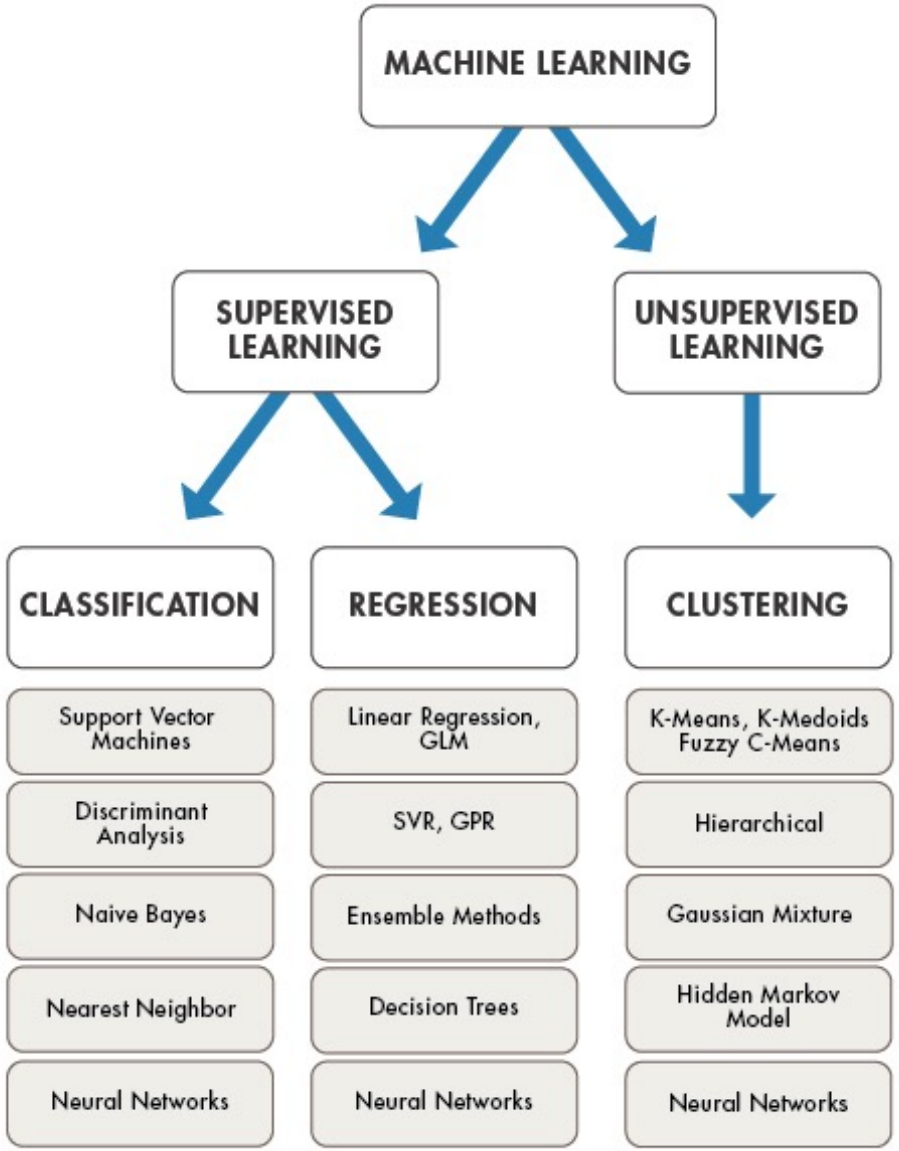
- According to Jurafsky and Martin (2008), "The goal of classification is to **take a single observation**, **extract some useful features**, and thereby **classify the observation** into one of a **set of discrete classes**"
- One method for classifying text is to use handwritten rules. There are many areas of language processing where handwritten rule-based classifiers constitute a state-of-the-art system
- Rules can be fragile, however, as situations or data change over time, and for some tasks, humans cannot always generate satisfactory rules
- Most cases of **classification** in natural language processing are instead done via **supervised machine learning**
- The task of **supervised classification** is to take an input x and a fixed set of output classes $Y = y_1, y_2, ..., y_M$ and return a predicted class $y \in Y$
- For text classification, we often use c (for "class") instead of y as our output variable, and d (for "document") instead of x as our input variable
- In the supervised situation, we have a training set of N documents that have each been hand-labeled with a class: $(d_1, c_1), ..., (d_N, c_N)$
- The goal is to learn a classifier capable of mapping from a new document d to its correct class $c \in C$
- **Classifiers** can be constructed with **unsupervised models**, i.e., cluster analysis
- There are two major text classification variants:
  - **Content-based classification** is a type of classification where priorities and weights are given to specific subjects or topics in the text content
  - **Request-based classification** depends on user requests and is targeted toward specific user groups or audience

Developing a Text Classifier

- A **text classifier** is a machine learning model that is capable of labeling texts based on their content
- **Classification** is an ML model that predicts outcomes based on distinct categories. The outcome is therefore **categorical**
- There are two types of machine learning techniques to classify text: **supervised** and **unsupervised** models
- However, **reinforcement-learning** and **semi-supervised** models can also be used, although not as popular
- A text **classifier** will help understand whether a random text statement is sarcastic or not when trained
- **Text classifiers** make it possible to classify large amounts of text data
- To develop **an end-to-end classifier,** you proceed through these steps:
    1. Clean and tokenize the corpus
    2. Extract the features using TF-IDF
    3. Divide the dataset into training and validation sets. Several machine learning algorithms, such as logistic regression, random forest, and XGBoost can be used to develop classification models
    4. Measure the performance of the models using parameters such as **confusion matrix**, **accuracy**, **precision**, **recall**, **F1-plot curve**, and **ROC** curve in the case of **supervised models**
- There are two types of **classifiers:**
    - **Generative classifiers** like Naive Bayes build a model of how a class could **generate** some input data. Given an observation, they return the class most likely to have generated the observation
    - **Discriminative classifiers** like logistic regression instead learn what features from the input are most useful to **discriminate** between the different possible classes
- While discriminative systems are often more accurate and, hence more commonly used, generative classifiers still have a role

UMBC

Text Classification: Supervised v. Unsupervised Models



Text Classification: Other Variants

- There are several variants of classification tasks based on the number of classes to predict:
  - **Binary classification**. There are only two classes i.e., ‘0’ or ‘1’
  - **Multi-class** or **multinomial**. There are more than two classes i.e., ‘0’ for neutral, ‘1’ for positive, ‘2’ for negative.
  - **Multi-label**. The model classifies the outcomes based on multi-labels i.e., ’Republican’, ’Democrat’, and ‘Independent’. This can be achieved with SVM Classifier

Text Classification: Algorithms for Multi-Class Classification

Inherently multiclass:

- sklearn.naive_bayes.BernoulliNB
- sklearn.tree.DecisionTreeClassifier
- sklearn.tree.ExtraTreeClassifier
- sklearn.ensemble.ExtraTreesClassifier
- sklearn.naive_bayes.GaussianNB
- sklearn.neighbors.KNeighborsClassifier
- sklearn.semi_supervised.LabelPropagation
- sklearn.semi_supervised.LabelSpreading
- sklearn.discriminant_analysis.LinearDiscriminantAnalysis
- sklearn.svm.LinearSVC (setting multi_class=”crammer_singer”)
- sklearn.linear_model.LogisticRegression (setting multi_class=”multinomial”)
- sklearn.linear_model.LogisticRegressionCV (setting multi_class=”multinomial”)
- sklearn.neural_network.MLPClassifier
- sklearn.neighbors.NearestCentroid
- sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis
- sklearn.neighbors.RadiusNeighborsClassifier
- sklearn.ensemble.RandomForestClassifier
- sklearn.linear_model.RidgeClassifier
- sklearn.linear_model.RidgeClassifierCV

Multiclass as One-Vs-One:

- sklearn.svm.NuSVC
- sklearn.svm.SVC.
- sklearn.gaussian_process.GaussianProcessClassifier (setting multi_class = “one_vs_one”)

Multiclass as One-Vs-The-Rest:

- sklearn.ensemble.GradientBoostingClassifier
- sklearn.gaussian_process.GaussianProcessClassifier (setting multi_class = “one_vs_rest”)
- sklearn.svm.LinearSVC (setting multi_class=”ovr”)
- sklearn.linear_model.LogisticRegression (setting multi_class=”ovr”)
- sklearn.linear_model.LogisticRegressionCV (setting multi_class=”ovr”)
- sklearn.linear_model.SGDClassifier
- sklearn.linear_model.Perceptron
- sklearn.linear_model.PassiveAggressiveClassifier

Support multilabel:

- sklearn.tree.DecisionTreeClassifier
- sklearn.tree.ExtraTreeClassifier
- sklearn.ensemble.ExtraTreesClassifier
- sklearn.neighbors.KNeighborsClassifier
- sklearn.neural_network.MLPClassifier
- sklearn.neighbors.RadiusNeighborsClassifier
- sklearn.ensemble.RandomForestClassifier
- sklearn.linear_model.RidgeClassifierCV

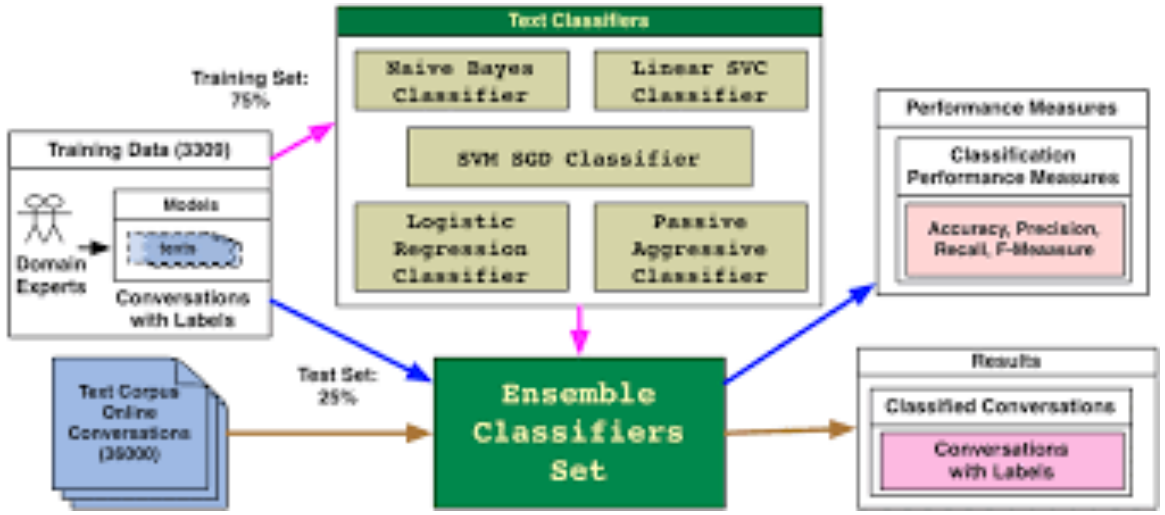
Support multiclass-multioutput:

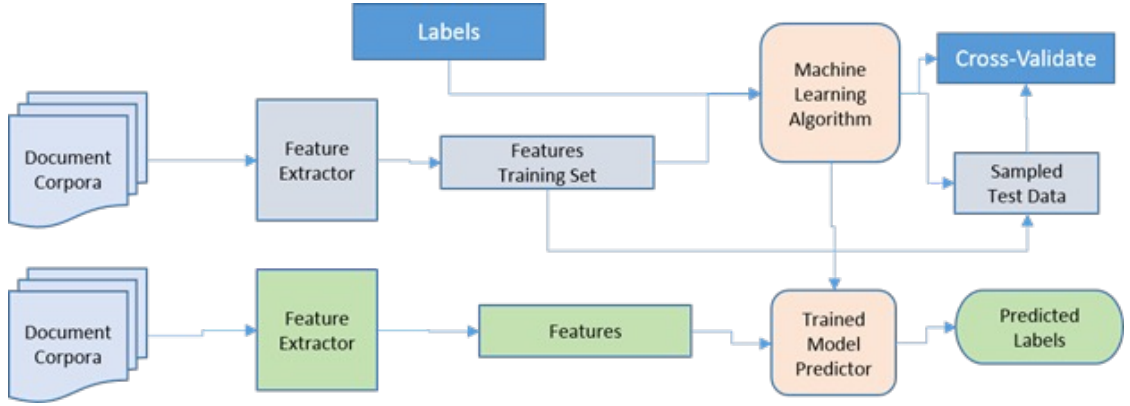
- sklearn.tree.DecisionTreeClassifier
- sklearn.tree.ExtraTreeClassifier
- sklearn.ensemble.ExtraTreesClassifier
- sklearn.neighbors.KNeighborsClassifier
- sklearn.neighbors.RadiusNeighborsClassifier
- sklearn.ensemble.RandomForestClassifier

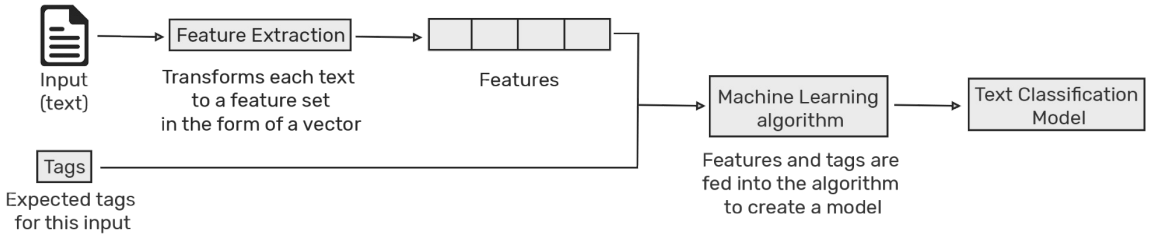
Warning

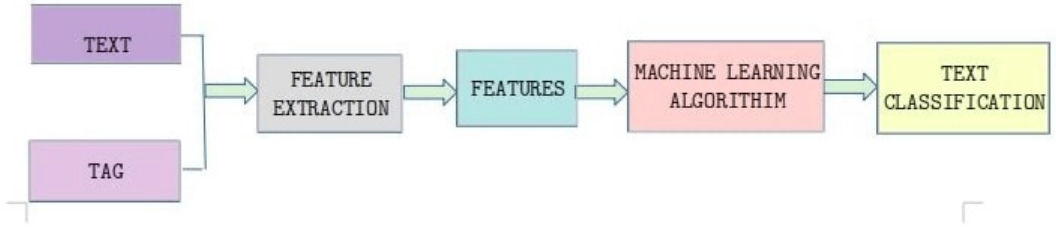
At present, no metric in sklearn.metrics supports the multioutput-multiclass classification task.

Text Classification Process









Text Classification Process and Measuring Model Accuracy

Process

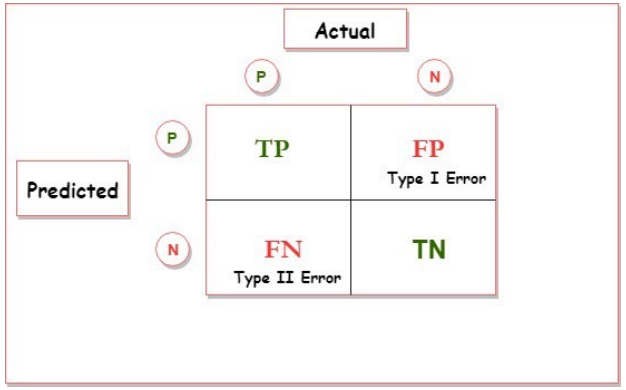
1. We prepare the train and test datasets (optionally, a validation set)
2. We preprocess and normalize text documents
3. We extract features (feature engineering i.e., BOW, TF-IDF, bag of n-grams, Word2Vec, GloVe, FastText)
4. We train the model
5. We predict the outcomes and evaluate the accuracy of the outcomes
6. We deploy the model

Tools to evaluate the classifier

- **Accuracy** is the overall proportion of correct predictions of the model**: (TP + TN) / (TP + FP + TN + FN)**
- **Precision (positive predictive value)** is the number of predictions made that are actually correct or relevant out of all the predictions based on the positive class**: TP/(TP + FP)**
- **Recall (sensitivity)** is the number of instances of the positive class that were correctly predicted: **TP/(TP + FN)**
- **F1- Score** is the harmonic mean of precision and recall and helps us optimize a classifier for balanced precision and recall performance: **(2*Precision + Recall)/(Precision + Recall)**

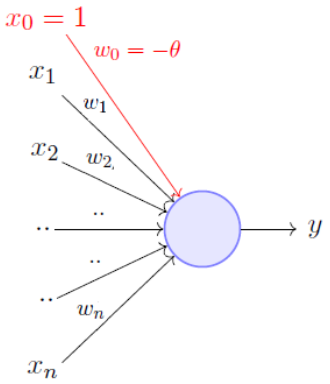
Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)



Text Classification: Perceptron

- A **perceptron**, a neuron's computational prototype, is categorized as the simplest form of a **neural network**. Frank Rosenblatt invented the perceptron at the Cornell Aeronautical Laboratory in 1957. It is a **linear classifier**
- The **perceptron** takes an input, aggregates it (weighted sum) and returns 1 only if the aggregated sum is more than some threshold else returns 0
- A single **perceptron** can only be used to implement **linearly separable** functions
- It takes both real and Boolean inputs and it associates a set of **weights** to them, along with a **bias**



A more accepted convention,

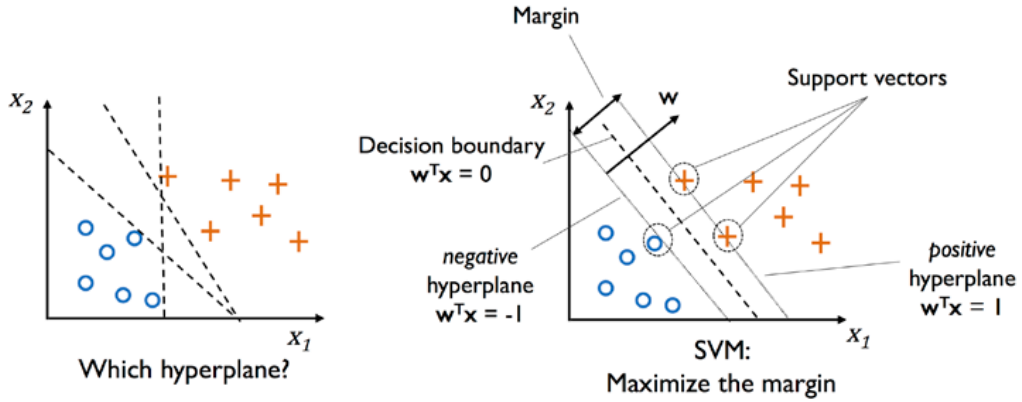
$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

$$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$$

- The **perceptron** is used as an algorithm or a linear classifier to facilitate supervised learning of binary classifiers. Some of the common problems of supervised learning include classification to predict class labels
- The **perceptron** takes an input, aggregates it (weighted sum) and returns 1 only if the aggregated sum is more than some threshold else returns 0. A single **perceptron** can only be used to implement **linearly separable** functions
- It takes both real and Boolean inputs and it associates a set of **weights** to them, along with a **bias.** The **bias** allows the classifier to turn its decision boundary around. The objective of the **bias** is to shift each point in a particular direction for a specified distance. **Bias** allows for higher quality and faster model training

- The **SVM** defines the criterion that identifies a decision surface or **hyperplane** that is maximally far away from any data point
- It is a **supervised machine learning algorithm** that can be used for **regression** and **classification**
- In SVMs, the optimization objective is to maximize the margin
  - The *margin* is defined as the distance between the separating hyperplane (decision boundary) and the training examples that are closest to this hyperplane, which are the so-called **support vectors**
  - In other words, this distance from the decision surface to the closest data point determines the *margin* of the classifier
- This method of construction implies that the decision function for an SVM is fully specified
- A (usually small) subset of the data defines the position of the separator
- These points are referred to as the *support vectors*
- The goal is *to maximize the distance between the two boundary hyperplanes to reduce the probability of misclassification*



Text Classification: Naïve Bayes

- **Naïve Bayes** is a **probabilistic and generative** classifier
- A probabilistic classifier indicates the probability of the observation being in the class. This full distribution over the classes can be useful information for downstream decisions: avoiding making discrete decisions early on can be useful when combining systems
- The **Naive Bayes** algorithm is often applied to **text categorization** or **classification**, that is, the task of assigning a label or text categorization category to an entire text or document
- One common text categorization task is **sentiment analysis**:
  - It is the extraction of sentiment i.e., the 'positive', 'neutral', and 'negative' orientation that an individual or group of individual express(es) toward some issue(s). In this case, we are talking about **multinomial Naïve Bayes classification**
- **Spam detection** is another important commercial application of **text classification**, the **binary** classification task of assigning an email to one of the two classes: 'spam' or 'not-spam'
- **Text classification** can be used to identify the **language** to be studied, even **authorship** in some cases
- One of the oldest tasks in **text classification** is **assigning a library subject category or topic label to a text**
  - Deciding whether a research paper concerns epidemiology or instead, perhaps, embryology, is an important component of information retrieval
- **Language modeling** can be viewed as **classification**:
  - Each word can be thought of as a class, and so predicting the next word is classifying the context-so-far into a class for each next word
  - A **part-of-speech tagger** classifies each occurrence of a word in a sentence as, for instance, a noun or a verb

Text Classification: Naïve Bayes (Cont.)

- **Naïve Bayes** as a probabilistic classifier (definitions and example)

Naive Bayes

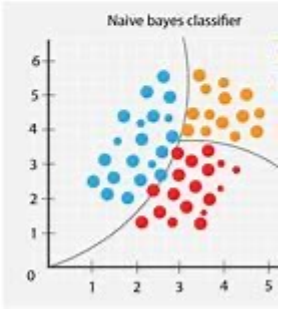
thatware.co

In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' with strong (naive) independence assumptions between the features.

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

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

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



What we want to know; the Posterior probability of Class j given a predictor x

The Likelihood; the probability of the predictor given a Class j. Its computed from the training-set.

The Prior probability of Class j; what we know about the class distribution before we consider x.

$$P(Class_j \mid x) = \frac{P(x \mid Class_j) \times P(Class_j)}{P(x)}$$

The Evidence. In practice, there's interest only in the numerator (denominator is effectively constant)

Applying the independence assumption

$$P(x \mid Class_j) = P(x_1 \mid Class_j) \times P(x_2 \mid Class_j) \times \ldots \times P(x_k \mid Class_j)$$

Substituting the independence assumption, we derive the Posterior probability of Class j given a new instance x' as...

$$P(Class_j \mid x') = P(x'_1 \mid Class_j) \times P(x'_2 \mid Class_j) \times \ldots \times P(x'_k \mid Class_j) \times P(Class_j)$$

Example of Naïve Bayes classifier

Name	Give Birth	Can Fly	Live in Water	Have Legs	Class
human	yes	no	no	yes	mammals
python	no	no	no	no	non-mammals
salmon	no	no	yes	no	non-mammals
whale	yes	no	yes	no	mammals
frog	no	no	sometimes	yes	non-mammals
komodo	no	no	no	yes	non-mammals
bat	yes	yes	no	yes	mammals
pigeon	no	yes	no	yes	non-mammals
cat	yes	no	no	yes	mammals
leopard shark	yes	no	yes	no	non-mammals
turtle	no	no	sometimes	yes	non-mammals
penguin	no	no	sometimes	yes	non-mammals
porcupine	yes	no	no	yes	mammals
eel	no	no	yes	no	non-mammals
salamander	no	no	sometimes	yes	non-mammals
gila monster	no	no	no	yes	non-mammals
platypus	no	no	no	yes	mammals
owl	no	yes	no	yes	non-mammals
dolphin	yes	no	yes	no	mammals
eagle	no	yes	no	yes	non-mammals

***X*: attributes**
***M*: class = mammal**
***N*: class = non-mammal**

$$p(X \mid M) = \frac{6}{7} \times \frac{6}{7} \times \frac{2}{7} \times \frac{2}{7} = 0.06$$

$$p(X \mid N) = \frac{1}{13} \times \frac{10}{13} \times \frac{3}{13} \times \frac{4}{13} = 0.0042$$

$$p(X \mid M)\,p(M) = 0.06 \times \frac{7}{20} = 0.021$$

$$p(X \mid N)\,p(N) = 0.004 \times \frac{13}{20} = 0.0027$$

Give Birth	Can Fly	Live in Water	Have Legs	Class
yes	no	yes	no	?

p(X | M) p(M) > p(X | N) p(N)
=> mammal

Jeff Howbert | Introduction to Machine Learning | Winter 2014 | ‹#›

Naïve Bayes Classification

Supervised learning, probabilistic classifier

- Features are independent of one another.
- Labels are assigned in advance.
- Feature detection is decided in advance.
- Classifiers are created by training them on labeled data.

Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. O'Reilly Media Inc.

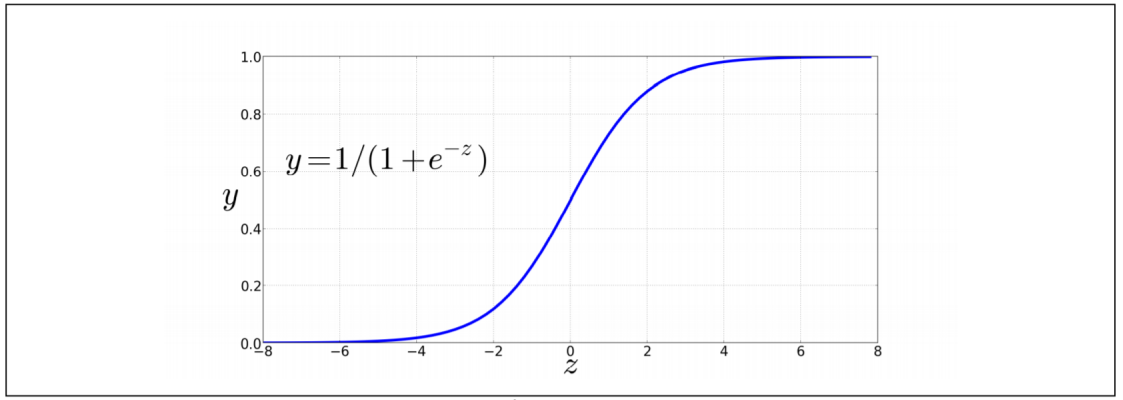
- **Logistic Regression** is a **supervised** and **discriminative** classifier
- **Logistic regression** is the baseline supervised machine learning algorithm for classification, and it also has a very close relationship with **neural networks.** A **neural network** can be viewed as a series of logistic regression classifiers stacked on top of each other
- **Logistic regression** can be used to classify an observation into one of two classes (like 'positive sentiment' and 'negative sentiment'), or into one of many classes (**multinomial logistic regression**)
- The most important difference between **Naive Bayes** and **Logistic Regression** is that **Logistic Regression** is a **discriminative classifier,** while **Naive Bayes** is a **generative classifier**
- A **discriminative** model is only trying to learn to distinguish the classes (dogs v. cats), while a **generative** model has the goal of understanding what dogs look like and what cats look like before classifying the animals
- The goal of **binary Logistic Regression model** is to train a classifier that can make a **binary decision** about the class of a new input observation
- **A Logistic Regression** model learns from a **training set**, a **vector of weights,** and a **bias** term
- Each weight w_i is a real number and is associated with one of the input features x_i
- The weight w_i represents how important that input feature is to the classification decision: It can be positive (meaning the feature is associated with the class) or negative (meaning the feature is not associated with the class)
- The **bias term**, also called the **intercept**, is another real number added to the weighted inputs
- The classifier first multiplies each x_i by its weight w_i , sums up the weighted features, and adds the bias term b
- The resulting single number z expresses the weighted sum of the evidence for the class

Text Classification: Logistic Regression (Cont.)

- The resulting single number z expresses the weighted sum of the evidence for the class as

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b \qquad \text{with} \qquad z = w \cdot x + b$$

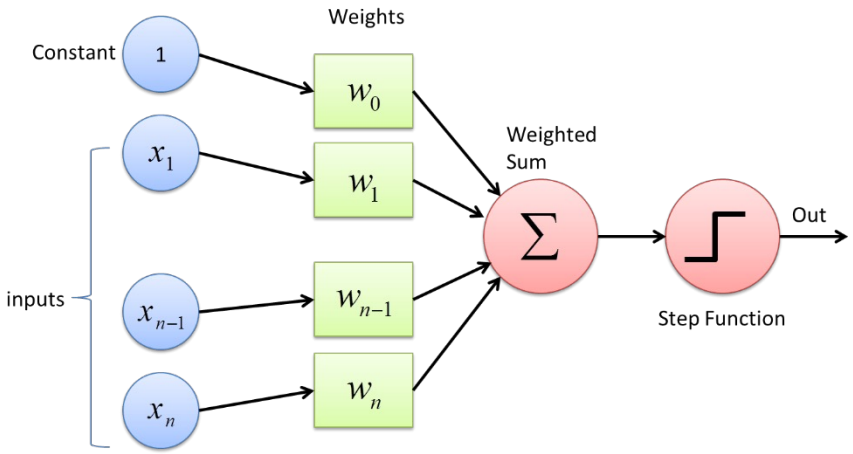
- To create a probability, we pass z through the **sigmoid function**, σ(z)
- The **sigmoid function** (named because it looks like an 's') is also called the **logistic function**
- The **sigmoid** has the following equation as the logistic function:



- The **sigmoid** has several advantages: it takes a real-valued number and maps it into the range [0,1]

Text Classification: Selected Models

Perceptron



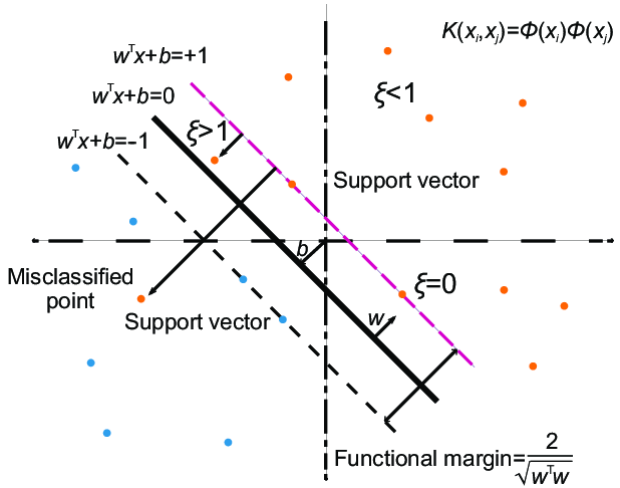
Naive Bayes

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

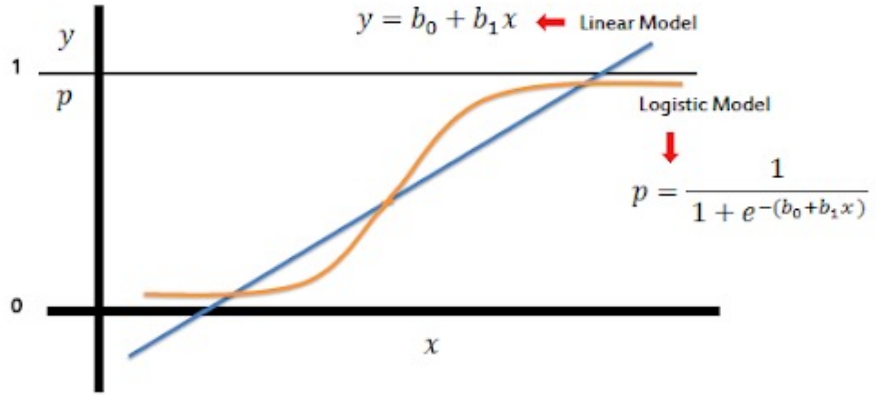
where A and B are events and $P(B) \neq 0$.

- $P(A \mid B)$ is a conditional probability: the likelihood of event A occurring given that B is true.
- $P(B \mid A)$ is also a conditional probability: the likelihood of event B occurring given that A is true.
- $P(A)$ and $P(B)$ are the probabilities of observing A and B independently of each other; this is known as the marginal probability.

Support Vector Machine



Logistic Regression



Text Classification: The Pros and Cons of Selected Models

Naive Bayes

- Pros: easy to implement; estimation is fast, requiring only a single pass over the data; assigns probabilities to predicted labels; controls overfitting with smoothing parameter
- Cons: often has poor accuracy, especially with correlated features

Perceptron

- Pros: easy to implement; online; error-driven learning means that accuracy is typically high, especially after averaging
- Cons: not probabilistic; hard to know when to stop learning; lack of margin can lead to overfitting

Support Vector Machine

- Pros: optimizes an error-based metric, usually resulting in high accuracy; overfitting is controlled by a regularization parameter
- Cons: not probabilistic

Logistic Regression

- Pros: error-driven and probabilistic; overfitting is controlled by a regularization parameter
- Cons: batch learning requires black-box optimization; logistic loss can “overtrain” on correctly labeled examples

Text Classification: The Pros and Cons of Selected Models (Cont.)

- One of the main distinctions is whether the learning algorithm offers a **probability** over **labels**
- This is useful in **modular architectures**, where the output of one classifier is the input for some other system
- In cases where probability is not necessary, the **Support Vector Machine** is usually the right choice, since it is no more difficult to implement than the **perceptron**, and it is often *more accurate*
- When probability is *necessary*, **logistic regression** is usually *more accurate* than **Naive Bayes**

UMBC

11. TOPIC MODELING

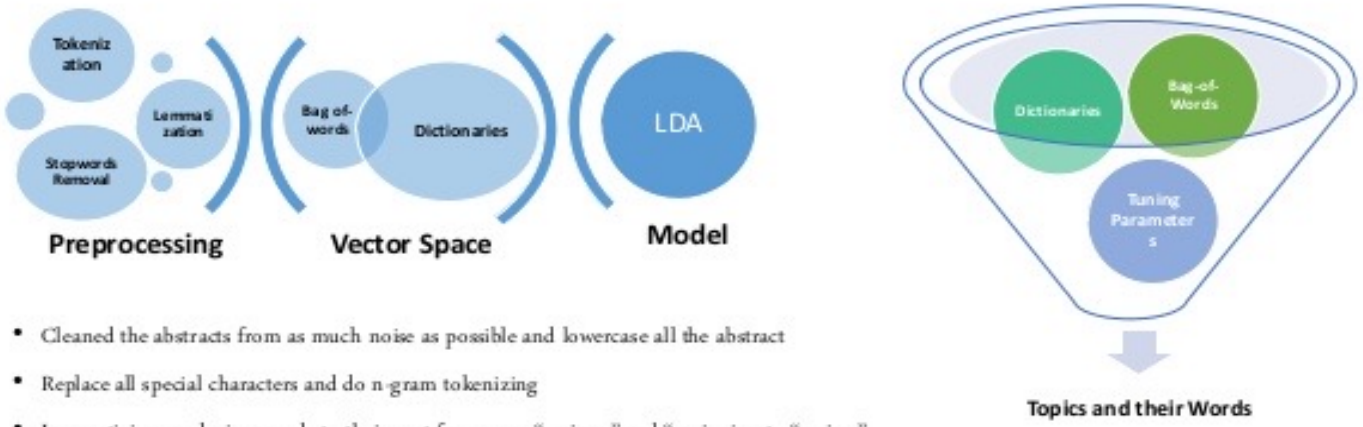
Topic Modeling: Latent Dirichlet Allocation

- **Latent Dirichlet Allocation** (LDA) is a generative probabilistic model of a corpus
- The basic idea is that documents are represented as **random** mixtures over **latent** topics, where each topic is characterized by a distribution over words
- It is a particularly popular method for **fitting a topic model**
- It **treats each document as a mixture of topics**, and **each topic as a mixture of words**
- This allows documents to "overlap" each other in terms of content, rather than being separated into discrete groups, in a way that mirrors the typical use of a natural language
- The probabilistic topic model estimated by **LDA** consists of two tables (matrices):
  - The first table **describes the probability or chance of selecting a specific part** when sampling a particular topic (category)
  - The second table **describes the chance of selecting a specific topic when sampling a particular document or composite**

Topic Modeling: Latent Dirichlet Allocation (Cont.)

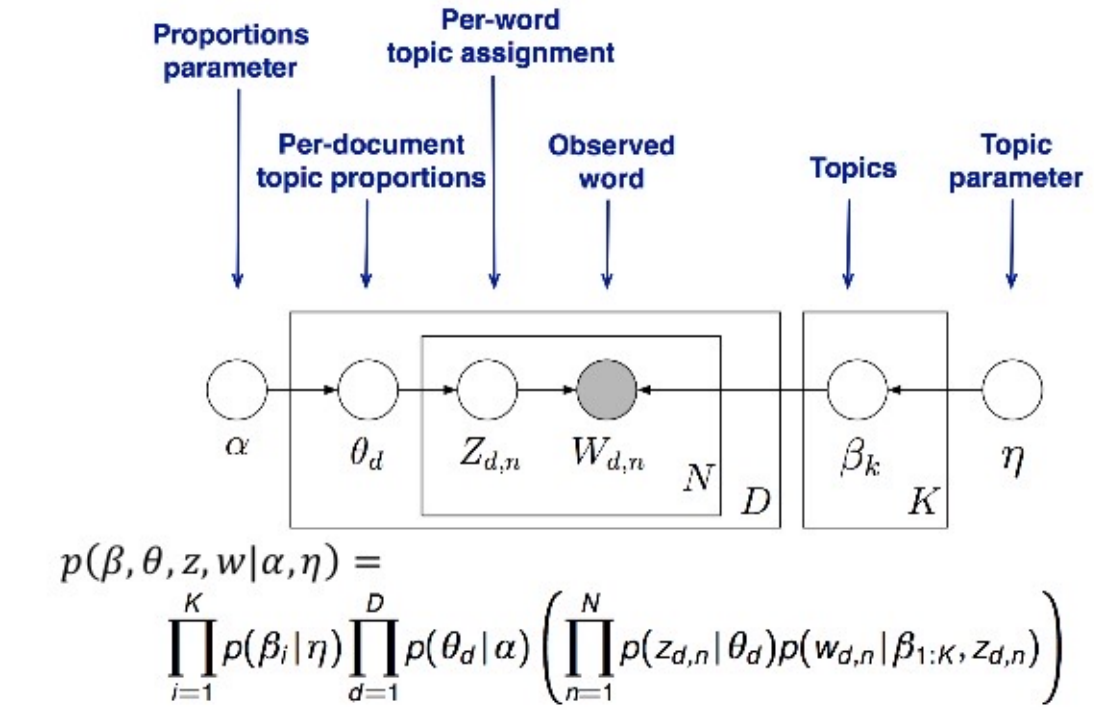
- There are two hyperparameters required in LDA:
  - The ***alpha*** **controls the mixture of topics for any given document**. If you turn it down, the documents will likely have less of a mixture of topics. If you turn it up, the documents will likely have more of a mixture of topics
  - The ***beta*** hyperparameter **controls the distribution of words per topic**. If you turn it down, and the topics will likely have less words. If you turn it up, the topics will likely have more words
- Ideally, we want our composites to be made up of only a few topics and our parts to belong to only some of the topics. Usually, *alpha* and *beta* are typically set below 1
- If you view the number of topics as several clusters and the probabilities as the proportion of cluster membership, then using **LDA** is a way of soft-clustering composites and parts

Topic Modeling with Latent Dirichlet Allocation



- Cleaned the abstracts from as much noise as possible and lowercase all the abstract
- Replace all special characters and do n-gram tokenizing
- Lemmatizing - reducing words to their root form, e.g., "reviews" and "reviewing to "review"
- Removing numbers (e.g., "2014") and removing HTML tags and symbols,
- Create Dictionaries, Corpus of Bag-of-Words
- Pass through LDA Algorithm and Evaluate

Topic Modeling: Latent Dirichlet Allocation (Cont.)



$$p(\beta, \theta, z, w|\alpha, \eta) = \prod_{i=1}^{K} p(\beta_i|\eta) \prod_{d=1}^{D} p(\theta_d|\alpha) \left(\prod_{n=1}^{N} p(z_{d,n}|\theta_d) p(w_{d,n}|\beta_{1:K}, z_{d,n}) \right)$$

Source: Blei, ICML 2012 tutorial

- **Alpha**: represents document-topic density i.e., the number of topics per document. Low value of alpha shows fewer number of topics in the document mix, while higher value indicates that the documents have a greater number of topics in the mix
- **Beta**: represents topic-word density i.e., the number of words per topic. Lower the value of beta, fewer are the number of words in the topic and vice-versa
- **K** = No of topics to consider

- **Every document is a mixture of topics**
- **Every topic is a mixture of words**
- LDA is a mathematical method for estimating both at the same time:
  - Finding the **mixture of words** that is **associated with each topic**
  - Determining the **mixture of topics** that **describes each document**

Topic Modeling: Latent Semantic Indexing

- **Latent semantic indexing** (**LSI**) is a concept used by search engines to discover how a term and content work together to mean the same thing--even if they do not share keywords or synonyms
- **Latent semantic indexing**, sometimes referred to as **latent semantic analysis**, is a mathematical method developed in the late 1980s to improve the accuracy of information retrieval
- It uses a technique called **singular value decomposition** (**SVD**) to scan unstructured data within documents and identify relationships between the concepts contained in the documents
- **SVD** finds the **hidden (latent) relationships** between words (semantics) in order to improve information understanding (indexing)
- It provides a significant step forward for the field of text comprehension as it accounts for the contextual nature of language
  - For example, the words 'hot' and 'dog' may seem easy to understand, but both have multiple definitions based on how they are used. Put both together and you have a whole new concept altogether
- **LSI** also allows documents to be clustered together based on their thematic commonalities, which was a very useful capability for early search engines

```
from gensim.models import LsiModel
lsimodel = LsiModel(corpus=corpus, num_topics=10, id2word=dictionary)
lsimodel.show_topics(num_topics=5) # Showing only the top 5 topics
```

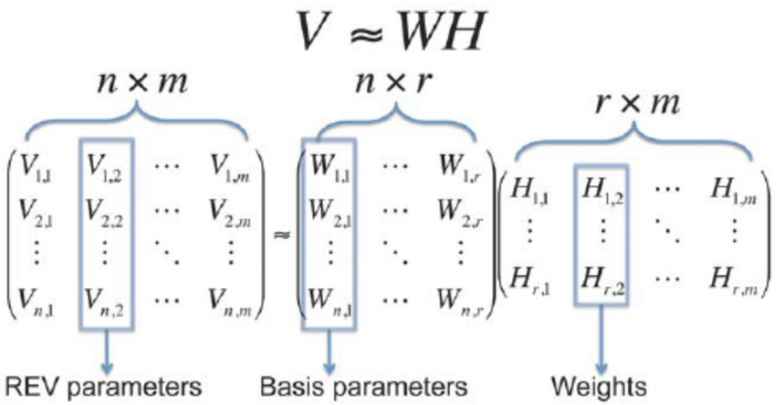
Topic Modeling: Hierarchical Dirichlet Process

- The other standard topic modeling algorithm popular in Gensim is **Hierarchical Dirichlet process** (**HDP**)
- It is also a brainchild of Michael I. Jordan and David Blei
- It is different from **LDA** and **LSI** because it is **non-parametric**: There is no need to specify the number of topics
- **HDP** is an extension of **LDA**, designed to address the case where the **number of mixture components** (the number of "topics" in document-modeling terms) is not known a priori
- Using **LDA** for document modeling, one treats each "topic" as **a distribution of words in some known vocabulary**
  - For each document, a mixture of topics is drawn from a **Dirichlet distribution**, and then each word in the document is an **independent draw from that mixture** (that is, selecting a topic and then using it to generate a word)
- **HDP** (applied to document modeling) also uses a **Dirichlet** process to capture the uncertainty in the number of topics
  - A common base distribution is selected, which represents the infinite set of possible topics for the corpus
  - Then, the finite distribution of topics for each document is sampled from this base distribution
- **HDP** has the advantage that the **maximum number of topics can be unbounded and learned from the data** rather than **specified in advance**
- The "hierarchical" portion of the name refers to another level being added to the generative model (the **Dirichlet** process producing the **number of topics**), not the topics themselves as the topics are still flat clusterings

```
from gensim.models import HdpModel
hdpmodel = HdpModel(corpus=corpus, id2word=dictionary)
hdpmodel.show_topics()
```

Topic Modeling: Non-Negative Matrix Factorization

- **NMF** has been widely used as a **clustering method**, especially for document data, and as a **topic modeling method**
- **NMF** can also be used for **dimensionality reduction**, **source separation**, or **topic extraction** using sklearn's 'decomposition' class
- **NMF** can be applied to **topic modeling**, where the input is a **term-document matrix**--typically normalized **TF-IDF**
- The goal of **NMF** is to find two non-negative matrices (W, H) whose product approximates the non-negative matrix V
- **NMF** has two main advantages when compared to LDA:
  - Its resolution depends on deterministic algorithms
  - NMF allows for an easier tuning and manipulation of its parameters



http://ceur-ws.org/Vol-2167/short5.pdf

Measuring the Predictive Power of a Generative Topic Model

- The predictive power of a generative topic model can be measured by analyzing the distribution of the generated corpus
- **Perplexity** is a measure of how close the distribution of the words in the generated corpus is to reality
- **Log perplexity** is a more convenient measure for this closeness. The formula is as follows:

$$\log perplexity = -\frac{1}{n}\sum_{k=0}^{n}\log P(w)$$

- Here, n is the number of words and P(w) is the probability associated with word w. The negative log likelihood is identical to the log perplexity
- A lower perplexity means a better performance because the probability distribution of words is not uniform: it is concentrated in a small subset of words. A non-uniform probability function causes a lower negative likelihood
- With the **gensim** library, the **CoherenceModel** function determines how well the topics were identified and how well they fit together. The **coherence score** indicates whether there is an overlap among topics
- **Entropy** is a **measure of the randomness** in the information being processed. The higher the **entropy**, the harder it is to draw any conclusions from that information
- **Entropy** measures the "amount of information" present in a variable. **Entropy** is estimated, not only based on the number of different values present in a variable, but also by the **amount of *surprise*** that this value of the variable holds. The **amount of information** in a message or text is directly **proportional to the amount of surprise** (information gain) available in the message

UMBC

Measuring the Predictive Power of a Generative Topic Model (Cont.)

- In information theory, **entropy** measures the amount of uncertainty or surprise in a message or data set. The more unpredictable or random the information, the higher its entropy
- The **more certain** or the **more deterministic** an event is, the **less information** it will contain
- **Uncertainty** or **impurity** is represented as the log to base 2 of the probability of a category (p_i)
- The index (i) refers to the number of possible categories
- Here, i = 2 as our problem is a binary classification

$$\text{entropy (p)} = -\sum_{i=1}^{N} p_i log_2 p_i$$

- The **heterogeneity** or **impurity function** is H(X) = $-\sum(p_i\ log_2\ p_i)$
- Imagine flipping a coin: heads and tails are equally likely, so the entropy is high because you are unsure of the outcome. Conversely, knowing a message in advance reduces its entropy to zero, as there is no surprise left
- Information entropy plays a crucial role in data compression, communication channels, and even understanding the limitations of information processing in general.

Measuring the Predictive Power of a Generative Topic Model (Cont.)

Computing the entropy index with Python (Example)

```
import math
from os import path
d = path.dirname("/content/file.txt")
# Read the whole text.
text1 = open(path.join(d, '/content/file_clean.txt')).read()
def Entropy(text1,base = 2.0):
    #make set with all unrepeatable symbols from string
    dct = dict.fromkeys(list(text1))
    #calculate frequencies
    pkvec =  [float(text1.count(c)) / len(text1) for c in dct]
    #calculate Entropy
    H = -sum([pk  * math.log(pk) / math.log(base) for pk in pkvec ])
    return H
print('Entropy Index: ', Entropy(text1))
```

Out: Entropy Index: 4.18 [bits]

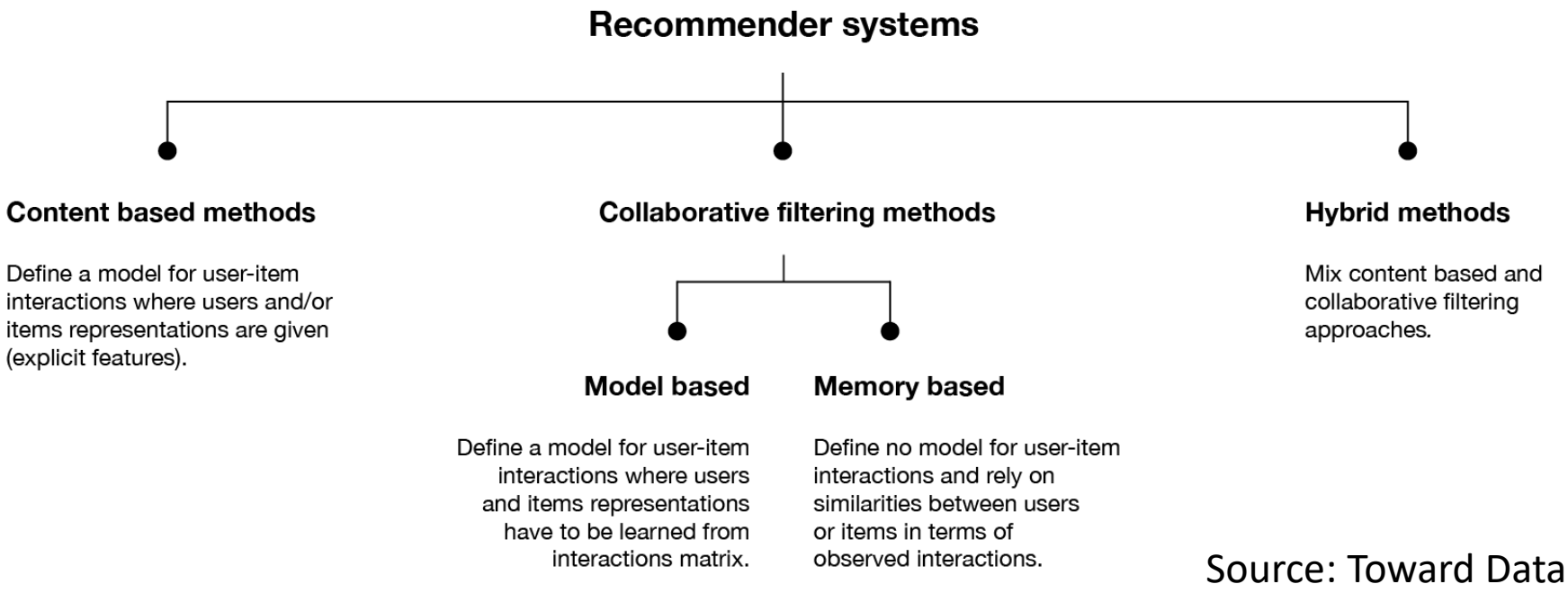
UMBC

12. RECOMMENDER SYSTEMS

Recommender Systems

- **Recommendation systems** are algorithms aimed to suggest relevant items to users based on collected history or observed behavior such as movies to watch, text to read, or products to buy
- There are two paradigms: **collaborative** and **content-based methods**
  - **Collaborative methods** for recommender systems are methods based on past interactions recorded between users and items to generate new recommendations. These interactions are stored in the so-called "user-item interactions matrix." The past user-item interactions represent the bases to detect similar users and/or similar items and to make predictions based on estimated proximities
    - The class of **collaborative filtering algorithms** is divided into two sub-categories called **memory-based** and **model-based** approaches:
      - **Memory-based** approaches directly work with values of recorded interactions, assuming no model, and are essentially based on nearest-neighbors search or KNN (i.e., find the closest users from a referenced user and suggest the most popular items among these neighbors)
      - **Model-based** approaches assume there is an underlying "generative" model that explains the user-item interactions and tries to identify it in order to make new predictions
  - **Content-based** approaches, unlike collaborative methods that only rely on **the user-item interactions**, use additional information about users and/or items
    - The idea behind **content-based** methods is to try to build a model relying on the available 'features' that may explain the observed user-item interactions
    - In this framework, we look at the profile of a specific user (age, sex, income, location, etc.…) and based on collected information, we compare it to others to determine relevant products

Summary: Types of Recommendation Systems



Source: Toward Data Science

Besides the hybrid system, there is also a **knowledge-based recommender system**, which uses explicit knowledge about items and users' needs to make recommendations. It requires knowledge of a domain and users' contextual information

Recommender Systems: Singular Value Decomposition

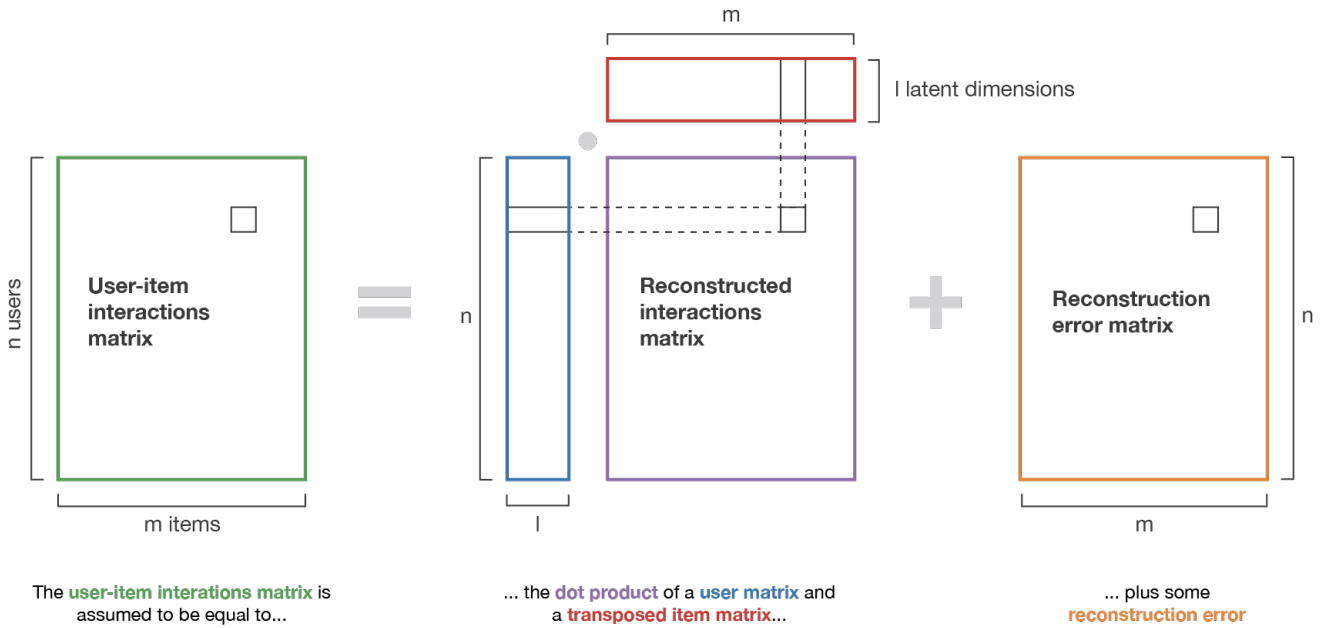
- Finding and recommending many suitable items that would be liked and selected by users is always a challenge
- There are many techniques used for this task and **Singular Value Decomposition** is one of those techniques
- The primary application of recommender systems is finding a relationship between user and products in order to maximize the user-product engagement
- In the context of the recommender system, the SVD is used as a collaborative filtering technique
- The **Singular Value Decomposition (SVD)** is a method from linear algebra
  - It has been generally used as a dimensionality reduction technique in machine learning. SVD is a matrix factorization technique
  - It reduces the number of features of a dataset by reducing the space dimension from N-dimension to K-dimension (where K<N)
  - It uses a matrix structure where each row represents a user, and each column represents an item
  - The elements of this matrix are the ratings that are given to items by users
  - The factorization of this matrix is done by the singular value decomposition
  - It finds factors of matrices from the factorization of a high-level (user-item-rating) matrix
  - The singular value decomposition is a method of decomposing a matrix into three other matrices as given as **A = U*S*V^T** where *A* is an *m x n* **utility matrix**, *U* is an *m x r* orthogonal left singular matrix, which represents the **relationship between users and latent factors**, *S* is an *r x r* diagonal matrix, which describes the **strength of each latent factor** and *V* is an *r x n* diagonal right singular matrix, which indicates the **similarity between items and latent factors**

Advantages and Disadvantages of Both Types of Recommendation Systems

	Advantages	Disadvantages
Collaborative	• Requires no information about users or items • Increased interactions generate better recommendations • No latent model assumed • Low bias	• Difficult to recommend anything to new users (no experience) • Too few interactions may not provide reliable info on behavior patterns • Complexity increases with data volume • High variance
Content-Based	• Way to bypass 'cold start' or insufficient records of past behaviors • Makes association of users with identified classes with similar features • Latent interactions assumed • Low variance	• Model trained to reconstruct user-item interactions values from its own representation of users and items • Assumed user-item interactions are dynamic • High bias
Knowledge-Based	• Can provide more controlled and explainable recommendations • Can handle complex constraints and relationships	• Require extensive knowledge engineering and maintenance • May not be as adaptable to changing user preferences or item availability

Examples of Strategies for Recommendations

Collaborative Approach (NMF)



Content-Based Approach (Naïve Bayes)

User described by some features

(features can be of various kind and define the inputs of the model)

Bayesian classifier for a given item

(parameters of the bayesian classifier are specific to the item and learned on past item interactions)

Predicted class ("like" or "dislike")

(output of the bayesian classifier model when inputs are the features of the user)

Source: Toward Data Science

Recommender Systems: Singular Value Decomposition (Continued)

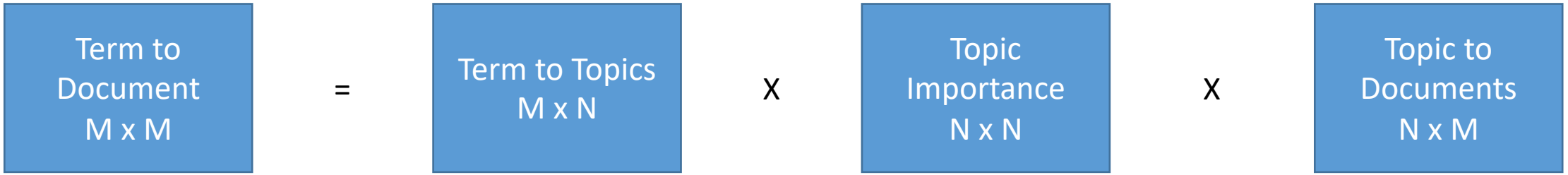
- The latent factors here are the characteristics of the items, for example, the genre of music
- The **SVD** decreases the dimension of the utility matrix *A* by extracting its latent factors
- It maps each user and each item into a *r*-dimensional latent space
- This mapping facilitates a clear representation of relationships between users and items
- Let each item be represented by a vector x_i , each user be represented by a vector y_u, and the expected rating by a user on an item be defined as $Y_{ui} = x_i^T * y_u$
  - Y_{ui} is a form of factorization in singular value decomposition
  - The x_i and y_u variables can be obtained in a manner that the square error difference between their dot product and the expected rating in the user-item matrix is a minimum value
- To let the model generalize well and not overfit the training data, a regularization term is added such that

$$Min(x,\ y) \sum_{(u,\ i) \in K} (r_{ui} - x_i^T . y_u)^2 + \lambda(||x_i||^2 + ||y_u||^2)$$

- To reduce the error between the value predicted by the model and the actual value, the algorithm uses a **bias term**. For a user-item pair (u, i), μ is the average rating of all items, b_i is the average rating of item i minus μ and b_u is the average rating given by user *u* minus *μ*, the final equation after adding the regularization term and bias can be given as

$$Min(x,\ y,\ b_i,\ b_u) \sum_{(u,\ i) \in K} (r_{ui} - x_i^T . y_u - \mu - b_i - b_u)^2 + \lambda(||x_i||^2 + ||y_u||^2 + b_i^2 + b_u^2)$$

Recommender Systems: Singular Value Decomposition (Continued)



- SVD separates matrices based on the formula M = U Σ VT

- **U** captures the "rotation" or "orientation" of the original data in a new coordinate system
- **Σ** captures the "scaling" or "importance" of different dimensions in this new coordinate system
- $\mathbf{V}^{\mathbf{T}}$ captures the "correlations" or "relationships" between the original variables
- Works for any real or complex matrix
- Reveals latent structures and relationships in data
- Enables dimensionality reduction and compression
- SVD helps discover hidden patterns and relationships in user-item interactions to make personalized recommendations

UMBC

Recommender Systems: Summary

- **Recommendation algorithms** can be divided into two main categories:
- **Collaborative approaches** (such as user-user, item-item, and matrix factorization) that are only based on user-item interaction matrix. They include
    - **Memory-based collaborative methods** do not assume any **latent model** and have **low bias** but **high variance**
    - **Model-based collaborative approaches** assume **latent interactions.** They need to learn from both users and items from scratch. They have a **higher bias but lower variance**
- **Content-based approaches** (such as regression or classification models) use prior information about users and/or items
    - **Content-based methods** assume a **latent model** built around explicitly given **users and/or items features**. They have the **highest bias and lowest variance**
- **Knowledge-based models** rely on the knowledge of a domain**. Hybrid models** combine the benefits of **collaborative** and **content-based models**
- **Scale considerations** must be considered when designing the system (i.e., better use of sparsity, iterative methods for factorization or optimization, approximate techniques for nearest neighbor search)

- **Recommender systems** are **difficult to evaluate**:
    - It is difficult to apply metrics such as MSE, accuracy, recall, or precision
- **Sampling and testing** may be the best way to evaluate models and make recommendations until there is enough information to improve model accuracy

13. CHATBOTS AND SPEECH-TO-TEXT

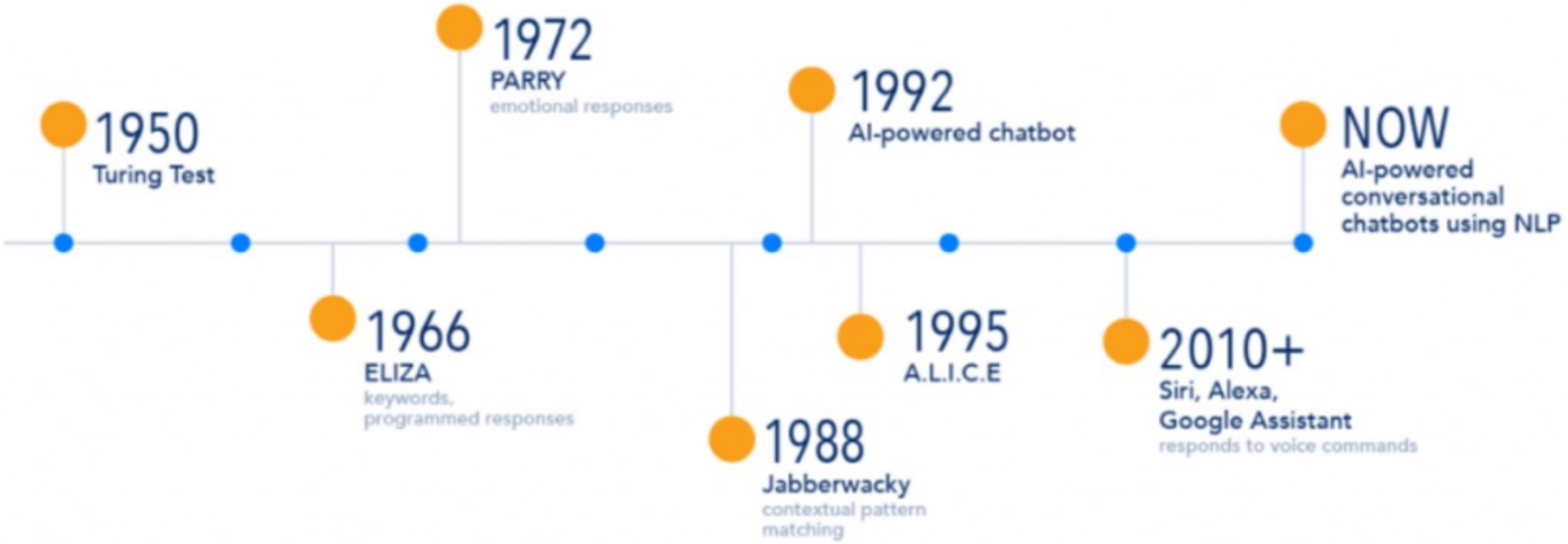
UMBC

Chatbots

- A **chatbot** is a program that communicates with a user
  - It simulates and processes human conversations (either spoken or written)
- **Chatbots** are not new. They were created in the 1960's
  - In 1966, Weizenbaum invented a computer program called **ELIZA** which imitated the language of a psychotherapist from only 200 lines of code
  - The first move away from text chatbots occurred in 1988 when Rollo Carpenter started the **Jabberwacky** project
- They are useful when we do not want to recreate the same task/process by going through the same set of procedures over again
- They are trained to provide answers to specific questions and interact with users (**Siri** and **Alexa**)
- One of the biggest successes ultimately came from the technology within smartphones
- In 2010, Apple launched **Siri** for iOS
  - **Siri** was the first multi-functional, voice-commanded bot
  - It paved the way for intelligent personal home assistants, such as Amazon's **Alexa and Google**'s Assistant
- Chatbots are difficult to create especially when they try to interpret natural language
- The latest innovation is **ChatGPT** launched by **OpenAI** in November 2022

UMBC

A Chatbot Timeline



Source: Siemens

Chatbots (Cont.)

- There are six types of **chatbots**:
  - **Rule-based**: They respond to very specific commands (for instance, weather forecast at a selected location)
  - **Machine-Learning-based**: They try to understand the sentiment and meaning of the language used and not to rely on predetermined command
  - **Text-based**: A bot answers the user's questions via text interface
  - **Voice-based**: A bot answers the user's questions via a human voice interface
  - **Retrieval-based:** They provide an answer from a predefined set of answers
  - **Generative-based:** They are most often based on basic probabilistic and machine learning models
    - They need to be trained to generate new content
    - **Markov chains** have originally been used for the task of **text generation**
    - **Sequential models such as RNN, LSTM, and GRU** are often used for chatbots

- There are two types of **chatbots**:
  - **Task-Oriented**: focused on specific tasks, uses rule-based scenarios
  - **Data-Driven and Predictive**: they are virtual assistants, interactives and personalized, and apply predictive intelligence

The Fundamental NLP Tools to Create a Chatbot

- **POS Tagging** are important to assign parts of speech to each word or token such as nouns, verbs, etc.
  - POS are important to identify some entity in a sentence
  - They reduce the complexity of understanding a text that cannot be trained or is trained with less confidence
  - We can identify parts of the text input and do string matching only for those parts
- **Tokenization** is the splitting of text into meaningful segments
- **Stemming** is the process of reducing inflected words to their word stem (base form)
- **Lemmatization** is the algorithmic process of determining the lemma of a word based on its intended meaning
- **Named-Entity Recognition** is the process of finding and classifying name entities existing in the given text into pre-defined categories. spaCy is a tool that enables fast entity recognition
- **Stop Words** are high frequency words that need to be filtered out of a document before processing
- **Dependency Parsing,** especially with spaCy, can be used for sentence boundary detection and makes it possible to iterate over base noun phrases or 'chunks'
  - We also look for **ancestors** in dependency parsing (the rightmost token of a token's syntactic descendant) as well as **children** (immediate syntactic dependent of the token)
- **Noun Chunks** are base-noun phrases
- Finding **similarity** between sets of sentences

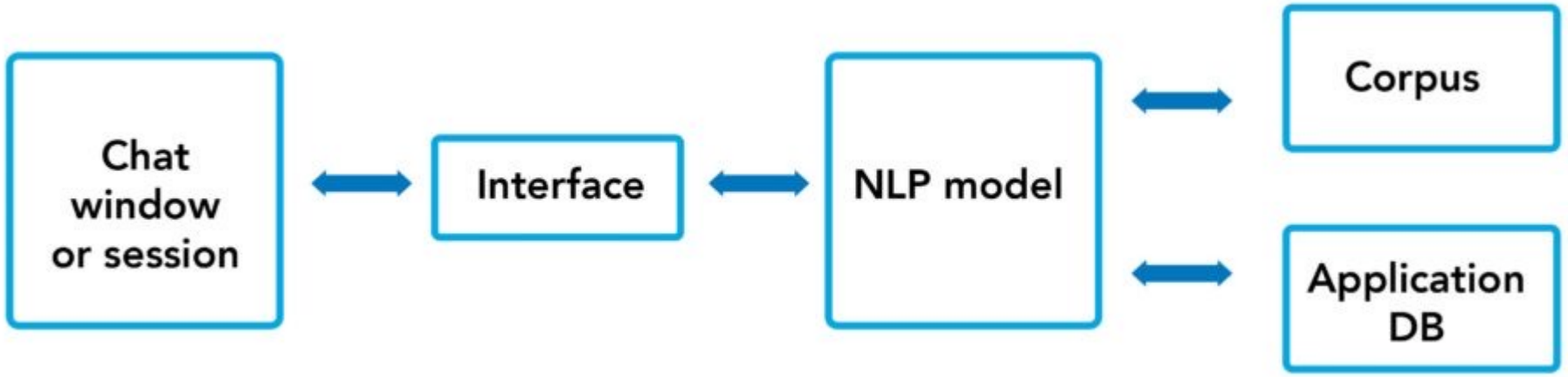
Chatbots (Cont.)

The Components of a Chatbot

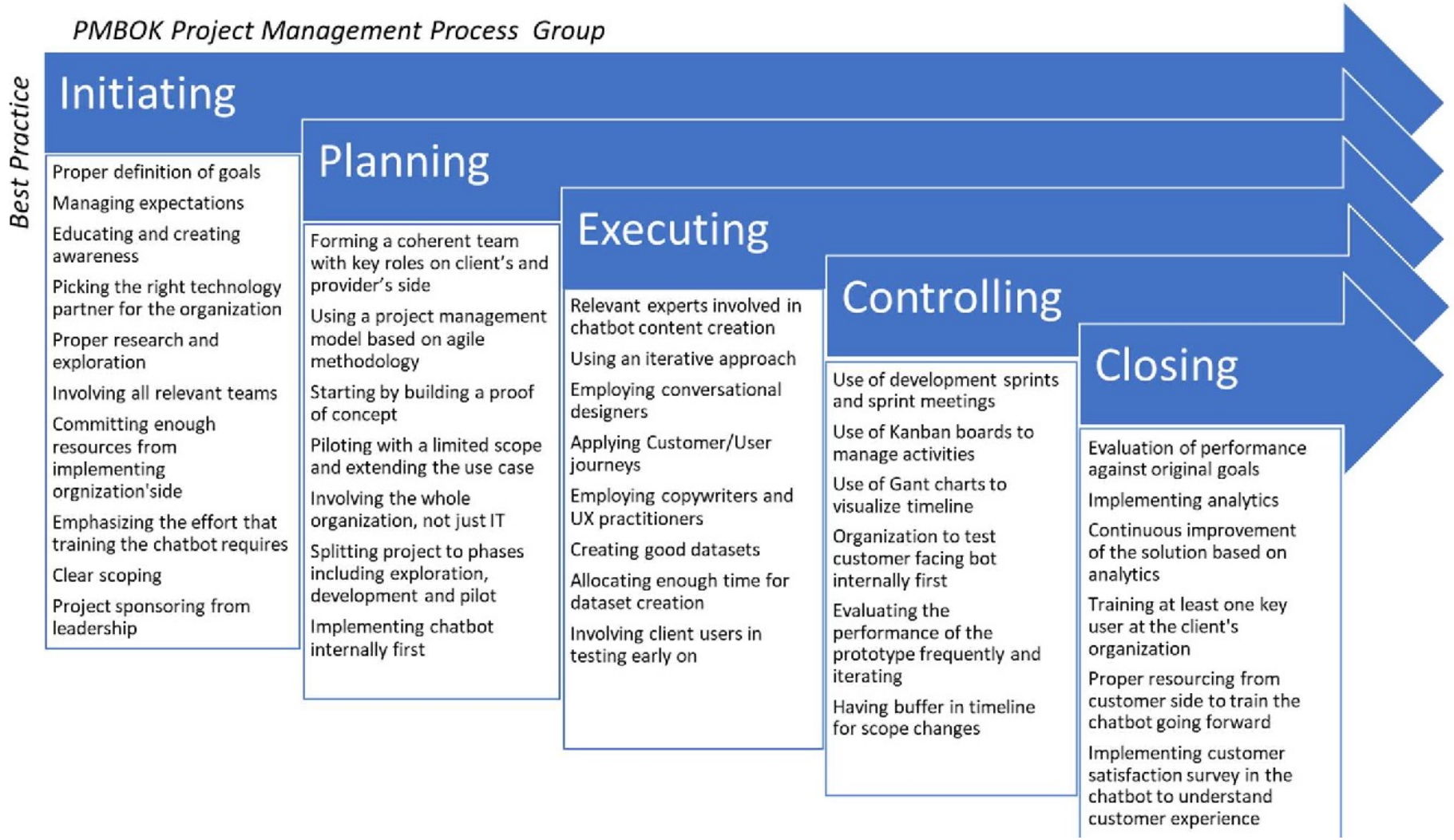
- **Agent:** It is a Natural Language Understanding module that is trained to answer questions. It handles conversations and routes necessary actions
- **Intents**: They refer to the reasons for communicating, fulfilling commands and performing tasks. They have elements such as context, training, phase, actions and parameters, and responses
- **Entities**: Intents have metadata called entities. For instance, in the sentence 'book a movie ticket,' 'book a ticket' is the intent and 'movie' is the entity
- **Utterances** are different forms of asking the same question and expressing the same intents
- **Training the bot** refers to build a model that will learn from the existing set of defined intents/entities and utterances
- **Confidence Score**: Every time a user tries to find what intents an utterance may belong to, the computer will come up with a confidence score
- **Fulfillment** is a connecting service that allows an agent to take actions based on the end user's expression and send dynamic responses back to the user
- **Wake words** are words that wake up chatbots to get ready to receive commands. 'Alexa' or 'Hey Siri' are examples
- **Launch words** are words that launch an action such as 'order'
- **Error planning and default cases** are elements to allow chatbots to handle unforeseen conditions
- A **webhook** is an HTTP push API or web callback. It sends data from the application to the application consumers as soon as the event occurs

Chatbot Architecture

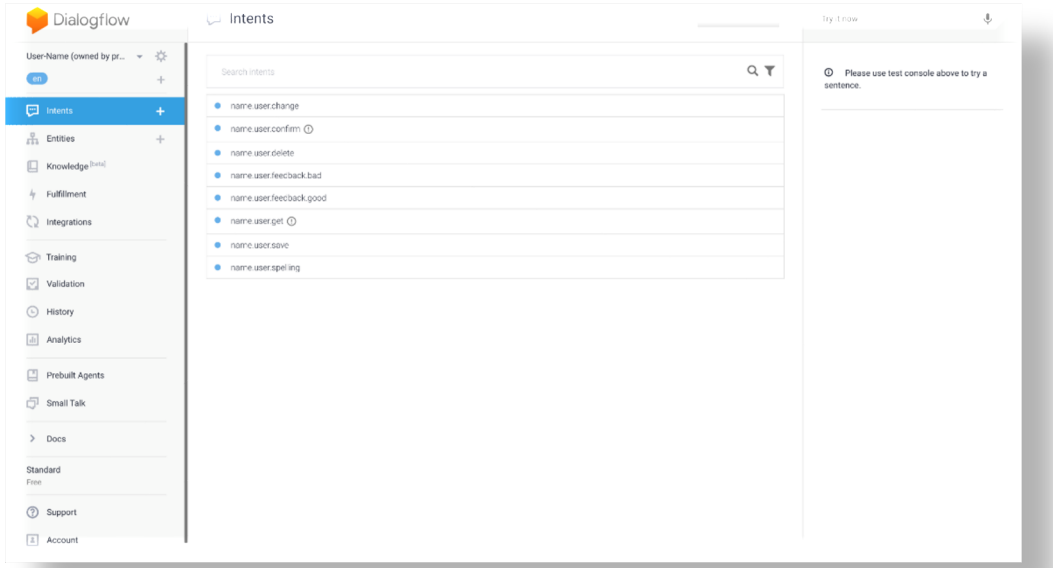
- Typical **chatbot architecture** should consist of the following items:
  - **Chat window** (Communication User Interface or CUI)/session/or front-end application interface
  - **Deep learning model** for Natural Language Processing
  - **Corpus** or data to train the NLP model
  - **Application database** for processing actions to be performed by the chatbot
- Some of the best platforms include Chatfuel, Wotnot, DialogFlow, LivePerson, among many others

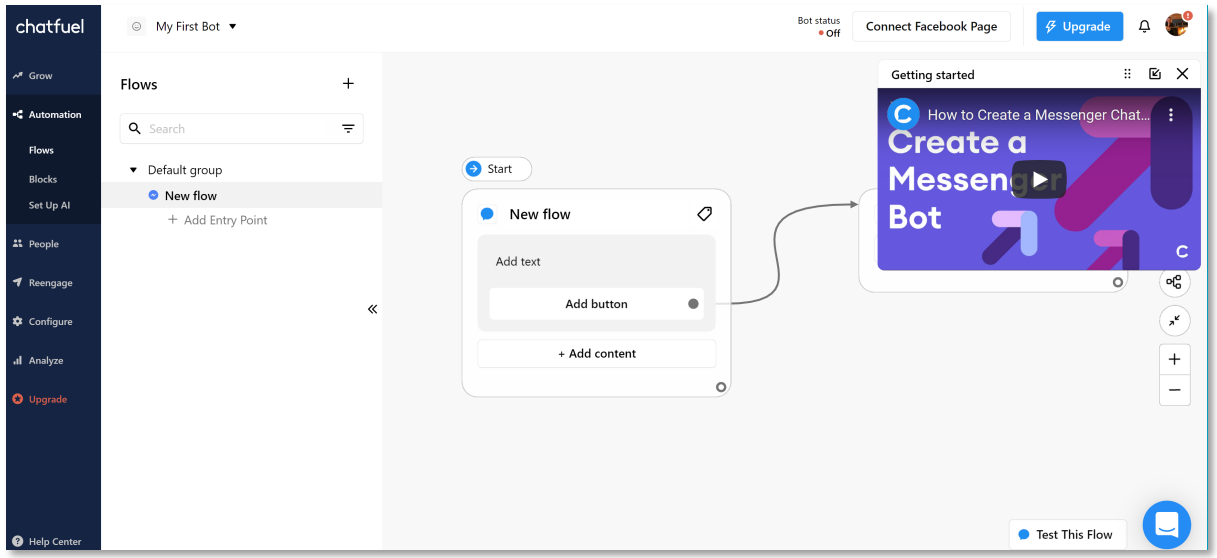


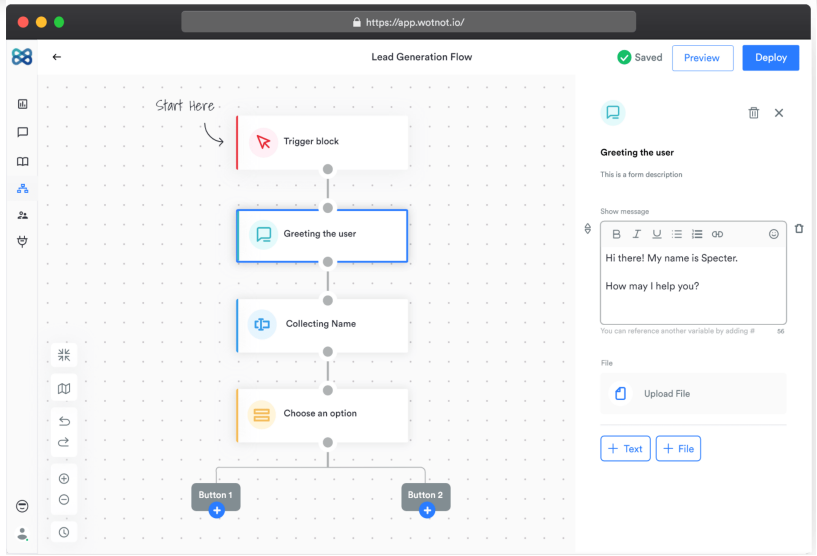
Project Management Best Practice Framework for Chatbot Implementation

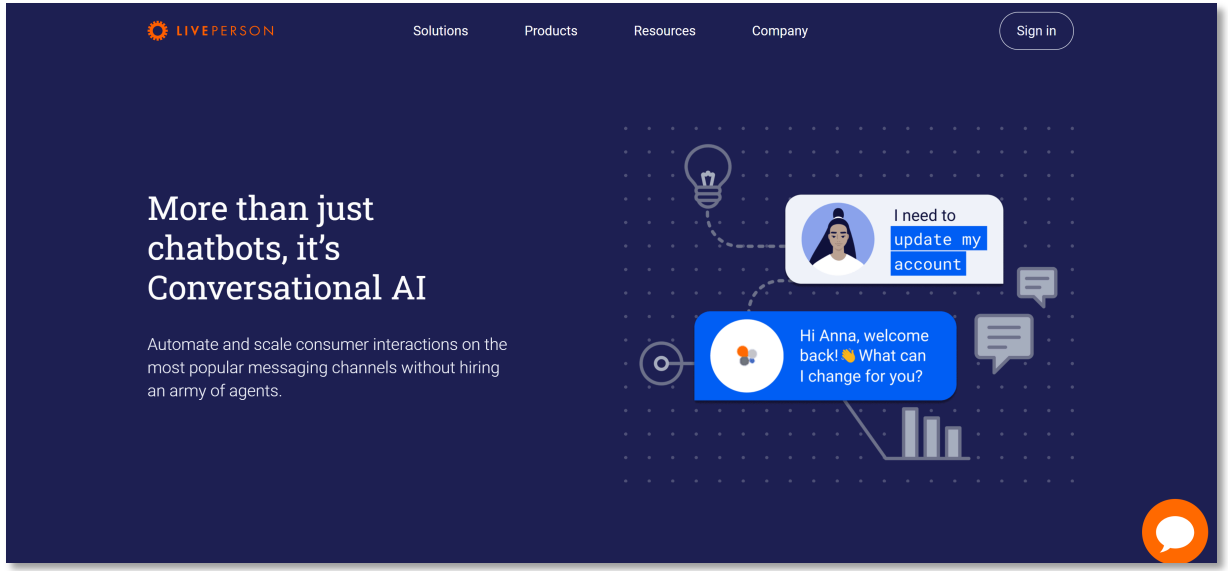


Chatbots: Creating Chatbots with Selected Platforms



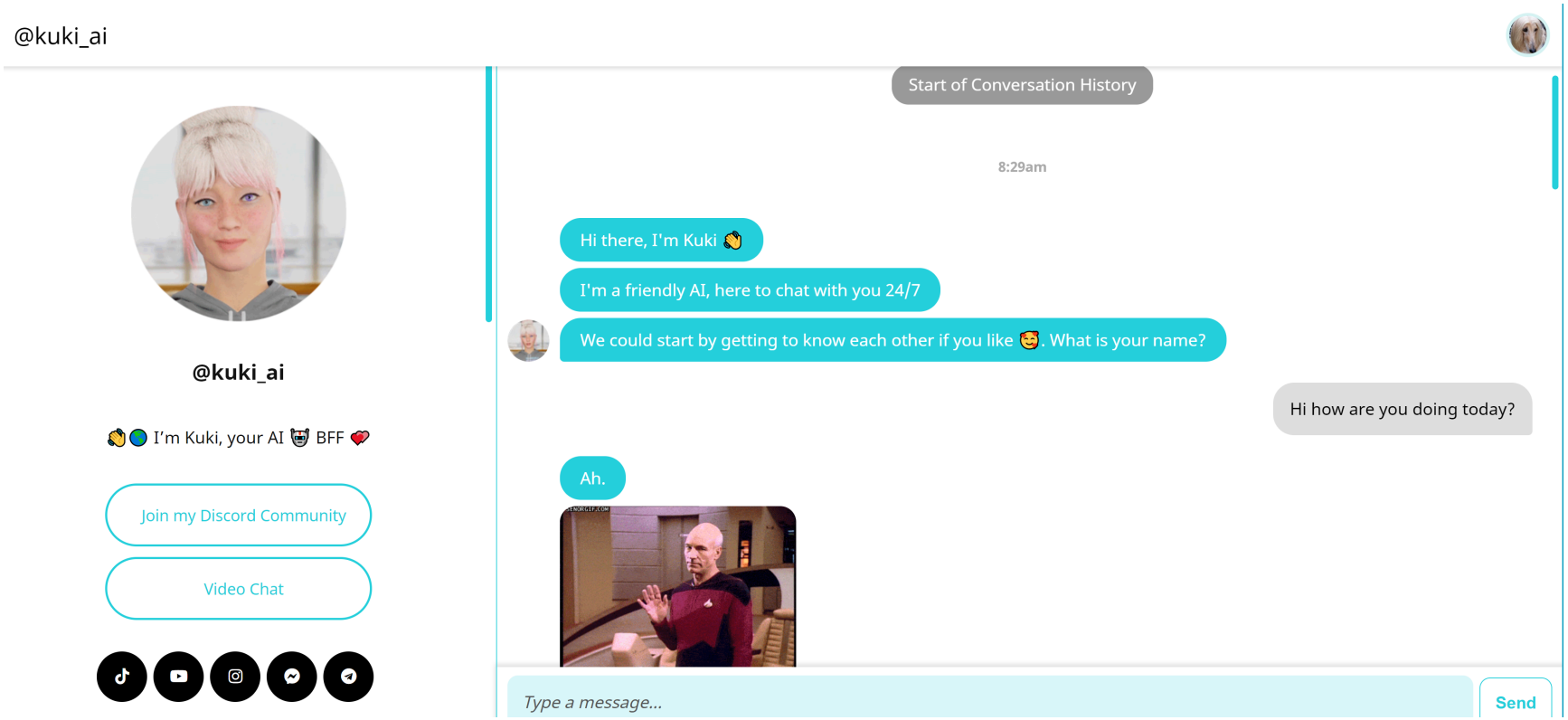






World's Best Chatbot

- The world's best chatbot is Kuki, a record-breaking human-like chatbot and five-time winner of the Loebner Prize Turing Test
- The website is https://chat.kuki.ai/chat

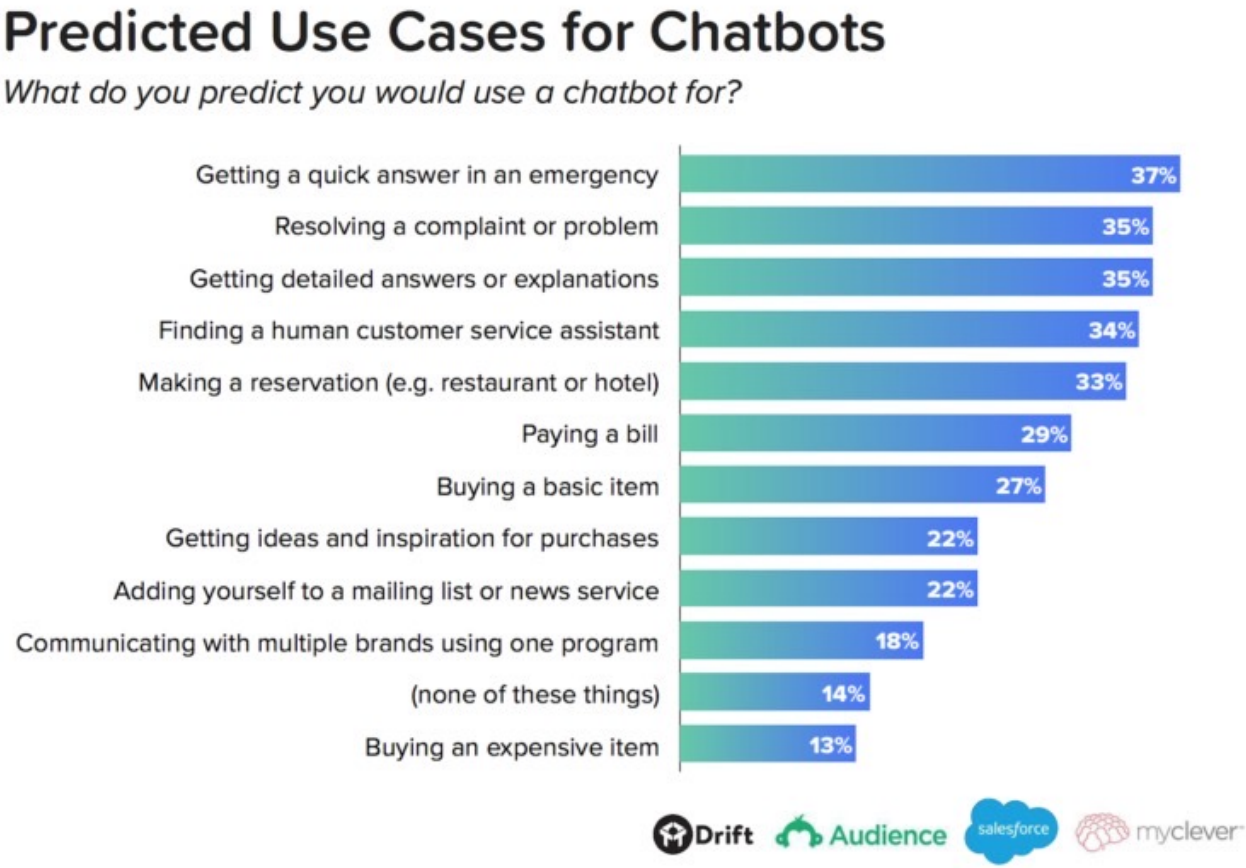


Chatbots (Cont.)

- Here are some of the major platforms for creating **chatbots**:
  - https://woebot.io
  - https://dialogflow.com
  - https://rasa.com
  - https://wit.ai
  - https://luis.ai
  - https://www.liveperson.com/
  - https://chatfuel.com
- Here is the link to the inventory of all the **chatbots** in the world:
  - https://www.chatbots.org/chatbot/
- The best chatbots in 2022:
  - https://manychat.com/blog/chatbot-examples/

The Future of Chatbots

- Chatbots can improve customer service especially if they address some of the problems of online experience:



The Future of Chatbots: Toward ChatGPT

Source: Slide Team

The Future of Chatbots: ChatGPT

- **ChatGPT** is a chatbot fine-tuned from a **GPT-3.5** series model. It finished training in early 2022. ChatGPT 4.0 is available at a price
- **ChatGPT** is trained to follow an instruction in a prompt and provide a detailed response
- **OpenAI** trained this model using **Reinforcement Learning from Human Feedback (RLHF)**, using the same methods as **InstructGPT**, but with slight differences in the data collection setup
- **OpenAI** trained an initial model using supervised fine-tuning:
  - Human AI trainers provided conversations in which they played both sides—the user and an AI assistant
  - The trainers got access to model-written suggestions to help them formulate their responses
  - **OpenAI** mixed this new dialogue dataset with the **InstructGPT** dataset, which was transformed into a dialogue format
- To create a reward model for reinforcement learning, **OpenAI** collected comparison data, which consisted of two or more model responses ranked by quality. This was based on conversations that AI trainers had with the chatbot
- **OpenAI** randomly selected a model-written message, sampled several alternative completions, and had AI trainers rank them
- Using the reward models, **OpenAI** fine-tuned the model using **Proximal Policy Optimization** and performed several iterations of this process

14. MACHINE TRANSLATON

Machine Translation

- **Machine translation** is high in demand in a global context
- You can use Google Translate or you can create your own API with Huggingface designed by researchers at the University of Helsinki, Finland
- **Machine translation** (**MT**) refers to fully automated software that can translate source content into target languages
- Humans may use **MT** to help them translate text and speech into another language without human intervention
- MT systems require "training" in the desired domain and language pair to increase quality

Types of Machine Translation

- **Generic MT** usually refers to platforms such as Google Translate, Bing, Yandex, and Naver. These platforms provide MT for ad hoc translations to millions of people. Companies can buy generic MT for batch pre-translation and connect to their own systems via API
- **Customizable MT** refers to **MT** software that has a basic component and can be trained to improve terminology accuracy in a chosen domain (medical, legal, IP, or a company's own preferred terminology). For example, WIPO's specialist MT engine translates patents more accurately than generalist MT engines, and eBay's solution can understand and render into other languages hundreds of abbreviations used in electronic commerce
- **Adaptive MT** offers suggestions to translators as they type in their CAT-tool and learns from their input continuously in real time. Introduced by Lilt in 2016 and by SDL in 2017, adaptive **MT** is believed to improve translator productivity significantly and can challenge translation memory technology in the future

Machine Translation (Cont.)

- To process any translation, human or automated, the meaning of a text in the original (source) language must be fully restored in the target language, i.e., the translation
- Translation is not a mere word-for-word substitution
  - A translator must interpret and analyze all the elements in the text and know how each word may influence another
  - This requires extensive expertise in grammar, syntax (sentence structure), semantics (meanings), etc., in the source and target languages, as well as familiarity with each local region

Machine Translation Approaches

There are three main approaches to machine translation:

- First-generation **rule-based (RbMT)** systems rely on countless algorithms based on the grammar, syntax, and phraseology of a language
- **Statistical systems (SMT)** arrived with search and big data. With lots of parallel texts becoming available, SMT developers learned to pattern-match reference texts to find translations that are statistically most likely to be suitable. These systems train faster than RbMT, provided there is enough existing language material to reference
- **Neural MT (NMT)** uses machine learning technology to teach software how to produce the best result This process consumes large amounts of processing power, and this explains why it relies on GPU
  - NMT started gaining visibility in 2016
  - Many MT providers are now switching to this technology

A combination of two different MT methods is called **Hybrid MT**

Rule-Based Machine Translation Technology

- **Rule-based machine translation** relies on countless built-in linguistic rules and millions of bilingual dictionaries for each language pair
- The software parses text and creates a transitional representation from which the text in the target language is generated. This process requires extensive **lexicons** with morphological, syntactic, and semantic information, and large **sets of rules**. The software uses these **complex rule sets** and then **transfers** the **grammatical structure** of the source language into the **target language**
- Translations are built on gigantic dictionaries and sophisticated linguistic rules. Users can improve the out-of-the-box translation quality by adding their terminology into the translation process. They create **user-defined dictionaries** which override the system's default settings
- In most cases, there are two steps:
  - An initial investment that significantly increases the quality at a limited cost, and
  - An ongoing investment to increase quality incrementally
- While rule-based MT brings companies to the quality threshold and beyond, the quality improvement process may be long and expensive

Rule-Based MT (Cont.)

Rule-Based MT (SYSTRAN, Apertium, GramTrans)

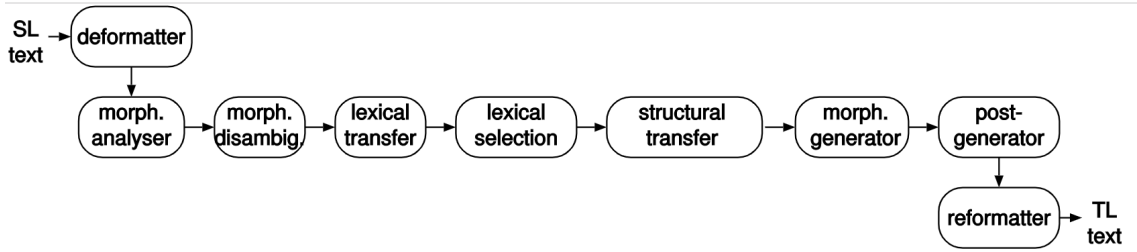
- **Rule-based MT** provides good out-of-domain quality and is by nature predictable
  - Dictionary-based customization guarantees improved quality and compliance with corporate terminology.

Advantages

- No bilingual text required
- Domain-independent
- Total control (a possible new rule for every situation)
- Reusability (existing rules of languages can be transferred when paired with new languages)

Disadvantages

- Translation results may lack the fluency readers expect
- In terms of investment, the customization cycle needed to reach the quality threshold can be long and costly
- The performance is high even on standard hardware
- Requires good dictionaries
- Manually set rules (requires expertise)
- The more the rules the harder to deal with the system



Statistical Machine Translation

- **Statistical machine translation** utilizes statistical translation models whose parameters stem from the analysis of monolingual and bilingual corpora
- Building statistical translation models is a quick process, but the technology relies heavily on existing multilingual corpora
- A minimum of 2 million words for a specific domain and even more for general language are required
- Theoretically it is possible to reach the quality threshold, but most companies do not have such large amounts of existing multilingual corpora to build the necessary translation models
- Additionally, statistical machine translation is CPU-intensive and requires an extensive hardware configuration to run translation models for average performance levels

Statistical MT (Cont.)

Statistical MT (SMT) provides good quality when large and qualified corpora are available

- The translation is fluent, meaning it reads well and therefore meets user expectations
- However, the translation is neither predictable nor consistent
- Training from good corpora is automated and cheaper
- But training on general language corpora, meaning text other than the specified domain, is poor
- Furthermore, statistical **MT** requires significant hardware to build and manage large translation models
- An **SMT** requires three steps:
  - A **Language Model** (what is the correct word given its context?)
  - A **Translation Model** (what is the best translation of a given word?)
  - A method to find the right order of words

Advantages

- Less manual work from linguistic experts
- One **SMT** suitable for more language pairs
- Less out-of-dictionary translation: with the right language model, the translation is more fluent

Disadvantages

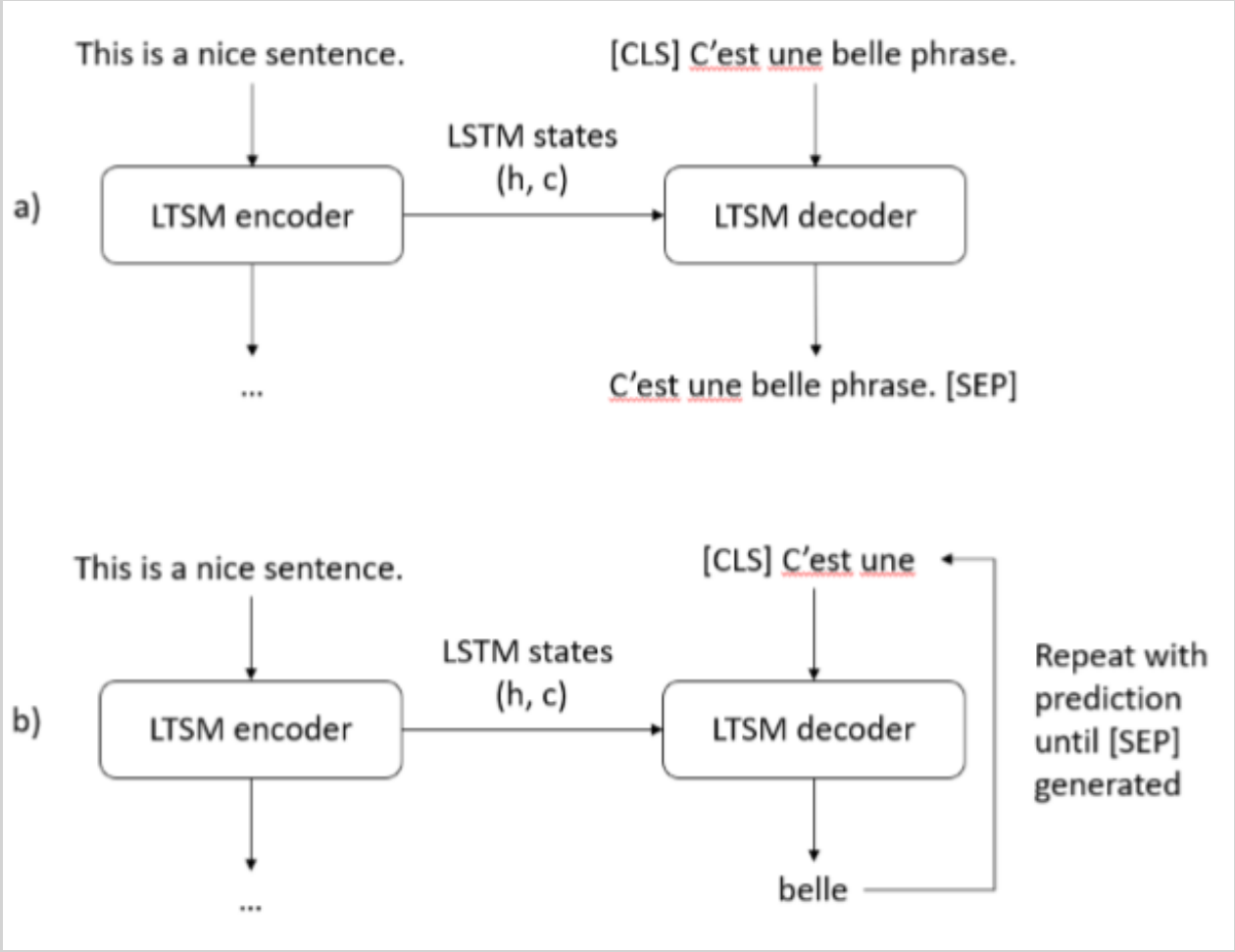
- Requires bilingual corpus
- Specific errors are hard to fix
- Less suitable for language pairs with big differences in word order

- **Google Translate** (between 2006 and 2016, when they announced to change to **NMT**)
- **Microsoft Translator** (in 2016 changed to **NMT**)
- **Moses**: Open-source toolkit for statistical machine translation

UMBC

Neural MT (NMT)

- **NMT**s can be built with one network instead of a pipeline of separate tasks
- In 2014, sequence-to-sequence models were introduced opening new possibilities for neural networks in NLP
- Before the **seq2seq** models, the neural networks needed a way to transform the sequence input into computer-ready numbers (one-hot encoding, embeddings)
- With **seq2seq**, the possibility of training a network with input and output sequences became possible



UMBC

Neural MT (Cont.)

- The **NMT** emerged quickly as a choice
- After a few years of research, these models outperformed the **SMT**s
- Many translator provider companies changed their networks to neural-based models including Google and Microsoft
- If the training data are unbalanced, the model cannot learn from rare and more frequent ones
- In the case of languages, it is a common problem as there are many rare words used only a few times
- To train a model that is not biased towards frequent words can be challenging
- Facebook researchers introduced an unsupervised MT model working with both **SMT** and **NMT** that requires only large monolingual corpora and not a bilingual one
- The main bottleneck of the previous examples was the lack of large database with translations to train on. This model shows promise to resolve this issue

  Advantages
  - End-to-end models (no pipeline of specific tasks)

  Disadvantages
  - Requires bilingual corpus
  - Rare word problem
- **Main users include:**
  - **Google Translate** (from 2016)
  - **Microsoft Translate** (from 2016)
  - **FacebookOpenNMT**: An open-source neural machine translation system

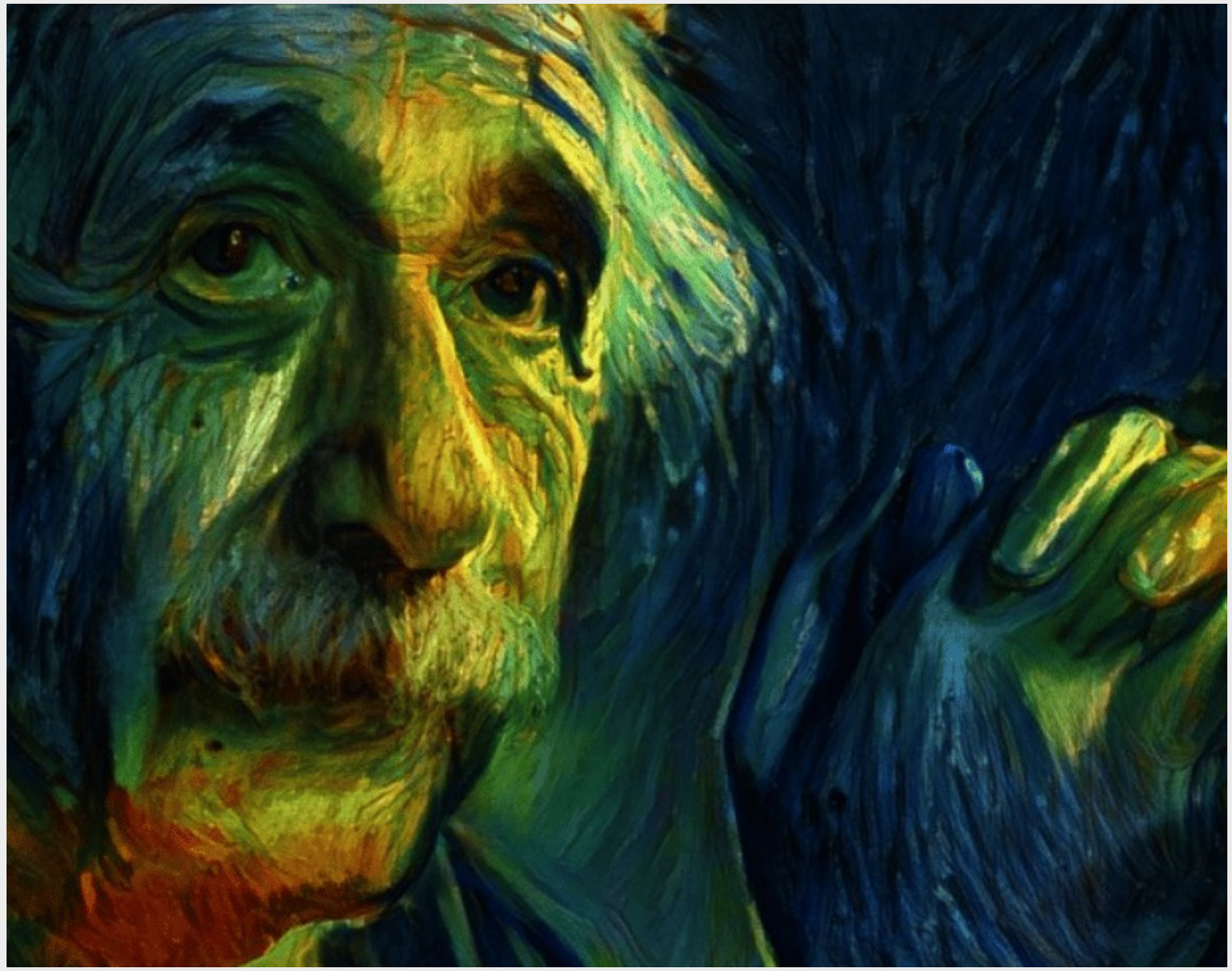
Machine Translation (Cont.)

Availability: API, Cloud, Server, Desktop

- Google, Microsoft, IBM, Amazon, Yandex, and many others run **MT** software on their own infrastructure and can provide it as a Cloud API service, priced per symbol. For example, it costs $20 to translate 1 million characters with Google Translate
- In contrast, developers of customizable **MT**, including Systran and Promt, offer server and desktop products priced per license
- In professional translations, **MT** is most often integrated into the CAT-tool
- The human linguist can pick a suggestion from **MT** as they go through the text, if they do not find a better match in the translation memory

Evaluating MT Quality

- Translation companies and departments typically evaluate MT quality by the effort it takes for a human to post-edit the output. It is often measured in pages per hour, or in the number of keystrokes per segment
- Specialists' training **MT** engines rely on automated tests and metrics. They are better suited for A/B or Control/Variation testing and experimentation. They can show the impact of the tiniest changes where humans might not notice the difference
- The mainstay metric for auto-testing is called **BLEU**
  - '**Bilingual evaluation understudy (BLEU)**' shows how closely **MT** translation corresponds to human translation of the same text
  - It compares parallel translations and produces a score between 0 (worst) and 1 (best)
  - While **BLEU** scores are widely used by **MT** researchers, they can be manipulated, and it takes a specialist to make sense of results

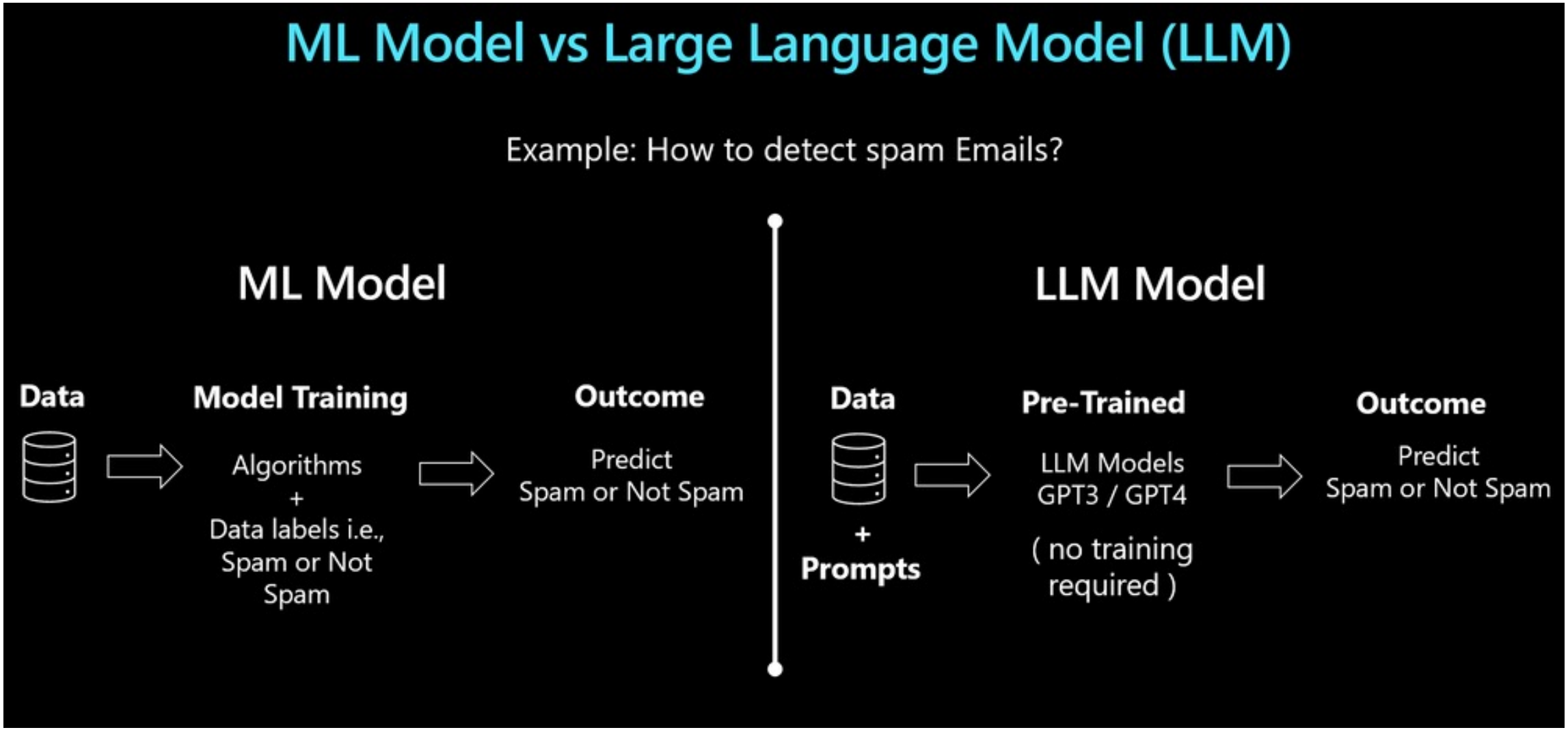


15. GENERATIVE AI and LLM

Large Language Models

- **LLMs** are a type of artificial intelligence (AI) that have been trained on massive amounts of text data
- There are two kinds of language modeling: **autoencoding** and **autoregression**
  - **Autoencoding models** ask a model to fill in missing words from any portion of a phrase from a known vocabulary. These models correspond to the autoencoder section of the transformer. They create a bidirectional representation of the sentence (**BERT models**)
  - **Autoregressive models** ask a model to generate the next most likely token of a given phrase from a know vocabulary. They are trained to predict the next tokens in the phrase (**GPT models**)
- **LLMs** can be either autoregressive, autoencoding, or both such as the **T5 models**
- **LLMs** use data to learn how to **generate text**, **translate languages**, write different kinds of **creative content**, and **answer questions** in an informative way
- They typically have billions of parameters—the variables the model uses to make predictions
- Every **LLM** has been pre-trained on a large corpus of text and on specific language modeling-related tasks
- **LLMs** use a type of **neural network** called a **transformer** to process text
- **Transformers** work by encoding the input text into a sequence of numbers and decoding those numbers to produce an output prediction
- **LLMs** are trained using a process called **supervised learning**, where they are given a set of input text and the corresponding output text

Large Language Models (Cont.)



Source: Microsoft

Large Language Models: How do they work?

- An **LLM** needs to be trained on a large volume -- sometimes referred to as a *corpus* -- of data that is typically petabytes in size
- The training can take multiple steps, usually starting with an **unsupervised learning approach**
- In that approach, the model is trained on **unstructured data** and **unlabeled data**
- The benefit of training on unlabeled data is that more data is often available. At this stage, the model begins to derive relationships between different words and concepts
- The next step for some **LLMs** is training and fine-tuning with a form of **self-supervised learning**
- Some data labeling has occurred, assisting the model to more accurately identify different concepts
- Next, the **LLM** undertakes deep learning as it goes through the **transformer neural network process**
- The **transformer model architecture** enables the **LLM** to understand and recognize the relationships and connections between words and concepts using a self-attention mechanism
- That mechanism can assign a score, commonly referred to as a weight, to a given item (called a token) to determine the relationship
- Once an **LLM** has been trained, a base exists on which the AI can be used for practical purposes
- By querying the **LLM** with a prompt, the AI model inference can generate a response, which could be an answer to a question, newly generated text, summarized text, or a sentiment analysis report

Large Language Models: What is Hallucination?

- Hallucinations happens when **LLMs** produce responses that do not accurately reflect the given context, are not supported by evidence, or deviate from the expected behavior based on their training data
- Here are some examples of hallucinations in **LLM**-generated outputs:

1. Factual Inaccuracies: The **LLM** produces a statement that is factually incorrect
2. Unsupported Claims: The **LLM** generates a response that has no basis in the input or context
3. Nonsensical Statements: The **LLM** produces a response that doesn't make sense or is unrelated to the context
4. Improbable Scenarios: The **LLM** generates a response that describes an implausible or highly unlikely event

Large Language Models: How to Measure Hallucination? (Cont.)

Language Modeling:

- **Perplexity**: Measures how well the model predicts the probability distribution of the given test data. Lower perplexity indicates a better language model
- **Cross-Entropy Loss**: Measures the average negative log-likelihood of the true probability distribution given the model's predicted probability distribution

Text Classification and Sentiment Analysis:

- **Accuracy**: The proportion of correctly classified instances out of the total instances.
- **Precision, Recall, and F1-score**: These metrics measure the trade-off between false positives and false negatives, and their harmonic mean, respectively
- **AUC-ROC (Area Under the Receiver Operating Characteristic Curve)**: Measures the trade-off between true positive rate and false positive rate at various classification thresholds

Large Language Models: How to Measure Hallucination? (Cont.)

Machine Translation:

- **BLEU (Bilingual Evaluation Understudy)**: Measures the similarity between the model-generated translations and reference translations by computing n-gram precision
- **METEOR (Metric for Evaluation of Translation with Explicit Ordering)**: Considers n-gram matches and alignments between the translation and reference, including synonyms and stemming
- **TER (Translation Edit Rate)**: Measures the number of edits (insertions, deletions, substitutions) required to transform the model-generated translation into the reference translation

Text Summarization:

- **ROUGE** (Recall-Oriented Understudy for Gisting Evaluation): A set of metrics (ROUGE-N, ROUGE-L, ROUGE-S) that measure the overlap of n-grams, longest common subsequences, and skip-bigrams between the generated summary and reference summaries

Large Language Models: How to Measure Hallucination? (Cont.)

Named Entity Recognition:

- **Precision, Recall, and F1-score**: These metrics are used to evaluate named entity recognition tasks, considering exact matches of entity boundaries and entity types

Question Answering:

- **F1-score**: The harmonic mean of precision and recall, considering exact token matches between the model-generated answer and the reference answer.
- **EM (Exact Match)**: A binary metric that measures whether the model-generated answer exactly matches the reference answer

- **LLMs** can be used for a variety of tasks, including:
  - Generating text, like poems, code, scripts, musical pieces, email, letters, etc.
  - Translating languages
  - Writing different kinds of creative content
  - Answering questions in an informative way
  - Summarizing documents
  - Chatting with humans in a natural way

Large Language Models: Challenges

- **LLMs** can be biased, as they are trained on data that may reflect human biases
- **LLMs** can be easily fooled, as they can be tricked into generating text that is factually incorrect or offensive
- **LLMs** can be expensive to train and run, as they require a lot of computing power
- **LLMs** generally require large quantities of expensive graphics processing unit hardware and massive data sets, which makes development costly
- **LLMs** are prone to **hallucination**. AI hallucination occurs when an LLM provides an inaccurate response that is not based on trained data
- **LLMs** are prone to **glitch tokens**—maliciously designed prompts that cause an LLM to malfunction, an emerging trend since 2022

Examples of **LLMs**

- Some of the most well-known LLMs include GPT-3, Jurassic-1 Jumbo, LaMDA, and Megatron-Turing NLG

Large Language Models: Types

Among the common types are the following:

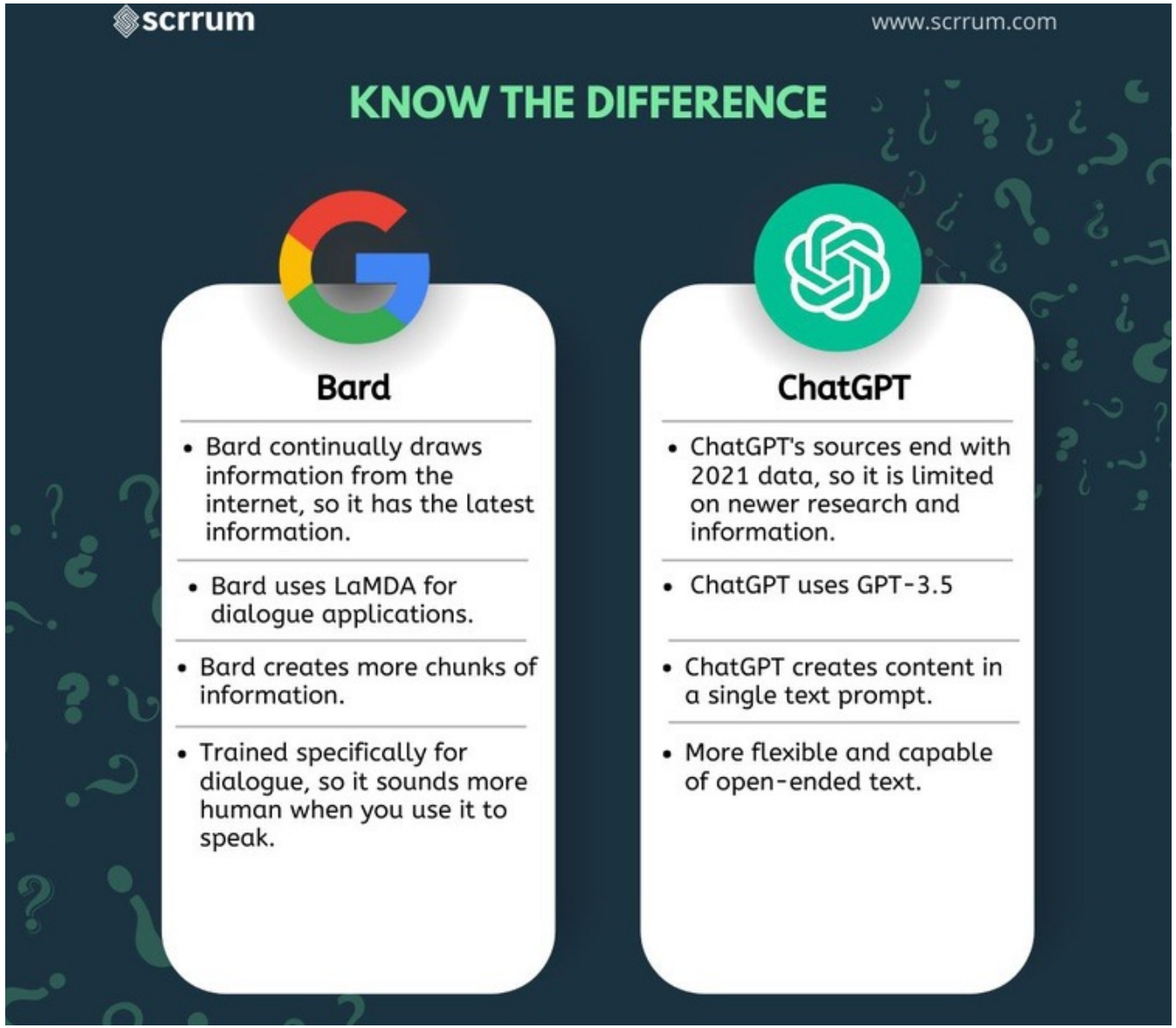
- **Zero-shot model**. This is a large, generalized model trained on a generic corpus of data that can give a fairly accurate result for general use cases without additional training. **GPT-3** is often considered a zero-shot model
- **Fine-tuned or domain-specific models**. Additional training on top of a zero-shot model like **GPT-3** can lead to a fine-tuned, domain-specific model. One example is OpenAI Codex, a domain-specific LLM for programming based on **GPT-3**
- **Language representation model**. One example of a language representation model is **Bidirectional Encoder Representations from Transformers (BERT)**, which makes use of **deep learning** and **transformers** well suited for NLP
- **Multimodal model**. Originally, **LLMs** were specifically tuned just for text, but with the multimodal approach, handling both text and images is possible. **GPT-4** is an example of this type of model

Recent Developments in NLP (Cont.)

- **GPT-4** It was trained on **Microsoft Azure AI supercomputers**. Azure's AI-optimized infrastructure also allows **OpenAI** to deliver **GPT-4** to users around the world
- **GPT-4** can solve math problems, answer questions, make inferences, or tell stories. In addition, GPT-4 can summarize large chunks of content
- **GPT-4** has limitations. For example, GPT-4 does not check if its statements are accurate. It tends to create 'hallucinations,' which is the artificial intelligence term for inaccuracies. It does not know if sensitive information has been used for training
- **GPT-4** does not incorporate information more recent than September 2021 in its lexicon
- There are two other serious competitors to ChatGPT: Google's **Bard** and Anthropic' **Claude**
- One of **GPT-4**'s competitors, **Google Bard**, does have up-to-the-minute information because it is trained on the contemporary internet

UMBC

Comparison of Model Architectures



UMBC

Getting the Most of out ChatGPT

- Avoid information overload, open-ended questions, and lack of constraints. So, you need to use the concepts of:
  - **Chain of Thought** → The method divides intricate problems into smaller, manageable steps
  - **Active prompting** → Querying the LLM with a few CoT examples and generating *k* possible answers for a set of training questions
  - **Reason and Act** (**ReAct**) →This approach is based on human intelligence's ability to seamlessly combine task-oriented actions with verbal reasoning
- Querying Model: **Functions** → **Agents** → **Artificial General** Intelligence
  - AGI is a group of agents working together to surpass human-level intelligence
  - An agent is an AI entity that can operate and make decisions independently

UMBC

The Future of LLMs

- Expanded use of techniques such as **reinforcement learning from human feedback**, which OpenAI uses to train ChatGPT, could also help improve LLMs' accuracy
- There is a class of **LLMs** based on the concept known as *retrieval-augmented generation*—including Google's **Realm** (short for Retrieval-Augmented Language Model)—that will enable training and inference on a very specific corpus of data, much like how a user today can specifically search content on a single site
- There is also ongoing work to optimize the overall size and training time required for LLMs, including the development of Meta's Llama model. **Llama 2**, which was released in July 2023, has less than half the parameters that **GPT-3** has, and a fraction of the number **GPT-4** contains, though its backers claim it can be more accurate
- On the other hand, the use of large language models could drive new instances of **shadow IT** in organizations
- CIOs will need to implement usage guardrails and provide training to avoid data privacy problems and other issues. LLMs could also create new cybersecurity challenges by enabling attackers to write more persuasive and realistic phishing emails or other malicious communications

UMBC

PROJECT DISCUSSION AND WRAP-UP

UMBC

Project Description

- Students will work in a team of five to prepare a presentation on their project
- Each group of students will have a team leader who will keep the professor informed on their progress
- Each group will have a choice of creating either a **chatbot**, a **recommender system**, a **machine translator**, a **spam filter**, a **stock prediction model based on investors' sentiments**, or a **sentiment analysis** based on digital and social media on a current topic
- The grading will consider a group presentation to the class as well as a final report delivered **on time**
- The research paper should not exceed 20 pages with appendices and notebook and should include the following items:
  - A description of the project
  - An area of application (for instance, clothes recommendation, book selection, pizza ordering, customer complaints)
  - The problem that the chatbot, recommender system, machine translator, spam filtering, stock predictive model, or sentiment analysis is designed to help with (for instance, increasing sales through recommendations, creating a trouble-shooting guide, anticipating stock fluctuations based on investors' sentiments, or informing the engineering department of potential or actual customer problems, etc.)
  - The data used to support the chatbot, the recommender system, or sentiment analysis
  - The algorithms powering the chatbot, recommender system, machine translator, investors' sentiments guiding stock fluctuations, or sentiments of groups or individuals (for instance, neural networks, PCA, etc.)
- Each group will have an opportunity to demonstrate the chatbot, recommender system, machine translator, investors' sentiments guiding stock fluctuations, or sentiments of groups or individuals in one of the last two classes of the semester
- Presentation can be either a PowerPoint or a video
- Innovative ideas and presentation formats will be rewarded

UMBC

REFERENCES AND CREDITS

- Charu C. Aggarwal, “Machine Learning for Text,” Springer, 2018
- Charu C. Aggarwal, “Recommender Systems: The Textbook,”, Springer 2016
- John Atkinson-Abutridy, “Text Analytics,” John Atkinson-Abutridy, 2020
- Rachel Batish, “Voice Bot and Chatbot Design,” Packt, 2018
- Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan et al. "Language models are few-shot learners." *arXiv preprint arXiv:2005.14165* (2020)
- Noam Chomsky, “Syntactic Structure,” Mouton de Gruyter, 2002
- Jacob Eisenstein, “Introduction to Natural Language Processing,” MIT Press, 2018
- Kim Falk, “Practical Recommender Systems,” Manning, 2019
- Shohom Gosh and Dwight Gunning, “Natural Language Processing Fundamentals,” Packt Publishing, 2019
- Philipp Koehn, “Statistical Machine Translation,” Cambridge University Press, 2010
- Philipp Koehn, “Neural Machine Translation,” Cambridge University Press, 2020
- Rob High and Tanmay Bakshi, “Cognitive Computing with IBM Watson,” Packt Publishing, 2019
- Nitin Indurkhya and Fred J. Damerau, editors, "Handbook of Natural Language Processing," CRC Press, 2010 (2nd edition)
- Udday Kamath et al., Deep Learning for NLP and Speech Recognition, Springer, 2019
- Daniel Jurafsky and James H, Martin, "Speech and Language Processing: An introduction to speech recognition, computational linguistics, and natural language processing," Prentice Hall, 2009 (2nd edition)

- Hobson Lane et al., “Natural Language Processing in Action,” Manning, 2019
- Bing Liu, “Sentiment Analysis: Mining Opinions, Sentiments, and Emotions,” Cambridge University Press, 2020
- Inderjeet Mani and Mark T. Maybury, “Advances in Automatic Text Summarization,” MIT Press, 1999
- Christopher D. Manning and Hinrich Schütze, "Foundations of Statistical Natural Language Processing," MIT Press, 2000
- Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze, “Introduction to Information Retrieval,” Cambridge University Press, 2008
- Sumit Raj, “Building Chatbots with Python Using Natural Language Processing and Machine Learning,” Apress, 2019
- Dennis Rothman, “Transformers for Natural Language Processing,” Packt Publishing, 2022
- Dipanjan Sarkar, “Text Analytics with Python: A Practical Real-World Approach to Gaining Actionable Insights from Your Data,” Apress, 2019
- Amir Shevat, “Designing Bots: Creating Conversational Experiences,” O’Reilly, 2019 (2nd release)
- Bhargav Srinivasa-Desikan, “Natural Language Processing and Computational Linguistics,” Packt Publishing, 2018
- Jeffrey Strickland, “Data Science Applications Using Python and R: Text Analytics,”, Jeffrey Strickland, 2020
- Oliver Theobald, “Machine Learning: Make your Own Recommender System,” Scatterplot Press, 2018
- Lewis Tunstall, Leandro von Werra, and Thomas Wolf, “Natural Language Processing with Transformers: Building Language Applications with Hugging Face,” O’Reilly, 2022
- Yue Zang and Zhiyang Teng, “Natural Language Processing: A Machine Learning Perspective,” Cambridge University Press, 2021