8. Text processing pt. 2
LING 471

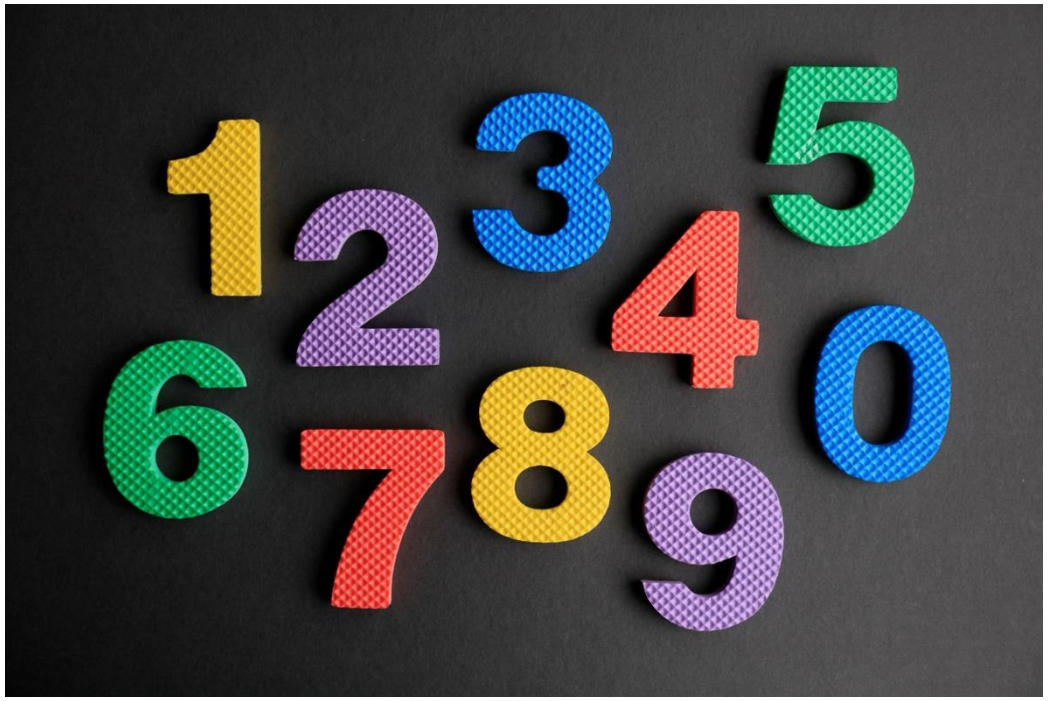
Learning outcomes

- Describe how textual data is stored on computers
- Describe what Unicode and ASCII are
- Write and describe simple list and generator comprehensions
  - With optional conditionals
- Discuss different components of evaluation for NLP systems

Unicode and encodings

Encodings

- An **encoding** is a method for storing something in a particular way
- A prominent type of encoding is **text encoding**
- Every computer *must* store data **as numbers**, so text must be encoded as a number



Why use encodings?

- More convenient from a hardware perspective to only need to handle one kind of data (numbers)
- Sometimes, it is not clear how else to represent something (e.g., letters)
- Encodings can take up a lot less space
  - A letter is an image (or at least visual), which takes up a lot of space
- Comparing some objects is hard (how do you "compare" letters?)
  - We can at least compare numbers, so make letters into numbers

Decimal	Hexadecimal	Binary	Octal	Char
048	30	0110000	060	0
049	31	0110001	061	1
050	32	0110010	062	2
051	33	0110011	063	3
052	34	0110100	064	4
053	35	0110101	065	5
054	36	0110110	066	6
055	37	0110111	067	7
056	38	0111000	070	8
057	39	0111001	071	9
058	3A	0111010	072	:
059	3B	0111011	073	;
060	3C	0111100	074	<
061	3D	0111101	075	=
062	3E	0111110	076	>
063	3F	0111111	077	?
064	40	1000000	100	@
065	41	1000001	101	A
066	42	1000010	102	B
067	43	1000011	103	C
068	44	1000100	104	D
069	45	1000101	105	E
070	46	1000110	106	F
071	47	1000111	107	G
072	48	1001000	110	H
073	49	1001001	111	I
074	4A	1001010	112	J
075	4B	1001011	113	K
076	4C	1001100	114	L
077	4D	1001101	115	M
078	4E	1001110	116	N
079	4F	1001111	117	O
080	50	1010000	120	P
081	51	1010001	121	Q
082	52	1010010	122	R
083	53	1010011	123	S
084	54	1010100	124	T
085	55	1010101	125	U
086	56	1010110	126	V
087	57	1010111	127	W
088	58	1011000	130	X
089	59	1011001	131	Y
090	5A	1011010	132	Z
091	5B	1011011	133	[
092	5C	1011100	134	\
093	5D	1011101	135	]
094	5E	1011110	136	^
095	5F	1011111	137	_

ASCII: The first convention

- American Standard Code for Information Exchange
- More or less the first standardized way to represent text data
- Original implementation only had 128 possible characters, based on English letters and numbers
- Set up as a table (dict) that mapped characters to numbers

More on ASCII

- Widely used until fairly recently
  - Python 2 still defaults to using ASCII
  - You will probably come across text files encoded with ASCII at some point in your life
- Problems
  - 128 characters is not enough for other languages, especially if they don't use the Roman alphabet (e.g., Chinese, Japanese, Korean)
  - Not entirely clear how new characters should be added

Unicode Standard: An extensible encoding scheme

- Unicode is an encoding scheme from the Unicode Consortium
- Central goal is to catalog all known characters and assign numbers to them
- Started with ASCII values and then built from there
- Not all writing systems are well-represented
  - See how Mongolian orthography is represented
- Still, probably the best system we currently have (and is *the* standard)

Text data in Unicode

- Earlier, we talked about how we need to convert text data into numbers
- But, text is already represented as numbers on computers...
- This encoding is not necessarily useful for processing language, but it is better than nothing
- Python lets you get the numbers associated with a letter with the `ord` function

```
s = 'The quick brown fox jumped
over the lazy dog'
for c in s:
    print(ord(c))
```

Unicode and fonts

- To a rough approximation, a computer font is just a mapping between a (Unicode) number and an image/symbol
- In a text document, webpage, etc., the text data is stored as numbers
  - Your computer then looks up what images to show for the stored numbers based on the font

Programming activities

- Assign the following text to a variable: “`bUT i dOn'T liKE eAtINg vegETAblEs!`”
- Use a combination of a for loop, string concatenation, conditionals, and the ord() and chr() functions to change the string to all lower case
  - Then change everything to upper case
  - Hint: “A” is 65, “B” is 66, …, and “a” is 97, “b” is 98, …
- Do not use regular expressions or the lower() and upper() functions to change the string

List and generator comprehensions

Working with lists

- So far, we have made lists with
  - Manual entry (actually type the values in)
  - Some kind of loop with `list.append` or by calling the `list` function
- There are other ways we can work with lists, though
  - Some of them are particularly helpful for filtering out some elements

```
a = [1, 2, 3] # manual entry

a = [] # list.append function
for i range(1, 4):
    a.append(i)

a = list(range(1, 4)) # list function

# how to filter?
b = []
for x in a:
    if x % 2 == 0:
        b.append(x)
```

List comprehensions

- Syntactic method to create a list programmatically (as opposed to manually typing all the elements manually)
- "Comprehenison" comes from a similar meaning as "comprehensive" and not "understanding"
- Notation is related to set builder notation from math

- Set of all integers:
$$\{n \in \mathbb{Z}\}$$
- Set of all even integers:
$$\{n \in \mathbb{Z} \mid \exists k \in \mathbb{Z}, n = 2k\}$$
- Set of all squares of integers:
$$\{n \in \mathbb{Z} \mid \exists k \in \mathbb{Z}, n = k^2\}$$
- List comprehensions are easier to understand and write than the above examples!

Simple list comprehensions

- [EXPR for VAR in ITERABLE]
  - [- opening bracket
  - EXPR – any expression that evaluates to a value
  - for – standard for keyword
  - VAR – variable of iteration in for loop
  - in – standard in operator
  - ITERABLE – any iterable
  -] – closing bracket

```
a = [x for x in range(1, 4)]
# a == [1, 2, 3]

a = [y**2 for y in range(1, 4)]
# a == [1, 4, 9]

a = [1 for i in range(1, 4)]
# a == [1, 1, 1]
```

For loop and list comprehension equivalency

- Effectively, remove the brackets, and make the EXPR the body of the loop
  - Still need to store the results somehow…

```
a = list()
for x in range(1, 4):
    a.append(x**2)
```

```
a = [x**2 for x in range(1, 4)]
```

List comprehensions with conditionals

- You can add a conditional (if-statement) to a list comprehension to filter some values out
  - E.g., you only want even numbers, or you only want words that end with "s"
- [EXPR for VAR in ITERABLE if COND]
  - if – standard if keyword
  - COND – Boolean condition

```
# list of only even numbers up to 20
a = [x for x in range(21) if x % 2 == 0]

# select only even numbers from list
b = [1, 2, 3, 4, 5, 6, 7]
b = [x for x in b if x % 2 == 0]

# select only words ending in "s"
sentence = 'This is how cats and dogs get
along'
c = [x for x in s.split() if x.endswith('s')]
```

Generator comprehensions

- Like list comprehensions, but used in other scenarios where you don't necessarily want or aren't able to store a list
  - E.g., calculating a sum, making a dict
- Just remove the square brackets!
- Generators (like `range`) only give values when asked to
  - i.e., they are **lazy** instead of **eager**
  - This is often desirable with large sets of data

```
# sum 0 up to and including 10
s = sum(x for x in range(11))

# dict of squares from 0 up to and
# including 10
d = {k : k**2 for k in range(11)}
```

More on comprehensions

- They can be hard to use at first, but I think they are ultimately easier to use and read with once you understand them
- Often considered to be the "Pythonic" (idiomatic, standard for Python) way to make lists and filter out certain values
- Usually best when the expression is simple and for single for loops
  - `[math.cos(int(x.strip())**2) if int(x.strip()) % 2 == 1 else 0 for x in s.split()]`
    - Hard to understand
  - `[(x, y, z) for x in range(10) for y in range(10) for z in range(10)]`
    - Tricky triple loop, but sometimes useful… (can make permutations easily this way)
- Often slightly faster than using list.append to build lists
- Usually don't want to use functions with side-effects in comprehensions, e.g., don't use the `print` function in them

Programming activities

- Using list comprehensions, do the following:
  - Create a list from 0 up to and including 10000
  - Create a list that squares all the numbers from the previous list
  - Create a list that only includes odd numbers from 0 up to and including 9877
  - Create a list that has converted each word in the following string to lowercase: "tHIs iS A sENtENce thaT hAS miXeD cASe"
  - Create a list that has called ord on the characters in the example string on the slide "Text data: Is Unicode enough?"
- Using a generator comprehension
  - Create a dict that maps all integers from 0 up to and including 10000 to their square roots
  - Sum the value of all even numbers from 0 up to and including 4578

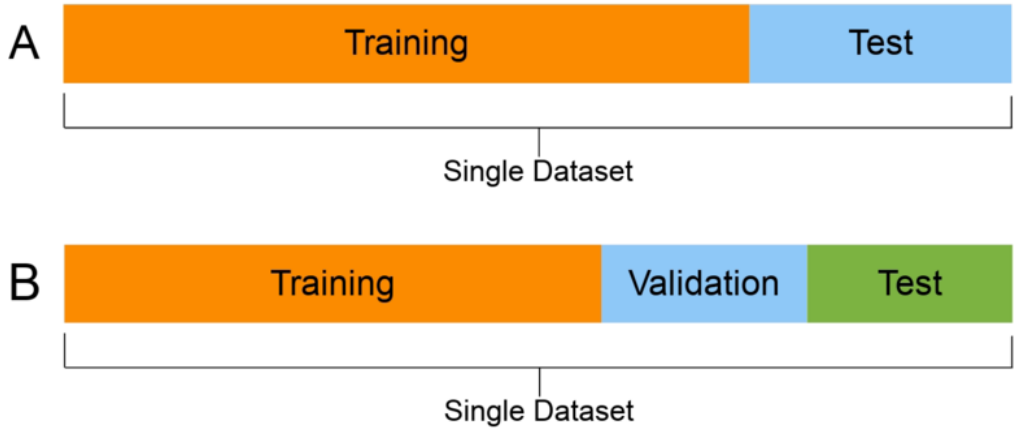
Evaluation in NLP and data science

Evaluation in computational fields

- Computational approaches allow for
  - Numerical evaluation
  - System comparison
  - Feedback on system changes
- Computational fields are often defined by evaluation
  - What they do is driven by evaluation scores on concrete datasets

Evaluation in machine learning: Data set splitting

- Machine learning **trains** algorithms on labeled data points
- To evaluate, you need **unseen** data points
- Often split data into train/dev/test (or train/val/test)
  - Dev/val: used to tune various parameters
- Usually you want to evaluate on the test set



Evaluating without a train/test split: Cross validation

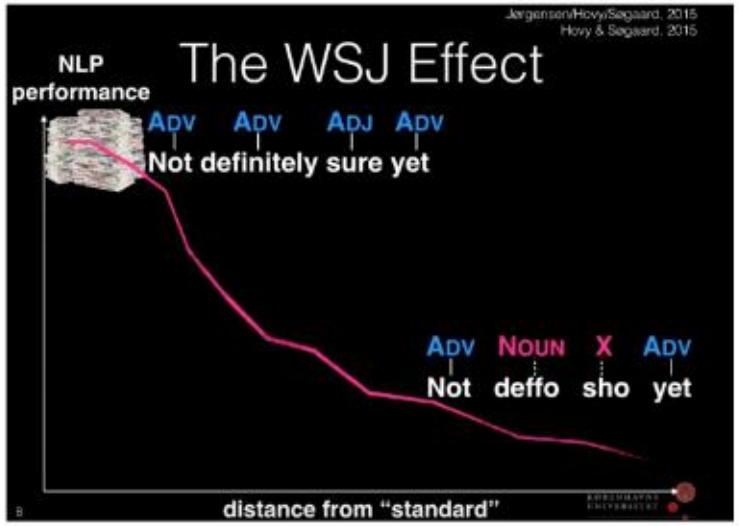
- Sometimes there is not enough data to split do test/train splits
- Cross-validation
  - Reserve a fraction of data in each iteration of training
  - At every iteration, perform the evaluation on the small held-out data

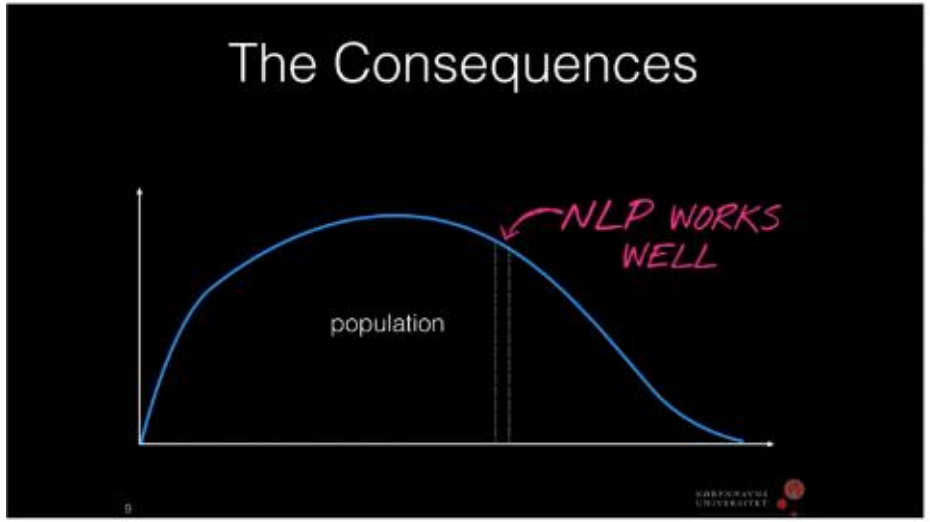
EXAMPLES

- K-fold cross validation
  - Split data into *k* different fractions (usually 5 or 10)
- Leave one out (LOO) cross validation
  - Leave one data point out for evaluation and train on the rest
  - Repeat until all points used

Evaluation in NLP and data science

- NLP is defined and driven by evaluation
- Trained and evaluated on classic data sets like the Wall Street Journal corpus
- This is a common practice in speech recognition as well (e.g., also using the Wall Street Journal corpus, among others)
- A system is evaluated based on how well it compares to other systems trained on the same data
  - Could argue that this is to control for variation in the training data





The WSJ effect

- Years of training and retraining systems on WSJ enshrined certain biases in NLP
- Over training on WSJ led systems to adapt to the test set
  - Even though the train/test division in the data set was observed
  - Reason: We keep choosing systems that performed well on the test data…
- With the rise of LLMs, the field has largely moved towards benchmarks such as MMLU, HELM, etc.
  - Previously: “we're overfitting to WSJ"
  - Now: "we're overfitting to benchmarks in general”& "we're training on the entire internet, what biases does that introduce?"

Hovy, D. (2019). Hidden biases: Ethical issues in NLP, and what to do about them. Available from https://hch19.cl.uni-heidelberg.de/program/slides/I/HCH19_lecture_Dirk_Hovy.pdf

Evaluation: Baseline

- Choose a starting point for your comparison
- What performance do you want to beat?
  - 0% accuracy?
  - Random/chance performance (could you flip a coin and get better results?)
  - Most common value?
    - E.g., can you do better than classifying every word as a determiner? "the" is the most common word in English, and "the" is a determiner, so it's not an abysmal guess…
  - An older system?

"Gold" standards or "ground truth"

- Labels are treated as correct during the evaluation
  - The labels provided in the IMDB data set are an example
- Often, this is not entirely true and somewhat subjective
  - See the transcriptions used in speech recognition
  - Try to find a label in the IMDB set you disagree with!
- I have heard jokes about renaming some of these as "bronze" standards…

Intrinsic vs. extrinsic evaluation

INTRINSTIC

- How well does the system perform on its own criteria?
- Example
  - You've made a system that tags part of speech (verb, noun, etc.)
  - How well did it do at tagging the parts of speech?

EXTRINSIC

- How much does the system improve others that rely on it?
- Example:
  - You have your part of speech tagger from before
  - How much does your new tagger improve a system that parses sentence structure?

Evaluation drives NLP

- People are happy to have **incremental** improvement
- Many experiments are **designed** to have incremental improvement
  - And sometimes the designers worry less about whether the numbers the system learns are meaningful
  - Sometimes leads to "scoreboarding" or "leaderboarding" (both pejorative) where the only real target is increasing performance on the evaluation metric