

Daniel Jurafsky & James H. Martin. Speech and Language Processing.
Глава 2, с. 2-11. <https://web.stanford.edu/~jurafsky/slp3/>

Регулярны
е
выражени
а текста
я

**Basic Text
Processing**

Regular Expressions

Крижановский А.А.
Локализация, добавление примеров и вопросов.

Regular expressions

- A formal language for specifying text strings
- How can we search for any of these?
 - самовар
 - вар
 - варешка
 - Варвара





Степа, тараща глаза, увидел, что на маленьком столике сервирован поднос, на коем имеется нарезанный белый хлеб, паюсная икра в вазочке, белые маринованные грибы на тарелочке, что-то в кастрюльке и, наконец, водка в объемистом ювелиршином графинчике. Особенно поразило Степу то, что графин запотел от холода. Впрочем, это было понятно – он помещался в полоскательнице, набитой льдом. Накрыто, словом, было чисто, умело.

~

~

```
/[бел]\{3,\}|л.д|чисто|ч\S*k
```

Формальные языки (ФЯ)

- РЕ описывают регулярные языки в теории формальных языков.
- ФЯ состоит из слов, порождённых РЕ.
- РЕ включают константы и операторы, которые задают:
 - множество строк,
 - и операций над ними

Константы, являющиеся RE

- над конечным алфавитом Σ
- (пустое множество) \emptyset
- (пустая строка) ε — множество, содержащее только пустую строку
- (символьный литерал) $a \in \Sigma$ — множество, содержащее только символ a .

Операции, генерирующие RE 1

- над конечным алфавитом Σ
- R, S — это RE
- (сцепление, конкатенация) RS
 - Пример: $\{"ab", "c"\}\{"d", "ef"\} = \{"abd", "abef", "cd", "cef"\}$.
- (дизъюнкция, чередование) $R|S$
 - $\{"ab", "c"\}|\{"ab", "d", "ef"\} = \{"ab", "c", "d", "ef"\}$.

Операции, генерирующие RE 2

- (замыкание Клини, звезда Клини) R^*

минимальное надмножество множества R , которое содержит ε и замкнуто относительно конкатенации

– $\{"ab", "c"\}^* = \{\varepsilon, "ab", "c", "abab", "abc", "cab", "cc", "ababab", "abscab", \dots\}$.

– $\{"0", "1"\}^* =$

Области применения RE

- поиск по шаблону
- замена (pattern substitution)
- chatbot (тест Тьюринга)
- **ТОКЕНИЗАЦИЯ**

Regular Expressions: Disjunctions

- Letters inside square brackets []

Pattern	Matches
<code>[wW]oodchuck</code>	Woodchuck, woodchuck
<code>[1234567890]</code>	Any digit

- Ranges `[A-Z]`

Pattern	Matches	
<code>[A-Z]</code>	An upper case letter	<u>D</u> renched Blossoms
<code>[a-z]</code>	A lower case letter	<u>m</u> y beans were impatient
<code>[0-9]</code>	A single digit	Chapter <u>1</u> : Down the Rabbit Hole

Regular Expressions: Negation in Disjunction

- Negations `[^Ss]`
 - Carat means negation only when first in []

Pattern	Matches	
<code>[^A-Z]</code>	Not an upper case letter	O <u>y</u> fn pripetchik
<code>[^Ss]</code>	Neither 'S' nor 's'	<u>I</u> have no exquisite reason"
<code>[^e^]</code>	Neither e nor ^	Look h <u>e</u> re
<code>a^b</code>	The pattern a carat b	Look up <u>a^b</u> now

Regular Expressions: More Disjunction

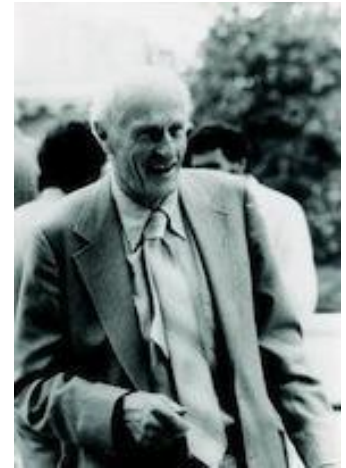
- Woodchucks is another name for groundhog!
- The pipe | for disjunction

Pattern	Matches
<code>groundhog woodchuck</code>	
<code>yours mine</code>	yours mine
<code>a b c</code>	= <code>[abc]</code>
<code>[gG]roundhog [Ww]oodchuck</code>	



Regular Expressions: ? * + .

Pattern	Matches	
<code>colou?r</code>	Optional previous char	<u>color</u> <u>colour</u>
<code>oo*h!</code>	0 or more of previous char	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
<code>o+h!</code>	1 or more of previous char	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
<code>baa+</code>		<u>baa</u> <u>baaa</u> <u>baaaa</u> <u>baaaaa</u>
<code>beg.n</code>		<u>begin</u> <u>begun</u> <u>begun</u> <u>beg3n</u>



Stephen C Kleene
Kleene *, Kleene +

Regular Expressions: Anchors **^** **\$**

Pattern	Matches
<code>^[A-Z]</code>	<u>P</u> alo Alto
<code>^[^A-Za-z]</code>	<u>1</u> <u>"Hello"</u>
<code>\.\$</code>	The end <u>.</u>
<code>.\$</code>	The end <u>?</u> The end <u>!</u>

Токенизация (две стратегии разбиения на слова)

- Пробелы?
- Слова?
 - Какие символы?

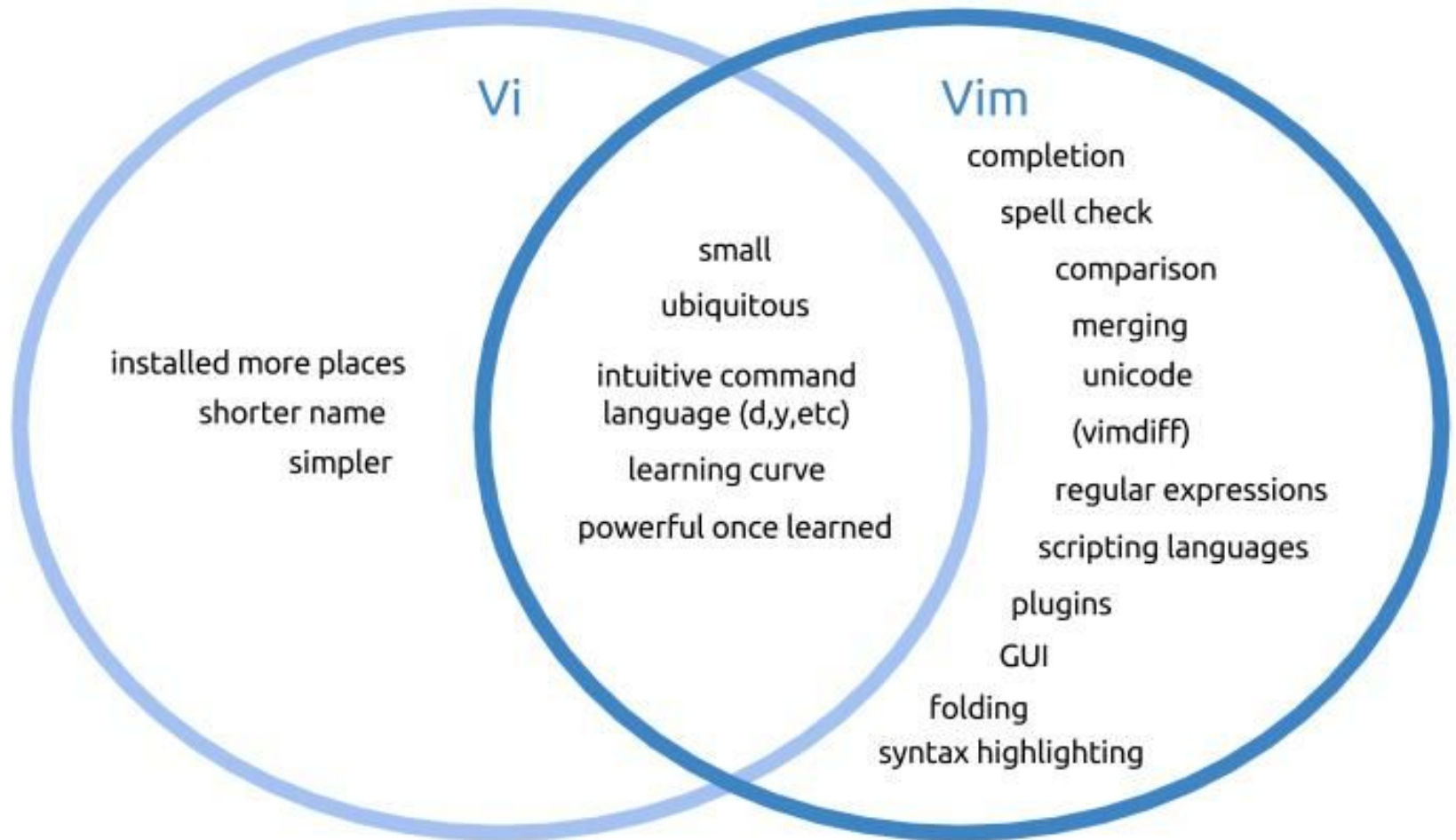
Разбиваем на слова (VIM):

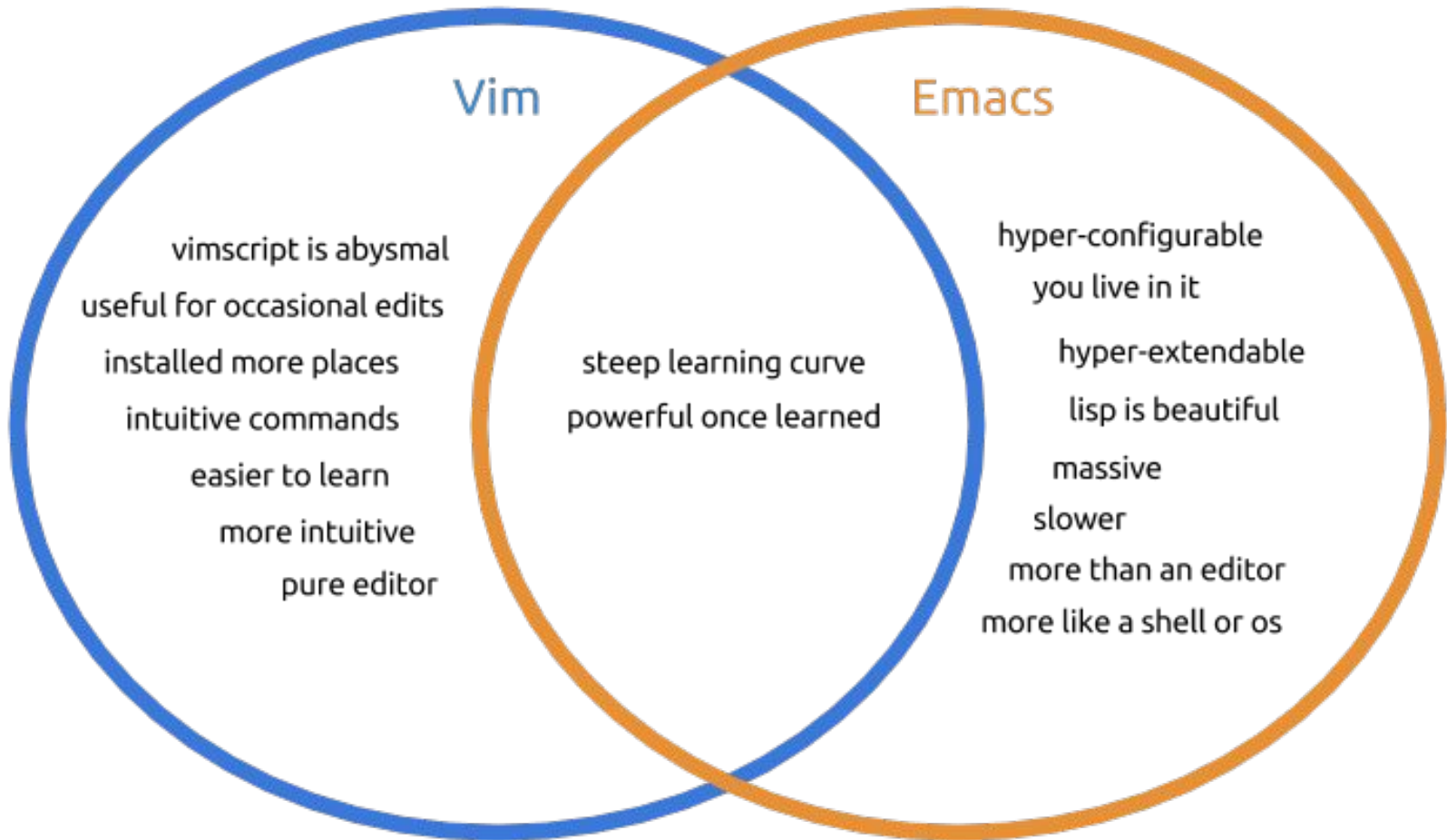
(1) ищем границы слов

- Но он не умер. Открыв слегка глаза, он увидел себя сидящим на чем-то каменном. Вокруг него что-то шумело. Когда он открыл, как следует, глаза, он увидел, что шумит море, и что даже больше того, — волна покачивается у самых его ног, и что, короче говоря, он

VIM

Vi, Vim and Emacs





Vim, Emacs

- Bill Joy
- 1976
- Unix modularity
- work like a language =>?
intuitive
installed nearly
- Richard Stallman
- 1976
- power and configurability
- live in the system as much as possible

Just memorize this vi / vim cheat sheet and you're ready for lightning-quick editing.

vi / vim graphical cheat sheet

Esc normal mode	external file	@, play macro	# prev. dir	\$ out	% goto %2 match	^ "soft" bol	& repeat	% next front	(begin (sentence)) end (sentence)	"soft" bol + next line	+ next line
~ toggle case	1	2	3	4	5	6	7	8	9	0 "hard" bol	- prev line	= auto. throw
Q ex mode	W next WORD	E end WORD	R rephrase mode	F back till	T "till" Y yank	U undo line	I insert at bol	O open above	P paste before	f begin (para)	F end (para)	. misc
q record (macro)	W next word	E end word	R rephrase char	F back till	T "till" Y yank	U undo	I insert mode	O open below	P paste after	f misc	. misc	. misc
A append at eol	C subst line	D delete in eol	P "back" find char	G "to" find char	H screen top	J join lines	K help	screen up	. ex cmd line	H reg. / gutters	. misc	. misc
Z append	S subst char	D delete char	f find char	g, extra find char	H screen top	j down	K help	screen down	. repeat / /?/?/	. goto mk, bol	. not used!	. not used!
Z quit	X back-space	C change to eol	V visual WORD	N next (find)	M screen mid	n next (find)	M, set, reverse	> indent	P, find (ex-c)	. not used!	. not used!	. not used!
Z, extra eol	X delete char	C change	V visual mode	b prev next	n next (find)	M, set, reverse	V, CTRL	< unindent	P, find (ex-c)	. not used!	. not used!	. not used!

motion moves the cursor, or defines the range for an operator
command direct action command, if col, it enters insert mode
operator requires a motion afterwards, operates between cursor & destination
extra special functions, requires extra input commands with a dot need a clear argument afterwards
Q bol = beginning of line, eol = end of line.
 mk = mark, yank = copy
 words: `quax[efol] bax[]`
 WORDS: `ruux [foo] bax[]`

Main command line commands ('ex'):
 w (save), zq (quit), zQ (quit w/o saving)
 :f (open file f), :Ba/s/v (replace 's' by 'v' filewide), :h (help in vim), :mew (new file in vim).

Other important commands:
 CTRL-R: redo (vim), CTRL-]/-/: page up/down, CTRL-]/-/: scroll line up/down, CTRL-V: block-visual mode (vim only)

Visual modes:
 Move around and type operator to act on selected region (vim only)

Notes:
 (1) use % before a yank/paste/del command to use that register (Clipboard) (e.g., %a, %*)
 (2) type in a number before any action to repeat it that number of times (e.g.: 2p, d2w, j1, d4)
 (3) duplicate operator to act on current line (dd = delete line, >> = indent line)
 (4) ZZ to save & quit, ZQ to quit w/o saving
 (5) zt: scroll cursor to top, zb: bottom, zz: center
 (6) gg: top of file (vim only), gG: open file under cursor (vim only)

For a graphical vi/vim tutorial & more tips, go to www.viemu.com - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

vi / vim graphical cheat sheet

Esc
normal mode

~ toggle case	! external filter	@. play macro	# prev ident	\$ eol	% goto match	^ "soft" bol	& repeat :s	* next ident	(begin sentence) end sentence	"soft" bol down	+ next line
. goto mark	1	2	3	4	5	6	7	8	9	0 "hard" bol	- prev line	= auto ³ format
Q ex mode	W next word	E end word	R replace mode	T back 'till	Y yank line	U undo line	I insert at bol	O open above	P paste before	{ begin parag.	}	end parag.
q record macro	w next word	e end word	r. replace char	t. 'till	y yank ^{1,3}	u undo	i insert mode	o open below	p paste ¹ after	[. misc]. misc	
A append at eol	S subst line	D delete to eol	F."back" find ch	G eof/goto ln	H screen top	J join lines	K help	L screen bottom	. ex cmd line	" reg. ¹ spec	bol/goto col	
a append	s subst char	d delete ^{1,3}	f. find char	g. extra ⁶ cmds	h ←	j ↓	k ↑	l →	. repeat ; t/T/f/F	' goto mk. bol	\ not used!	
Z quit ⁴	X back-space	C change to eol	V visual lines	B prev word	N prev (find)	M screen mid'l	< un- ³ indent	> indent ³	? find (rev.)			
Z extra ⁵ cmds	X delete char	c change ^{1,3}	V visual mode	b prev word	n next (find)	m. set mark	reverse ; t/T/f/F	. repeat cmd	/ find			

- motion** moves the cursor, or defines the range for an operator
- command** direct action command, if **red**, it enters insert mode
- operator** requires a motion afterwards, operates between cursor & destination
- extra** special functions, requires extra input

q. commands with a dot need a char argument afterwards
 bol = beginning of line, eol = end of line, mk = mark, yank = copy
 words: `quux(foo, bar, baz);`
 WORDS: `quux(foo, bar, baz);`

Main command line commands ('ex'):
 :w (save), :q (quit), :q! (quit w/o saving)
 :e f (open file f),
 :%s/x/y/g (replace 'x' by 'y' filewide),
 :h (help in vim), :new (new file in vim),
Other important commands:
 CTRL-R: redo (vim),
 CTRL-F/-B: page up/down,
 CTRL-E/-Y: scroll line up/down,
 CTRL-V: block-visual mode (vim only)
Visual mode:
 Move around and type operator to act on selected region (vim only)

- Notes:**
- (1) use "x before a yank/paste/del command to use that register ('clipboard') (x=a..z,") (e.g.: "ay\$ to copy rest of line to reg 'a')
 - (2) type in a number before any action to repeat it that number of times (e.g.: 2p, d2w, 5l, d4j)
 - (3) duplicate operator to act on current line (dd = delete line, >> = indent line)
 - (4) ZZ to save & quit, ZQ to quit w/o saving
 - (5) zt: scroll cursor to top, zb: bottom, zz: center
 - (6) gg: top of file (vim only), gf: open file under cursor (vim only)

Разбиваем на слова (VIM):

(1) ищем границы слов

Но он не умер. Открыв слегка глаза, он увидел себя сидящим на чем-то каменном. Вокруг него что-то шумело. Когда он открыл, как следует, глаза, он увидел, что шумит море, и что даже больше того, — волна покачивается у самых его ног, и что, короче говоря, он сидит на самом конце мола, и что под ним голубое сверкающее море, а сзади — красивый город на горах. ■



vim_re_ex1.txt + ...screenshot) - GVIM1



Но он не умер. Открыв слегка глаза, он увидел себя сидящим на чем-то каменном. Вокруг него что-то шумело. Когда он открыл, как следует, глаза, он увидел, что шумит море, и что даже больше того, — волна покачивается у самых его ног, и что, короче говоря, он сидит на самом конце мола, и что под ним голубое сверкающее море, а сзади — красивый город на горах.

~

/[.!? ,]*\s

Токенизация

- Пробелы,
но не всегда:
 - Saint Petersburg
 - всё равно
(частица)
 - ?
- Слова?
 - Поздравляю с 8
марта :) #весна

Токенизация

- Пробелы,
но не всегда:
 - Saint Petersburg
 - всё равно
(частица)
 - ?
- Слова?
 - Поздравляю с 8 марта :) #весна
 - + смайлики,
 - + хештеги

!Китайский язык: 1755年1月25日俄罗斯女沙皇伊丽莎白·彼得罗芙娜下令建立莫斯科大学, 同年4月26日该大学开始授课。至今为止在俄罗斯1月25日是大学生节。

Разбиваем на слова:
(2) ищем сами слова

- The detailed elaborations on the development of even a short program form a long story, indicating that careful programming is not a trivial subject. If this paper has helped to dispel the widespread belief that programming is easy as long as the programming language is powerful enough and the available computer is fast enough, then it has achieved one of its purposes. #NiklausWirth

#programming

Niklaus Wirth. Program Development by Stepwise Refinement. 1995.

<http://sunnyday.mit.edu/16.355/wirth-refinement.html>

Разбиваем на слова (2): ищем сами слова

- 1) ищем слова *program* и *programming*,
 - поиск по двум словам, какое раньше?
 - поиск слов с корнем *program*,
 - RE: **Quantifiers, Greedy and Non-Greedy**
*, *?, +? VIM: *, \+, \=, \{-}
- 2) замена (substitute) на WORD,

Жадные (Ленивые) квантификаторы:

? * + .

Vim greedy	Vim non-greedy	Matches	
*	\{-}	0 or more of previous char	<u>color</u> <u>colour</u>
\+	\+?	1 or more of previous char	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
{m, n}	\{-n, m\}	from m to n of previous char	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>

Разбиваем на слова: ИЩЕМ САМИ СЛОВА

- The detailed elaborations on the development of even a short program form a long story, indicating that careful programming is not a trivial subject. If this paper has helped to dispel the widespread belief that programming is easy as long as the programming language is powerful enough and the available computer is fast

Разбить текст на предложения

- Но он не умер. открыв слегка глаза, он увидел себя сидящим на чем-то каменном. Вокруг него что-то шумело. Когда он открыл, как следует, глаза, он увидел, что шумит море, и что даже больше того, — волна покачивается у самых его ног, и что, короче говоря, он сидит на самом конце мола, и что под

vim_re_ex1.txt + (/da...nce/screenshot) - GVIM1



Но он не умер. открыв слегка глаза, он увидел себя сидящим на чем-то каменном. Вокруг него что-то шумело. Когда он открыл, как следует, глаза, он увидел, что шумит море, и что даже больше того, – волна покачивается у самых его ног, и что, короче говоря, он сидит на самом конце мола, и что под ним голубое сверкающее море, а сзади – красивый город на горах.

1,141-83

All



Но он не умер. открыв слегка глаза, он увидел себя сидящим на чем-то каменном. Вокруг него что-то шумело. Когда он открыл, как следует, глаза, он увидел, что шумит море, и что даже больше того, – волна покачивается у самых его ног, и что, короче говоря, он сидит на самом конце мола, и что под ним голубое сверкающее море, а сзади – красивый город на горах.

~
~
~

/[.!?]\{1,3}\s[АБВГДЕЁЖЗК]

Example

- Find me all instances of the word “the” in a text.

`the`

Misses capitalized examples

`[tT]he`

Incorrectly returns `other` or `theology`

`[^a-zA-Z][tT]he[^a-zA-Z]`

Errors

- The process we just went through was based on **fixing two kinds of errors**
 - Matching strings that we should not have matched (**there, then, other**)
 - **False positives (Type I)**
 - Not matching things that we should have matched (The)
 - **False negatives (Type II)**

Errors cont.

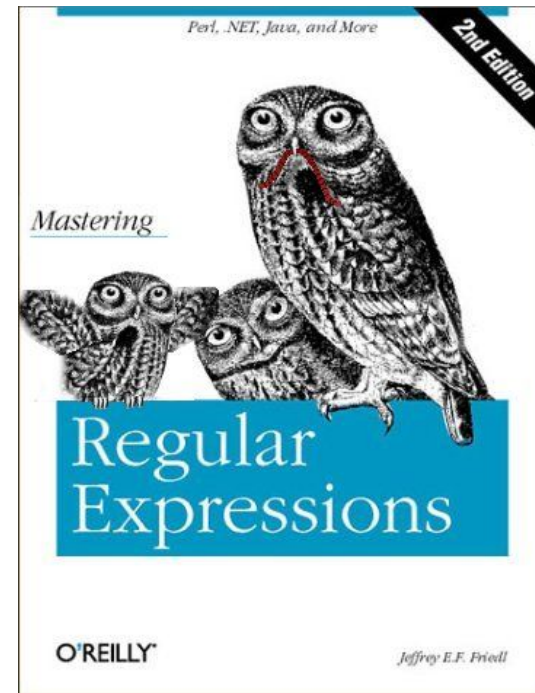
- In NLP we are always dealing with these kinds of errors.
- Reducing the error rate for an application often involves two antagonistic efforts:
 - **Increasing accuracy or precision** (minimizing false positives)
 - **Increasing coverage or recall** (minimizing false negatives).

Summary

- Regular expressions play a surprisingly large role
 - Sophisticated sequences of regular expressions are often the first model for any text processing text
- For many hard tasks, we use machine learning classifiers
 - But regular expressions are used as features in the classifiers
 - Can be very useful in capturing generalizations

Литература

- Фридл, Дж. Регулярные выражения. — СПб.: «Питер», 2001. — 352 с. (Mastering Regular Expressions)
- Miessler D. The Differences Between Vi, Vim, and Emacs.



ССЫЛКИ

- <http://vimregex.com>
-



Basic Text Processing

Regular Expressions