Natural Language and Speech Processing

Lecture 1: Introduction
Basic Text Processing
Regular Expressions

Tanel Alumäe

Goal of the 1s part of the lecture

- Organization of the course
  - Prerequisites
  - Course “logistics”
  - Reading material
  - Assessment
- Explain what this course is about
- What you are expected to learn

Why natural language processing for computer science students?

- Natural language based interaction with devices is becoming very wide-spread
- The most innovative companies in natural language processing research:
  - Google
  - Facebook
  - Microsoft
  - Apple
  - OpenAI
- Making natural language based computer systems is typically complicated
- But also often more profitable, as it requires more knowledge and expertise
- The course will focus on statistical/neural NLP, where models will be learned from data

Course Logistics

- Instructor: Tanel Alumäe
  - Contact: tanel.alumae@taltech.ee
  - TalTech PhD in 2007
  - Associate Processor of Speech Processing
  - Head of the Laboratory of Language Technology (Department of Software Science)
  - Research interests: speech recognition, other speech processing problems, deep neural networks, NLP
- One lecture and one lab exercise per week
  - Lecture: Thursday 10:00, ICT-315
  - Lab: Thursday 12:00, ICT-403/404
- Register at https://moodle.taltech.ee/user/index.php?id=37215 (automatic registration from OIS)
- The course is given for the 9th time
- Course is evolving, new topics introduced this year

Prerequisites

- Required:
  - Statistics, probability theory
  - Linear algebra
  - Algorithms as data structures
  - **Programming (Python)**
    - Tutorial: http://cs231n.github.io/python-numpy-tutorial/
  - Knowing some machine learning is a plus
  - Linux highly recommended for some tasks
    - Tutorial: https://ryanstutorials.net/linuxtutorial/

Learning outcomes

- Official expected learning outcomes:
  - You are familiar with the main subfields of natural language processing (NLP) and speech processing
  - You understand the ideas behind the main NLP methods
  - You can select and apply a suitable method to solve a new NLP task efficiently
  - You can propose suitable model architectures and training methods for NLP and speech processing tasks
  - You understand how text-based generative AI (e.g. ChatGPT) works

Assessment

- Maximum total points: 100
- Short tests in lectures (about the topic of the last lecture): 10
- Three homeworks
  - Consists of both theoretical part as well as practical component (programming)
  - Each worth **15 points** of the final grade
- Independent project
  - Solve a practical NLP task
  - Write a report (a small research paper)
  - Worth 25 **points**
- Exam
  - Written theoretical part
  - Interview (questions about the submitted homework and project)
  - Worth **20 points**
- NB! In order to pass the course the student must get at least 50% from homeworks (threshold 23 points in total), at least 50% from the project (threshold 13 points) and at least 50% from the exam (threshold 10 points).

Lab exercises

- Lab exercises introduce you to some tools (mostly Python libraries) that are useful for solving practical NLP tasks
  - Pynini (for working with finite state transducers)
  - EstNLTK (for Estonian language processing)
  - spaCy (for processing other languages)
  - Sklearn (for text classification)
  - **Pytorch (for neural networks)**
  - HuggingFace Transformers

Homeworks

- Practical tasks: e.g., write a simple program that handles some NLP problem (typically extends lab exercises)
- Submitted via Moodle
- **AI use is allowed, but understanding is required**
- NB! Must be completed by the student alone (consulting with co-students is of course allowed)
- **Too similar submissions result in zero points for both submitters, unrecoverable**
- Plagiarism is not tolerated
  - Also applies to program code
- **NB!** Homeworks will be fully accepted only after short interviews (conducted in lab sessions).
  - I can ask questions about every program line!
  - Failure to explain your code will result in a deduction of points (proportional to the severity of the failure)

Project

- The goal of the project is to give you:
  - Deeper experience in some particular NLP field
  - Experience with scientific writing
- Consists of code and report
- Report should include all the necessary components of a research paper – introduction, description of the problem, description of the solution, evaluation/experiments, conclusion
  - Analysis of previous/related work can be skipped
- Report can be in Estonian or English, around 3000 words (2 pages with 2 columns, template will be given)
- Submitted project will be reviewed during the exam
- To see (mostly) excellent student project samples, check https://nlp.stanford.edu/courses/cs224n/
- You may use external libraries (and it is encouraged), but you must acknowledge all such usages

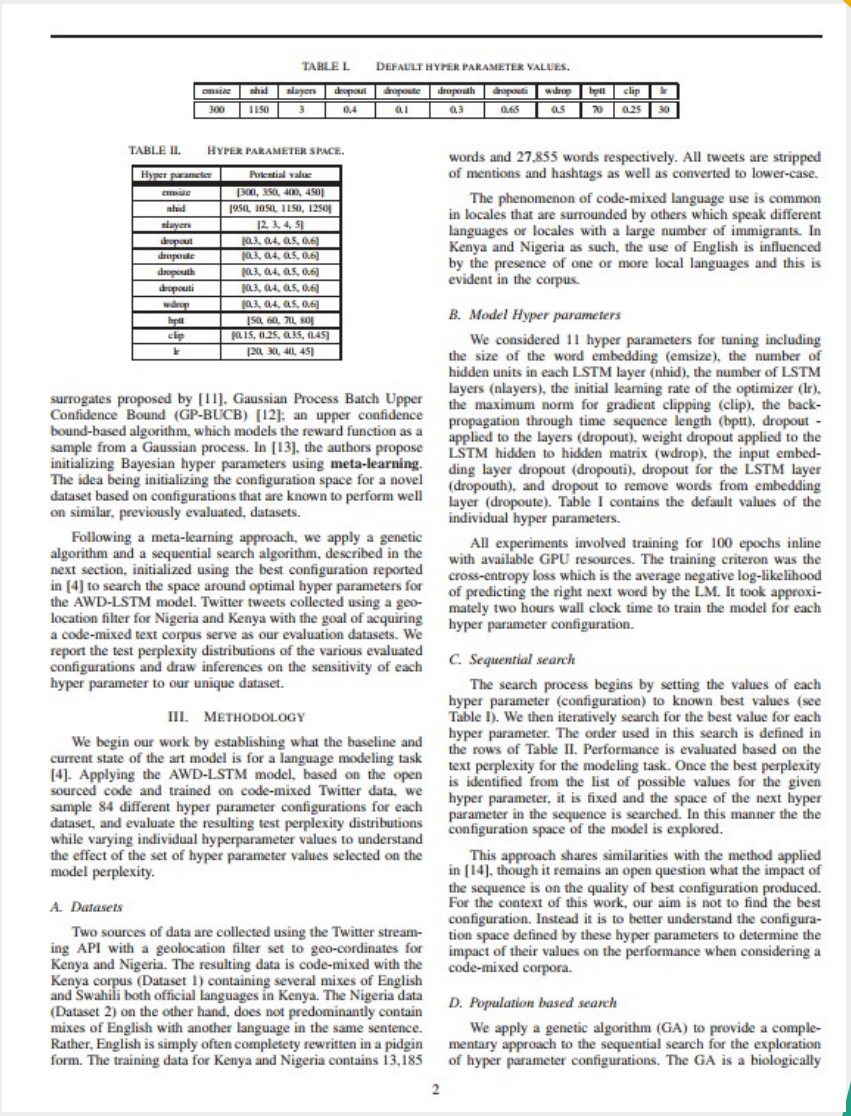


TABLE I. DEFAULT HYPER PARAMETER VALUES.

emsize	nhid	nlayers	dropout	dropoute	dropouth	dropouti	wdrop	bptt	clip	lr
300	1150	3	0.4	0.1	0.3	0.65	0.5	70	0.25	30

TABLE II. HYPER PARAMETER SPACE.

Hyper parameter	Potential value
emsize	[300, 350, 400, 450]
nhid	[950, 1050, 1150, 1250]
nlayers	[2, 3, 4, 5]
dropout	[0.3, 0.4, 0.5, 0.6]
dropoute	[0.3, 0.4, 0.5, 0.6]
dropouth	[0.3, 0.4, 0.5, 0.6]
dropouti	[0.3, 0.4, 0.5, 0.6]
wdrop	[0.3, 0.4, 0.5, 0.6]
bptt	[50, 60, 70, 80]
clip	[0.15, 0.25, 0.35, 0.45]
lr	[20, 30, 40, 45]

surrogates proposed by [11], Gaussian Process Batch Upper Confidence Bound (GP-BUCB) [12]; an upper confidence bound-based algorithm, which models the reward function as a sample from a Gaussian process. In [13], the authors propose initializing Bayesian hyper parameters using **meta-learning**. The idea being initializing the configuration space for a novel dataset based on configurations that are known to perform well on similar, previously evaluated, datasets.

Following a meta-learning approach, we apply a genetic algorithm and a sequential search algorithm, described in the next section, initialized using the best configuration reported in [4] to search the space around optimal hyper parameters for the AWD-LSTM model. Twitter tweets collected using a geo-location filter for Nigeria and Kenya with the goal of acquiring a code-mixed text corpus serve as our evaluation datasets. We report the test perplexity distributions of the various evaluated configurations and draw inferences on the sensitivity of each hyper parameter to our unique dataset.

III. METHODOLOGY

We begin our work by establishing what the baseline and current state of the art model is for a language modeling task [4]. Applying the AWD-LSTM model, based on the open sourced code and trained on code-mixed Twitter data, we sample 84 different hyper parameter configurations for each dataset, and evaluate the resulting test perplexity distributions while varying individual hyperparameter values to understand the effect of the set of hyper parameter values selected on the model perplexity.

A. Datasets

Two sources of data are collected using the Twitter streaming API with a geolocation filter set to geo-cordinates for Kenya and Nigeria. The resulting data is code-mixed with the Kenya corpus (Dataset 1) containing several mixes of English and Swahili both official languages in Kenya. The Nigeria data (Dataset 2) on the other hand, does not predominantly contain mixes of English with another language in the same sentence. Rather, English is simply often completety rewritten in a pidgin form. The training data for Kenya and Nigeria contains 13,185 words and 27,855 words respectively. All tweets are stripped of mentions and hashtags as well as converted to lower-case.

The phenomenon of code-mixed language use is common in locales that are surrounded by others which speak different languages or locales with a large number of immigrants. In Kenya and Nigeria as such, the use of English is influenced by the presence of one or more local languages and this is evident in the corpus.

B. Model Hyper parameters

We considered 11 hyper parameters for tuning including the size of the word embedding (emsize), the number of hidden units in each LSTM layer (nhid), the number of LSTM layers (nlayers), the initial learning rate of the optimizer (lr), the maximum norm for gradient clipping (clip), the back-propagation through time sequence length (bptt), dropout - applied to the layers (dropout), weight dropout applied to the LSTM hidden to hidden matrix (wdrop), the input embedding layer dropout (dropouti), dropout for the LSTM layer (dropouth), and dropout to remove words from embedding layer (dropoute). Table I contains the default values of the individual hyper parameters.

All experiments involved training for 100 epochs inline with available GPU resources. The training criteron was the cross-entropy loss which is the average negative log-likelihood of predicting the right next word by the LM. It took approximately two hours wall clock time to train the model for each hyper parameter configuration.

C. Sequential search

The search process begins by setting the values of each hyper parameter (configuration) to known best values (see Table I). We then iteratively search for the best value for each hyper parameter. The order used in this search is defined in the rows of Table II. Performance is evaluated based on the text perplexity for the modeling task. Once the best perplexity is identified from the list of possible values for the given hyper parameter, it is fixed and the space of the next hyper parameter in the sequence is searched. In this manner the the configuration space of the model is explored.

This approach shares similarities with the method applied in [14], though it remains an open question what the impact of the sequence is on the quality of best configuration produced. For the context of this work, our aim is not to find the best configuration. Instead it is to better understand the configuration space defined by these hyper parameters to determine the impact of their values on the performance when considering a code-mixed corpora.

D. Population based search

We apply a genetic algorithm (GA) to provide a complementary approach to the sequential search for the exploration of hyper parameter configurations. The GA is a biologically

2

Default project topic

- The default independent project is multi-label classification of German book summaries (blurbs)
- It's about multi-label classification of so-called blurbs -- short summaries of books (think about an online bookstore). The language of the blurbs and the labels is German
- The task is multi-label classification: that is, each blurb can be classified to one or many classes. Actually, the classes are hierarchical (e.g., Fantasy -> Urban Fantasy) but we don't really use the hierarchical nature of the classes
- Actually, GermEval 2019 Task 1 has two tasks: task A is about predicting the most general class, and task B is about predicting all labels. We only focus on task B
- There are a total of 343 different categories and sub-categories.
- The paper describing the task is available here
- More information and starter code is available here

Other possible project topics

- Identify offensive social media language
  - https://competitions.codalab.org/competitions/20011
- Semantic similarity of texts
  - Predict which Quora pairs of questions contain two questions with the same meaning
  - https://www.kaggle.com/c/quora-question-pairs
- Community question answering
  - http://alt.qcri.org/semeval2017/task3/
  - Given (i) a new question and (ii) a large collection of question-answer threads created by a user community, rank the answer posts that are most useful for answering the new question
- Text normalization
  - Convert, e.g. “A baby giraffe is 6ft tall” → “A baby giraffe is 6 feet tall”
  - English: https://www.kaggle.com/c/text-normalization-challenge-english-language
  - Russian: https://www.kaggle.com/c/text-normalization-challenge-russian-language
- Named Entity Recognition on code-switched (mixed-language) data
  - https://competitions.codalab.org/competitions/18724
- Multilingual offensive language detection
  - https://sites.google.com/site/offensevalsharedtask/results-and-paper-submission

Project assessment

- Technical quality of writing
  - Structure
  - Description of the problem
  - Description of the solution
  - Description of the dataset
  - Presentation of results
  - Analysis and discussion
- Complexity of the used methods
  - **You have to use something more than a very straightforward approach to get perfect score**
- Implementation
- Results, compared to the state-of-the-art (if available)

Amount of work

- 6 ECTS = 6*26=156 hours of intensive work
  - On average, assuming prerequisites are mostly fulfilled
- 16 lectures = 2*15=32h
- 16 lab exercises = 2*16=32h
- 10 hours per homework = 3*10=30h
- **56 hours for project**
- 6 hours for studying for the exam
- Total: 156 hours

Dates

- Home assignments:
  - March 6
  - April 3
  - May 5
- Project:
  - Three days before you come to the exam

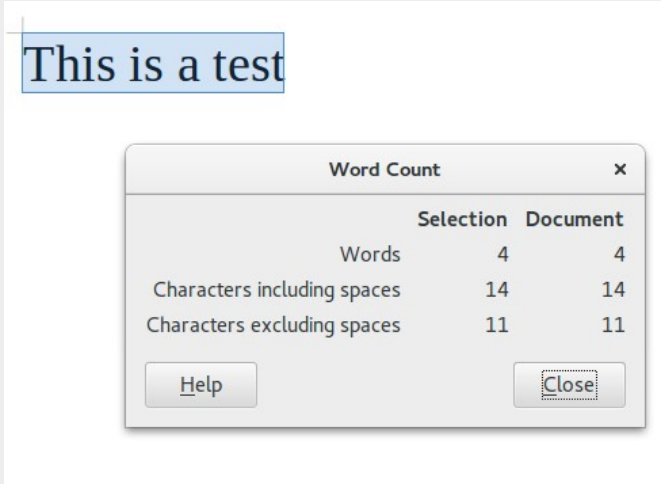
Late policy: each late day reduces points by 10%. Decay is linear. E.g., if you submit a homework 2 days late, and your real score is 11, your reduced score is 11*0.80=8.8≈9. That is, the number of maximum late days is 10. **NB! Project deadline is strict!**

Reading material

- Main textbook: *Speech and Language Processing* by Jurafsky and Martin
  - Draft of 3rd edition available at https://web.stanford.edu/~jurafsky/slp3/
- Lewis Tunstall, Leandro von Werra, and Thomas Wolf. *Natural Language Processing with Transformers*
- Stanford course: CS224N: Natural Language Processing with Deep Learning
  - All materials (lecture videos, assignments, background material) available: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1254/
- If you have no background in neural networks but would like to take the course anyway, this site will give you more background:
  - Michael A. Nielsen. Neural Networks and Deep Learning

What is natural language processing

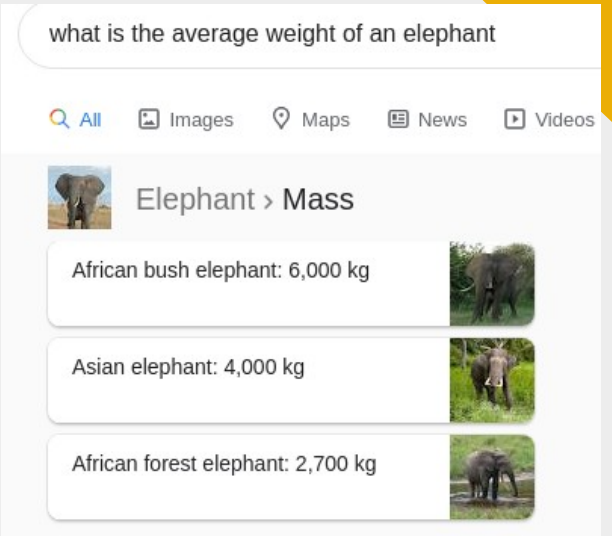
- NLP is a collective term referring to automatic computational processing of human languages
- Computational techniques that use some kind of knowledge of language
  - E.g.: counting words in document (what is a *word?*)
  - “Open the pod bay doors, HAL”



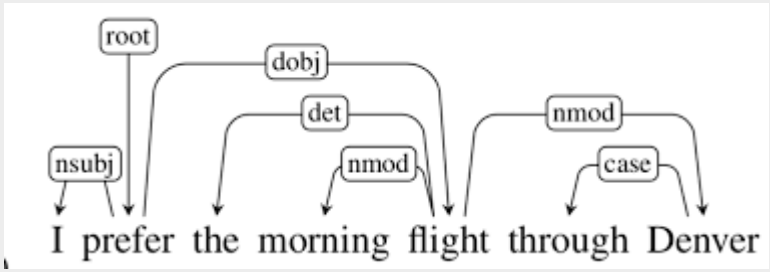
Some NLP tasks

- **Information retrieval** (e.g. Google)
- **Document classification**
  - e.g. *sports* vs *business*, *offensive* vs *unoffensive* comment in social media
- **Spelling and grammar checking and correction**
- **Machine translation**
- **Information extraction** (e.g. "*Yesterday, New York based Foo Inc. announced their acquisition of Bar Corp.*" → Merger(Foo, Bar, date))
- **Open-ended dialogue (ChatGPT)**
- **Speech recognition** (speech-to-text)
- **Speech synthesis** (text-to-speech)
- **Speaker recognition** (who speaks?)

Some more NLP tasks



- **Question answering**
  - Closed domain
    - Given: text about the subject, and a question
    - Find: answer in the text (or no answer)
  - Open domain
- **Summarization**
  - Extractive: extract most important sentences
  - Abstractive: “true” summarization
- **Keyphrase extraction**
- **Parsing**



Space Odyssey 2001

- Let’s look at the *Space Odyssey 2001* *ex*ample in more detail: https://youtu.be/dSIKBliboIo?t=52

Open the pod bay doors, HAL

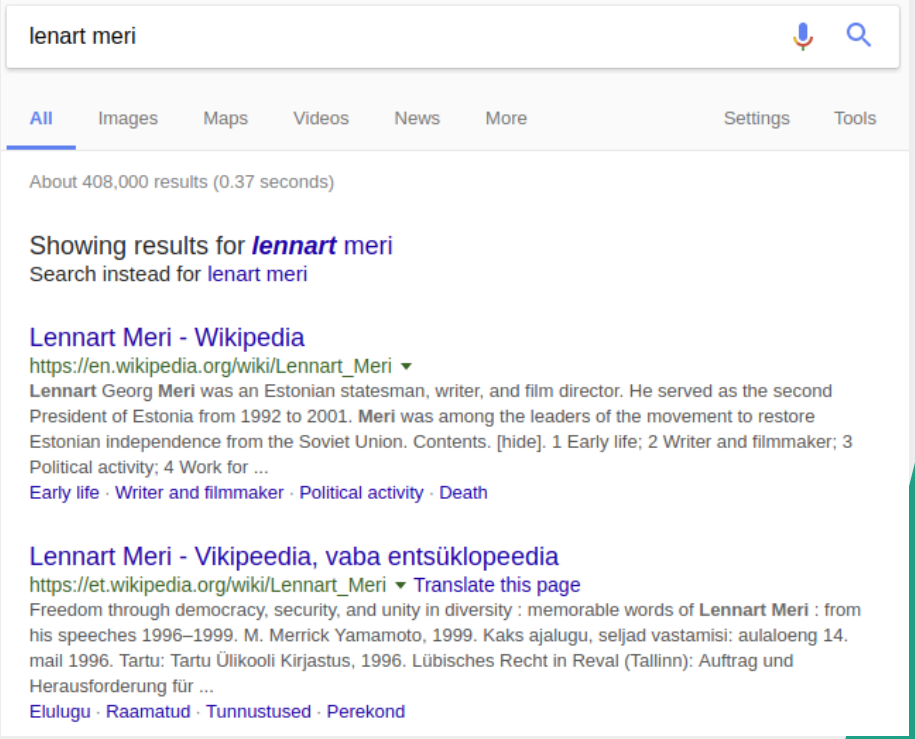
NLP components of HAL

I'm sorry Dave, I'm afraid I can't do that

- Speech recognition (speech-to-text)
- Audio-visual speech recognition (lip reading)
- Speech synthesis (text-to-speech)
- Analyze the structure of human language
  - Dialog act type identification (identify that the utterance is a request *vs* a simple statement)
  - Syntax analysis (identify that the "pod bay doors" must be "opened")
  - Semantic analysis (map "pod bay doors" and "open" to certain objects and actions that HAL knows about)
- Natural language generation
  - Discourse (use if "that" in the response, *vs* "bay doors")
  - Pragmatics (use of "I'm afraid I can't do that" *vs* "No")

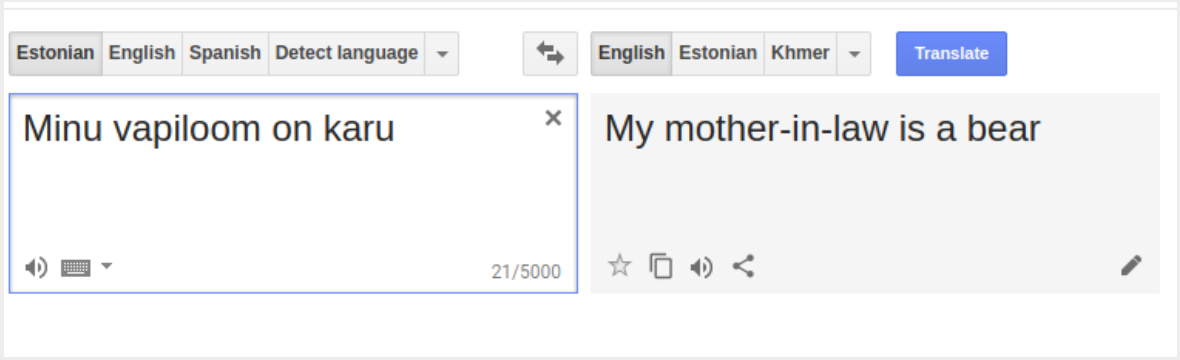
Information Retrieval

- Retrieving and ranking documents, given a search query (e.g. Google)
- Possible NLP components:
  - Spell-check and auto-correct the query
  - Lemmatize the query and documents (e.g., for a query *pöial*, also show documents containing *pöidla*
  - Synonyms and closely related terms, e.g. for query *mad cow disease* also show documents containing *BSE* and *bovine spongiform encephalopathy*

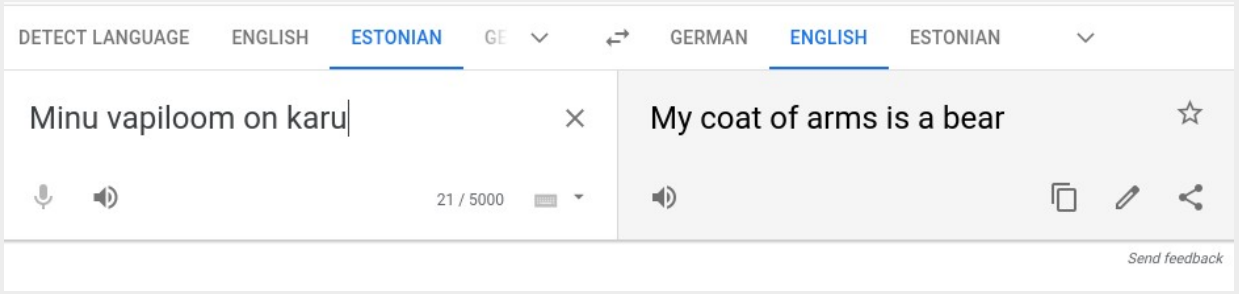


Machine translation

- Used as standalone, or as an aid for human translators
- Great progress in recent years
  - 2019

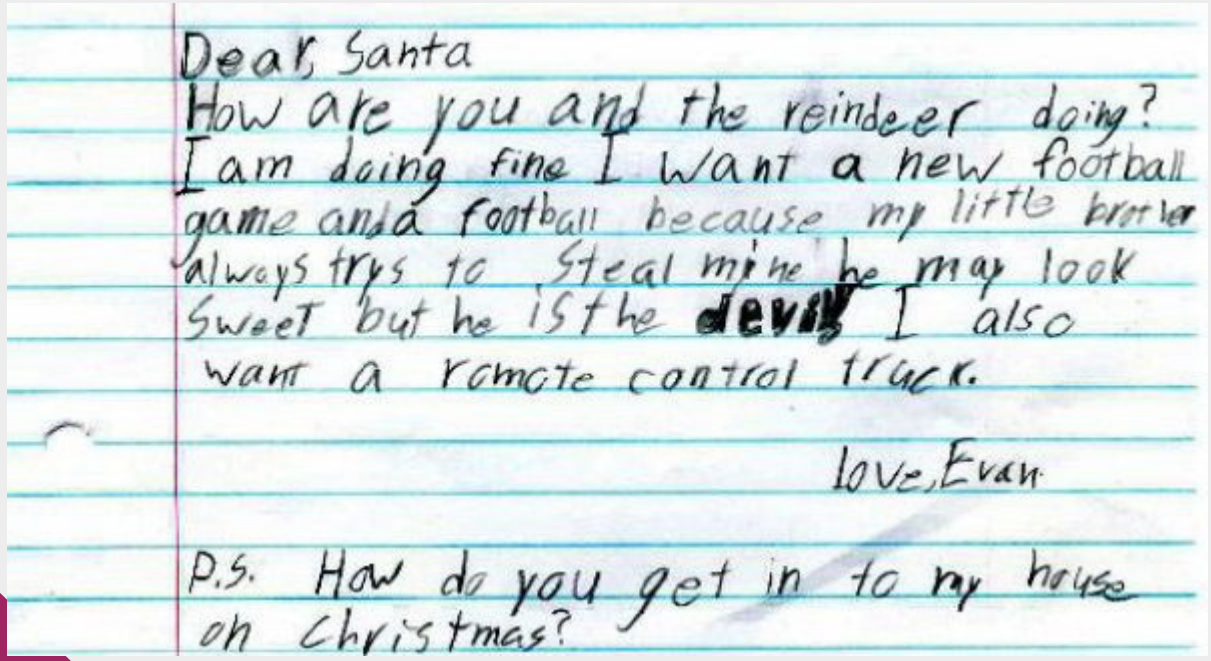


  - 2021



Information extraction

- Finding structured information from unstructured or semi-structured text



→ Football game
Football
Remote control truck

Why is processing natural language difficult?

- Full human-level error-free NLP is **AI-complete**: it requires human-level AI
- It is difficult to represent all the different meanings in natural language abstractly:
  - *Tallinn is the capital of Estonia* → Capital(*Estonia*, *Tallinn*)
  - *Jerusalem is key to any peace agreement → ???*
- Lots of relationships between concepts
- Different ways to represent the same meaning:
  - *Firefighter dies battling huge California wildfire*
  - *Firefighter dies, thousands more take on California blaze*
  - *Cal fire family mourns San Diego firefighter killed in Thomas fire*
  - *Firefighter killed battling Thomas fire*

Question

- Read the sentence:

  “I saw a man with a telescope”

- How many different interpretations are there?

- Paraphrase: “Using a telescope, I saw a man.”
  - Meaning: The instrument for the seeing action is a telescope that I am using.
- Paraphrase: “I saw a man who had (or was carrying/using) a telescope.”
  - Meaning: The man himself is in possession of the telescope.
- Paraphrase: “I use a saw on a man who had a telescope,” or “Using a saw, I cut a man, and there was also a telescope involved.”
- Paraphrase: I perform the action of sawing a man, and I employ a telescope as my “saw.”
  - Comment: This is semantically bizarre (“using a telescope as a saw”), but syntactically it’s on par with “I used X to do Y with a saw,” just substituting telescope for saw as the instrument phrase.

Why is processing natural language easy?

- Language is highly redundant:
  - E.g. movie review: *I hated this movie. Hated hated hated hated hated this movie. Hated it. Hated every simpering stupid vacant audience-insulting moment of it. Hated the sensibility that thought anyone would like it. Hated the implied insult to the audience by its belief that anyone would be entertained by it.* → **Rating: 1/5**
- Many crude methods provide surprisingly good results

Ambiguity

Natural language is often highly ambiguous at many levels

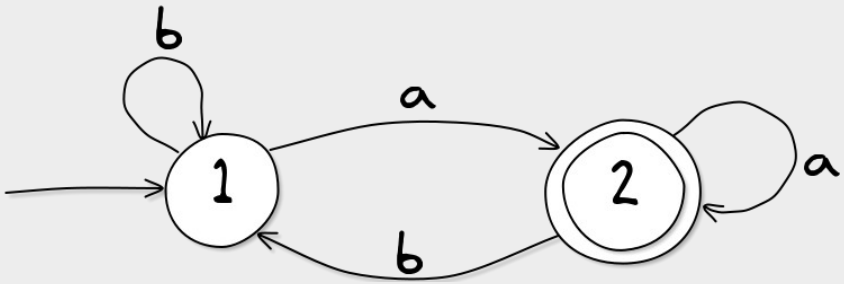
- Speech:
  - *The tail of the dog ↔ The tale of the dog*
  - *It's easy to recognize speech ↔ It's easy to wreck a nice beach ↔ It's easy to wreck an ice beach*
  - Estonian: *sõidutasime autoga ↔ sõidutas imeautoga*
- Lexical: e.g. *back:* noun (*my back*) vs adverb (*to back away*) vs adjective (*back door*)
- Syntactic: *I heard his cell phone ring* ***in my office***
- Semantic: *I don't like it when my father* ***smokes***
- Metonymy: ***White House*** *said to be considering replacing Tillerson with CIA chief*
- Metaphors: *food for thought*, *lose one's way*

Course contents

- Finite state automata, regular expressions
- Morphology
- N-gram language models
- Text classification with statistical models
- Word classification
- Word embeddings, neural network language models
- Convolutional neural network models for sentences
- Recurrent neural networks for NLP
- Neural attention mechanism, Transformer models
- Large language models: BERT, BART, GPT
- Machine translation
- Large language model, e.g. ChatGPT
- Speech recognition
- Speaker and language recognition
- Speech synthesis (?)

Finite state automata, regular expressions

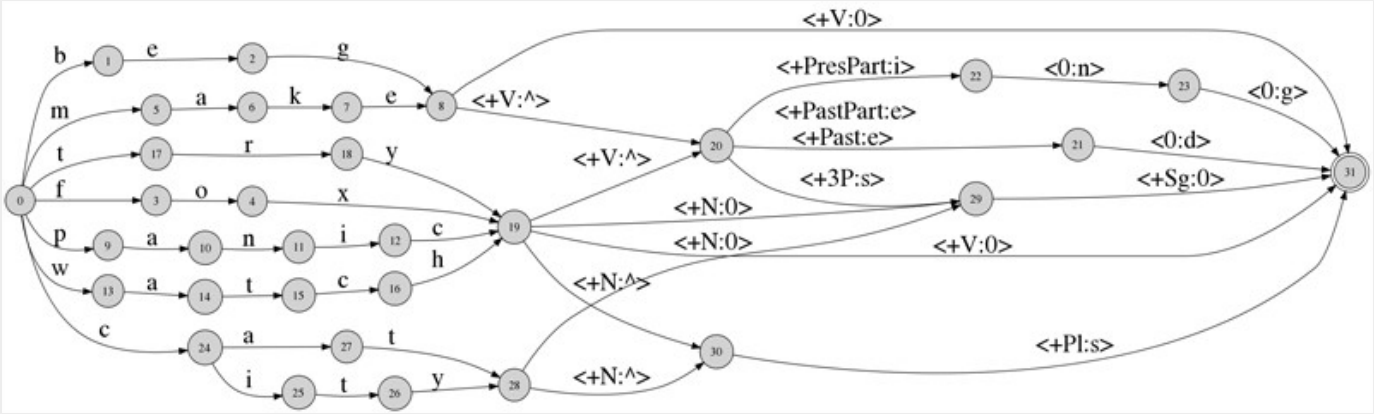
Symbol	Example	Definition
.	.␣␣,␣A.C␣	match any character
[]	[ABCabc]␣,␣[A-Ca-c]	match any char in the list
[^]	[^Z]␣,␣[^XYZ]␣,␣[^x-z]	match all except the chars in the list
^	^C␣,␣^A.*	next token must be the first part of string
$	[CO]G$	prev token must be the last part of string
*	C*␣,␣[ab]*	match 0 or more copies of prev char or regular expression token
+	C+␣,␣[ab]+	match 1 or more copies of the prev token
|	C|O	match either the 1st token or the 2nd
\(\)	\(CA\)+	combines multiple tokens into one



Morphology, finite state transducers

- Morphological parsing and generation

Input word	Output morphemes
cats	cat +N +PL
cat	cat + N + SG
cities	city + N + PL
walks	walk + V + 3SG
cook	cook +N +SG or cook +V



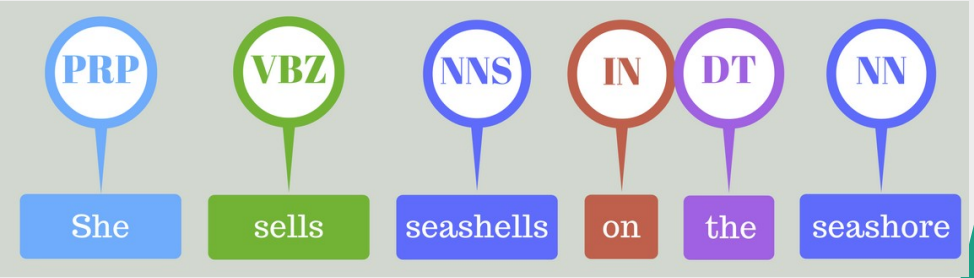
Word classification with statistical models

- Named entity recognition

In 1917, Einstein applied the general theory of relativity to model the large-scale structure of the universe. He was visiting the United States when Adolf Hitler came to power in 1933 and did not go back to Germany, where he had been a professor at the Berlin Academy of Sciences. He settled in the U.S., becoming an American citizen in 1940. On the eve of World War II, he endorsed a letter to President Franklin D. Roosevelt alerting him to the potential development of "extremely powerful bombs of a new type" and recommending that the U.S. begin similar research. This eventually led to what would become the Manhattan Project. Einstein supported defending the Allied forces, but largely denounced using the new discovery of nuclear fission as a weapon. Later, with the British philosopher Bertrand Russell, Einstein signed the Russell–Einstein Manifesto, which highlighted the danger of nuclear weapons. Einstein was affiliated with the Institute for Advanced Study in Princeton, New Jersey, until his death in 1955.

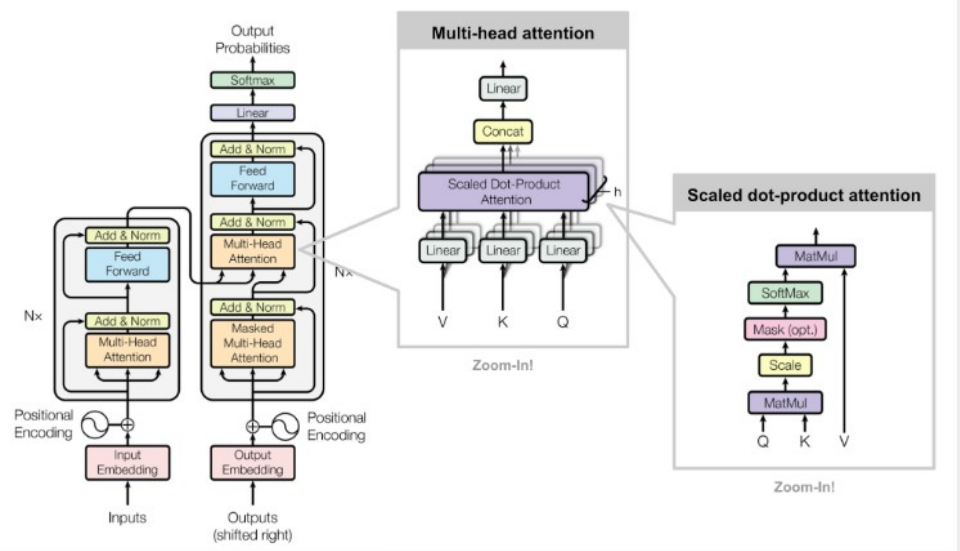
Tag colours:
LOCATION TIME PERSON ORGANIZATION MONEY PERCENT DATE

- Morphological tagging



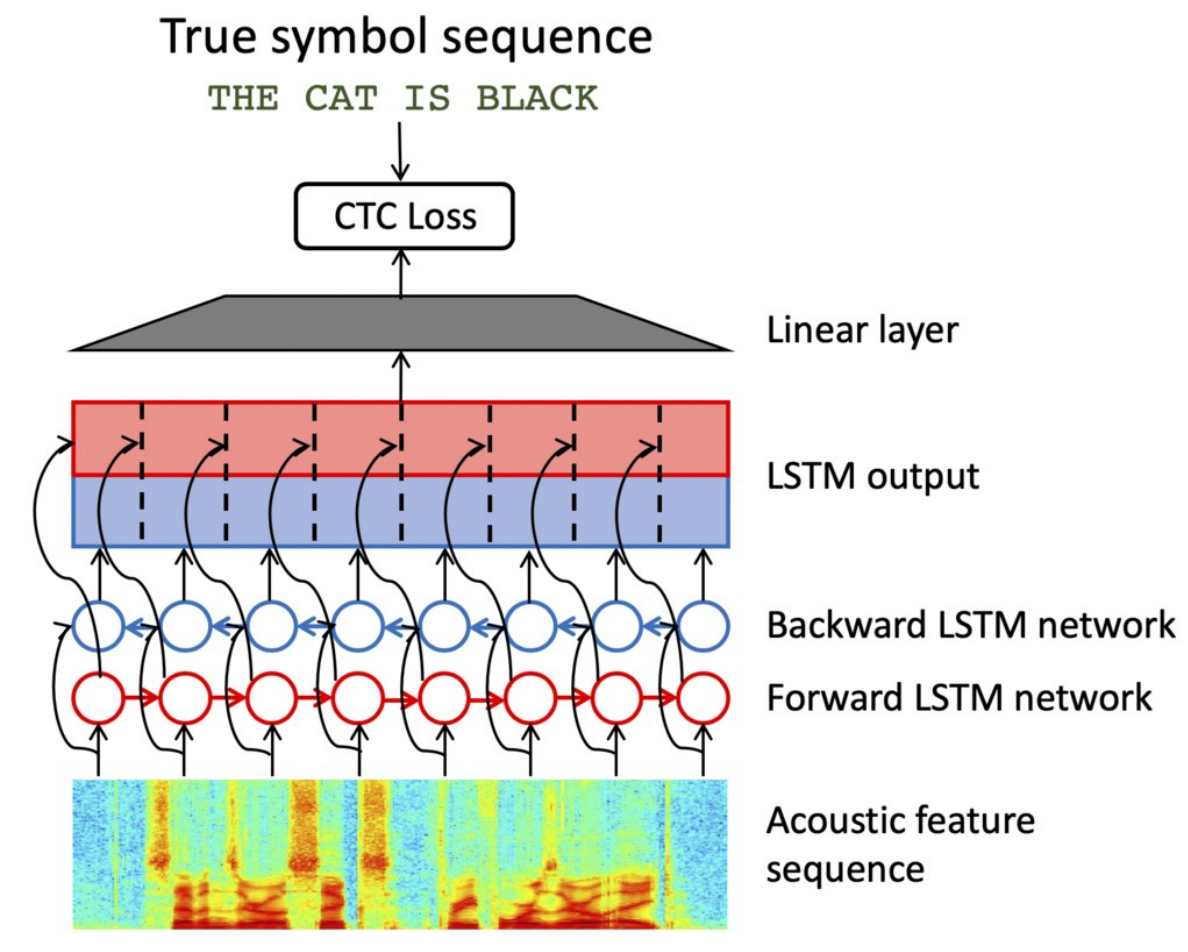
Transformer models, BERT

- Neural attention: mechanism to process sequences without recurrent neural nets
- Most popular neural building block in modern NLP
- BERT (and many similar models): pretrained “self-supervised” model
  - Trained on large amounts of internet texts and books
  - Trained to predict “masked” words
  - Easy to port and finetune to various tasks



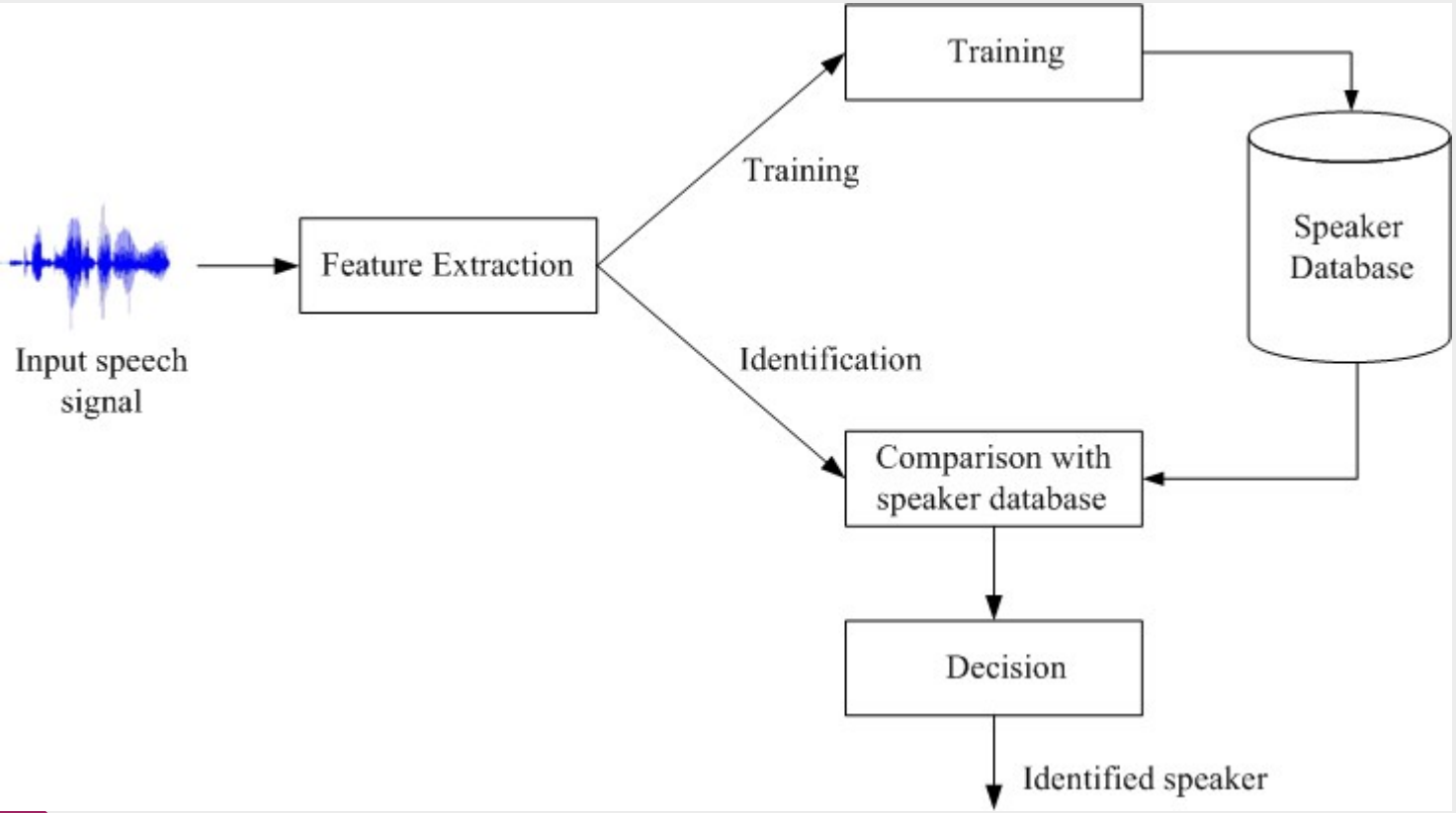
Speech recognition

- Converting speech into text



Speaker and language identification

- Identifying speaker or spoken language (e.g., Estonian, English or Russian?) from speech



Summary

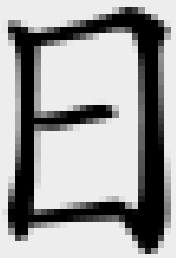
- Natural language processing is difficult
- But very interesting
- Nowadays, many new methods and ideas in AI (particularly deep learning) come from the NLP community
- Learning modern NLP gives you a good background for doing any machine learning or data science work professionally
- Interesting Master thesis topics available (contact me as soon as possible if interested – I have limited supervising capacity)

Encoding text in computer

Writing systems

- Different types of writing systems
  - Alphabetic (English, Estonian, Russian)
  - Syllabic: symbols represent *syllables* (Japanese *kana*, Cherokee)
  - Logographic: single symbols represent complete words (Japanese kanji, Chinese, Ancient Egyptian), but also symbols such as $, %, €

ア	阿	イ	伊	ウ	宇	エ	江	オ	於
カ	加	キ	機	ク	久	ケ	介	コ	己
サ	散	シ	之	ス	須	セ	世	ソ	曽
タ	多	チ	千	ツ	川	テ	天	ト	止
ナ	奈	ニ	仁	ヌ	奴	ネ	祢	ノ	乃
ハ	八	ヒ	比	フ	不	ヘ	部	ホ	保
マ	末	ミ	三	ム	牟	メ	女	モ	毛
ヤ	也			ユ	由			ヨ	與
ラ	良	リ	利	ル	流	レ	礼	ロ	呂
ワ	和	𠮷	井			ヱ	恵	ヲ	乎
ン	尓								



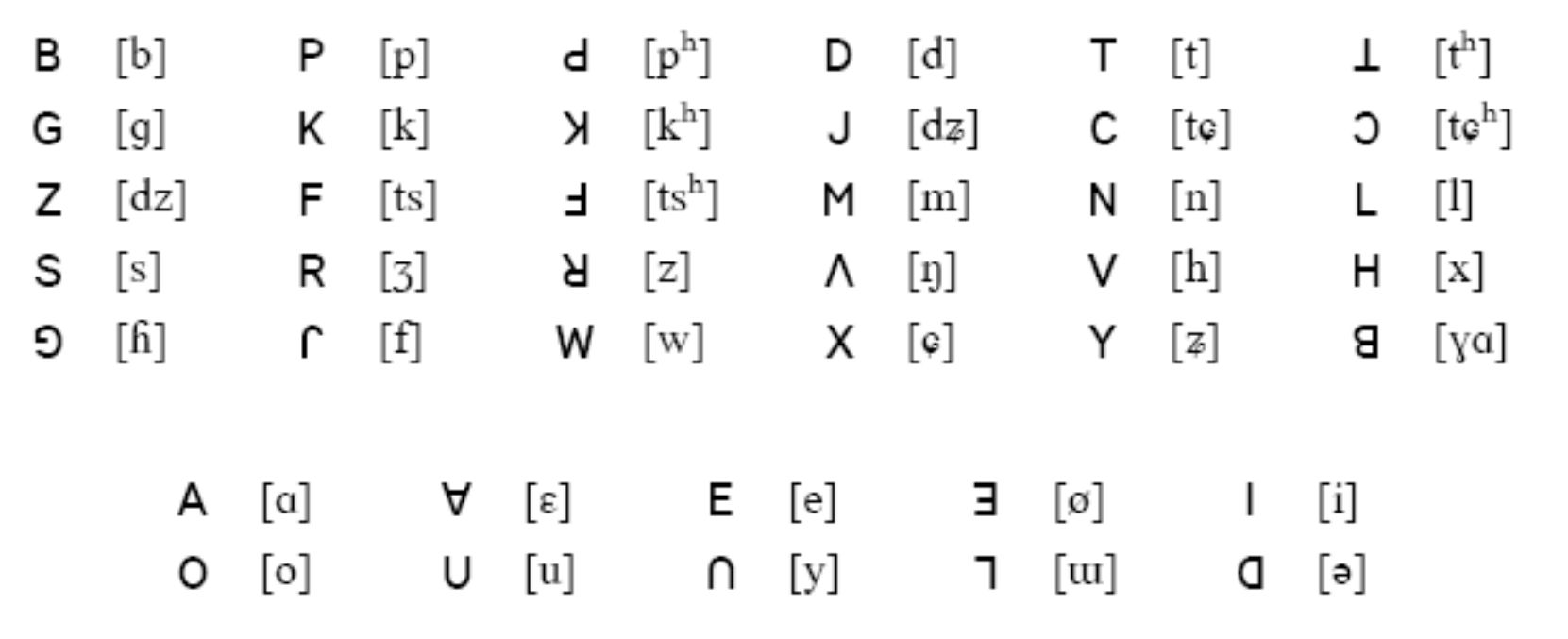
Alphabetic symbols

- Alphabets (phonemic alphabets)
  - Represent all sounds (consonants and vowels)
- Abjads (consonant alphabets)
  - Represent consonants only (in some cases also some vowels)
  - vowel sounds have to be inferred by the reader
  - Examples: Arabic, Aramaic, Hebrew
  - Arabic alphabet is used with many other languages, e.g. Kurdish, Pashto, Kazakh (in some areas)

العربية

Alphabet example

- Fraser alphabet, used to write *Lisu*, a Tibeto-Burman language spoken by about 700 000 people in South-East Asia



Encoding written language in computer

- Using a single byte (8 bits), one can represent 256 different characters
- Earliest character encoding: ASCII
  - Uses only 7 bits = 128 possible characters
  - English letters, numbers, punctuation marks
  - First 31 codes reserved for control characters (backspace, line feed, etc)

0 NUL	10 DLE	20	30 0	40 @	50 P	60 `	70 p
1 SOH	11 DC1	21 !	31 1	41 A	51 Q	61 a	71 q
2 STX	12 DC2	22 "	32 2	42 B	52 R	62 b	72 r
3 ETX	13 DC3	23 #	33 3	43 C	53 S	63 c	73 s
4 EOT	14 DC4	24 $	34 4	44 D	54 T	64 d	74 t
5 ENQ	15 NAK	25 %	35 5	45 E	55 U	65 e	75 u
6 ACK	16 SYN	26 &	36 6	46 F	56 V	66 f	76 v
7 BEL	17 ETB	27 '	37 7	47 G	57 W	67 g	77 w
8 BS	18 CAN	28 (	38 8	48 H	58 X	68 h	78 x
9 HT	19 EM	29)	39 9	49 I	59 Y	69 i	79 y
A LF	1A SUB	2A *	3A :	4A J	5A Z	6A j	7A z
B VT	1B ESC	2B +	3B ;	4B K	5B [	6B k	7B {
C FF	1C FS	2C ,	3C <	4C L	5C \	6C l	7C |
D CR	1D GS	2D -	3D =	4D M	5D]	6D m	7D }
E SO	1E RS	2E .	3E >	4E N	5E ^	6E n	7E ~
F SI	1F US	2F /	3F ?	4F O	5F _	6F o	7F DEL

Different coding systems

- ASCII contains only English letters
- What about other languages and characters, such as *õ, š, Ш, 见*?
- ASCII was extended to 8-bit, to include extra characters
- However, this gives only 128 extra symbols
- Result: different encodings for different languages:
  - ISO-8859-1: includes extra letters needed for French, German, Spanish
  - ISO-8859-7: Greek alphabet
  - ISO-8859-8: Hebrew alphabet
  - ISO-8859-13: Baltic languages (including Estonian)

Problem with character sets

- Conflicts
  - Two different encodings can use the same number for different characters (e.g. ***š*** in ISO-8859-13 is encodes as 240, which in ISO-8859-1 corresponds to **ð**)
  - Sometimes, the same character corresponds to different codes in different character sets
- Also, a text file using a language-specific encoding (e.g. ISO-8859-13) cannot mix in letters from other encodings (e.g. Greek or Chinese)
- Also, when opening a text file using a ISO-8859-X encoding, you have to ***know*** the encoding, otherwise the content will be garbled
  - That's why you sometimes see something like "*Öösel sõin ma ðokolaadi*" when opening text files

Unicode

- Unicode tries to fix this mess by having a single representation for every possible character in all world languages
- The latest version of Unicode (17.0) contains 159801 characters covering 172 scripts
  - E.g., Chinese has ~100000 characters in Unicode
- Unicode uses 32 bits, meaning we can store 2^{32} = over 4 billion different characters

Unicode, cont.

- Each unicode code points consists of a number and a description
- The actual visual representation (glyph) is not stored in Unicode – this is determined by the ***font***

U+0061	a	LATIN SMALL LETTER A
U+0062	b	LATIN SMALL LETTER B
U+0063	c	LATIN SMALL LETTER C
U+00F9	ù	LATIN SMALL LETTER U WITH GRAVE
U+00FA	ú	LATIN SMALL LETTER U WITH ACUTE
U+00FB	û	LATIN SMALL LETTER U WITH CIRCUMFLEX
U+00FC	ü	LATIN SMALL LETTER U WITH DIAERESIS
U+8FDB	进	
U+8FDC	远	
U+8FDD	违	
U+8FDE	连	
U+1F600	😀	GRINNING FACE
U+1F00E	🀎	MAHJONG TILE EIGHT OF CHARACTERS

UTF-8

- The most straightforward way to store Unicode text is to use UTF-32, where each character is represented using 4 bytes
- However, this uses up a lot of memory, especially for English text: 4 times more than ASCII
- Therefore, 90% of texts on the web is stored using **UTF-8**:
  - Characters are represented using one or more bytes
  - Highest bit of the first byte is used a flag:
    - Highest bit 0: single character (0xxxxxxx)
    - Highest bit 1: part of a multibyte character
      110xxxxx + 10xxxxxx
      1110xxxx + 10xxxxxx + 10xxxxxx
  - Nice consequence: ASCII text is valid UTF-8 text
  - Estonian text in UTF-8 takes only slightly more space than ISO-8859-13 (only *öäüõš* use 2 bytes)
  - The higher is the Unicode ID of the character, the more bytes it uses in UTF-8

ISO-8859-13 *vs* UTF-8 example

- “käru”
- ISO-8859-13:
  01101011 (k) 11100100 (ä)
  01110010 (r) 01110101 (u)
- UTF-8:
  01101011 (k) 11000011 10100100 (ä)
  01110010 (r) 01110101 (u)

Problems with Unicode

- Very similar characters with different codepoints (called “confusables”)
  - e.g. “A” has codepoints in both Greek and Latin script
  - Can be used to make **spoofing attacks**
  - String matching causes **hard-to-debug errors**
- Characters with diacritics can be written in two ways:
  - The actual codepoint for the character
    - e.g. õ (00F5)
  - The base character codepoint + “combining character”
    - E.g. o (006F) + ˜ (0342) results in õ (006F 0342)

Regular expressions

Regular expressions

- Most important tool for describing *text patterns*
- Can be used to defining string patterns, e.g. that match:
  - Dollar amounts: $199, $25, $24.99
  - Western style person names: Donald Trump, Toomas Hendrik Ilves, George W. Bush (but be careful with names)
  - Hashtags: #yolo, #nlp, #regexp
- And for transforming:
  - $199 → 199 dollars
  - Donald Trump → D. Trump, Toomas Hendrik Ilves → T. H. Ilves, George W. Bush → G. W. Bush
- Highly useful in practical NLP, e.g. for data cleaning and transforming between simple formats

Basic Regular Expression Patterns

- Simplest regular expression is a sequence of simple characters, e.g. /woodchucks/ (note that '/' characters are not part of the regex, they simply denote that the part between /../ is a regex)
- Regular expressions are case-sensitive

RE	Example Patterns Matched
/woodchucks/	"interesting links to woodchucks and lemurs"
/a/	"Mary Ann stopped by Mona's"
/!/	"You've left the burglar behind again!" said Nori

Sets

- Square braces denote a set of characters from which one has to match:

RE	Match	Example Patterns
/[wW]oodchuck/	Woodchuck or woodchuck	"Woodchuck"
/[abc]/	'a', 'b', *or* 'c'	"In uomini, in soldati"
/[1234567890]/	any digit	"plenty of 7 to 5"

- Sets can also include a character range (e.g. [A-Z], be careful with *öäüõ*)

RE	Match	Example Patterns Matched
/[A-Z]/	an upper case letter	"we should call it 'Drenched Blossoms' "
/[a-z]/	a lower case letter	"my beans were impatient to be hoed!"
/[0-9]/	a single digit	"Chapter 1: Down the Rabbit Hole"

Negation and disjunction

- Caret (^) specifies what a single symbol cannot be
- Can be used for excluding single symbols or character sets

RE	Match (single characters)	Example Patterns Matched
/[^A-Z]/	not an upper case letter	"Oyfn pripetchik"
/[^Ss]/	neither 'S' nor 's'	"I have no exquisite reason for't"
/[^\.]/	not a period	"our resident Djinn"
/[e^]/	either 'e' or '^'	"look up ^ now"
/a^b/	the pattern 'a^b'	"look up a^ b now"

- Pipe symbol | denotes disjunction operator ("or"):
  - /cat|dog/ matches cat or dog but not cadog

Quantors

- Optional preceding symbol or expression: **?**
  - /Tallinn?/ matches Tallinn, Tallin
- Zero or more preceding symbols or expressions: *
  - /Tallinn*/ maches Tallin, Tallinn, Tallinnnnn
  - /a[bc]*/ matches a, acbb, but not adc
- One or more preceding symbols or expressions: **+**
  - /ba+!/ matches ba!, baa! but not b!
- Exactly *n* repetitions of the previous symbol/expression: **{n}**
  - /ba{4}/ matches baaaa
- From *n* to *m* repetitions: **{n,m}**:
  - /ba{2,4}/ matches baa, baaa, baaaa

Wildcard

- Dot (“.”) is a **wildcard** that matches **any** single character
  - /beg.n/ matches *begin*, *began*, *beg n*, *beg9n*
  - Can be combined with quantors:
    - /.*/ matches string of any length
    - /aa.*bb/ matches *aabb*, *aaabb*, *aaccggrrttbb*

Anchors

- Anchors are special characters that *anchor* regexes to special places in string:
  - ^ - start of string
  - $ end of string
  - \b word boundary
- Examples:
  - /^dog/ matches *dog* at the beginning of string, but not in a sentence like “I have a dog”
  - /dog$/ matches a *dog* at the end of a string
  - /\bthe\b/ matches the but not other
  - /\b99\b/ matches *99* in “I have 99 dollars” but not in “This costs $99” or “This happened in 1999”

Operator precedence

- Order of operator precedence:

Parenthesis	()
Counters	* + ? {}
Sequences and anchors	the ^my end$
Disjunction	|

- Examples:
  - **/the*/** matches the, theeee, but not thethe
  - **/(the)+/** matches the, thethe, thethethe
- When in doubt, use parenteses
  - **/cit(y)|(ies)/** matches city, cities

Predefined sets

RE	Expansion	Match	First Matches
\d	[0-9]	any digit	Party␣of␣5
\D	[^0-9]	any non-digit	Blue␣moon
\w	[a-zA-Z0-9_]	any alphanumeric/underscore	Daiyu
\W	[^\w]	a non-alphanumeric	!!!!
\s	[␣\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in␣Concord

Special symbols

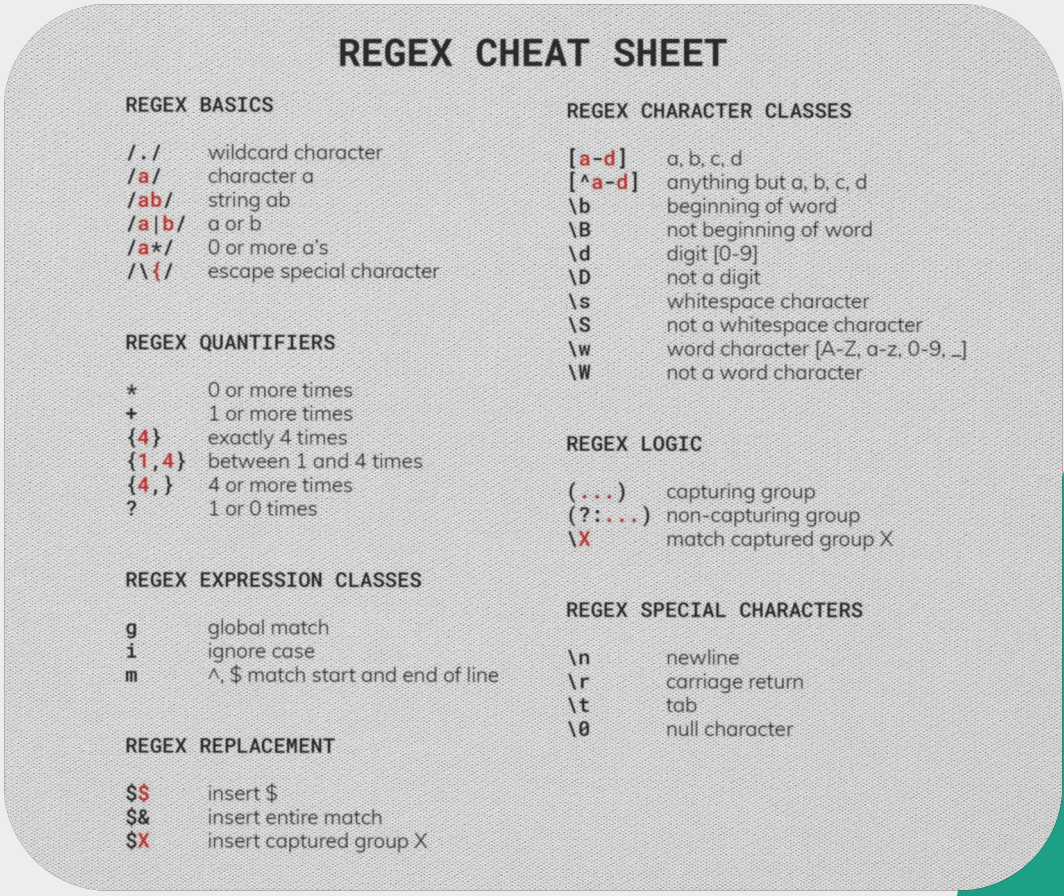
RE	Match	First Patterns Matched
*	an asterisk "*"	"K*A*P*L*A*N"
\.	a period "."	"Dr. Livingston, I presume"
\?	a question mark	"Why don't they come and lend a hand?"
\n	a newline	
\t	a tab	

Practical examples

- Find words ending with “a”: **/\S*a\b/**
- Find e-mail addresses that use .ee domain:
  /\b[a-z._]+@([a-z_]+\.)+ee\b/
- Find dates in format 03/11/2017:
  /\b\d{2}\/\d{2}\/\d{4}\b/ (careful: also finds 91/88/0000)

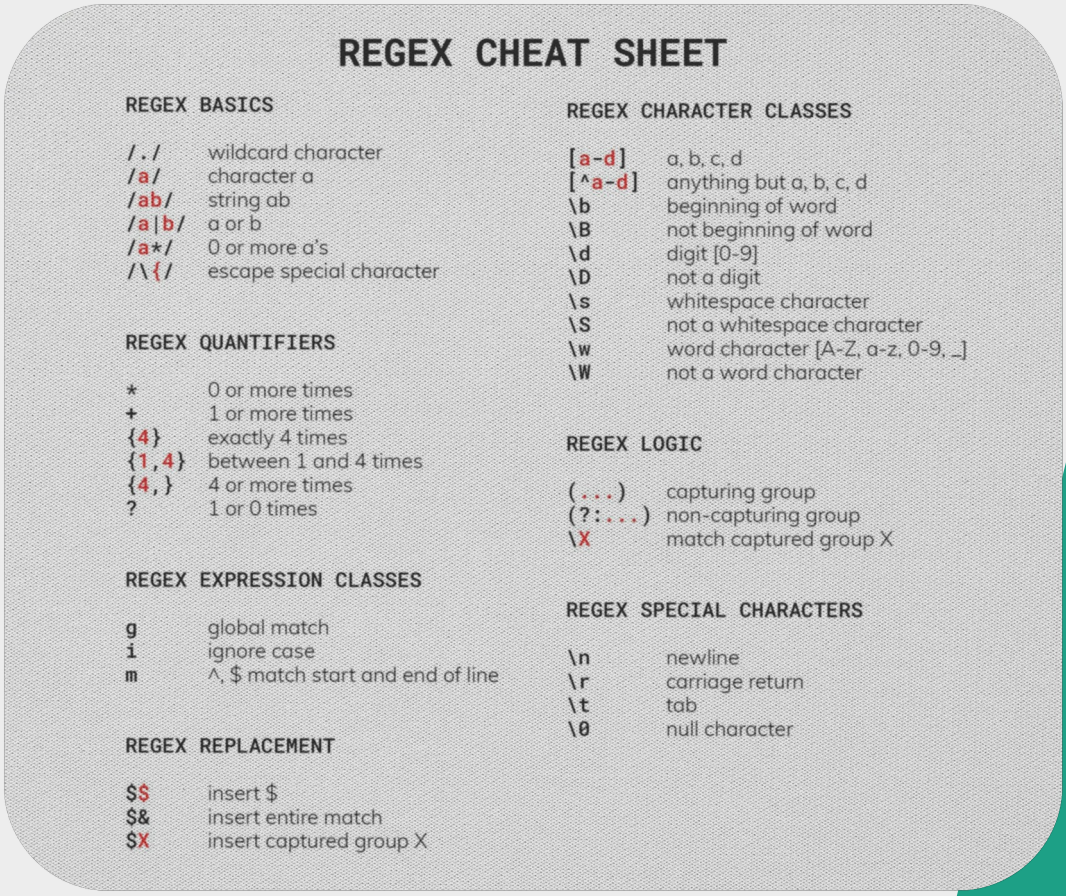
Regexp exercise

- Write a regex that matches all lowercase alphabetic strings ending with ‘b’, e.g.:
  - abb
  - aab
- But not:
  - Ab
  - bba
  - 123ab



Regexp exercise

- Write a regex that matches the set of all strings from the alphabet a,b such that each ‘a’ is immediately preceded by and immediately followed by ‘b’, e.g.:
  - bbbabbb
  - babab
- But not:
  - a
  - baba
  - babcc



REGEX CHEAT SHEET

REGEX BASICS

/./	wildcard character
/a/	character a
/ab/	string ab
/a|b/	a or b
/a*/	0 or more a's
/\{/	escape special character

REGEX QUANTIFIERS

*	0 or more times
+	1 or more times
{4}	exactly 4 times
{1,4}	between 1 and 4 times
{4,}	4 or more times
?	1 or 0 times

REGEX EXPRESSION CLASSES

g	global match
i	ignore case
m	^, $ match start and end of line

REGEX REPLACEMENT

$$	insert $
$&	insert entire match
$X	insert captured group X

REGEX CHARACTER CLASSES

[a-d]	a, b, c, d
[^a-d]	anything but a, b, c, d
\b	beginning of word
\B	not beginning of word
\d	digit [0-9]
\D	not a digit
\s	whitespace character
\S	not a whitespace character
\w	word character [A-Z, a-z, 0-9, _]
\W	not a word character

REGEX LOGIC

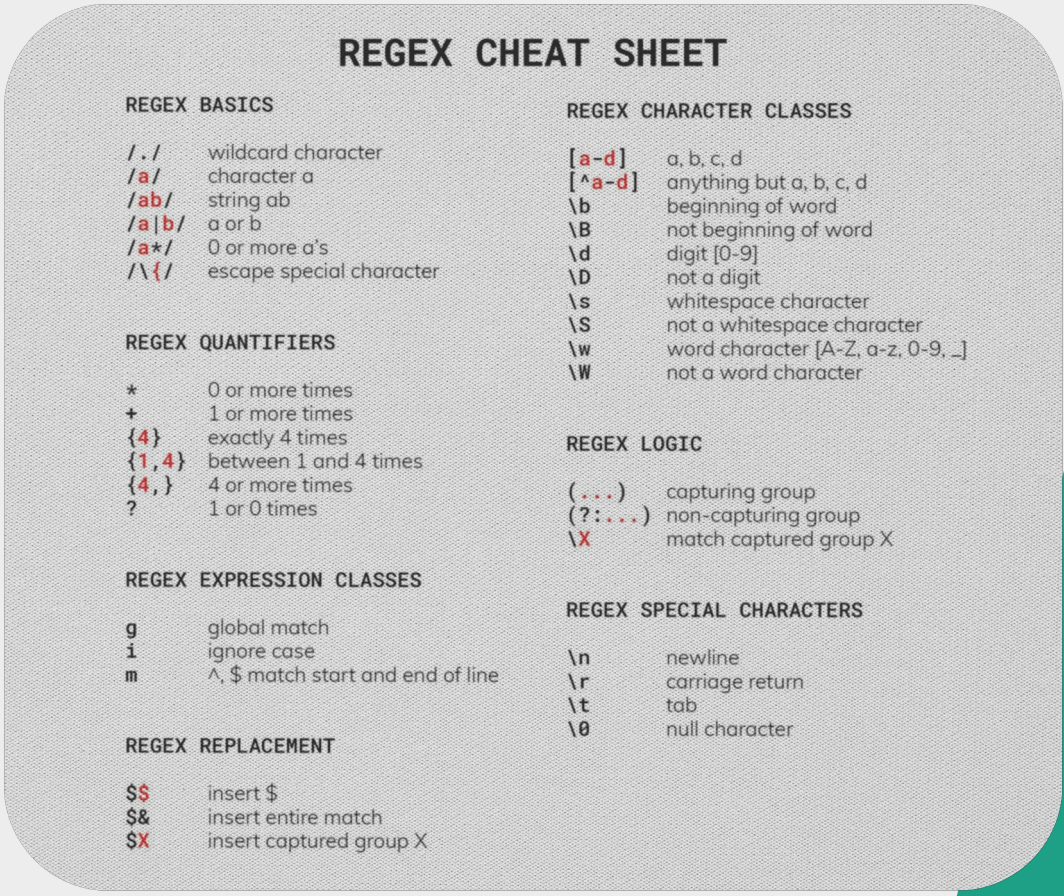
(...)	capturing group
(?:...)	non-capturing group
\X	match captured group X

REGEX SPECIAL CHARACTERS

\n	newline
\r	carriage return
\t	tab
\0	null character

Regexp exercise

- Write a regex that matches the set of all strings consisting of two consecutive repeated words, e.g.:
  - Foo Foo
  - 123 123
- But not:
  - Big Bag
  - ab ab c



REGEX CHEAT SHEET

REGEX BASICS

/./	wildcard character
/a/	character a
/ab/	string ab
/a|b/	a or b
/a*/	0 or more a's
/\{/	escape special character

REGEX QUANTIFIERS

*	0 or more times
+	1 or more times
{4}	exactly 4 times
{1,4}	between 1 and 4 times
{4,}	4 or more times
?	1 or 0 times

REGEX EXPRESSION CLASSES

g	global match
i	ignore case
m	^, $ match start and end of line

REGEX REPLACEMENT

$$	insert $
$&	insert entire match
$X	insert captured group X

REGEX CHARACTER CLASSES

[a-d]	a, b, c, d
[^a-d]	anything but a, b, c, d
\b	beginning of word
\B	not beginning of word
\d	digit [0-9]
\D	not a digit
\s	whitespace character
\S	not a whitespace character
\w	word character [A-Z, a-z, 0-9, _]
\W	not a word character

REGEX LOGIC

(...)	capturing group
(?:...)	non-capturing group
\X	match captured group X

REGEX SPECIAL CHARACTERS

\n	newline
\r	carriage return
\t	tab
\0	null character

Substitutions

- Regular expressions are often used for **substitution**
- In many Unix tools (*perl, vim, sed*), substitution can be invoked using:
  s/regexp/substitution/
- For example:
  `perl -npe 's/colour/color/g' f1.txt > f2.txt`
  replaces all instances of *colour* in *f1.txt with color* and saves the result to *f2.txt*
- Mastering regular expression substitutions makes many routine and awkward data cleaning and transformation tasks very easy

Substitutions using references

- Using references makes regular expression substitutions really powerful
- Numeric references (\1, \2, ...) ***refer*** to parts in parentheses (and) in the search expression:
  - **s/(\d+)/<\1>/** surrounds all number sequences with <>, e.g.
    `foo 44 bar → foo <44> bar`
  - **s/(\w+) (\w+)/\2 \1/** switches the first two words of the line, e.g.:
    `John Smith 56998874 → Smith John 56998874`

Practical substitution examples

- Very hacky HTML markup remover:
  s/<[^>]*>/ /g
  `a <b>nice</b> day → a  nice  day`
- Add a +372 prefix to all (probable) Estonian phone numbers starting with 6 or 7:
  s/\b(\d{6,7})\b/+372\1/g
  `6888999 → +3726888999`
- Replace names like *John Smith* with *J. Smith*:
  s/([A-Z])[a-z]* ([A-Z][a-z]+)/\1. \2/
  (note that this fails with non-English characters and any word that looks different)

Basic text processing

- Basic NLP pre-processing usually consists of:
  - Tokenization
  - Text normalization
  - Sentence segmentation

Tokenization: what are words?

- How many words (**tokens**) in the following sentence: *Tom's bike is red, my bikes are red.*
  - 8 words if not counting punctuation
  - 10 if counting punctuation
  - Whether to treat punctuation as a word depends on the task
  - But how many different words (word **types**)?
    - 7 (without punctuation)
  - But are *bike* and *bikes* different words?
    - They have the same **lemma** (*bike)* but are different **wordforms**

How many different words are there?

- English:

Corpus	Tokens = N	Types = $\|V\|$
Shakespeare	884 thousand	31 thousand
Brown corpus	1 million	38 thousand
Switchboard telephone conversations	2.4 million	20 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13 million

- Estonian
  - Number of word types increases more rapidly because of inflections (*koer, koera, koeraga, …*) and compound words (*hundikoer, koerakutsikas, …*)

How to split text into words?

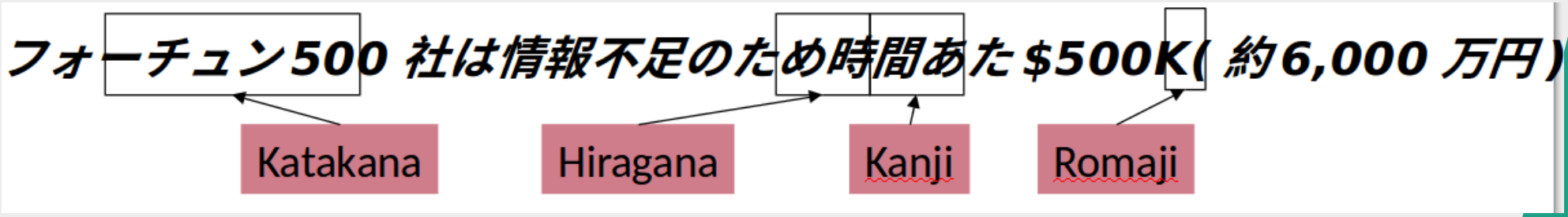
- Also called tokenization
- Split at whitespace?
  - What about punctuation? We usually want:
    The bike is red. → The bike is red .
  - However, word-internal punctuation is usually kept:
    John’s bike is red. → John’s bike is red .
    AT&T → AT&T, m.p.h → m.p.h
  - Also, word-ending punctuation is not always a separate token:
    - “Mr. Big”
    - In Estonian “See juhtus 1976. aastal”

Tokenization: language issues

- French:
  - *l'ensemble* → l'ensemble or le *enseble*
- *English:*
  - *doesn't* → *doesn't* or *does n't*
- Finnish:
  - *Honkaharjun sairaalan osasto 3:lla* → *Honkaharjun sairaalan osasto 3:lla*

Tokenization: language issues

- Chinese and Japanese no spaces between words:
  莎拉波娃现在居住在美国东南部的佛罗里达。
  莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
  Sharapova now lives in US southeastern Florida
- Further complicated in Japanese, with multiple alphabets intermingled



- Dates/amounts in multiple formats

Tokenization algorithm

- Text tokenization is language-specific
- Often done using rules, for example:
  - Split at whitespace
  - For every resulting token:
    - If it looks like an URL or e-mail, don't do anything
    - Separate token ending , ! ? : ; …
    - Separate quotes at word beginning and ends
    - Separate . at token ends, unless:
      - The token before . is numeric (e.g. *44.*)
      - The token is in a list of known abbreviations (e.g. *m.h.*) or is a single uppercase letter (e.g. *M.*)
      - The token after the current token starts with a lowercase letter
- Tokenization algorithm needs to be robust – it often needs to deal with typos, grammar mistakes.
- E.g., the algorithms above fails if there is no space after period in the sentence end: *Koer sööb konti.Kass sööb kala. → [Koer] [sööb] [konti.Kass] [sööb] [kala] [.]*
- Therefore, complicated heuristic rules are often added

Practical tokenization

- Pretty good tokenization (and many other NLP tools) for Estonian is implemented in the **EstNLTK** python package:

```
>>> from estnltk import Text
>>> text = Text('M. Õun elab 3. korrusel.')
>>> print (" ".join([w['text'] for w in text['words']]))
M. Õun elab 3. korrusel .
```

- For English, **spaCy** toolkit is a good choice:

```
>>> import spacy
>>> nlp = spacy.load('en_core_web_sm')
>>> doc = nlp("Apple isn't interested in buying U.K. startup for $1 billion.")
>>> print(" ".join([w.text for w in doc]))
Apple is n't interested in buying U.K. startup for $ 1 billion .
```

Text normalization

- During text normalization, we standardize the words:
  - Depends on the task
  - Can include of the following steps:
    - Fixing most common orthographic problems (e.g. *š* is often written as *s~* or sh in Estonian text)
      Masha ja Karu → Maša ja Karu
    - Normalizing punctuation marks to common UNICODE codepoints:
      John`s → John’s
      “great” → ’’great’’
    - Recasing text written in all-caps
      THIS IS GREAT → this is great
    - Converting abbreviations to common form:
      U.S. → US
    - True-casing sentence-beginning words:
      Mari kasvab metsas → mari kasvab metsas
      Mari on tore tüdruk → Mari on tore tüdruk
    - Sometimes, we **lemmatize words** (useful for information retrieval)
      mari kasvab metsas → mari kasvama mets
      tõstsin pöidla üles → tõstma pöial üles
    - Expanding numbers, units (e.g., needed for speech synthesis):
      $45 → fourty five dollars
      kaalun üle 80 kg → kaalun üle kaheksakümne kilogrammi
  - Usually implemented using a mixture of rule-based and machine-learning methods
  - Often complicated, boring and time-consuming to implement, but very important for overall success

Sentence segmentation

- Sentence segmentation divides running text into sentence:
  I *see a dog. The dog eats a bone.* →
  [I see a dog .], [The dog eats a bone.]
- Usually quite simple, if tokenization has been done before (just split the text on separate .!?):
  I *see a dog . the dog eats a bone .* →
  [I see a dog .], [the dog eats a bone .]

Questions?