Natural Language Processing

Week 1

Agenda

- Instructor & Student Introductions
- Course Introduction
- Regular Expressions
- Text Normalization
- Minimum Edit Distance

About the Instructor

Professional Experience

5 years experience in data analytics

Applied ML in 2 domains: finance, law

Education

MS in Applied Data Science, University of Chicago

BS in Physics, University of Chicago

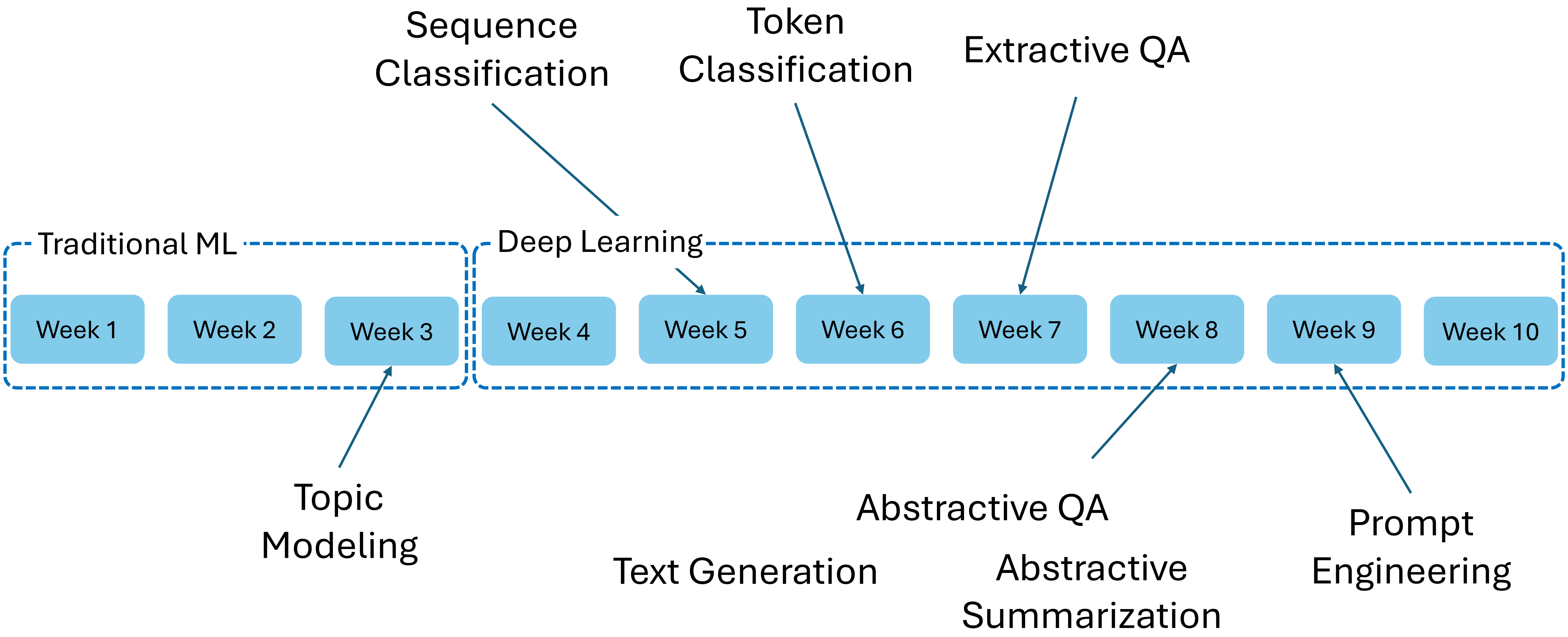
CFA Charterholder

Ignas Grabauskas (ignasg@uchicago.edu)
Data Scientist @ Simpson Thacher Bartlett
Lecturer @ UChicago Data Science Institute

Student Introductions

- Name
- Part-time? Full-time?
- Education / work background
- How far along you are in the program
- What you hope to get out of the class

Course Introduction



Syllabus Highlights

- If contacting instructor, TA, or grader please use email, **not** Canvas messages or comments
- Please bring laptops to class
    - **Create a python environment dedicated to the class**
    - Starting Week 2 there will short review quizzes
- Homework must be submitted individually
- Gen AI is allowed
- Two 24-hour grace periods are allowed for homework only. Must be requested before-hand (for other accommodations please see me).
- Absences must be requested

Regular Expressions

Regular Expressions (regex)

- A language for specifying patterns in text
- Useful when searching for specific patterns in text

Questions? Please do not hesitate to contact us at support@ourcompany.com (please allow 24 hours for our customer care team to respond to your questions). For more immediate service, you can also reach out to our customer care team at 1-555-555-5555. We are dedicated to ensuring your satisfaction and saving you $$$.

```
[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}
```

Character Matching

- Most characters will simply match themselves literally (keep case sensitivity in mind)

Questions? Please do not hesitate to contact us at support@ourcompany.com (please allow 24 hours for our customer care team to respond to your questions). For more immediate service, you can also reach out to our customer care team at 1-555-555-5555. We are dedicated to ensuring your satisfaction and saving you $$$.

`"Questions"` `"customer care team"` `"24"`

Metacharacters ^ $ * + ? { } [ ] \ | . ( )

- Special characters that don't match themselves (need to be "escaped" for literal match)
- They signal some out of the ordinary thing or modify behavior in some non-literal way

Questions? Please do not hesitate to contact us at support@ourcompany.com (please allow 24 hours for our customer care team to respond to your questions). For more immediate service, you can also reach out to our customer care team at 1-555-555-5555. We are dedicated to ensuring your satisfaction and saving you $$$.

"\?"   "\."   "\("   "\)"   "\$"

Character Classes / Character Sets [ ]

- Denote a set a characters that can be matched (character-level "or")

Behavior	Regex Pattern	Matches	Notes
List characters individually	`[Hh]ello`	"Hello" **or** "hello"	
Provide range of characters	`[A-Za-z1-4]`	Any capital letter **or** any lowercase letter **or** 1 through 4	
Negate or "compliment" a set of characters	`[^A-Za-z1-4]`	Any character that's **not**: capital A through Z **or** lowercase a through z **or** 1 through 4	^ character negates only if at the beginning of the set
Certain metacharacters are not active	`[$?.\]^]`	$ **or** ? **or** . **or**] **or** ^	\ ] must be escaped ^ - may need to be escaped

Escaping and Special Sequences: \

- Used to escape metacharacters to allow for literal matches
- Denotes a special sequence for commonly used sets of characters

`\d` matches any digit character, equivalent to `[0-9]`

`\D` matches any non-digit character, equivalent to `[^0-9]`

`\s` matches any whitespace character, equivalent to `[ \t\n\r\f\v]`

`\S` matches any non-whitespace character, equivalent to `[^ \t\n\r\f\v]`

`\w` matches any alphanumeric character, equivalent to `[a-zA-Z0-9_]`

`\W` matches any non-alphanumeric character, equivalent to `[^a-zA-Z0-9_]`

re — Regular expression operations — Python 3.12.2 documentation

Dot .

- Used to match any character except newline character (though can be forced to match newline character as well)
- Useful for situations where the exact character at a certain position is unknown

Behavior	Regex Pattern	Matches	Notes
List characters individually	`.at`	aat bat cat dat eat !at 9at	

Repetition / Quantifiers `{ } + ? *`

- Used to specify that portions of the regex pattern must be repeated a certain number of times

Behavior	Regex Pattern	Matches	Notes
Matches 0 or more times	`ca*t`	ct cat caaaaat	
Matches 1 or more times	`ca+t`	cat caaaat	
Matches 0 or 1 times	`scikit-?learn`	scikitlearn scikit-learn	
Matches at least *m* and at most *n* times	`\d{3,5}`	123 1234 12345	`{m,}` no upper bound `{,n}` no lower bound `{m}` exact count

Anchors ^ $

- Special characters that require a pattern to be to a particular area of a string

Behavior	Regex Pattern	Matches	Notes
Matches start of string	`^hello`	"**hello**, world" but not "she said hello"	
Matches end of string	`world$`	"hello, **world**" but not "world of wonder"	

Grouping ( )

- Allows us to look at a sequence of characters as a unit
- Useful for extracting parts of a pattern or applying quantifiers to a group of chars

Behavior	Regex Pattern	Matches	Notes
Applies quantifier to characters as a group	`(ab)+`	ab abab	
Allows extraction of subsequence in pattern	`(\d{3})-\d{3}-\d{4}`	555-555-5555 312-555-5555	Matches phone number and can extract the area code specifically, referred to as "capturing groups". For non-capturing groups, use (?:)

Look-ahead and look-behind （ ）

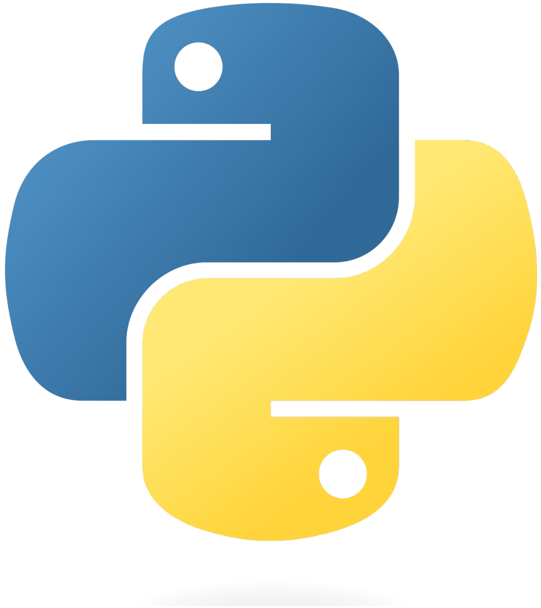
- Allows us to create conditions for a match based on what follows or precedes

Behavior	Regex Pattern	Matches	Notes
Match "x" only if it is followed by "y"	`Springfield(?=, IL)`	"**Springfield**, IL" not "Springfield, MA"	Positive Lookahead
Match "x" only if it is not followed by "y"	`Springfield(?!, IL)`	"**Springfield**, MA" or "**Springfield**, MO" not "Springfield, IL"	Negative Lookahead
Match "x" only if it is preceded by "y"	`(?<=Mr\. )Brown`	"Mr. **Brown**" not "Dr. Brown"	Positive Lookbehind
Match "x" on if it is not preceded by "y"	`(?<!Mr\. )Brown`	"Dr. **Brown**" not "Mr. Brown"	Negative Lookbehind

Alternation |

- Pattern-level "or"

Behavior	Regex Pattern	Matches	Notes
Matches several patterns	`cat|dog|bird`	“cat" **or** “dog“ **or** “bird”	
Matches several characters	`a|c`	“c” **or** “b”	Better to use character sets for character-level or
Limits the reach of the alteration	`(bat|bird)man`	Matches “batman” **or** “birdman”, **not** “bat”	



Text Normalization

Text Normalization

- Preprocessing text to convert it into a more uniform format for NLP applications
- Typically includes
  - Tokenizing (segmenting) words
  - Normalizing word formats
  - Segmenting sentences
  - Removing stop words
- The extent of text normalization depends on the task at hand
  - Named entity recognition relies on capitalization
  - Domain-specific scenarios

Word Tokenization

- The task of segmenting running text into words (or subwords)

```
"Behold! An example of word tokenization!"
["Behold", "!", "An", "example", "of", "word", "tokenization", "!"]
```

- Punctuation as separate tokens (commas useful for parsing, periods and question marks useful for sentence boundaries, etc.)
- Intra-word punction: *AT&T*, *Ph.d.*, *m.p.h.*
- Intra-word special characters: *$5.99, https://www.python.org, #LastMinuteHomework*
- Phrases: *Rock n' Roll*, *Salt Lake City*

Word Tokenization

- Language dependency
- German (noun compounds not segmented)

  `Lebensversicherungsgesellschaftsangestellter`
- Mandardin (no spaces between words)

  我喜欢学习

Word Tokenization

- Top-down (rule-based) tokenization
    - Begins with a comprehensive view of language (structure, grammatical conventions)
    - Rules based on syntax, grammar, or specific patterns identified in text
    - NLTK Treebank Tokenizer or spaCy

- Bottom-up tokenization, statistics of letter sequences are used to break up words into subword tokens
    - Begins with text data and uses statistical methods to identify patterns in letter sequences
    - Byte Pair Encoding (BPE), SentencePiece, and WordPiece

Word Normalization

- The task of putting words/tokens into a standard format
  - Reducig words to their base form (lemmatization & stemming)
  - Case folding useful for generalization in information retrieval or speech recognition)
  - Case folding not useful for sentiment analysis, information extraction (US vs us)
  - Reducing word forms (USA <--> US)
- Applications are several:
  - Make solutions more generalizable (document retrieval searching for "cars" vs "car")
  - In some cases, makes identifying signal easier

Word Normalization: Lemmatization

- Lemma: a form of a word chosen to represent a lexeme (set words related through inflection)
- Lemmatization: the task of removing word inflections, returning a word to its lemma form

Variations	Lemmatized
`running, ran, runs`	`run`
`are, was, being, were`	`be`
`He visited libraries searching for books`	`he visit library search for book`

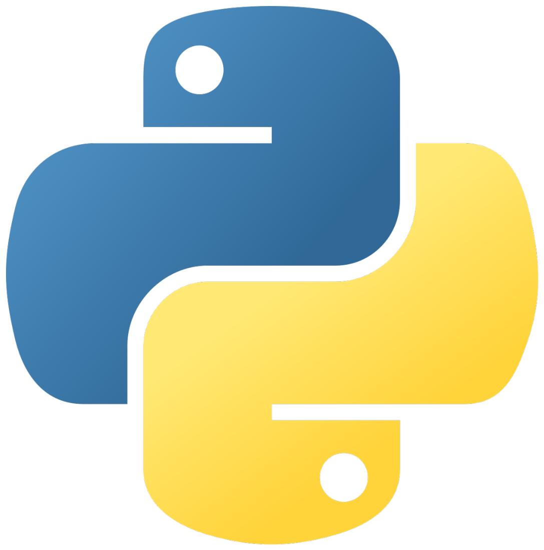
- Requires a sophisticated analysis of a word (grammatical tense, plurality, pos)
- Useful in cases where an accurate understanding of a word's meaning is important (because lemmatization always results in a valid word)

Word Normalization: Stemming

- Stem: the part of the word that remains unmodified during inflection (generally)
- Crudely chopping off affixes (word elements attached to stem)

Variations	Base Form
`running, ran, runs`	`running, ran, run`
`are, was, being, were`	`are, wa, be, were`
`He visited libraries searching for books`	`he visit librari search for book`

- Porter Stemmer is designed handle wide range of irregularities without being too aggressive
- Lancaster Stemmer uses a series of 120 rules to aggressively stem a word

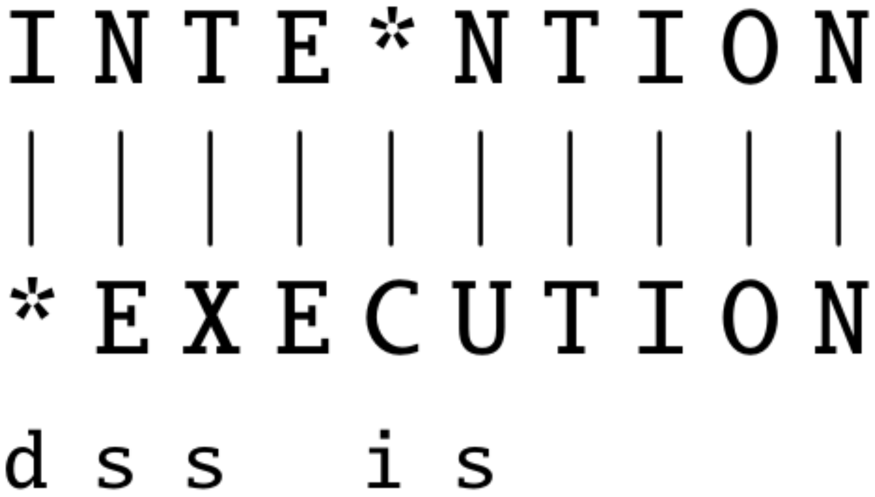


Minimum Edit Distance

Minimum Edit Distance

- Much of NLP is concerned with measuring how similar two piece of text are, including individual words
- We can compare how close two strings are to each other by calculating minimum edit distance
- **Minimum edit distance**: minimum number of edit operations (insertion, deletion, substitution) needed to transform one string to another

Alignment



```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Levenshtein	1	1	1	1	1	$\sum = 5$
Weighted Levenshtein	1	2	2	1	2	$\sum = 8$

Jurafsky, Speech and Language Processing 3rd Edition, p. 26, Figure 2.14

Variations

- There are different variations of minimum edit distance, for different applications
- **Damerau-Levenshtein:** allows transposing of adjacent characters. Useful for typographical errors
- **Weighted Levenshtein:** assigns different cost to different operations. Useful for keyboard layout optimization or OCR
- **Jaro-Winkler:** gives favorable scores to strings that match from the beginning. Useful for short names and record linkage
- And so on…