Regular Expressions

Natalie Parde

UIC CS 421

One way to preprocess text is by using regular expressions.

- Regular expressions: A formal language for specifying text strings.
- How can we search for any of these?
  - Donut
  - donut
  - Doughnut
  - doughnut
  - Donuts
  - doughnuts

Regular Expression Terminology

Regex: Common abbreviation for **reg**ular **ex**pression

Disjunction: Logical OR

Range: All characters in a sequence from c_1-c_2

Negation: Logical NOT

Scope: Indicates to which characters the regex applies

Anchor: Matches the beginning or end of a string

Regular Expressions: Disjunctions (and Ranges)

- Disjunction: Letters inside square brackets [az]
- Range: Hyphen between the first and last characters in the range [a-z]

Pattern	Matches	Example
[dD]onut	donut, Donut	This morning would be better with a **donut**.
[0123456789]	Any digit	This morning would be better with **5** donuts.
[A-Z]	An uppercase letter	**D**onuts are an excellent way to start the day.
[0-9]	Any digit	I just ate **5** donuts.

Regular Expressions: Negation in Disjunction

- Negation: A caret (^) at the beginning of a disjunction [^az]
  - The caret must be at the beginning of the disjunction to negate it

Pattern	Matches	Example
[^dD]onut	Any letter except “d” or “D” before the sequence “onut”	This morning would be better with a co**c**onut.
[^A-Z]	Not an uppercase letter	D**o**nuts are an excellent way to start the day.
[^^]	Not a caret	**W**hat is your favorite kind of donut?
D^o	The pattern “D^o”	Is **D^o**nut a good name for my donut shop?

Regular Expressions: More Disjunction

- The pipe | indicates the union (logical OR) of two smaller regular expressions
- a|b|c is equivalent to [abc]

Pattern	Matches	Example
d|D	“d” or “D”	This morning would be better with a **d**onut.

Regular Expressions: Special Characters

- *****: Means that there must be 0 or more occurrences of the preceding expression
- **.**: A wildcard that can mean any character
- **+**: Means that there must be 1 or more occurrences of the preceding expression
- **?**: Means that there must be 0 or 1 occurrences of the preceding expression
- **{m}**: Means that there must be *m* instances of the preceding expression
- **{m,n}**: Means that there must be between *m* and *n* instances of the preceding expression
- **(abc)**: Means that the operation should be applied to the specified sequence

Regular Expressions: Special Characters

Pattern	Matches	Example
donuts*	“donut” or “donuts” or “donutss” or “donutsss”….	This morning I had many **donuts**.
.onut	Any character followed by “onut”	Can I have a co**conut donut**?
donuts+	“donuts” or “donutss” or “donutsss”….	Do you want one donut or two **donuts**?
donuts?	“donut” or “donuts”	Do you want one **donut** or two **donuts**?
donuts{1}	“donuts”	Do you want one donut or two **donuts**?
donuts{0,1}	“donut” or “donuts”	Do you want one **donut** or two **donuts**?
.o(nut)?	Any character followed by “o” or “onut”	Can I have a dis**co donut**?

Regular Expressions: Anchors

- Indicate that a pattern should be matched only at the beginning or end of a word

Pattern	Matches	Example
^Donuts	"Donuts" only when it is at the beginning of a string	**Donuts** are an excellent way to start the day.
donuts\.$	"donuts." only when it is at the end of the string	I just ate 5 **donuts.**

Case Example: Regex for “the”

the

Fails on: The

[tT]he

Fails on: other

^[tT]he$

Fails on: Grab the disinfectant!

[^.][tT]he[^.]

Fails on: My favorite word is “the.”

Errors

- In iterating through possible solutions to avoid failures, we were trying to fix two types 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

- This is a recurring theme in NLP!
- 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)

Regular Expressions: Takeaway Points

Regular expressions are a surprisingly powerful tool!

They are critical to text tokenization and normalization.

They may also be used to extract **features** for machine learning classifiers.