REGULAR EXPRESSIONS AND TEXT NORMALIZATION

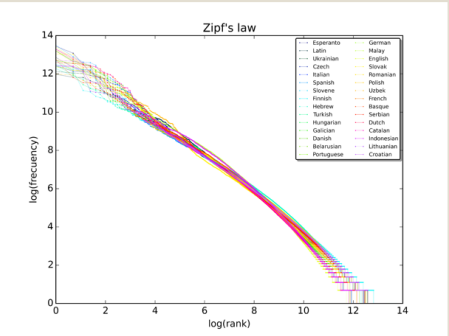
CS 445 - NATURAL LANGUAGE PROCESSING

Instructor: Dilara Keküllüoğlu
Fall 2024 – Week 2

Recap

1. What is NLP?
2. Why is NLP hard? – Variability, Ambiguity, Sparsity, Robustness, Context Dependence, Unknown Representation, Language Diversity, and many others.
3. Corpora – Selection, Characteristics, Bias

- **Homophones**: hear - here
- **Word sense**: block (rock or neighbourhood)
- **Part of speech**: smell (verb or noun?)
- **Syntactic structure**: I saw a man with a telescope.
- **Quantifier ambiguity**: Every student did not pass the exam.
- **Multiple meanings**: I saw her duck.
- **Reference**: The girl told her mom about the fight. She was upset.
- **Discourse**: I will not cook today. Alice ordered take out.



Sergio Jimenez, A plot of the rank versus frequency for the first 10 million words in 30 Wikipedias (dumps from October 2015) in a log-log scale.

Tophat Exercise

- What was the most interesting thing you have learned last week in the university?

Learning Goals (Week 2)

1. **Define** regular expressions
2. **Write** regular expressions for given patterns
3. **Differentiate** false positive and false negative cases
4. **Apply** tokenization and normalization to text

Regular Expressions

- Combination of characters for matching patterns in text
- Used to find specific text and patterns in textual data

Let's say we want to find all occurrences of word in a document:

- Cat
- Cats
- cat
- cats
- kedi
- kediler

Regular Expressions - Disjunctions

- [] <- Anything inside the bracket will match
- | <- pipe for representing or

Pattern	Match
[abc]	a, b, c
[Cc]at	Cat, cat
[1234567890]	Single digit
[abc]d[1234]5	ad15, cd35, … (12 versions)
[Cc]at|[Kk]edi	Cat, cat, Kedi, kedi

Regular Expressions - Ranges

Pattern	Match
[A-Z]	An upper case letter
[a-z]	A lower case letter
[0-9]	A single digit
[A-Za-z0-9]	A letter from all of the above

Negation

- ^ just before negates the list in brackets

Pattern	Match
[^A-Z]	Not an upper case letter
[^Ss]	Not an ‘S’ or ‘s’
[^?]	Not a question mark
[A^]	A or ^

Aliases for common patterns

Pattern	Expansion	Example
\w	[0-9a-zA-Z_]	**A**dverb
\W	[^\w]	Hi**!**
\d	[0-9]	**0**07 Bond
\D	[^0-9]	**C**risp
\s	[\r\t\n\f]	Good news
\S	[^\s]	**G**ood news

Getting interesting! Wildcards

- . – matches any character
- ? – Optional for previous character (0 or 1 times)
- * - 0 or many times for the previous character
- + - 1 or many times for the previous character
- {} – limits the number of characters
- ^ - start of the word (when not in brackets)
- $ - end of the word

Pattern	Match
Ch.r	Char, Cher, Ch r, Ch4r
Cats?	Cat or Cats
Vay*	Va, Vay, Vayyyy, Vayyyyyy
Me+rhaba	Merhaba, Meeeeerhaba
Vay{5}	Vayyyyy
^[\w].*[\W]$	All good!
[\w]+	
[\d]*x?	

Try out regex

- Go into https://regex101.com/ mobile or laptop (you can try to do it on paper too)
- Write a regular expression that would match **afs4596!cc.xx**
- Write a regular expression that can match email addresses

Email Validation

- How do emails look? – unique email name, @, service provider name, domain name
- Unique email name – [\w\-\.]+
- @ - @
- Service provider – [\w\-]+\.
- Domain name – [\w\-]{2,4}

- All together - ^[\w\-\.]+ @ [\w\-]+\. [\w\-]{2,4}$

Group Exercise

- Make groups of 2-3 people and complete following tasks.
- Write a regex to match =) and =(only
  - ^(=\(|=\))+$
- Write a regex to catch ages in a text
  - \d{1,2}
  - [123456789][\d]?
  - ^I am [123456789][\d]?

Iterative Process

- It's OK to try a few times before getting it right. You need to check whether your regex matches as you want in your dataset.
- the – misses capitalized The
- [Tt]he – also matches words like other, Theology
- **\W[Tt]he\W**

False positive – False negative

- False Positive (FP) – Some words match even though they should not
  - [Tt]he – also matches words like other, Theology

- False Negative (FN) – Some words do not match even though they should
  - the – misses capitalized The

These measures are used to evaluate performance.

FP and FN

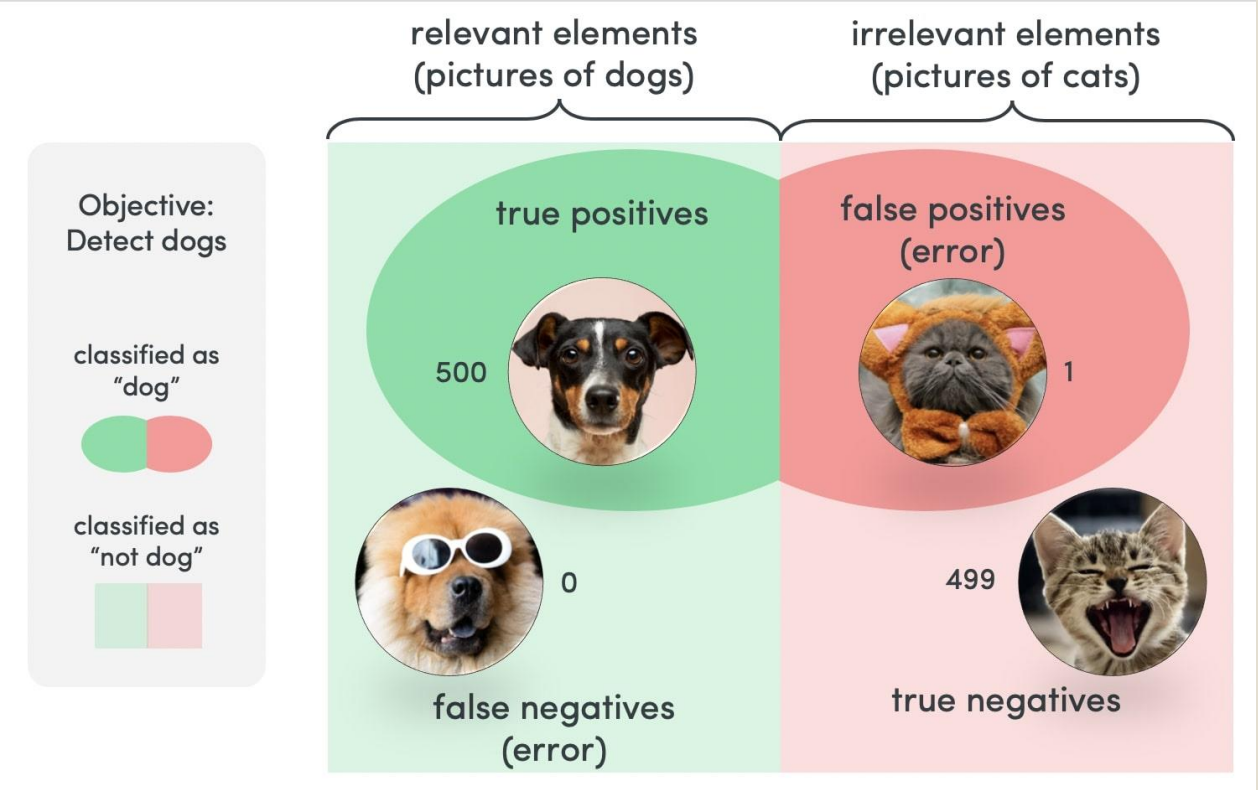


Photo: https://levity.ai/blog/precision-vs-recall

Precision - Recall

- We want to reduce both FP and FN while increasing True Positive (TP) and True Negative (TN) for good performance.
- Precision – Lower the FP, better the precision
- Recall – Lower the FN, better the recall

Precision - Recall

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN}$$

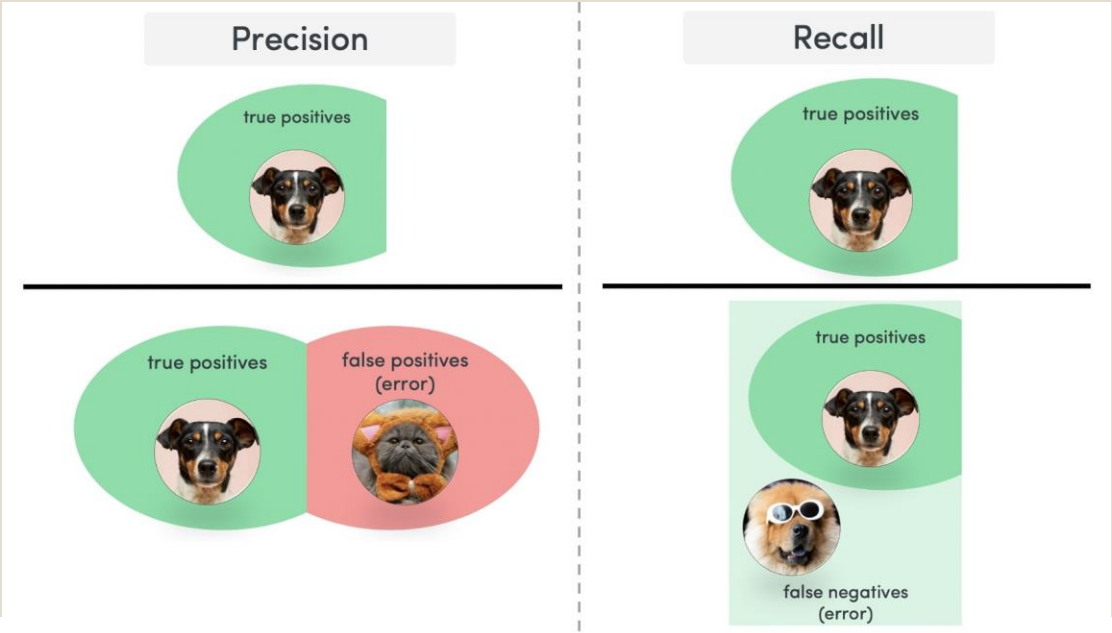
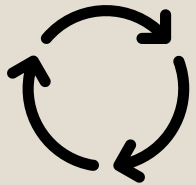


Photo:
https://levity.ai/blog/precision-vs-recall

Regex Use

- Used for text validation, text processing, finding patterns
- Personally in my research with Twitter data, I used to find phrases, 2-digit numbers, etc.

Text Normalization



- Essential step in all of NLP tasks
- Transforming text into a uniform format for standardization
- U.S.A – USA <- represent the same entity
- cat, cats, cat’s -> cat

Text Normalization

- Steps that could be taken for text normalization
  - Case-folding – lower casing the words
  - Tokenization
  - Lemmatization
  - Stemming
  - Sentence segmentation
- Steps might change depending on context. For example, emojis might need to be removed for social media data.

Tokenization

- Dividing text into smaller meaningful units – tokens
- How to separate tokens?
  - Spaces
  - Punctuations
- Should hyphens stay?
  - up-to-date
  - well-being
  - check-in

Top-down vs Bottom-up Tokenization

- Should we have rules or use statistical patterns for breaking down tokens?
- Rule-based implementation
  - Predictable
  - Easy to write with regex
- Statistical patterns
  - Do not need to extract tokenization rules
  - Output might change with different dataset

Top-down Tokenization

- Considerations while creating rules for tokenization in English
  - **Punctuations** are important
    - Separate: Sentence-marking, commas for parsers
    - Keep-in: dots in numbers (5.99$), abbreviations Ph.D.
  - **Clitics**: 'm, 're, 'll ->expand or keep as they are?
    - I am, there are, she will
  - Keep **hyphenated compound words** as one.
  - Do not break down **named entities** – "Sabancı University", "New York", etc.

What about other languages?

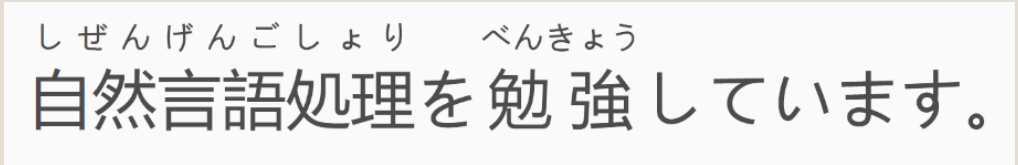
- Would these rules work in every language?
- Languages are **diverse**, some do not have space or punctuations.

姚明进入总决赛 - Yao Ming reaches the finals

- In Chinese, characters (hanzi) can be divided one-by-one and it would be meaningful.

What about other languages?

- However, that would not work for Japanese.



- I am studying NLP.

- We need to have word segmentation.
- It is not easy to model where to divide the text.

Bottom-Up Tokenizers

- LLM tokenizers - train tokenizers with the data
- Token learner + Token segmenter
- Given the train data – learn the most frequent subwords
- Use segmenter to tokenize the test data
- Byte-Pair Encoding algorithm (Sennrich et al., 2016)

Train data	Test data
newer, higher, stronger --- x+er	lower - low+er

Byte Pair Encoding (BPE) token learner

Let vocabulary be the set of all individual characters
= {A, B, C, D,…, a, b, c, d….}

- Repeat:
  - Choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')
  - Add a new merged symbol 'AB' to the vocabulary
  - Replace every adjacent 'A' 'B' in the corpus with 'AB'.
- Until *k* merges have been done.

BPE token learner algorithm

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

Byte Pair Encoding (BPE) Addendum

- Most subword algorithms are run inside space-separated tokens.
- So we commonly first add a special end-of-word symbol '__' before space in training corpus
- Next, separate into letters.

BPE token learner

- Original (very fascinating 🙄) corpus:

low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new

- Add end-of-word tokens, resulting in this vocabulary:

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w
2 l o w e s t _	
6 n e w e r _	
3 w i d e r _	
2 n e w _	

Jurafsky & Martin Slides

BPE token learner

corpus

5 l o w _
2 l o w e s t _
6 n e w e r _
3 w i d e r _
2 n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w

Merge e r to er

corpus

5 l o w _
2 l o w e s t _
6 n e w er _
3 w i d er _
2 n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er

Jurafsky & Martin Slides

BPE

corpus

5 l o w _
2 l o w e s t _
6 n e w er _
3 w i d er _
2 n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er

Merge er _ to er_

corpus

5 l o w _
2 l o w e s t _
6 n e w er_
3 w i d er_
2 n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_

Jurafsky & Martin Slides

BPE

corpus

5 l o w _
2 l o w e s t _
6 n e w er_
3 w i d er_
2 n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_

Merge n e to ne

corpus

5 l o w _
2 l o w e s t _
6 ne w er_
3 w i d er_
2 ne w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_, ne

Jurafsky & Martin Slides

BPE

The next merges are:

Merge	Current Vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

Jurafsky & Martin Slides

BPE token segmenter algorithm

On the test data, run each merge learned from the training data:

- Greedily
- In the order we learned them
- (test frequencies don't play a role)

So: merge every e r to er, then merge er _ to er_, etc.

- Result:
  - Test set "n e w e r _" would be tokenized as a full word
  - Test set "l o w e r _" would be two tokens: "low er_"

Properties of BPE tokens

- Usually include frequent words
- And frequent subwords
  - Which are often morphemes like *-est* or *–er*
- A **morpheme** is the smallest meaning-bearing unit of a language
  - *unlikeliest* has 3 morphemes *un-*, *likely*, and *-est*

Tokenizer Try

- https://huggingface.co/spaces/Xenova/the-tokenizer-playground
- Write a few sentences with different tokenizers and try.
  - English
  - Turkish or any other language
- You can see “I’m John” is tokenized correctly but not “I’m Dilara” =) Try out your name.

Lemmatization

- **Lemma**: "a form of a word that appears as an entry in a dictionary and is used to represent all the other possible forms." – Cambridge Dictionary
- Converting tokens to their **root** meanings
- Cats and cat should be treated the same
- Morphological parsing – **Stems** + Affixes

Lemmatization

- She is running → She be run
- I ate an apple → I eat a apple

- What about Turkish?? Too many affixes.
- Morphological parsing and lemmatization is complex.

- Uygarlaştıramadıklarımızdanmışsınızcasına
- `(behaving) as if you are among those whom we could not civilize'
- Uygar `civilized' + laş `become'
  + tır `cause' + ama `not able'
  + dık `past' + lar 'plural'
  + ımız 'p1pl' + dan 'abl'
  + mış 'past' + sınız '2pl' + casına 'as if'

Stemming

- Another step of normalization
- "Lazy" **alternative** to lemmatization
- Chop off the affixes of the words
- Simpler rules

- Porter Stemmer sample rules

$$\text{ATIONAL} \rightarrow \text{ATE} \quad (\text{e.g., relational} \rightarrow \text{relate})$$
$$\text{ING} \rightarrow \epsilon \quad \text{if stem contains vowel (e.g., motoring} \rightarrow \text{motor)}$$
$$\text{SSES} \rightarrow \text{SS} \quad (\text{e.g., grasses} \rightarrow \text{grass})$$

Lemmatization vs Stemming

- Lemmatization is **dictionary-based**
- Stemming is **rule-based**
- Full lemmatization might require sophisticated analysis – POS tagging + multiple affixes extraction
- Stemming might be less accurate in grouping words but it might be faster

Sentence Segmentation

- Deciding the boundary of the sentences
- Punctuation such as .,?,!
- However, not every “.” is a sentence-ender.
- Abbreviations dictionary to differentiate abbreviations vs. sentence ender “.”s.

Text Normalization

- Important step for text analysis and NLP tools
- Lots of decisions to make!
- Documentation every step is essential for reproducibility and credibility of the results

Learning Goals (Week 2) - revisited

1. **Define** regular expressions
2. **Write** regular expressions for given patterns
3. **Differentiate** false positive and false negative cases
4. **Apply** tokenization and normalization to text

Following lectures

- Edit distance
- Text analysis – text cleaning, sentiment analysis example

Further resources

- 2nd chapter of Jurafsky&Martin
- https://huggingface.co/spaces/Xenova/the-tokenizer-playground
- https://seohorsesense.com/free/lemmatization.php

TEXT NORMALIZATION CONT.

CS 445 - NATURAL LANGUAGE PROCESSING

Instructor: Dilara Keküllüoğlu
Fall 2024 – Week 2.2

Text Normalization Cont.

- Case-folding – lower casing the words
- Tokenization
- Lemmatization
- Stemming
- Sentence segmentation
- …

Text Cleaning

- Another step of pre-processing the data for given tasks
- Not every token is uniformly informative
- Focusing on informative tokens will increase the performance
- Limited processing powers – might need to cut some data

Photo by Matilda Wormwood

Stopwords

- We did see how ‘the’, ‘a’, ‘an’, etc. is dominating the corpora
- However, these are usually not very informative
- Stopwords – List of insignificant words to be removed from a text
- NLTK has stopword lists for 29 languages

English and Turkish Stopwords in NLTK

```
>>> ", ".join(stopwords.words('english'))
"i, me, my, myself, we, our, ours, ourselves, you, you're, you've, you'll, you'd, your, yours, yourself, yourselves, he, him, his, himself,
she, she's, her, hers, herself, it, it's, its, itself, they, them, their, theirs, themselves, what, which, who, whom, this, that, that'll, t
hese, those, am, is, are, was, were, be, been, being, have, has, had, having, do, does, did, doing, a, an, the, and, but, if, or, because, a
s, until, while, of, at, by, for, with, about, against, between, into, through, during, before, after, above, below, to, from, up, down, in,
 out, on, off, over, under, again, further, then, once, here, there, when, where, why, how, all, any, both, each, few, more, most, other, so
me, such, no, nor, not, only, own, same, so, than, too, very, s, t, can, will, just, don, don't, should, should've, now, d, ll, m, o, re, ve
, y, ain, aren, aren't, couldn, couldn't, didn, didn't, doesn, doesn't, hadn, hadn't, hasn, hasn't, haven, haven't, isn, isn't, ma, mightn,
mightn't, mustn, mustn't, needn, needn't, shan, shan't, shouldn, shouldn't, wasn, wasn't, weren, weren't, won, won't, wouldn, wouldn't"
>>> ", ".join(stopwords.words('turkish'))
'acaba, ama, aslında, az, bazı, belki, biri, birkaç, birşey, biz, bu, çok, çünkü, da, daha, de, defa, diye, eğer, en, gibi, hem, hep, hepsi,
 her, hiç, için, ile, ise, kez, ki, kim, mı, mu, mü, nasıl, ne, neden, nerde, nerede, nereye, niçin, niye, o, sanki, şey, siz, şu, tüm, ve,
veya, ya, yani'
>>>
```

Creating your stopwords list

- Depending on your dataset domain, you might want to remove certain words from processing
  - Customer Service = [help, DM, Please, Thanks,…]
  - Birthday Celebrations = [happy, birthday, belated, congratulations,…]
- You can easily add your own words to the stopwords list

Other Information to Remove

- Depending on your task, you might need to remove other information.
- URLs
- HTML Tags
- Usernames in social media data
- Emojis - Emoticons

Pattern Matching to Remove

- URLs
  - https?:\/\/\S+|www\.\S+
- HTML Tags
  - <.*?>
- Usernames
  - ^@[\w_]{1,15}

Emojis (😀) and Emoticons (XD)

- Removing emojis is a difficult task =)
- Lots of variety and new emerging uses
- You can use regex – link at further resources

```
def remove_emoji(string):
    emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', string)
```

Text Cleaning

- A normalization step for your text data
- Usually comes after tokenization but not all the time
  - For ex. You might want to do part of speech tagging and extra words could be informative to the POS tagger.
- Decision to remove what depends on the task!!

Emojis (😀) and Emoticons (XD) - again

- Could be really informative!
  - Sentiment Analysis,
  - Identity Studies,
  - Social Media Analysis
- How can we use them in text analysis?
- Emoji converters!
  - 😂 - face-with-tears-of-joy
  - 😱 - face-screaming-in-fear

Informal Text Conversion

- You can convert not only emoji/emoticons but also informal texts
  - AFAIK=As Far As I Know
  - IMHO=In My Honest/Humble Opinion
  - IJBOL=I Just Burst Out Laughing
  - ROFL=Rolling On The Floor Laughing

https://www.apoven.com/internet-slang-guide/

- Find the tokens – Replace with expanded versions

Spellcheckers

- Typos are common
- Causes separate words for same intended meaning
- We need to check for typos and fix them
- Available libraries to apply spellchecking
- How does spellchecker know the intended words?
- Edit distance! We will get more into it next week.

Following lectures

- Edit distance
- Text analysis with sentiment analysis example

Further Resources

- Robertson, Alexander, Walid Magdy, and Sharon Goldwater. "Black or White but never neutral: How readers perceive identity from yellow or skin-toned emoji." Proceedings of the ACM on Human-Computer Interaction 5.CSCW2 (2021): 1-23.
- Emoji remover - https://gist.github.com/slowkow/7a7f61f495e3dbb7e3d767f97bd7304b
- Emojipedia - https://emojipedia.org/
- Slang list for informal text conversion - https://github.com/rishabhverma17/sms_slang_translator/blob/master/slang.txt