01 INTRODUCTION

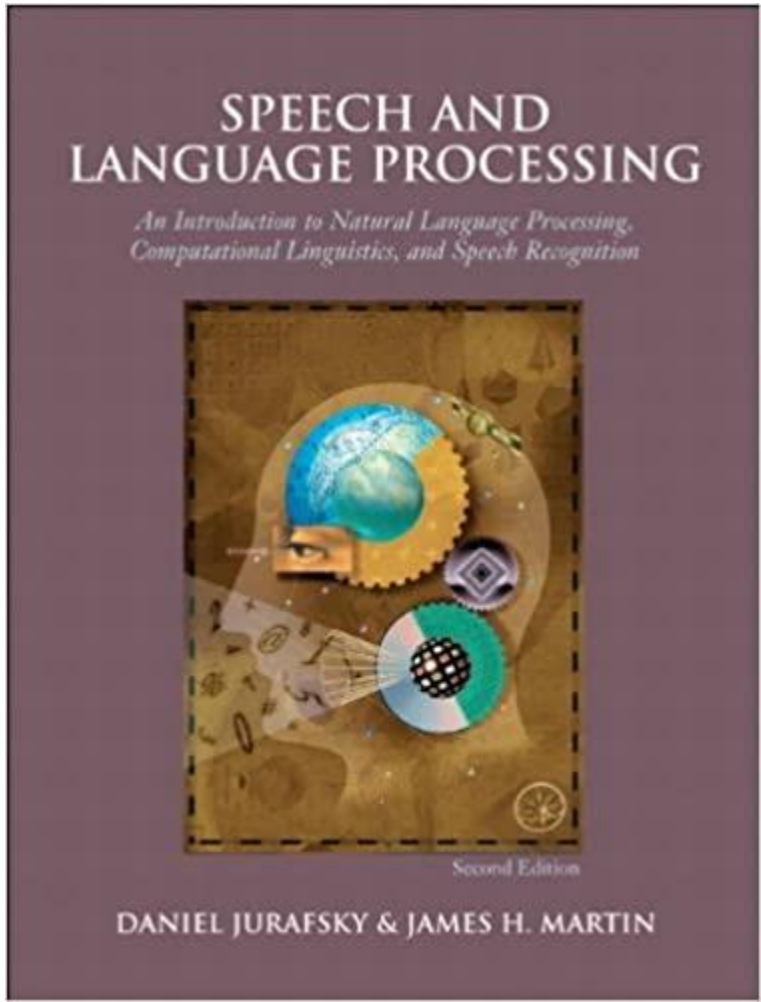
Spring 2023

CS6431 Natural Language Processing

B1:

Speech and Language Processing (Third Edition draft – Jan2022)

Daniel Jurafsky, James H. Martin



Credits

1. B1
2. https://regex101.com/
3. https://monkeylearn.com/natural-language-processing/
4. https://www.youtube.com/watch?v=6I-AIfkr5K4

Assignment

Read:

B1: Chapter 2

Problems:

End of chapter exercise of B1: Chapter 2

Outline

- Introduction to NLP
- Regular expression
- Text normalization
- Lemmatization
- Stemming
- Sentence Segmentation
- Edit Distance

Introduction to NLP

Natural Language Processing (NLP)

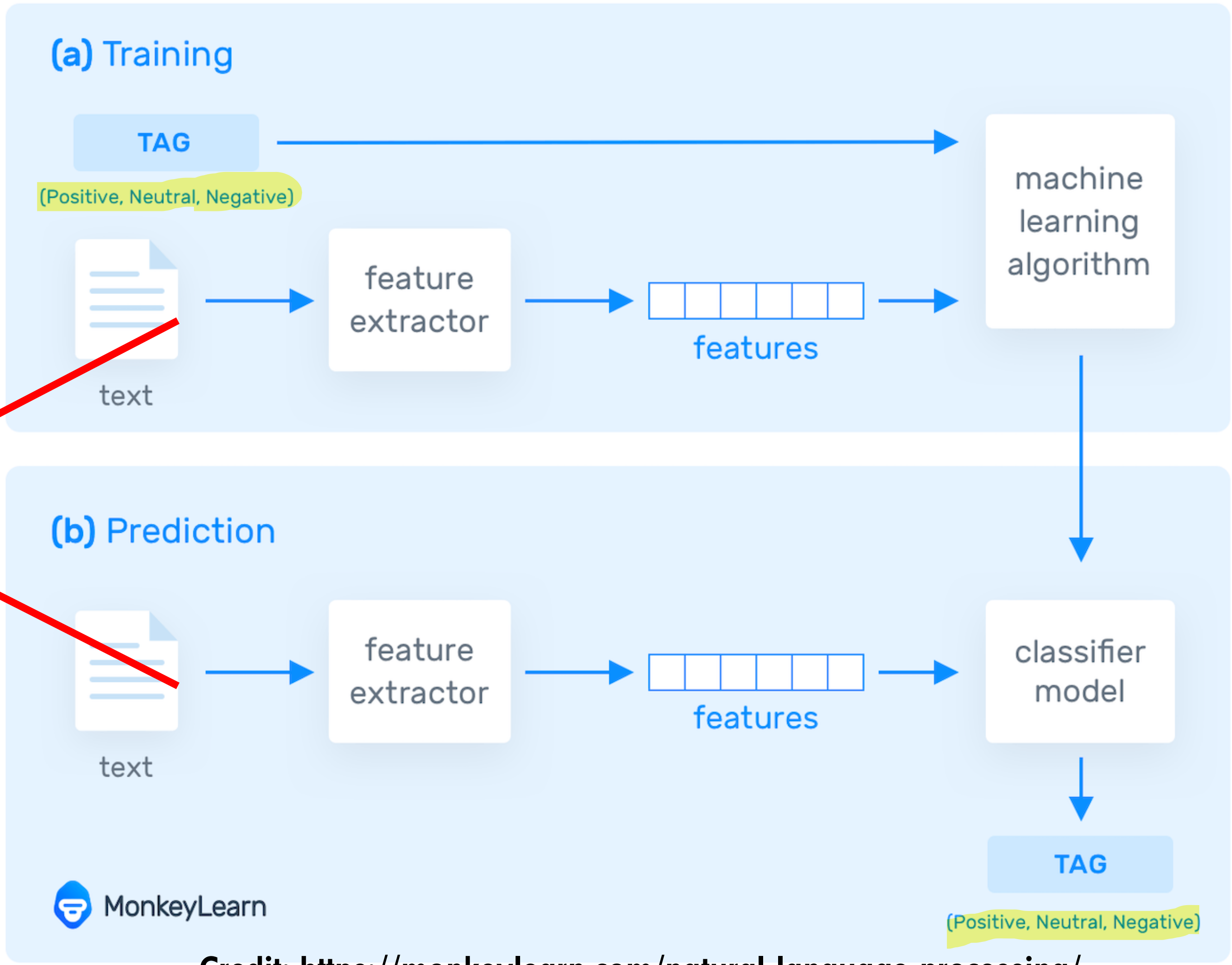
- Allows computers to understand human language
  - Translation software
  - Chatbots
  - Spam filters
  - Search engines
  - Grammar correction software
  - Voice assistants
  - Social media monitoring tools.

Email filter
virtual assistants
online search engine
predictive text - sentence comp option
sentiment analysis
text classification
chat bots
customer Relation management
automatic text summarization - summary
Natural language generation - chatGPT

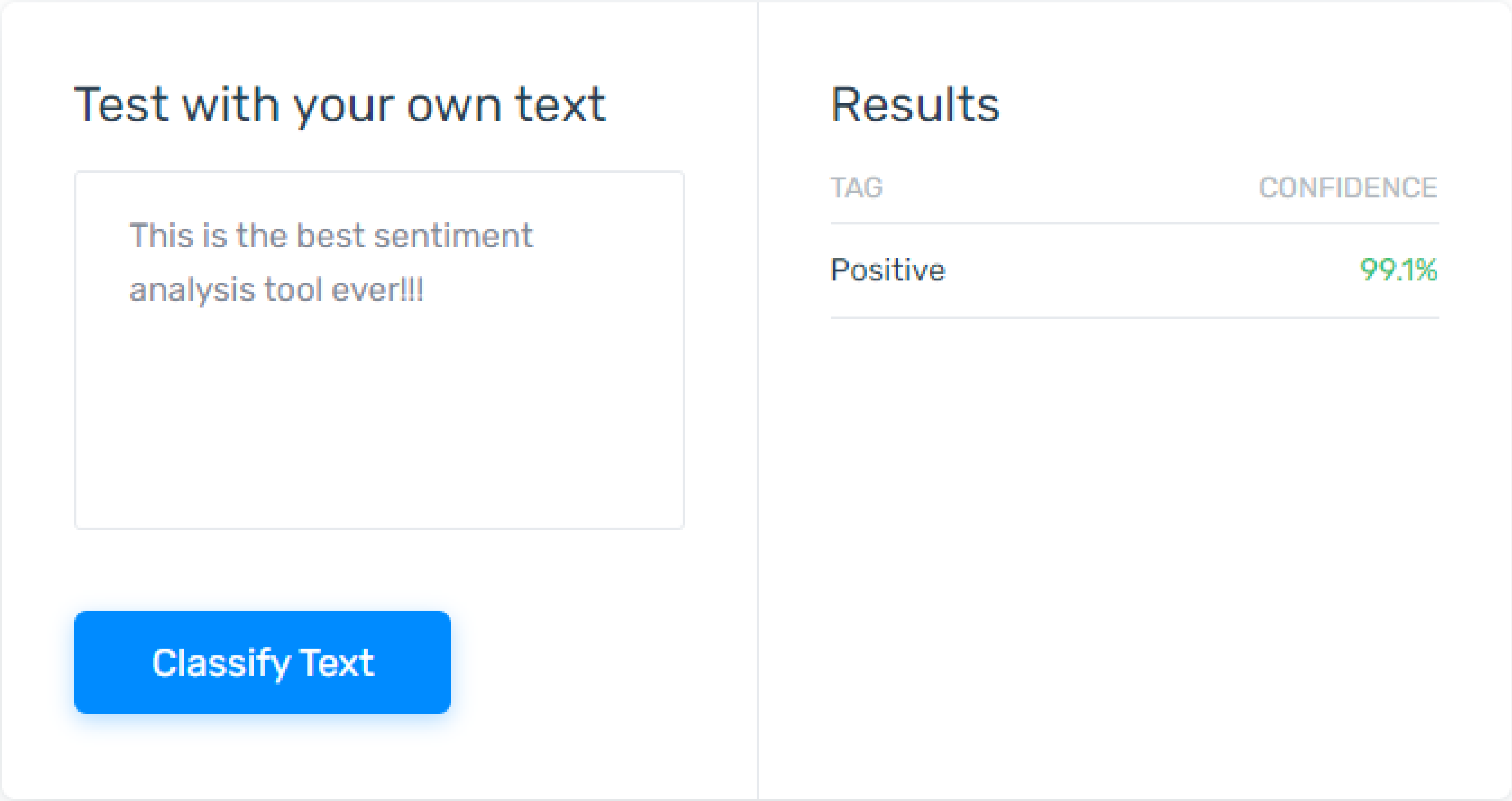
NLP-AI Pipeline

Text pre-processing (Segmentation, tokenization, POS tagging, NER)

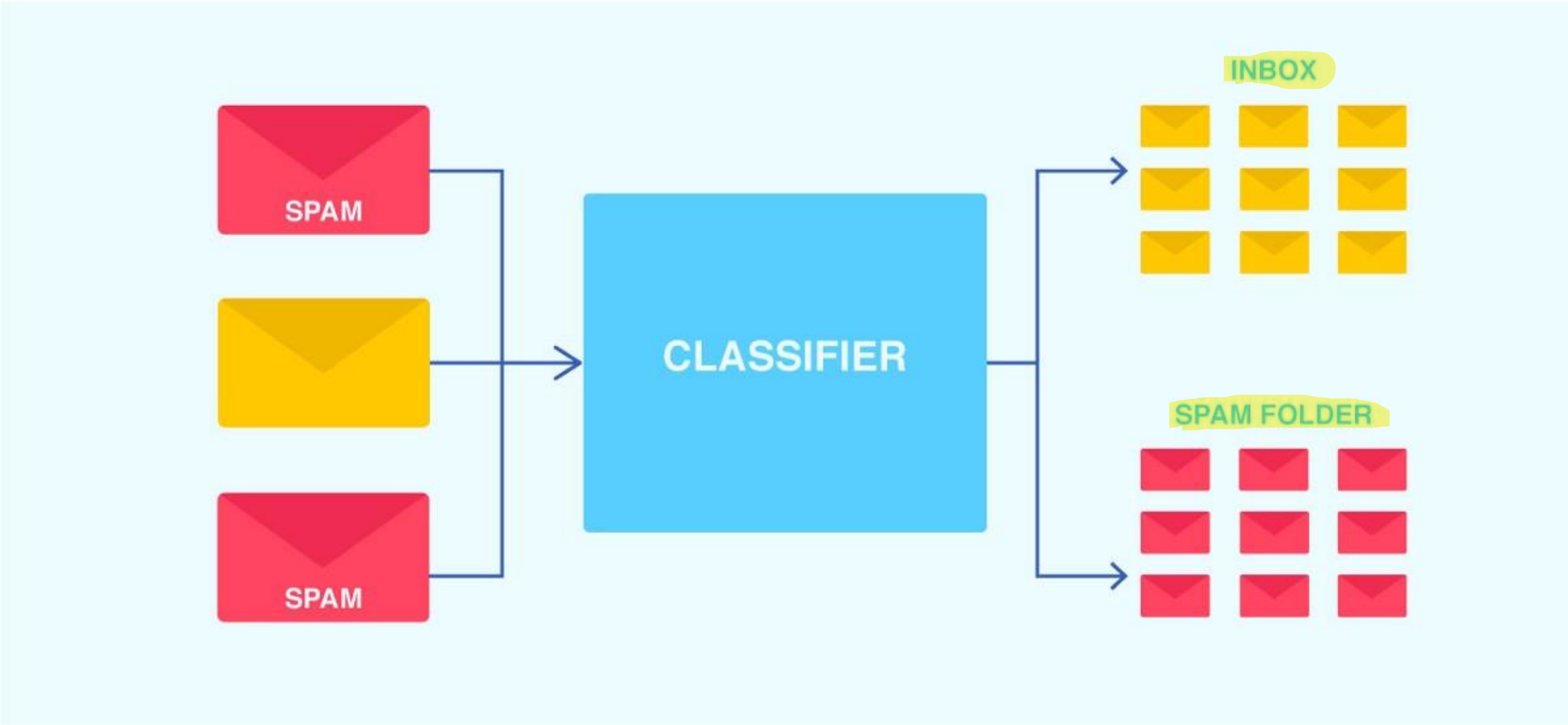
STPTN



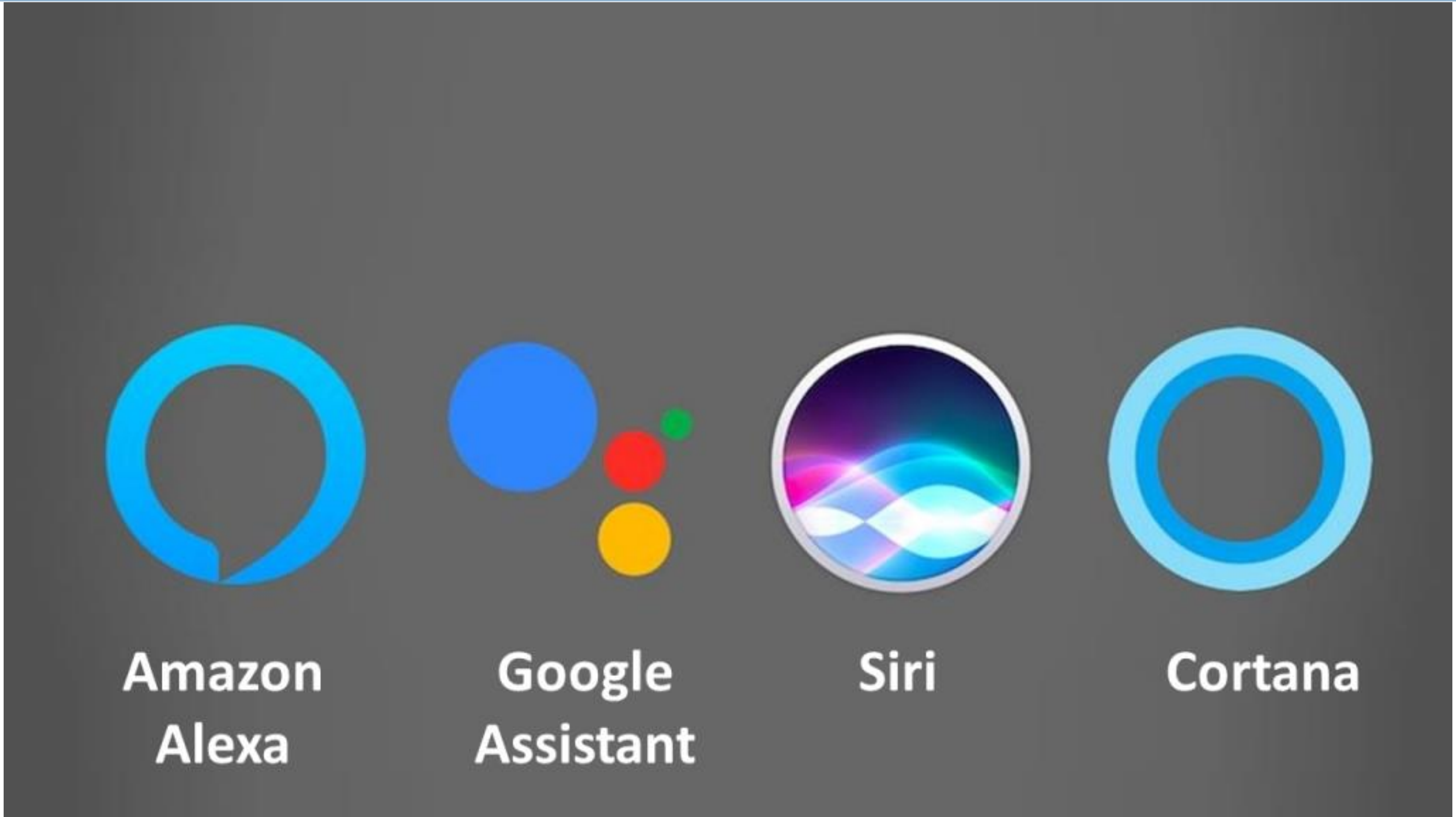
Credit: https://monkeylearn.com/natural-language-processing/


Test with your own text
This is the best sentiment analysis tool ever!!!
Classify Text
Results
TAG
CONFIDENCE
Positive
99.1%

Applications of NLP: (1) Email Filters

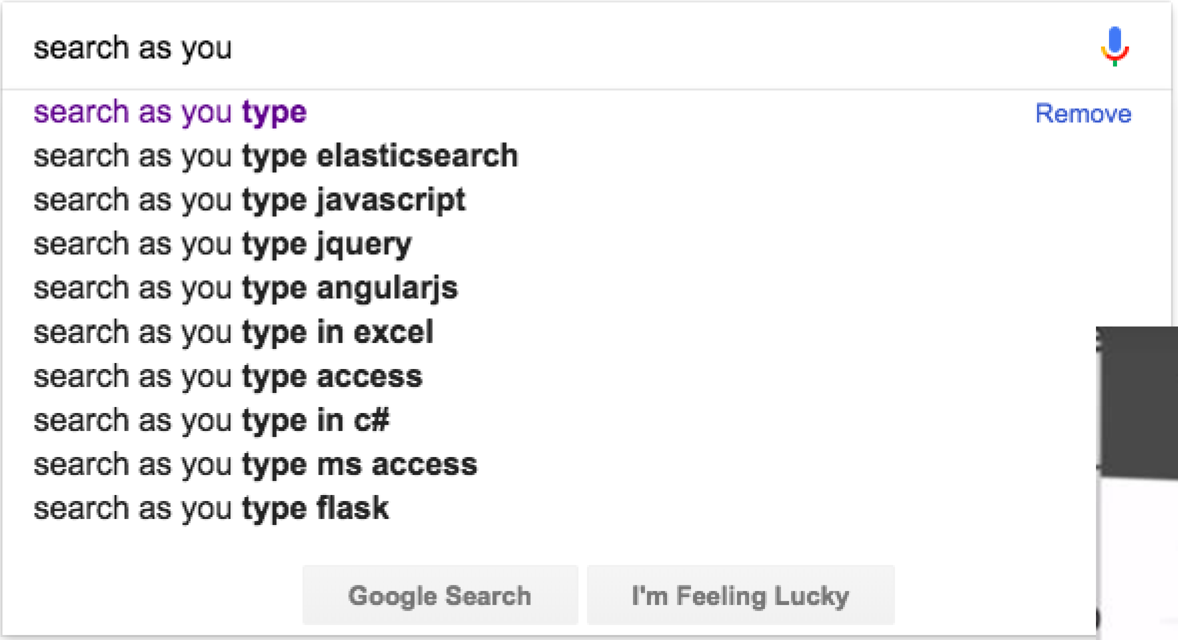


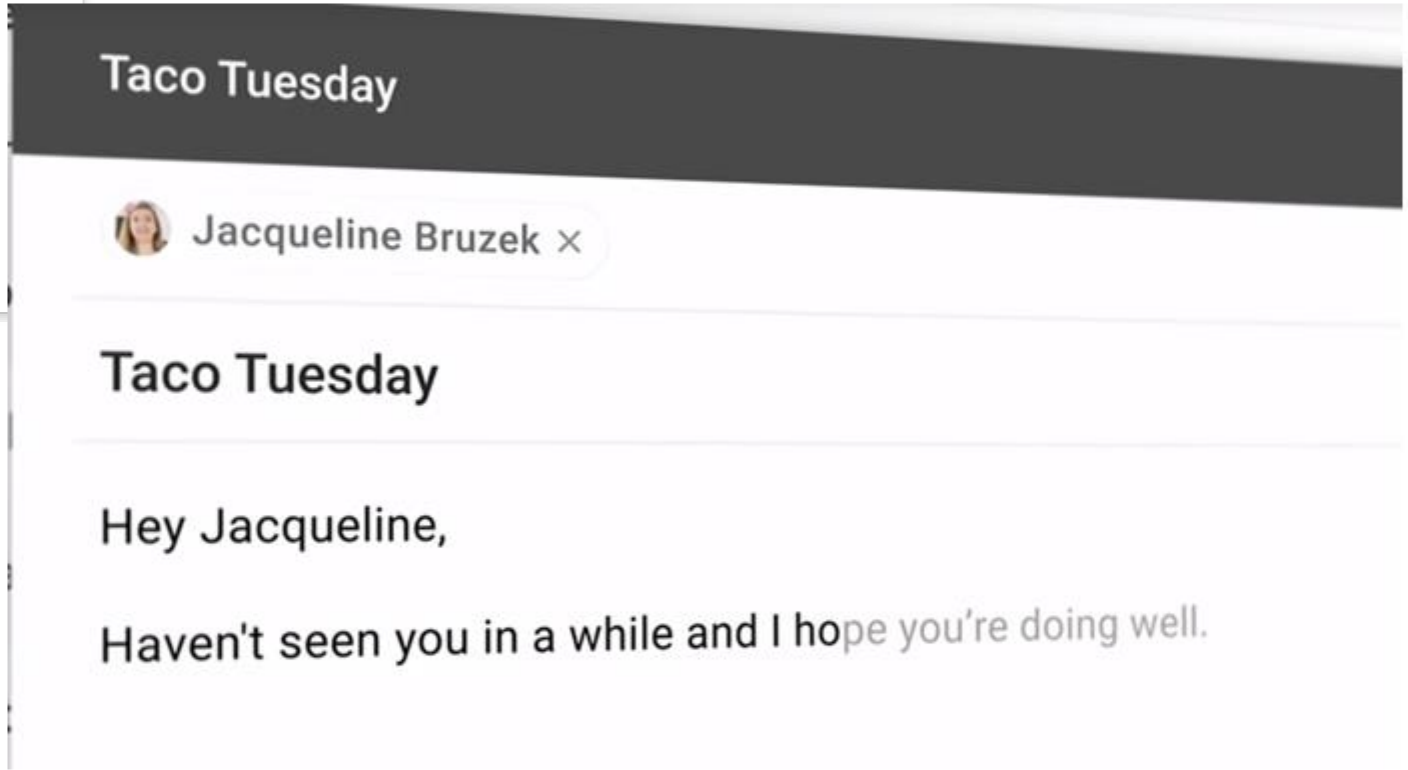
(2) Virtual assistants



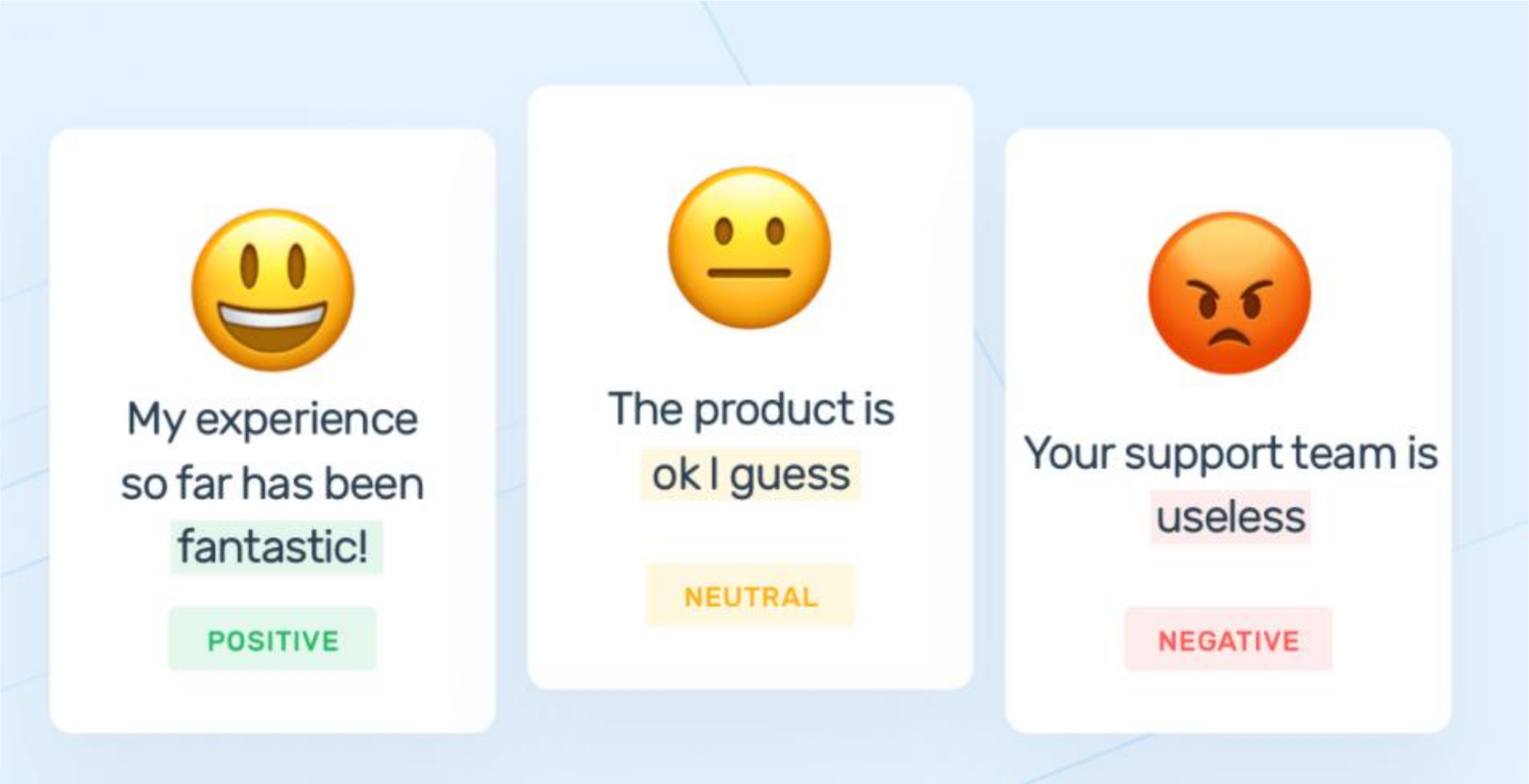
(3) Online search engines

(4) Predictive text



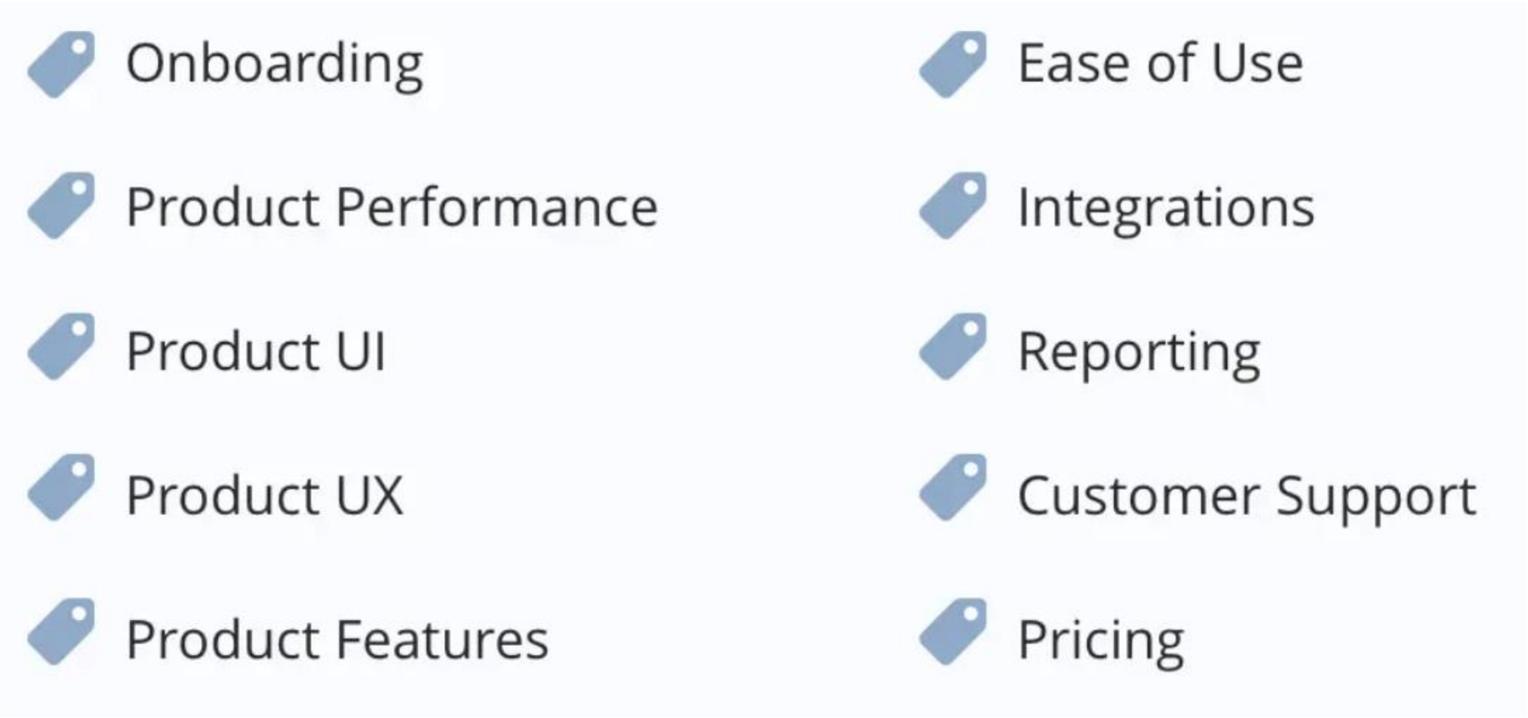


(5) Sentiment Analysis



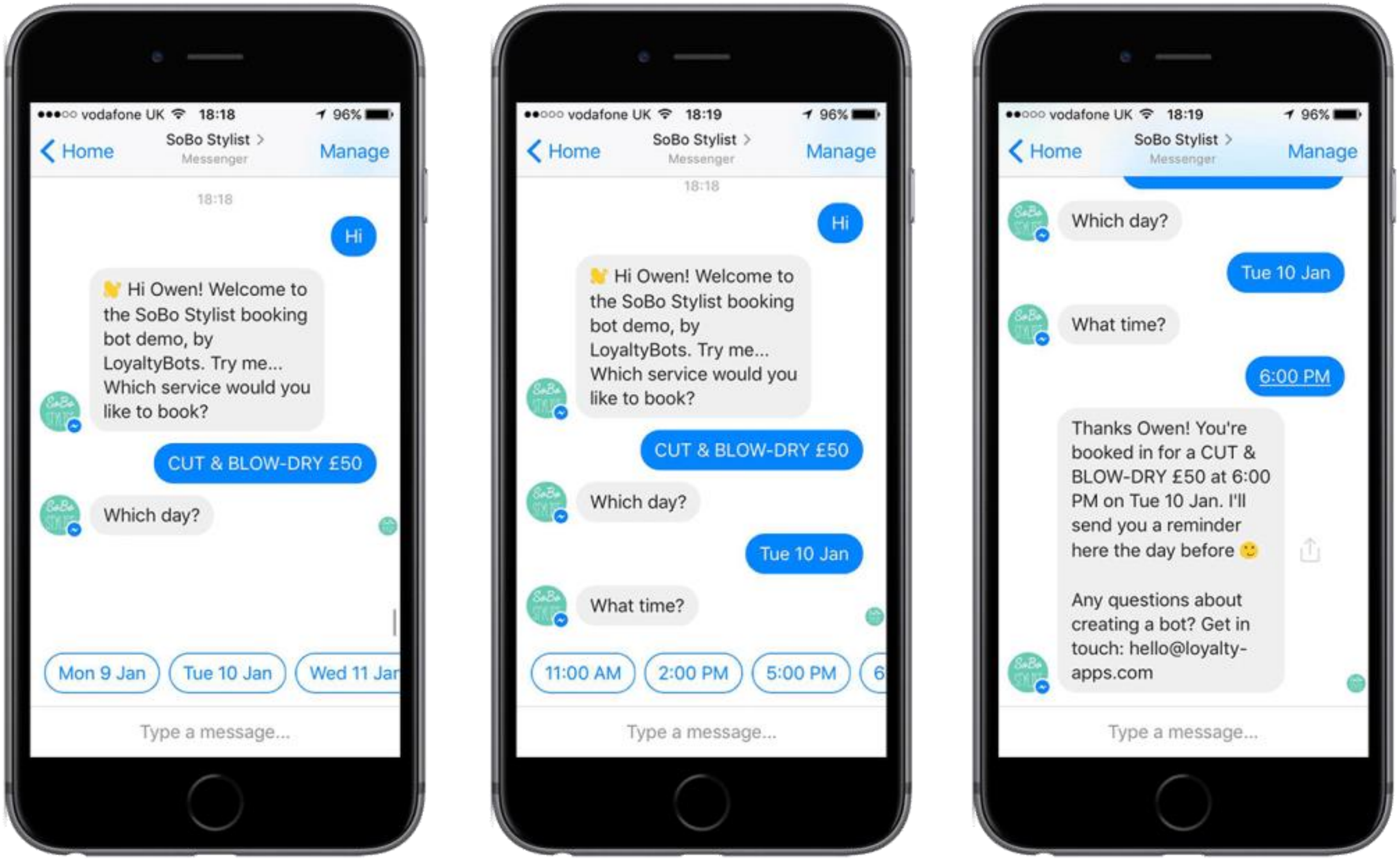
(6) Text Classification

- E.g., Sorting Customer Feedback

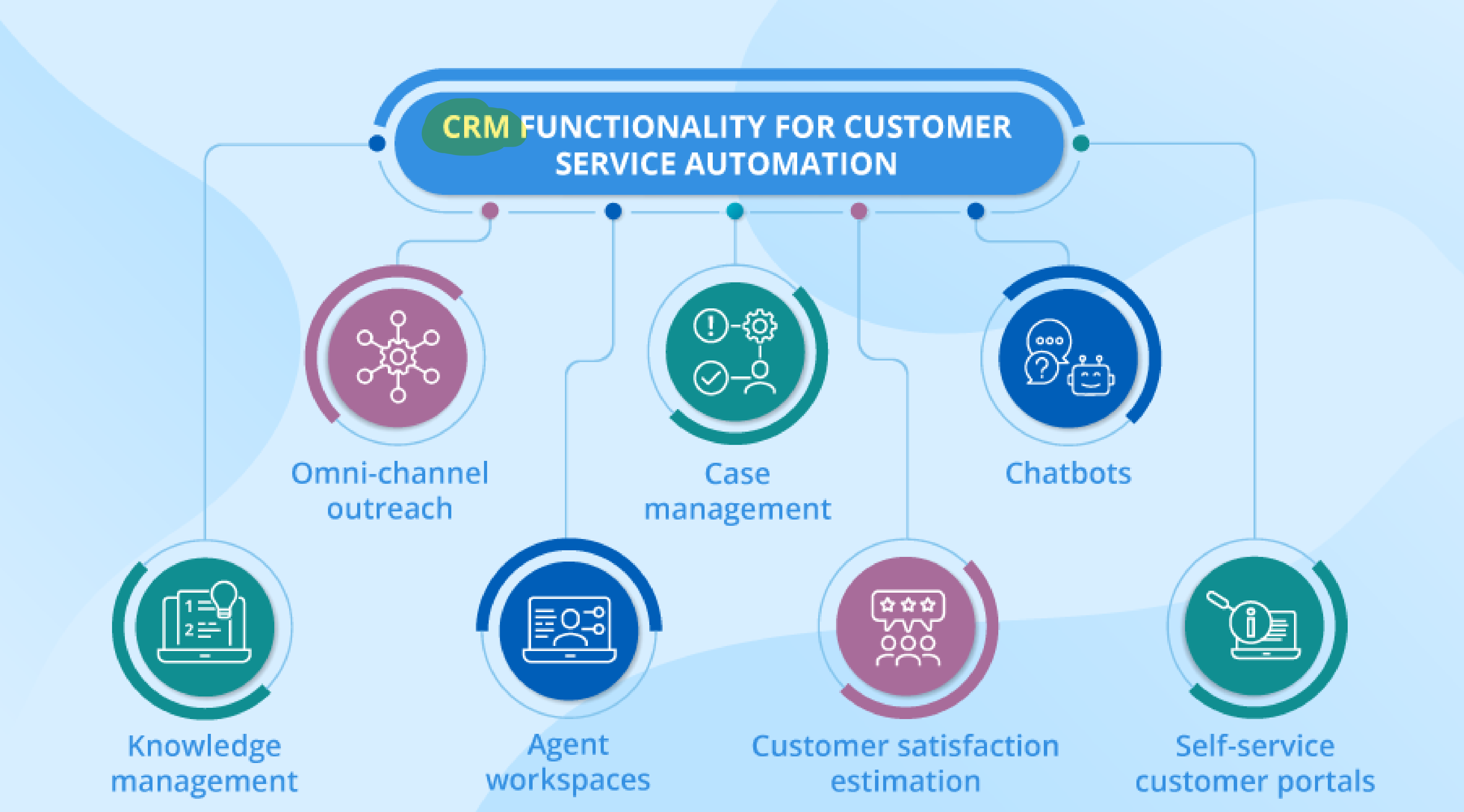


Response Tags Analysis
Promoters
Passives
Detractors
0
40
60
80
100
120
140
160
180
200
220
240
260
280
300
Product Features
50
160
90
Product UX
220
30
20
Customer Support
190
10
20
Integrations
30
60
10
Pricing
60
10
Ease of Use
40
Product UI
40
Product Performance
20
10

(7) Chatbots

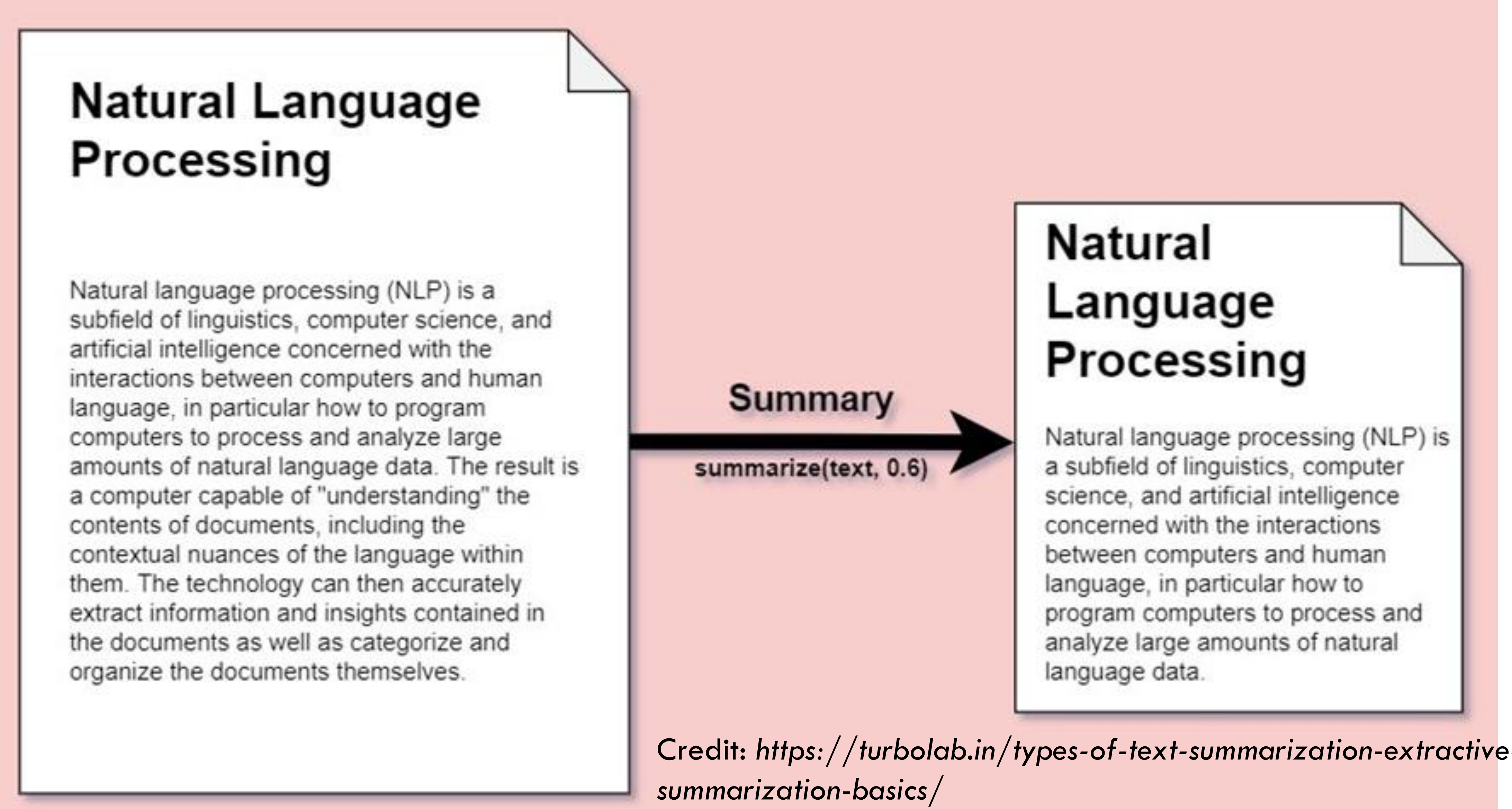

vodafone UK 18:18 96%
Home
SoBo Stylist
Messenger
Manage
18:18
Hi
Hi Owen! Welcome to the SoBo Stylist booking bot demo, by LoyaltyBots. Try me... Which service would you like to book?
CUT & BLOW-DRY £50
Which day?
Mon 9 Jan
Tue 10 Jan
Wed 11 Jan
Type a message...
vodafone UK 18:19 96%
Home
SoBo Stylist
Messenger
Manage
18:18
Hi
Hi Owen! Welcome to the SoBo Stylist booking bot demo, by LoyaltyBots. Try me... Which service would you like to book?
CUT & BLOW-DRY £50
Which day?
Tue 10 Jan
What time?
11:00 AM
2:00 PM
5:00 PM
Type a message...
vodafone UK 18:19 96%
Home
SoBo Stylist
Messenger
Manage
Which day?
Tue 10 Jan
What time?
6:00 PM
Thanks Owen! You're booked in for a CUT & BLOW-DRY £50 at 6:00 PM on Tue 10 Jan. I'll send you a reminder here the day before
Any questions about creating a bot? Get in touch: hello@loyalty-apps.com
Type a message...

Customer Relation Management



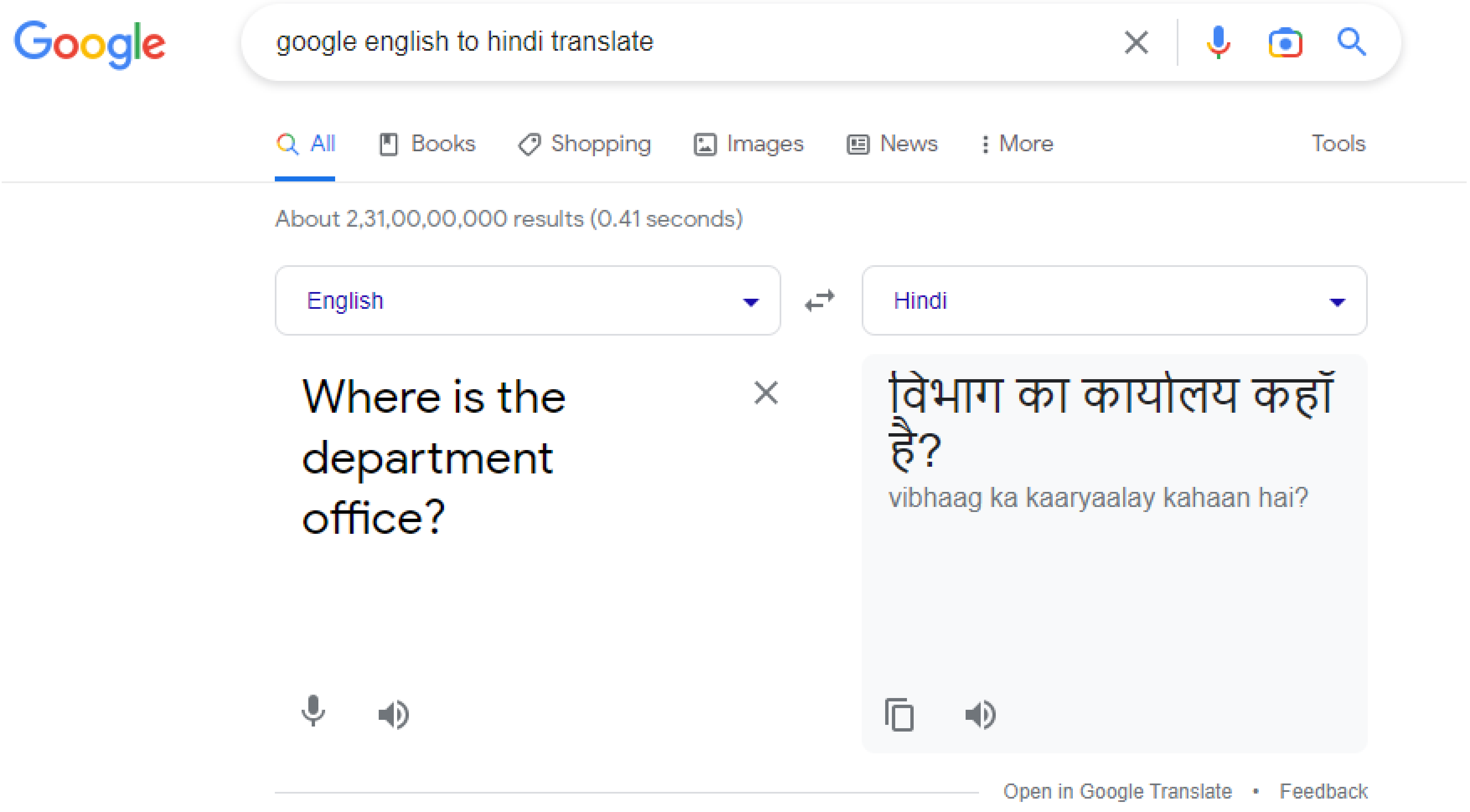
Credit: https://www.scnsoft.com/blog/customer-service-automation

(8) Automatic Text Summarization

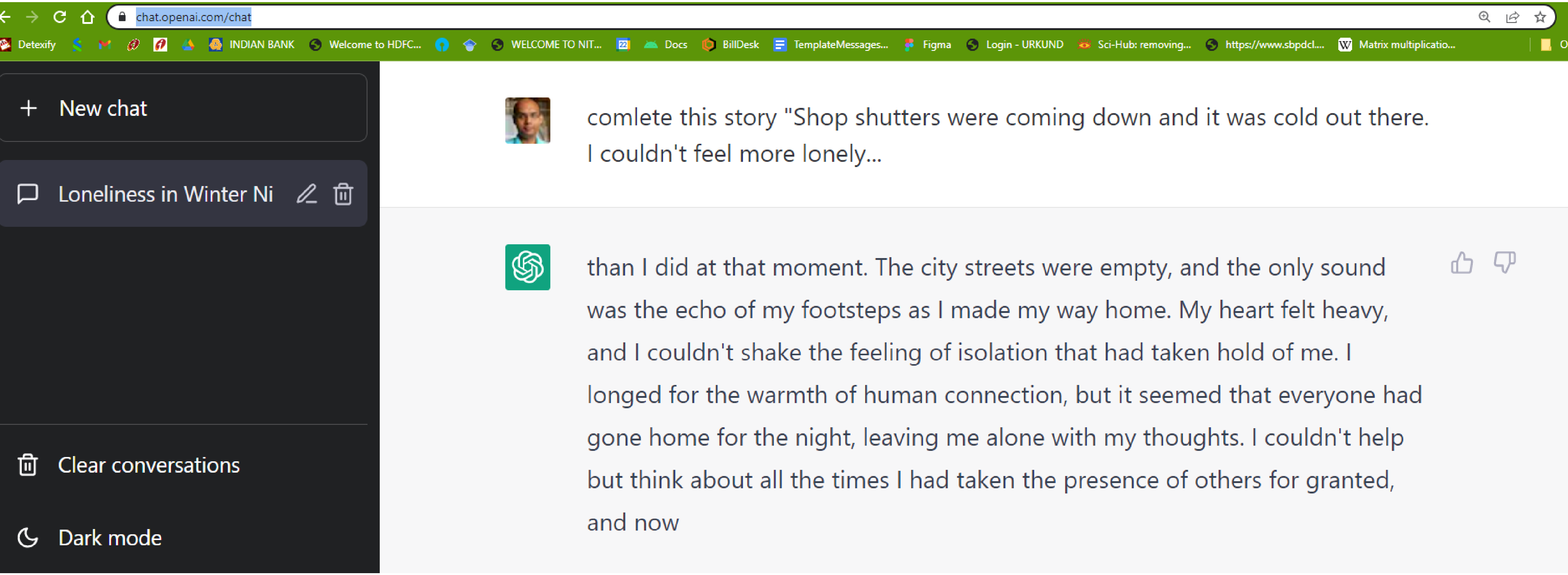


Credit: *https://turbolab.in/types-of-text-summarization-extractive-and-abstractive-summarization-basics/*

(9) Machine Translation



(10) Natural language generation



Regular Expression

Regular Expression

- *An algebraic notation for characterizing a set of strings*
- Case sensitive
- ***Concatenation***: sequence of characters

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

- ***We'll show regular expressions delimited by slashes but note that slashes are not part of the regular expressions***
- ***More than one match is possible, but we will consider only the first match***

- ***Disjunction:*** OR of characters
  - square braces [and]

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"

- /[ABCDEFGHIJKLMNOPQRSTUVWXYZ]/ can be inconvenient for 'any capital letter'
- ***Range***: range of characters

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

- ***Negation***: ^ caret only if the first symbol after [
  - If ^ is the first symbol, resulting pattern is negated
  - Anywhere else, it 'usually' matches a ^

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"

- How to match woodchuck and woodchucks?
  - ***Question mark*** ‘?’: zero or one instances of the previous character

RE	Match	Example Patterns Matched
`/woodchucks?/`	woodchuck or woodchucks	“woodchuck”
`/colou?r/`	color or colour	“color”

- How to match sheep language

  baa!

  baaa!

  baaaa!

  baaaaa!

  . . .

- Kleene star '*' : zero or more occurrences of the previous character
  - /a*/ matches 'aa', 'a', 'new' (has zero 'a's)
  - /aa*/ matches 'a', 'aa', but not 'new'
  - /[ab]*/ means "zero or more a's or b's"
  - /[0-9][0-9]*/ matches an integer
- '+': one or more occurrences
  - / [0-9]+/ is the same as /[0-9][0-9]*/

- **Wildcard** ‘.’: matches any single character (except a carriage return)

RE	Match	Example Matches
/beg.n/	any character between *beg* and *n*	begin, beg’n, begun

- /aardvark.*aardvark/ : matches a line in which ‘aardvark’ appears twice

- Anchors: anchor REs at particular places
  - ‘^’ (start) and ‘$’ (end)
  - /^The/ matches ‘The’ only at the start of a line
  - ^ has three uses
    1. To match the start of a line
    2. Negation when used with square brackets
    3. Just to mean ^
  - /^The dog\.$/ matches a line that contains only ‘The dog.’ (‘.’ has been escaped)

- ***Word boundary***: ‘\b’
  - A “word” is any sequence of digits, underscores, or letters
  - /\bthe\b/ matches ‘the’ but not ‘other’
  - /\b99\b/ matches ‘99’ in ‘There are 99 bottles of honey on the wall’ but not in ‘There are 299 bottles of honey on the wall’
- ***Non-word boundary***: ‘\B’ matches whatever ‘\b’ does not

RE	**Match**
^	start of line
$	end of line
\b	word boundary
\B	non-word boundary

- Disjunction: ‘|’ or operator
  - /cat|dog/ matches either the string cat or the string dog
- Precedence
  - How to match ‘guppy’ and ‘guppies’
    - /guppy|ies/ matches ‘guppy’ and ‘ies’ because sequences takes **precedence** over ‘|’ operator.
    - () can enforce precedence
    - /gupp(y|ies)/ matches ‘guppy’ and ‘guppies’

- ‘Column 1 Column 2 Column 3’
  - /Column [0-9]+ */ vs /(Column [0-9]+ *)*/

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

Precedence Order () * + ? {} seq ^ $ |

  - /the*/ matches ‘theeeee’ but not ‘thethe’

?

- Consider /[a-z]*/ on ‘once upon a time’
  - ‘o’, ‘on’, ‘onc’, ‘once’ all match
  - Greedy behaviour – returns largest match ‘once’
- Non-greedy
  - *?: Kleen star that matches as less as possible
  - +?: Kleen plus that matches as less as possible

Hands on

- Write a RE to find cases of the English article ‘the’
- /the/
  - Misses ‘The’
- /[tT]he/
  - Also matches ‘the’ in ‘other’ or ‘theology’
- /\b[tT]he\b/
  - What if we want to catch ‘the’ in ‘the_’ or ‘the25’
- /[^a-zA-Z][tT]he[^a-zA-Z]/
  - Misses when ‘The’ is at start of the line
- /(^|[^a-zA-Z])[tT]he([^a-zA-Z]|$)/

- Counters: {} – matches a given no of occurrences
  - /a\.{24}z/ a followed by 'exactly' 24 dots followed by z
  - /{n,m}/ - specifies range from *n to m* of the previous char/expr
  - /{n,}/ - at least *n* of the previous char/expr

RE	Match
*	zero or more occurrences of the previous char or expression
+	one or more occurrences of the previous char or expression
?	exactly zero or one occurrence of the previous char or expression
{n}	*n* occurrences of the previous char or expression
{n,m}	from *n* to *m* occurrences of the previous char or expression
{n,}	at least *n* occurrences of the previous char or expression
{,m}	up to *m* occurrences of the previous char or expression

- Aliases for common sets of characters.

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

- Escaped special characters

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	

- Also '\+', '\^'

Hands on - II

- A user wants to buy a computer “any machine with at least 6 GHz and 500 GB of disk space for less than $1000”
- Let us try a RE for price
- /$[0-9]+/
  - Does not match decimal
- /$[0-9]+\.[0-9][0-9]/
  - Allows $199.99 but not $199
- /(^|\W)$[0-9]+(\.[0-9][0-9])?\b/
  - Allows $199999.99 – too expensive
- /(^|\W)$[0-9]{0,3}(\.[0-9][0-9])?\b/

- Disk space - 500 GB
  - /\b[0-9]+(\.[0-9]+)? *(GB|[Gg]igabytes?)\b/

- Exercise: match only more than 500GB

- ***Substitution***: 's'
  - s/regexp1/pattern/

s/colour/color/ changes "Red colour" to "Red color"

  - Matches substring can be referred to

s/([0-9]+)/<\1>/ changes "the 35 boxes" to "the <35> boxes"

  - ***Capture group***: storing patterns in memory

/the (.*)er they were, the \1er they will be/

Matches "the bigger they were, the bigger they will be" but not "the bigger they were, the faster they will be"

/the (.*)er they (.*), the \1er we \2/ matches "the faster they ran, the faster we ran" but not "the faster they ran, the faster we ate"

    - \1, \2… refer to the first, second… ***capture groups***

- ***Non-capturing group:*** putting ?: after open parenthesis (?: pattern)

/(?:some|a few) (people|cats) like some \1/ matches “some cats like some cats” but not “some cats like some a few”

Text normalization

Words

- What counts as a word?

  “He stepped out into the hall, was delighted to encounter a water brother.”

  13 words w/o punctuation, 15 words with punctuation

- Punctuation can be useful
  - Denotes boundaries – commas, periods, colons
  - Identifies sentence meaning – question marks, exclamation marks, quotation marks

- Spoken speech can have no punctuations

  “I do uh main- mainly business data processing”

  - Fragments: broken off words like ‘main-’
  - Fillers: ‘uh’, ‘umm’
- Should we consider fragments and fillers as words?

- Should ‘the’ and ‘The’ be treated as same?
  - Depends on the application

Lemma and Wordform

- Lemma: basic form of the word
- Wordform: derived/inflected form of the word
  - ‘cat’ and ‘cats’ have the same lemma ‘cat’
  - ‘slowly’ and ‘slowness’ are different wordforms of ‘slow’

Types and Tokens

No of

- Types: #unique words
- Tokens: #running words

'They picnicked by the pool, then lay back on the grass and looked at the stars.'

- 16 tokens and 14 types

16 Words 14 unique words

Corpus

- A collection of written or spoken texts
  - Speakers (gender)
  - Language
    - Code switching

      dost tha or ra- hega ... dont wory ... but dherya rakhe

      [“he was and will remain a friend ... don’t worry ... but have faith”]
  - Dialect
  - Time
  - Place / Surroundings
  - Genre
    - Spoken / Written

Text Normalization

Steps

1. Tokenizing (segmenting) words
2. Normalizing word formats
3. Segmenting sentences

Tokenizing

- Running text to words
- Punctuation: as a word or not?
  - Yes, indicate boundary
  - No, when occur internally
    - m.p.h, Ph.D., AT&T, cap'n, '100,000'
- Number expressions and special characters
  - Prices: ₹45.55
  - Dates: (14/01/2022)
  - URLs: https://www.nitp.ac.in
  - Hashtags: #goNITP
  - Email: students.cs.ug20@nitp.ac.in

- Clitic contractions – marked by apostrophes
  - 'what're' - > what, are
  - 'we're' -> we, are
- Apostrophes
  - Genetic Markers: 'the book's cover'
  - Quotative: 'He acknowledged with "Thank you".'
- Multi-word tokens
  - New Delhi
  - Rock 'n' roll
- NER: the task of detecting names, dates,
  - Plays an important role in tokenization

- An example from nltk.regexp_tokenize function of the Python-based Natural Language Toolkit (NLTK) (Bird et al. 2009;

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     ([A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(-\w+)*        # words with optional internal hyphens
...   | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
...   | \.\.\.            # ellipsis
...   | [][.,;"'?():-_`]  # these are separate tokens; includes ], [
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

- In languages like Chinese and Japanese, segmenting characters rather than words can make more sense for tokenization

Tokenization Scheme

- Learning tokenization from a standard corpus
  - Unknown word problem – in training corpus but not in test corpus
    - Training has ‘low’, ‘new’, ‘newer’, but not ‘lower’
  - Solution: tokenize subwords
    - ‘newer’ – ‘new’ and ‘er’
    - Unlikeliest – ‘un-’, ‘likely’, and ‘-est’
- Tokenization scheme

  input induces
  - Token learner: Training corpus -> vocabulary
  - Token segmenter: test sentence, vcabulary -> tokens

  input tokenizes it according to that vocabulary

Byte-Pair Encoding (BPE) for Tokenization

- "Neural machine translation of rare words with subword units.", Sennrich et al., 2016
  - Choose most frequently adjacent symbols, say 'A' and 'B'
  - Create new symbol 'AB'
  - Replace all adjacent occurrences of 'A' 'B' with 'AB
  - Repeat till *k* merges (creating longer symbols)

- An example

v me add....show...end of word

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 _	

times

er
e r -> replace er
next...
er _ -> replace er_

A morpheme is a smallest meaning bearing unit of a language

note: utube video

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

- Applying BPE on test data
  - Apply the merger rules in the order learned
    - Training data frequencies matter, test data's doesn't
  - Result
    - Most/Known words are converted to symbols
    - Rare/Unknown words are represented into parts

Word Normalization

- Putting words/tokens in a standard format
  - E.g.
    - 'USA' and 'US'
    - 'uh-huh' and 'uhhuh'
  - Spelling information is lost but that is OK
- Case folding
  - Converting everything to lower case
  - Useful in information retrieval and speech recognition
  - Not always useful
    - US vs. us
  - Depends on the application

Lemmatization

- Reducing words to their roots
  - ‘am’, ‘are’, and ‘is’ share lemma ‘be’
  - ‘dinner’ and ‘dinners’ share lemma ‘dinner’
- Involves morphological parsing – breaking down to ‘morphemes’
  1. **Stems:** main meaning, e.g., ‘cat’ in ‘cats’
  2. **Affixes:** additional meaning, e.g., ‘s’ in ‘cats’
- Stemming: one way of lemmatization
  simplest way
  - Chopping out affixes through rules

The Porter Stemmer

standard algorithm for stemming

ational - ate

- Proposed by Porter (1980) "An algorithm for suffix stripping."
  - Input

This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things-names and heights and soundings-with the single exception of the red crosses and the written notes.

  - Output:

Thi wa not the map we found in Billi Bone s chest but an accur copi complet in all thing name and height and sound with the singl except of the red cross and the written note

Sentence Segmentation

- Separating sentences
  - Cues: punctuation marks
    - '?' and '!' are relatively unambiguous
    - '.' is ambiguous, e.g., 'Mr.' or 'Inc.'
      - End of sentence or abbreviation
      - Resolved through rules or machine learning

Edit Distance

String Similarity

- A key task across NLP

minimum no of edit operation

- Edit Distance: Quantifies distance/dissimilarity between two strings
  - Edit operations: insertion (i), deletion (d), substitution (s)
- Minimum Edit Distance(S1,S2): min #edit operations to transform S1 to S2
  - E.g., 'INTENTION' and 'EXECUTION'
    - Min edit dist = 5

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

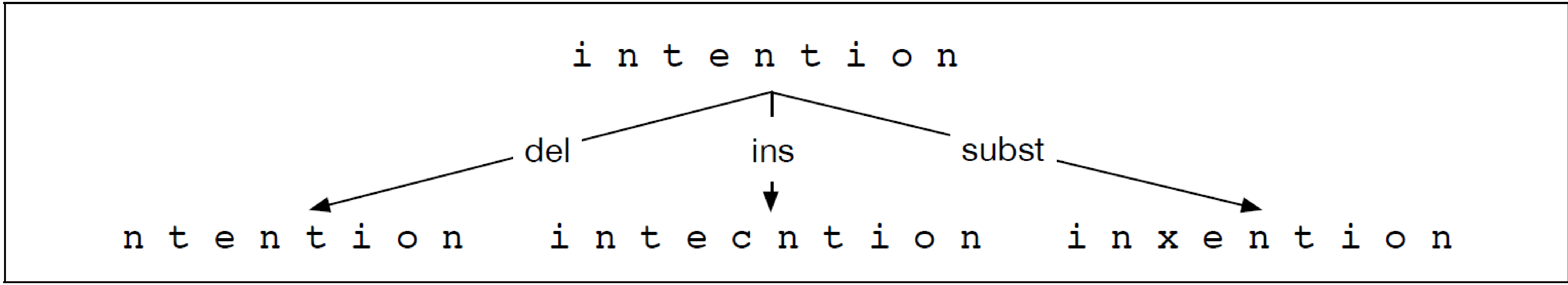
- Edit distance depends on how we align
  - Multiple alignments are possible

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

- Minimum Edit Distance corresponds to the alignment that gives minimum #edit operations
- Levenshtein, 1966, “Binary codes capable of correcting deletions, insertions, and reversals.” proposed cost to edit operations

- Edit operations, insertion (i), deletion (d), substitution (s), can have different costs
- Minimum Edit Distance(S1,S2): min possible ***cost*** of edit operations to transform S1 to S2

- How to find Minimum Edit Distance (S1,S2)?
  - Shortest path – a sequence of edits – from S1 to S2 in the search space



- We will use a dynamic programming approach

- Input string *X* be of length *m*
- Input string *Y* be of length *n*
- Let *D[i,j]* denote the MED between *X[1…i]* and *Y[1…j]*
  - That is the MED between the first *i* characters of *X* and the first *j* characters of *Y*.
  - *D[i,j]* also represents the corresponding alignment between *X[1…i]* and *Y[1…j]*

- Dynamic Programming approach: *D[i,j]* is computed from smaller values of *i* and *j*
- Three possible paths to reach cell *[i,j]* of *D*

$$D[i,j] = \min \begin{cases} D[i-1,j] + \text{del-cost}(source[i]) \\ D[i,j-1] + \text{ins-cost}(target[j]) \\ D[i-1,j-1] + \text{sub-cost}(source[i], target[j]) \end{cases}$$

```
function MIN-EDIT-DISTANCE(source, target) returns min-distance

  n ← LENGTH(source)
  m ← LENGTH(target)
  Create a distance matrix D[n+1,m+1]

  # Initialization: the zeroth row and column is the distance from the empty string
  D[0,0] = 0
  for each row i from 1 to n do
     D[i,0] ← D[i-1,0] + del-cost(source[i])
  for each column j from 1 to m do
     D[0,j] ← D[0, j-1] + ins-cost(target[j])

  # Recurrence relation:
  for each row i from 1 to n do
       for each column j from 1 to m do
          D[i,j] ← MIN( D[i−1,j] + del-cost(source[i]),
                        D[i−1,j−1] + sub-cost(source[i],target[j]),
                        D[i,j−1] + ins-cost(target[j]))
  # Termination
  return D[n,m]
```

- Let us find MED between words ‘intention’ and ‘execution’
- Cost of edit operations
  - Cost of insertion = cost of deletion = 1
  - Cost of substitution (in case of mismatch) = 2

Src\Tar	#	e	x	e	c	u	t	i	o	n
#	0	1	2	3	4	5	6	7	8	9
i	1	2	3	4	5	6	7	6	7	8
n	2	3	4	5	6	7	8	7	8	7
t	3	4	5	6	7	8	7	8	9	8
e	4	3	4	5	6	7	8	9	10	9
n	5	4	5	6	7	8	9	10	11	10
t	6	5	6	7	8	9	8	9	10	11
i	7	6	7	8	9	10	9	8	9	10
o	8	7	8	9	10	11	10	9	8	9
n	9	8	9	10	11	12	11	10	9	8

How to Retrieve Alignment?

- Two steps
  - Indicate ***back pointers*** in each cell
  - Perform ***backtrace*** through these back pointers from the last cell *[m,n]* to the initial cell *[0,0]*

	#	**e**	**x**	**e**	**c**	**u**	**t**	**i**	**o**	**n**
#	0	← 1	← 2	← 3	← 4	← 5	← 6	← 7	← 8	← 9
i	↑ **1**	↖←↑ 2	↖←↑ 3	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖ 6	← 7	← 8
n	↑ 2	↖←↑ **3**	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↑ 7	↖←↑ 8	↖ 7
t	↑ 3	↖←↑ 4	↖←↑ **5**	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖ 7	←↑ 8	↖←↑ 9	↑ 8
e	↑ 4	↖ 3	← 4	↖← **5**	← **6**	← 7	←↑ 8	↖←↑ 9	↖←↑ 10	↑ 9
n	↑ 5	↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ **8**	↖←↑ 9	↖←↑ 10	↖←↑ 11	↖↑ 10
t	↑ 6	↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖ **8**	← 9	← 10	←↑ 11
i	↑ 7	↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↑ 9	↖ **8**	← 9	← 10
o	↑ 8	↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↑ 10	↑ 9	↖ **8**	← 9
n	↑ 9	↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↖←↑ 12	↑ 11	↑ 10	↑ 9	↖ **8**

- Consider transformation ‘chat’ to ‘had’
  - Compute MED and backtrace the corresponding alignment
  - Cost of insertion = cost of deletion = 1
  - Cost of substitution (in case of mismatch) = 2

Src ↓ \Tar→	**#**	**h**	**a**	**d**
#				
c				
h				
a				
t				