R Programming Assignment **Isaac Murray**
MM706 - Programming for Analytics Due Date: 05/12/2016
Dr D Savostyanov

Introduction

Identifying gender from name data is a common problem for researchers [2]. This report describes a method for identifying gender from compound name data using the R programming language [4].

Investigation

The dataset supplied for this programming task contains name data that has the following general format: $\langle surname \rangle \langle , \rangle \langle honorific \rangle \langle firstname(s) \rangle$. There are some records within the data that deviate from this format e.g. first name is abbreviated to first initial or the honorific comes after the first names. Surnames contain spaces, hyphens and other characters. The honorific within the name data is a mixture of English and other European languages and includes military rank; religious order; title; and professional/academic. Some of the honorifics indicate gender (e.g. Mr, Mrs) whereas others do not (e.g. Dr).

The R `gender` package [3] has been developed to identify gender from first name. The `gender` package provides methods that can be incorporated into a solution for identifying gender where first name is present and can be extracted.

Solution

Exclusive use of either honorific or first name cannot be relied upon to identify gender in the supplied dataset. For this reason, a combination of honorific and first name approaches has been chosen. Visual checking of name indicates that characters to the left of the first comma can be considered surname. As surname is not useful for identifying gender, it is ignored. A regular expression [5] is used to define each gender by combining multiple gender-specific honorifics into a single regular expression; these are matched within the name to determine gender. Where honorific does not identify gender, first name is extracted from the name and processed by the `gender` package.

The solution includes a default list of all gender specific honorifics contained within the supplied dataset. It is also possible for the user to create bespoke lists of honorifics, creating a flexible solution that can adapt to new datasets. For example, if a dataset were being analysed where the name data contained sparse first names and large numbers of military honorifics, the user has the option to add the military honorifics and have them processed as male. A more extensive list of non-English language honorifics can be added if required.

Implementation

The solution is implemented as an R script, `Gender.R`. The method providing the functionality is `getGender`. This method has the following arguments:
`name`: Character or factor vector of name being checked for gender.
`honorifics`: Character matrix containing honorific and corresponding gender.

`firstname.pos`: Position of block of alpha characters to right of comma defining first name within name.
`default`: Returned when name cannot be matched.

The honorifics argument is a matrix of honorifics and gender codes (read from a file or defined in R code) e.g.

Mr	male
Master	male
Mrs	female
Miss	female
Ms	female

This matrix is translated into another matrix of gender code and regular expression:

male	`(^|[^A-Za-z])(Mr|Master)($|[^A-Za-z])`
female	`(^|[^A-Za-z])(Mrs|Miss|Ms)($|[^A-Za-z])`

These regular expressions translate as: *Consider name data matched where honorifics occur within the name data and are prefixed and suffixed with either no character or a non-alphabet character.*
The name data is tested for matches against each regular expression, returning the corresponding gender when matched.
With this approach, *Smith, Mr John; "Smith" , "Mr" "John" ; Smith,Mr John* all return **male**.
Using the above method, the name *Smith, Dr John* does not return gender as *Dr* can be applied to either gender. In such cases, the first name is extracted and the `gender` package is used. In order to process first name correctly, an assumption is made regarding the block of text within the name data that corresponds to first name. For the supplied dataset, the second block of text to the left of the comma is assumed to be first name although this can be overridden by the user. In the case of the name *Smith, Dr J.*, the gender cannot be identified and a default is returned.

Analysis & Testing

The following code listing provides some analysis and testing for the `getGender` method. It is assumed the supplied dataset `titanic-c.csv` is in the R working directory.
The commands have been run on Windows 10 Professional and Ubuntu 16.10 Linux operating systems using a laptop computer with the following specification:
Processor: Intel64 Family 6 Model 69 Stepping 1 GenuineIntel 2501 Mhz
Total Physical Memory: 16,281 MB

```
# Read in dataset.
ti <- read.csv("titanic-c.csv")

# Determine number of rows.
nrow(ti)
[1] 2208

# Time creation of and population of gender column.
system.time(ti$Gender <- getGender(ti$Name))
```

Elapsed time: 5.68/2.44 seconds (Windows 10/Ubuntu Linux).

```
# Show variety of resulting values.
unique(ti$Gender)
[1] "female" "male"

# Show sample of result.
library(dplyr)
sample_n(ti[c("Name","Gender")],20)
```

	Name	Gender
748	FISCHER, Mr Eberhard Thelander	male
513	RISIEN, Mrs Emma	female
2097	OLIVE, Mr Charles	male
140	REED, Mr James George	male
296	BROWN, Mr Joseph James	male
366	HUMBY, Mr Frederick	male
56	HIRVONEN, Mrs Helga Elisabeth Lindqvist	female
615	BAGGOTT, Mr Allen Marden	male
67	HARDER, Mrs Dorothy	female
1654	BAUMANN, Mr John D.	male
1747	LOCKE, Mr A.	male
12	REEVES, Mr F.	male
679	BROWN, Mr Thomas William Solomon	male
880	CORNELL, Mrs Malvina Helen	female
2107	NICHOLS, Mr Walter Henry	male
1259	DAVISON, Mrs Mary E.	female
168	MAIONI, Miss Roberta Elizabeth Mary	female
1323	LINDEBERG-LIND, Mr Erik Gustaf	male
261	COMBES, Mr George	male
1367	WARD, Mr J.	male

```
# Return default value.
getGender("Jones, Dr J",default = "XXX")
[1] "XXX"
```

Evaluation

The solution described in this report has correctly identified gender for records in the supplied dataset.

References

[1] infoplease. Active duty military female personnel by rank / grade. URL `http://www.infoplease.com/us/military/active-duty-personnel-women.html`.

[2] Lincoln Mullen. Predicting gender using historical data, 2015. URL `https://cran.r-project.org/web/packages/gender/vignettes/predicting-gender.html`.

[3] Lincoln Mullen. *gender: Predict Gender from Names Using Historical Data*, 2015. URL `https://github.com/ropensci/gender`. R package version 0.5.1.

[4] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016. URL `https://www.R-project.org/`.

[5] Wikipedia. Regular expression. URL `https://en.wikipedia.org/wiki/Regular_expression`.

Appendices

Reflection

The main challenge of the R programming task has been striking a balance between the specific requirements of the supplied dataset, and developing a useful program for novel datasets. An initial idea was to make no assumptions regarding the exact format of the name data and rely solely on regular expression matches to identify gender. This would provide a great deal of flexibility but would not have identified gender for all of the names in the supplied dataset due to the presence of the neutral *Dr* and military honorifics[1]. There is also the possibility that gender could be misreported in a name such as *Master, Dr Jane* where the surname is a coincidental honorific and the honofic is gender neutral.
Another idea considered was to allow the user to specify the postion within the name data of each part of the name. This could enable the program to match first on gender-specific honorifics and secondly on first name using the the `gender` package. This approach was rejected as the supplied dataset was not completely consistent in terms of the position of each part of the name and multi-part surnames could create inaccuracies. It was also discovered that splitting the name into chunks of characters had a detrimental effect on performance.
A compromise approach has been decided upon, where is is assumed the position of the first comma within the name delineates surname; the program attempts a match on a gender-specific honorific within the remainder of the name. If a gender match cannot be found using honorific, the program uses first name. This correctly identifies the gender of names within the supplied dataset and gives a degree of flexibilty to process similar but not necessarily identical name formats. The primary constraint is that the first comma within the name must delineate the surname.
The decision to allow customisation of the honorifics list was to provide usefulness in a setting where the `gender` package is not installed. The code could also easily be adapted for use in other settings e.g. identifying car manufacturer from car name data.

The main R language skills acquired in this programming task that can be transferred to other R programming projects are as follows:
Validating input: Input validation is a necessary aspect of any program that accepts user input. Ensuring that a program is processing data within expected parameters helps to prevent errors and inaccuracies.
String parsing and manipulation: Strings are ubiquitous throughout programming and most programs use some degree of string manipulation. Regular expressions are a powerful feature that can be applied to many types of string manipulation tasks.
Use of functions: Breaking a large program into smaller logical chunks improves readability, test-ability, code reuse and maintainability, among other advantages. The use of *sapply* enables functions to be applied over vectors. Being that vectors are fundamental constructs within the R programming model, this technique has widespread applications within R programming.

Instructions

The following instructions assume familiarity with basic R development concepts. All referenced files are assumed to be in the R working directory.

Setup

Source `Gender.R`
Install the `gender` package if not already installed.

Usage

If processing of a single name is required, a string literal can be used:

```
# Single name
getGender("Jones, Dr Indiana")
[1] "female"
```

Processing a vector with default overridden:

```
> # Vector of names.
names <- c("","Smith, Mr John","Lancelot, Sir G",
           "random junk","Of-Wonderland, Miss Alice")
getGender(names, default = "unmatched")
[1] "unmatched" "male"      "male"      "unmatched" "female"
```

Processing a dataset:

Name	Age
Bloggs, Dr Fred	23
Poppins, Miss Mary	16
Francis, Col. J	72

To process the dataset above dataset `test.csv` using program defaults, use the following commands (ds is arbitrary):

```
ds <- read.csv("test.csv")
getGender(ds$Name)
[1] "male"   "female" "u"
```

In the above example the third row cannot be matched as the *Col.*. honorific is not included in the default honorifics. To view the default honorifics issue the following command:

```
# Default honorifics.
default.honorifics
      [,1]     [,2]
 [1,] "Mr"     "male"
 [2,] "Master" "male"
 [3,] "Miss"   "female"
 [4,] "Ms"     "female"
 [5,] "Mrs"    "female"
 [6,] "Sig"    "male"
 [7,] "Mme"    "female"
 [8,] "Rev"    "male"
 [9,] "Mlle"   "female"
[10,] "Dona"   "female"
[11,] "Sir"    "male"
[12,] "Fr"     "male"
```

```
[13,] "Don"       "male"
[14,] "Countess"  "female"
[15,] "Lady"      "female"
```

To add new honorifics, take the following steps:

```
# Write default honorifics to a file
write.table(default.honorifics, "hons.csv",
            sep = ",", row.names = FALSE,   col.names=FALSE)
```

Open `hons.csv` and add the *Col.* honorific:

....	....
Countess	female
Lady	female
Col	**male**

Save `hons.csv`.
Note: It is not recommended to add the period (.) for abbreviated honorifics as this can affect the matching.

To read the honorifics file and reprocess the dataset, issue the following commands:

```
# Read honorifics file.
hons <- scan("hons.csv", what = character(), sep = ",")

# Convert file into suitable matrix.
hons <- matrix(hons, ncol=2,
               nrow=length(hons)/2, byrow = TRUE)

# Override default honorifics parameter.
getGender(ds$Name, honorifics = hons)
[1] "male"   "female" "male"
```

The *Col.* honorific now returns *male* when matched.

Processing alternative name format:

```
# Alertnative format.
getGender("Smith, Oncology, Dr Jane", firstname.pos = 3)
[1] "female"
```

To create and populate a new column on a dataset, issue the following commands:

```
# Read dataset.
ds <- read.csv("test.csv")

# Create and popluate Gender columns.
ds$Gender <- getGender(ds$Name, honorifics = hons)

#Check result.
ds
                  Name Age Gender
1     Bloggs, Dr Fred  23   male
2 Poppins, Miss Mary   16 female
3     Francis, Col. J  72   male
```

Code Listing

`Gender.R`

```
getGender <- function(name, honorifics = default.honorifics,
                      firstname.pos = 2, default = "u")
{
  # Matches honorific (mr, mrs, miss...) within name to corresponding gender.
  #   Where gender cannot be matched, uses first name matching in gender package.
  #
  # Args:
  #   name: Character or factor variable of name being checked for gender.
  #   honorifics: Character matrix containing honorific and corresponding gender code.
  #     Matrix format: E.g.
  #     Mr      male
  #     Master  male
  #     Mrs     female
  #     Miss    female
  #     etc.
  #     Default: see basic.honorifics function.
  #   firstname.pos: Position after comma, of block of alpha characters defining
  #     first name within name.
  #   default: If name cannot be matched, default is returned.
  #
  # Returns:
  #   Gender identified from name.

  # Check if gender package installed and load library otherwise warn.
  if("gender" %in% rownames(installed.packages()) == FALSE) {
    msg <- "Warning: getGender function uses 'gender' package."
    msg <- paste(msg, "Please install for improved matching")
    warning(msg)
    use.firstname <- FALSE
  } else {
    library(gender)
    use.firstname <- TRUE
  }

  # Convert input to correct data type.
  honorifics <- as.matrix(honorifics)

  # Section below validates input and halts exection if arguments not of correct type.
  if (!is.character(name) && !is.factor(name)) {
    stop("Error in getGender: 'name' argument must be character or factor.")
  }
  if (!is.character(honorifics) || !is.matrix(honorifics) ||
      !ncol(honorifics) == 2 || !nrow(honorifics) > 0) {
    msg <- "Error in getGender: 'honorifics' argument must coerce to character"
    msg <- paste(msg, "matrix nrow > 0; ncol == 2.")
    stop(msg)
  }
  if (!is.numeric(firstname.pos) || !firstname.pos == floor(firstname.pos)) {
    stop("Error in getGender: 'firstname.pos' argument must be integer.")
  }
  if (!is.character(default)) {
```

```
    stop("Error in getGender: 'default' argument must be character.")
  }
  # Raise an error if honorifics are duplicated:
  #   duplication could create ambiguity if one honorific
  #   mapped to more than one gender.
  a <- honorifics[,1]
  if (!length(a[duplicated(a)])==0) {
    stop("Error in getGender: honorifics must be unique.")
  }
  # Raise an error if honorifics contain empty strings.
  a <- trimws(honorifics)
  if (!length(a[a==""])==0) {
    stop("Error in getGender: honorifics must not contain empty strings.")
  }
  # End of validation section.

  # Convert matrix of honorifics and genders into
  #   matrix of genders and regular expressions.
  regex = transformTable(honorifics)

  # Apply matchName function to name argument and return result.
  return (
    as.character(
      sapply(name, function(x) matchName(x, regex, firstname.pos,
                                         use.firstname, default)
      )
    )
  )
}


#  Default list of honorifics.
m <- c("Mr",        "male",
       "Master",    "male",
       "Miss",      "female",
       "Ms",        "female",
       "Mrs",       "female",
       "Sig",       "male",
       "Mme",       "female",
       "Rev",       "male",
       "Mlle",      "female",
       "Dona",      "female",
       "Sir",       "male",
       "Fr",        "male",
       "Don",       "male",
       "Countess","female",
       "Lady",      "female")
default.honorifics <- matrix(m,length(m)/2,2,byrow = T)


matchName <- function(name, regex.matrix, firstname.pos, use.firstname, default)
{
  # Matches name to gender using regular expressions.
  #
  # Args:
```

```
  #   name: Character/factor variable of name being checked for gender.
  #   regex.matrix: Two column matrix:
  #     E.g.
  #     male     (^|[^A-Za-z])(Mr|Master)($|[^A-Za-z])
  #     female   (^|[^A-Za-z])(Mrs|Miss)($|[^A-Za-z])
  #   firstname.pos: See getGender.
  #   use.firstname: If false, do not attempt firstname processing.
  #   default: See getGender.
  #
  # Returns:
  #   Where name can be matched to regex.matrix[,2]
  #     returns corresponding entry regex.matrix[,1]
  #     else returns default.

  # Set return value to default argument.
  ret <- default
  # Get position of comma in name.
  pos <- regexpr(',', name)
  # Set local variable to characters to right of comma.
  tmpname <- trimws(substr(name, pos+1, nchar(as.character(name))))
  # Loop through all values in honorifics matrix.
  for (i in 1:nrow(regex.matrix)) {
    # Check whether remainder of name contains honorific.
    if (grepl(regex.matrix[i, 2], tmpname, ignore.case = TRUE)) {
      # If honorific found, set return value to corresponding gender.
      ret <- regex.matrix[i, 1]
      # Don't bother with rest of loop if we've already found match.
      break
    }
  }
  # If we've not matched on honorific and gender package installed,
  #   attemp to match on first name.
  if (ret == default && use.firstname) {
    ret <- matchFirstname(tmpname, firstname.pos, default)
  }
  return (as.character(ret))
}

matchFirstname <- function(name, firstname.pos, default)
{
  # Calls gender method of gender package to return gender from first name.
  #
  # Args:
  #   firstname.pos: See getGender.
  #   default: See getGender.

  # Extract first name from name by splitting name
  #   into chunks of alpha characters and taking that
  #   which corresponds to defined first name position.
  fname <- strsplit(gsub("[^A-Za-z -]", "", name), " +")[[1]][firstname.pos]
  # Call the gender method.
  ret <- as.character(gender(fname))[4]
  # If gender match return default,
  #   otherwise return gender.
  if (!as.character(ret) == "logical(0)") {
```

```
    return (ret)
  }
  return (default)
}

transformTable <- function(honorifics)
{
  # Transforms two column matrix of honorifics and gender
  #   into two column matrix of gender and regular expression of honorifics:
  #
  # Args:
  #   honorifics:
  #     E.g.
  #     Mr       male
  #     Master   male
  #     Mrs      female
  #     Miss     female
  #
  # Returns:
  #   E.g.
  #   male     (^|[^A-Za-z])(Mr|Master)($|[^A-Za-z])
  #   female   (^|[^A-Za-z])(Mrs|Miss)($|[^A-Za-z])

  # Set local variable.
  h <- honorifics
  # Get vector of genders.
  u <- as.character(unique(h[, 2]))
  # Create 2 column matrix with a row for each gender.
  ret <- matrix(u, nrow=length(u), ncol=2)
  # Create list of honorifics for each gender.
  for(i in 1:nrow(ret)){
    x <- h[h[, 2] == ret[i,1], 1]
    # Transform list into regular expression.
    x <- mkExp(x)
    ret[i,2] <- x
  }
  return (ret)
}

mkExp <- function(x, pref = "(^|[^A-Za-z])", suff = "($|[^A-Za-z])")
{
  # Converts character vector to regular expression string.
  #
  # Args:
  #   x: Character vector.
  #   pref: Prefix for regex.
  #   suff: Suffix for regex.
  #
  # Returns:
  #   E.g.
  #   (^|[^A-Za-z])(Mr|Master)($|[^A-Za-z])

  # Create '|' delimited string from character vector
  #   and bookend with prefix and suffix.
  ret <- paste0(pref,
```

```
                "(",
                paste(x, collapse = "|"),
                ")",
                suff)
  return (ret)
}
```