NAVAL POSTGRADUATE SCHOOL

OS4118 Statistical and Machine Learning

Regular Expressions

Prof. Sam Buttrey

Fall AY 2020

The Nation's Premiere Defense Research University

Monterey, California
WWW.NPS.EDU

NAVAL POSTGRADUATE SCHOOL

Regular Expressions

- Common in text-matching applications: every scripting language handles these
- A "regular expression" (regex) describes a set of rules to match or not match strings
- Generalizes the "wildcard" match (which is called a "glob" – `see glob2rx()`)
- Can be very powerful and very complicated

Regular Expressions

“Some people, when confronted with a problem, think ‘I know, I'll use regular expressions.’

“Now they have two problems.”

-- Jamie Zawinski

NAVAL POSTGRADUATE SCHOOL

Regular Expressions

- Regular expressions come in a number of varieties, each with slightly different rules
- Two of the main types are **POSIX**
  - …which includes “basic” and “extended”
  - Often with “GNU extensions,” as in R
- And **PCRE**, sometimes called “Perl,”
  - …although the Perl language uses its own variety which is a little different from PCRE
- Python re is “similar to Perl”

Regular Expressions

- In R, `grep` (and `regexpr` and relatives) use extended POSIX/Gnu by default
- `perl = TRUE` selects PCRE
- I’ve been using POSIX, but PCRE are also very popular
- Be sure you know which kind you’re using as you look for help etc.

Matching

- R `grep()` matches strings in a character **vector**
  - By default, returns indices of matching ones
- Common implemtation `egrep` in the bash shell matches **lines** in text file(s)
  - By default, returns every line that matches
- Lots of options to modify behavior, but regular expressions are essentially for finding **text** within **lines**

NAVAL POSTGRADUATE SCHOOL

Matching

- The goal is to identify strings that match a **pattern**
- Letters, numbers, and the space character match themselves in a pattern:
  - `grep ("cat", vec) # lines w/ "cat"`
  - `grep (" CA", vec) # find " CA "`
  - Don’t include spaces “for readability”
- If the pattern is fixed, we can use use `fixed=TRUE` in R for speed, convenience

NAVAL POSTGRADUATE SCHOOL

Regex, cont'd

- Some characters have special meanings in patterns
  - These are all non-alphanumeric and will usually need to be **escaped** in order to have literal meaning, depending on the variety of regex
- E.g. $ means "end of line"

`grep ("$", vec)` # find lines that end (!)

`grep ("t$", vec)` # find lines that end with a lower-case `t`

Special Characters (i)

- Special characters:
  `. \ | ( ) [ { ^ $ * + ?`
- To use one of these as itself, “escape” it by preceding it with a backslash…
- ..except that backslash is already a special character in R; we type it as \\

```
grep ('\\$',  vec) #lines w/$
grep ('\\\\', vec) #lines w/ \
= grep ('\\', vec, fixed= TRUE)
```

Special Characters (ii)

- ^ matches start of line, $ matches end
  - `grep ('^The', vec)`#lines starting with The
  - `grep ('^The', vec,`
    `ignore.case = TRUE)` # ignore case
  - `.` (dot) matches any one character, so `grep ('^.$', vec)` matches lines with exactly one character (dot is special)
  - `grep ('^\\.$', vec)` matches lines with exactly one dot and nothing else

More grep Arguments

- `grep()` tells you the indices of the strings that match (as a numeric vector)
- With `value = TRUE`, it returns the matching strings themselves
- With `value = FALSE` and `invert = FALSE`, gives the indices of strings where there is **no** match
- Also useful: `ignore.case` (default `FALSE`)
- `grepl()` returns a logical vector with `TRUE` for match and `FALSE` for no match

Special Characters (iii)

- [ ] introduces a **class** of characters
  - `grep ('ae', ...)` matches lines with ae
  - `grep ('[ae]', ...)` matches lns with a or e
- Ranges are permitted:
  - `grep ('[0-9]',...)` matches lines with a digit, but beware collation sequences for letters
- A number of classes are predefined:
  - `'[[:alpha:]]'` matches a letter; `[[:upper:]]`, `[[:digit:]]`, `[[:alnum:]]`, `[[:punct:]]`, and more are available

Character Classes (cont'd)

- A character class starting with ^ **excludes** characters in the class
  - `grep ('[^t]'`... # lines w/ at least one non-t
  - ^ elsewhere than at the start refers to itself, `so egrep '[t^]'` gets lines with t or ^
- Classes are for single characters: use **pipe** for "word1 or word2":
  - `egrep 'NA|[Mm]issing'` finds NA, missing, Missing – **parentheses** establish precedence

GNU shortcuts

- `\w` matches a character “in a word” – that is, letter, digit, underscore
  - = `[[:alpha:]]` ; `\W` = `[^[:alpha:]]`
- `\s`, `\S` = `[[:space:]]`, `[^[:space]]`
- `\b` matches empty string at beginning or end of word; `\<` and `\>` match at start, end; `\B` matches empty inside a word
  - '`dirty \Brat`' doesn’t match '`dirty rat`'

Repetition

- ? matches 0 or 1 times (“item is optional”)
- * matches 0 or more; + 1 or more times, so…
- '`^[^t]*$`' matches lines with no t
- '`[0-9]? [[:alpha]]+`' matches lines w/ 0+ digits, space, then a bunch of letters
- {m,n} matches m-n times, so…
- '`[0-9]{2,4}`' : 2-4 digits (or more, right?)
- '**`\\<`**`[0-9]{3}`**`\\>`**' : **word** w/exactly 3 digits

Example

- Suppose an e-mail address is “valid” if:
  - “Local part” starts with a letter or digit, and subsequent characters are letter, digit, dot;
  - An @ separates the local and domain part
  - The domain part is made of two or three “words” of letters and/or digits, separated by dot(s)
  - (The actual rules are mostly more permissive, though double-dots forbidden)
- Find lines with (only) e-mail addresses

E-mail example

- Start with alpha, follow with zero or more characters from (alpha or dot)
  - `[[:alnum:]]([[:alnum:]]|\.)*`
- Then the @ sign, representing itself: @
- Then one or two (word, dot) pairs, optionally followed by one more word: `([[:alnum:]]+\.){1,2}[[:alnum:]]+`

R Relatives, part 1 (recap)

- `grep()` tells you the indices of the strings that match (as a numeric vector)
- With `value = TRUE`, it returns the matching strings themselves
- `grepl()` returns a logical vector
- In each case the function describes an attribute of the containing string, not so much about the match itself
  - Does the string match the pattern or not?

R Relatives part 2

- Tools for extracting the matched text
- `regexpr ()` returns a vector telling **where** in the string the 1st match occurs (or –1), plus an indication of match length
- Of course lengths can vary, depending on the type of pattern being matched
- `gregexpr()` describes all matches
  - Use these with `regmatches()` to extract the matching part

NAVAL POSTGRADUATE SCHOOL

Greedy Matching

- By default matching is maximal, so the match length of '`b.*a`' in '`banana pear`' is 9
- Adding ? to the repetition operator will make the match minimal
  - `[[:alpha:]| ]+` – get the maximal set of words and spaces, starting at character 1
  - `[[:alpha:]| ]+?` –first word/space char

Relatives 3: Search and Replace

- R commands `sub()` and `gsub()` let you replace strings with other strings
  - `sub()`: first match; `gsub()`, all matches
- Stuff that matches inside parentheses can be used in the replace; those are **backreferences**, referred to by `\1`, `\2`, …
- Ex: turn “first last” names into “last, first”

-
  ```
  sub ("([[:alpha:]]+) ([[:alpha:]| ]+)",
                "\\2,\ \\1", nm)
  ```

Advanced Features

- Think of regex as a language, of which we now know a useful subset
- Features include conditionals, lookaheads/behinds/arounds, others
- Most flavors handle Unicode/UTF-8, but support differs from place to place
- Expect slight variations from one implementation to the next