Chapter 30: Pattern Matching and Replacement

This topic covers matching string patterns, as well as extracting or replacing them. For details on defining complicated patterns see Regular Expressions.

Section 30.1: Finding Matches

```
# example data
test_sentences <- c("The quick brown fox", "jumps over the lazy dog")
```

Is there a match?

`grepl()` is used to check whether a word or regular expression exists in a string or character vector. The function returns a TRUE/FALSE (or "Boolean") vector.

Notice that we can check each string for the word "fox" and receive a Boolean vector in return.

```
grepl("fox", test_sentences)
#[1]  TRUE FALSE
```

Match locations

`grep` takes in a character string and a regular expression. It returns a numeric vector of indexes.This will return which sentence contains the word "fox" in it.

```
grep("fox", test_sentences)
#[1] 1
```

Matched values

To select sentences that match a pattern:

```
# each of the following lines does the job:
test_sentences[grep("fox", test_sentences)]
test_sentences[grepl("fox", test_sentences)]
grep("fox", test_sentences, value = TRUE)
# [1] "The quick brown fox"
```

Details

Since the `"fox"` pattern is just a word, rather than a regular expression, we could improve performance (with either `grep` or `grepl`) by specifying `fixed = TRUE`.

```
grep("fox", test_sentences, fixed = TRUE)
#[1] 1
```

To select sentences that *don't* match a pattern, one can use `grep` with `invert = TRUE`; or follow subsetting rules with `-grep(...)` or `!grepl(...)`.

In both `grepl(pattern, x)` and `grep(pattern, x)`, the `x` parameter is vectorized, the `pattern` parameter is not. As a result, you cannot use these directly to match `pattern[1]` against `x[1]`, `pattern[2]` against `x[2]`, and so on.

Summary of matches

After performing the e.g. the `grepl` command, maybe you want to get an overview about how many matches where `TRUE` or `FALSE`. This is useful e.g. in case of big data sets. In order to do so run the `summary` command:

```r
# example data
test_sentences <- c("The quick brown fox", "jumps over the lazy dog")

# find matches
matches <- grepl("fox", test_sentences)

# overview
summary(matches)
```

Section 30.2: Single and Global match

When working with regular expressions one modifier for PCRE is `g` for global match.

In R matching and replacement functions have two version: first match and global match:

- `sub(pattern,replacement,text)` will replace the first occurrence of pattern by replacement in text
- `gsub(pattern,replacement,text)` will do the same as sub but for each occurrence of pattern
- `regexpr(pattern,text)` will return the position of match for the first instance of pattern
- `gregexpr(pattern,text)` will return all matches.

Some random data:

```r
set.seed(123)
teststring <- paste0(sample(letters,20),collapse="")

# teststring
#[1] "htjuwakqxzpgrsbncvyo"
```

Let's see how this works if we want to replace vowels by something else:

```r
sub("[aeiouy]"," ** HERE WAS A VOWEL** ",teststring)
#[1] "htj ** HERE WAS A VOWEL** wakqxzpgrsbncvyo"

gsub("[aeiouy]"," ** HERE WAS A VOWEL** ",teststring)
#[1] "htj ** HERE WAS A VOWEL** w ** HERE WAS A VOWEL** kqxzpgrsbncv ** HERE WAS A VOWEL**  ** HERE
WAS A VOWEL** "
```

Now let's see how we can find a consonant immediately followed by one or more vowel:

```r
regexpr("[^aeiou][aeiou]+",teststring)
#[1] 3
#attr(,"match.length")
#[1] 2
#attr(,"useBytes")
#[1] TRUE
```

We have a match on position 3 of the string of length 2, i.e: `ju`

Now if we want to get all matches:

```r
gregexpr("[^aeiou][aeiou]+",teststring)
#[[1]]
#[1]  3  5 19
#attr(,"match.length")
#[1] 2 2 2
```

```
#attr(,"useBytes")
#[1] TRUE
```

All this is really great, but this only give use positions of match and that's not so easy to get what is matched, and here comes `regmatches` it's sole purpose is to extract the string matched from regexpr, but it has a different syntax.

Let's save our matches in a variable and then extract them from original string:

```
matches <- gregexpr("[^aeiou][aeiou]+",teststring)
regmatches(teststring,matches)
#[[1]]
#[1] "ju" "wa" "yo"
```

This may sound strange to not have a shortcut, but this allow extraction from another string by the matches of our first one (think comparing two long vector where you know there's is a common pattern for the first but not for the second, this allow an easy comparison):

```
teststring2 <- "this is another string to match against"
regmatches(teststring2,matches)
#[[1]]
#[1] "is" " i" "ri"
```

Attention note: by default the pattern is not Perl Compatible Regular Expression, some things like lookarounds are not supported, but each function presented here allow for `perl=TRUE` argument to enable them.

Section 30.3: Making substitutions

```
# example data
test_sentences <- c("The quick brown fox quickly", "jumps over the lazy dog")
```

Let's make the brown fox red:

```
sub("brown","red", test_sentences)
#[1] "The quick red fox quickly"       "jumps over the lazy dog"
```

Now, let's make the `"fast"` fox act `"fastly"`. This won't do it:

```
sub("quick", "fast", test_sentences)
#[1] "The fast red fox quickly"       "jumps over the lazy dog"
```

`sub` only makes the first available replacement, we need `gsub` for global replacement:

```
gsub("quick", "fast", test_sentences)
#[1] "The fast red fox fastly"       "jumps over the lazy dog"
```

See Modifying strings by substitution for more examples.

Section 30.4: Find matches in big data sets

In case of big data sets, the call of `grepl("fox", test_sentences)` does not perform well. Big data sets are e.g. crawled websites or million of Tweets, etc.

The first acceleration is the usage of the `perl = TRUE` option. Even faster is the option `fixed = TRUE`. A complete example would be:

```r
# example data
test_sentences <- c("The quick brown fox", "jumps over the lazy dog")

grepl("fox", test_sentences, perl = TRUE)
#[1]  TRUE FALSE
```

In case of text mining, often a corpus gets used. A corpus cannot be used directly with **`grepl`**. Therefore, consider this function:

```r
searchCorpus <- function(corpus, pattern) {
  return(tm_index(corpus, FUN = function(x) {
    grepl(pattern, x, ignore.case = TRUE, perl = TRUE)
  }))
}
```