# Statistics for linguists: practical session in R

*Adnane Ez-zizi*

*30 November 2017*

## Introduction

In this R tutorial, you will apply some of the concepts you covered in the "basic statistics for language" session to a linguistic data set. You will learn to use R to: - Manipulate data (e.g., extract the values of a variable that satisfy a certain condition). - Perform exploratory data analyses by computing summary statistics and making plots such as bar plots and histograms. - Perform logistic regression to explore the relationship between your variable of interest (response variable) and your explanatory variables (predictors).

You will work on the same data set that was used by Bresnan et al. (2007) (the paper is included in the tutorial package) to study dative alternation in English (or almost the same to be precise). You will try to replicate some of their results (e.g., Figure 1-2 and Table 3 in the paper), and hopefully you will be in a position to implement the same types of analyses in your own research.

Dative alternation describes the alternation between a prepositional construction (e.g., Wang gave the book to Liu) and a double-object construction (e.g., Wang gave Liu the book). The prepositional structure is often shortened as NP PP, while the double-object structure is shortened as NP NP. In our example, Liu is called **the recipient** (or receiver), while the book is called **the theme** (or entity).

The goal of Bresnan et al. (2007)'s study was to find to what extent the dative alternation structure (whether NP NP or NP PP) could be predicted using linguistic variables such as the animacy of the recipient and the accessibility of the theme (i.e., whether or not the theme is clearly mentioned in the structure). Bresnan et al. (2007) built various models based on logistic regression and were able at the end to reach a prediction accuracy of 95%. Here we will concentrate on one of the simplest models that they used, which can, nevertheless, predict the correct dative alternation structure in 92% of the cases!

## Preliminary steps

Before starting to implement your program in R, you will need to install/load the R packages that are necessary to run your program.

```
### Install the necessary libraries
install.packages("languageR")

### Load the necessary libraries
library(languageR) # contains the dative dataset
```

## Loading the dataset into R

The next step is to get the dative data set into R. The data is contained in the package "languageR", which you've already called. So you can simply use the function *data()* to load the dataset. Note that this is a simplified version of the complete dative data set. I use it here to make these initial steps easier, but you will have the chance to play with the full data later in this tutorial.

```
### Load the dataset from the "languageR" package
data(verbs) # load the dataset
head(verbs, 10) # display the first ten rows of the dataset
```

The function *head(data, n)* prints the first n rows of data. To get information about the data set, type "?verbs". If the data set was saved in another format, say in a text format, you would have needed to use the *read.table()* function to upload it. Similarly, if the data was stored as a csv (comma separated value) file, you would have had to use *read.csv()*. To illustrate this, I will store the data set in both these formats and re-load it into R:

```
### Load the dataset from a text file
# First let's save the data frame verbs as a text file
write.table(verbs, "dative.txt")
rm(verbs) # remove the dataset from the workspace (check that is removed)
verbs = read.table("dative.txt", header = TRUE) # Import the dataset

### Load the dataset from a csv file (comma-separated value data)
# First let's save the data frame verbs as a text file
write.csv(verbs, "dative.csv", row.names = FALSE)
rm(verbs) # remove the dataset from the workspace (check that is removed)
verbs = read.csv("dative.csv", header = TRUE) # Import the dataset
```

## Data description

Most of the time, you will not have the luxury to have a document that describes your data in detail, so it is useful to use the following code lines. First, you can get the dimension of your dataset (i.e., number of observations and variables):

```
# Get the dimension of the dataset
dim(verbs)
```

The names of the variables can be obtained by typing:

```
# Get the variable names
names(verbs)
```

You can also get a summary of each of the variables in your data set by typing:

```
# Display the summary statistics
summary(verbs)
```

For qualitative vartiables like "verb" and "animacyOfRec", R displays the number of occurrences of each of their categories (or levels). Thus, for example, there are 822 clauses with animate recipients and 81 with inanimate recipients. For quantitative predictors like "LengthOfTheme"", you will get six statistics that describe the central tendency and range of your data. More details are given below in the section"exploratory data analysis".

## Data manipulation

You will usually want to examine some parts of a data set. For example, typing:

```
# extract the 10th observation
verbs[10, 2]
```

will select the element that corresponds to the 10th row and 2nd column—that is, the verb of the 10th construction (here "give"). The first number after the open-bracket symbol "[" always refers to the row, and the second number always refers to the column. The same outcome can be achieved by typing the name of the variable—here "verb"—directly.

```
# extract the 10th observation
verbs[10, "Verb"]
```

Similarly, you can display the logarithm of the theme length for the 10th construction by entering:

```
# extract the logarithm of the theme length for the 10th observation
verbs[10, "LengthOfTheme"]
```

To get the length of the theme, we need to exponentiate:

```
# To get the length
exp(verbs[10, "LengthOfTheme"])
```

To display the values of all variables for the 10th clause:

```
# extract the 10th observation
verbs[10, ]
```

You can also select multiple rows and columns at a time, by providing vectors as the indices.

```
# Define the vector of row numbers to select
row.select = c(10, 97, 433, 901) # vector (10, 97, 433, 901)

# Define the vector of column numbers to select
col.select = 1:3 # vector (1,2,3)
```

Here, the selected rows are 10, 97, 433 and 901. The selected columns are 1, 2 and 3, corresponding respectively to the variables "RealizationOfRec", "Verb", "AnimacyOfRec". These columns can also be selected using:

```
# Define the vector of column numbers to select
col.select = c("RealizationOfRec", "Verb", "AnimacyOfRec")
```

Let's now display the data for these selected rows and columns

```
# data for the first the selected rows and columns
verbs[row.select, col.select]
```

It is also possible to apply some mathematical operations to the selected data. For example, we can compute the sum and average of log-lengths of theme by typing successively:

```
# sum the lengths of theme for the selected rows
sum(verbs[row.select, "LengthOfTheme"])
```

```
# average length of theme for the selected rows
mean(verbs[row.select, "LengthOfTheme"])
```

To extract observations that verify a certain condition, say those in which AnimacyOfTheme has the value "animate":

```
# Extract constructions in which AnimacyOfTheme has the value "animate"
verbs[verbs$AnimacyOfTheme == "animate", ]
```

Alternatively:

```
# Alternatively:
subset(verbs, AnimacyOfTheme == "animate")
```

You can also use more complex conditions:

```
# Extract constructions with animate themes and with themes longer than 2 words
verbs[(verbs$AnimacyOfTheme=="animate") &
      (exp(verbs$LengthOfTheme)>2), ]
```

As a reminder, we used *exp(verbs$LengthOfTheme)* because the length of theme is log transformed in the data, so to get the original length, we need to apply the exponential.

After these basic operations on the data, let's now see how data can be further described.

## Exploratory data analysis

We will start by explaining in more details the *summary()* function that you've encountered above. Remember this function produce different outputs for qualitative variables in comparison with quantitative ones:

```
# Display the summary statistics
summary(verbs)
```

There is nothing to add for qualitative variables. You will basically get the number of occurrences for each category. Let's concentrate on the output of the quantitative predictor "LengthOfTheme":

```
# Summary of the quantitative variable "LengthOfTheme"
summary(verbs$LengthOfTheme)
```

### Measures of central tendency

The mean is simply the average value of the variable. It can be obtained by typing:

```
# average log-length
mean(verbs$LengthOfTheme)
```

The median is the value of the variable below which you find 50% of the variable values. Although the mean is used more in practice, it is always a good idea to compute both. A big difference between the two signals the impact of outliers; the median being less affected by them. To compute the median, type:

```
# median log-length
median(verbs$LengthOfTheme)
```

### Measures of dispersion

The first and last value in the summary table correspond respectively to the minimum and maximum log-length of the theme. These can be separately obtained as follows:

```
min(verbs$LengthOfTheme) # minimum length
max(verbs$LengthOfTheme) # maximum length
```

The remaining two values correspond to the first (the length value below which we find 25% of the lengths) and third () quartiles. These give the length values below which we find 25% and 75% of the lengths respectively.

```
quantile(verbs$LengthOfTheme, 0.25) # first quartile
quantile(verbs$LengthOfTheme, 0.75) # third quartile
```

The min/max and first-quartile/third-quartile values are used to construct two measures of dispersion, namely, the range and interquartile range, as follows:

```
# Range
Range_LT = max(verbs$LengthOfTheme) - min(verbs$LengthOfTheme)
Range_LT
```

```
# Interquartile-range
IQR_LT = quantile(verbs$LengthOfTheme, 0.75) - quantile(verbs$LengthOfTheme, 0.25)
```

```
names(IQR_LT) <- NULL
IQR_LT
```

The two most used measures of dispersion are the standard deviation and variance:

```
# variance
var(verbs$LengthOfTheme)
```

```
# standard deviation
sd(verbs$LengthOfTheme)
```

The bigger the values of the standard deviation and variance, the more spread out the data points are from their average value.

### Contingency tables

In this part, you will learn to produce contingency tables for qualitative variables. These display the frequency distribution of one or several variables. Let's start with one variable. The frequency table of the outcome is:

```
# Frequency table for the dependent variable
table(verbs$RealizationOfRec)
```

We've already encountered this output when we studied the *summary()* function. What about the frequency table between the outcome and the predictor "AnimacyOfRec"? You need to simply add the predictor using the same function:

```
# contingency table with two factors
table(verbs$RealizationOfRec, verbs$AnimacyOfRec)
```

This table shows, for example, that 34 of the "NP" constructions are inanimate. The same result can be obtained using *xtab()*:

```
# same
xtabs(~ RealizationOfRec + AnimacyOfRec, data = verbs)
```

Feel free to familiarise yourself more with this function by typing *?xtabs*. You can also use *xtabs()* for more than two variables. For example, you can construct a contingency table to study the association between the outcome variable "RealizationOfRec" and the predictors "AnimacyOfRec" and "AnimacyOfTheme":

```
# contingency table with 3 factors
xtabs(~ AnimacyOfRec + AnimacyOfTheme + RealizationOfRec, data = verbs)
```

As can be seen from this table, animate themes are very rare. It therefore makes sense to concentrate on clauses with inanimate themes. This can be done by conditioning on the rows of the data set:

```
# select only inanimate themes
sub.inanimate = (verbs$AnimacyOfTheme != "animate")
# re-construct the contingency table
verbs.xtabs = xtabs(~ AnimacyOfRec + RealizationOfRec,
                    data = verbs,
                    subset = sub.inanimate)
verbs.xtabs # display the contingency table
```

The table shows that recipients are more likely to be realised as an "NP" when animate, and as a "PP" when inanimate.

You can transform the contingency table into a table of proportions instead of simple counts by dividing each element in the table by the sum of all counts:

```
# Setting the number of decimal places to 2 when displaying values
options(digits = 2)
# To display the results in term of proportions
verbs.xtabs / sum(verbs.xtabs)
```

Check that the sum of all the cell values is 1. To display the values in terms of percentages, we simply multiply by 100:

```
# To display the results in term of percentages
(verbs.xtabs/sum(verbs.xtabs)) * 100
```

To display the results in term of relative frequencies with respect to the row totals:

```
# To display the results in term of relative frequencies with respect to the row totals
verbs.xtabs/rowSums(verbs.xtabs)
```

As you will notice, the values in each row sum to one. For example, the value 0.63 refers to the (conditional) probability of observing an "NP" realisation given that the recipient is animate. similarly, 0.37 is the (conditional) probability of observing an "NP" realisation given that the recipient is animate. That's why the sum of the two is 1. The same table can be obtained directly using the function *prop.table()*:

```
# same
prop.table(verbs.xtabs, 1)
```

which also allows to get the relative frequency with respect to the column total (conditional probability given the realisation of the recipient):

```
# To produce the relative frequencies with respect to the column totals
prop.table(verbs.xtabs, 2) # Here the columns sum to one
```

Here the table shows that, for example, given that the recipient is realised as a "PP", there is a 86% chance that the recipient is animate.

## Graphical data exploration

In this section and the remaining ones, we will use the full dative data set. You will learn to make plots to graphically represent or supplement the numerical statistics that we discussed in the last section. Perhaps more interesting to you here, you will learn to make the first two figures that were used in Bresnan et al. (2007)'s paper.

### Visualisation of one categorical variable

Here you will try to graphically represent the proportion table of "recipient realization" using a bar plot. The table looks like this:

```
# Proportion table for the dependent variable
outcome.tab = table(dative$RealizationOfRecipient) # frequenct table
outcome.percentage = prop.table(outcome.tab) * 100 # Trnasform it into a proportion table
outcome.percentage = round(outcome.percentage , 1) # To display only up to one decimal
outcome.percentage
```

To represent these two quantities as vertical bars, we use the bar plot:

```
# Bar plot (displaying the same data in the proportion table)
barplot(outcome.percentage, main = "Bar plot of recipient realization",
        xlab = "dative category", ylab = "Percentage (%)", col = "grey50",
```

```
        ylim = c(0,100), cex.names = 1.2, space=0.25)
grid()
```

### Visualisation of two categorical variables

Let's visualize the contingency table for the counts of clauses cross-classified by the animacy of the recipient and the realization of the recipient (NP versus PP), using a bar plot. The table is given by:

```
# Construct the contingency table
dative.xtabs = xtabs(~ AnimacyOfRec + RealizationOfRecipient,
                     data = dative,
                     subset = (AnimacyOfTheme != "animate"))
dative.xtabs # display the table
```

First, we will represent the contingency table using a stacked bar plot.

```
# Stacked bar plot
barplot(dative.xtabs, main = "Stacked bar plot of clause counts", ylab = "count",
        ylim = c(0,2500), legend.text = c("animate", "inanimate"), space=0.5)
grid()
```

It is a good time to use the help function. Type *?barplot* to get more information about the options that I used inside the function. For example, you should realise that "main =" was used to add the title. Another way to analyse such a function is to remove one option at a time and see how this affects the output. Could you find how to display the bars side by side without looking at the next code chunk? (You can cheat a little bit and use google if you wish.)

Here is how you do it:

```
# Side-by-side bar plot
barplot(dative.xtabs, main = "Side by side bar plot of clause counts",
        ylab = "count", ylim = c(0,2500), legend.text = c("animate", "inanimate"),
        beside = T)
grid()
```

The following R code puts the two bar plots in the same window:

```
par(mfrow = c(1,2), mar=c(5,5,5,4)) # Divide the window into 2 parts and change the margins
# Stacked bar plot
barplot(dative.xtabs, ylab = "count", ylim = c(0,2500),
        legend.text = c("animate", "inanimate"),
        args.legend = list(x = "topright", bty="n", inset=c(-0.3,0)))
grid()
# Side-by-side bar plot
barplot(dative.xtabs, ylab = "count", ylim = c(0,2500),
        legend.text = rownames((dative.xtabs)), beside = T,
        args.legend = list(x = "topright", bty="n", inset=c(-0.3,0)))
grid()
par(mfrow = c(1,1), mar=c(5.1, 4.1, 4.1, 2.1)) # Re-set the parameters to their default values
```

Let's now try to produce bar plots that show the distribution of discourse accessibility (for both the receiver/recipient and entity/theme) in "NP" and "PP" constructions, like those displayed in Figure 1 (page 75) of Bresnan et al. (2007)'s paper.

You will first need to construct the contingency table for the counts of realization of the recipient cross-classified by the accessibility of receiver (i.e., recipient) and entity (i.e., theme).

```
# Contingency table for the counts of realization of the recipient by accessibility
dative.xtabs = xtabs(~ AccessOfRec + AccessOfTheme+ RealizationOfRecipient,
                     data = dative)
dative.xtabs
```

From this table, you can create a contingency table for "NP" realizations, which gives the counts of each accessibility type for both the receiver and entity.

```
# Create a contingency table for NP NP
xtabs_receiverNP =  rowSums(dative.xtabs[, , 1])
xtabs_themeNP =  colSums(dative.xtabs[, , 1])
xtabs_NP = cbind(xtabs_receiverNP, xtabs_themeNP)
# Change the rows order to get the same plot as in Fig 1
xtabs_NP = xtabs_NP[c("given", "accessible", "new"), ]
# Change the column names
colnames(xtabs_NP) = c("Receiver", "Entity")
xtabs_NP # display the contingency table
```

You can do the same for "PP" realizations:

```
# Create a contingency table for NP PP
xtabs_receiverPP =  rowSums(dative.xtabs[, , 2])
xtabs_themePP =  colSums(dative.xtabs[, , 2])
xtabs_PP = cbind(xtabs_receiverPP, xtabs_themePP)
# Change the rows order to get to get the same plot as in Fig 1
xtabs_PP = xtabs_PP[c("given", "accessible", "new"), ]
# Change the column names
colnames(xtabs_PP) = c("Receiver", "Entity")
xtabs_PP # display the contingency table
```

Now you can generate the bar plots that represent the two contingency tables using what we've learned so far:

```
# Make the bar plots
# For NP NP constructions
par(mfrow = c(1,2))
barplot(xtabs_NP, beside = T, ylim = c(0,2500), main = "NP NP", ylab = "count",
        legend.text = c("given", "accessible", "new"),
        args.legend = list(x = "topright", bty="n", inset=c(-0.1,0)))
grid()
# For NP PP constructions
barplot(xtabs_PP, beside = T, ylim = c(0,2500), main = "NP PP", ylab = "count",
        legend.text = c("given", "accessible", "new"),
        args.legend = list(x = "topright", bty="n", inset=c(-0.1,0)))
grid()
```

**Question**: Try to make a figure like Figure 2 (page 76) in Bresnan et al. (2007)'s paper. You will first need to recode the variables "AccessOfRec" and "AccessOfTheme" by combining the categories "accessible" and "new" into a new category called "nongiven". You can use the function *levels()* to change the names of the categories. Once done, construct the new contingency tables and generate the bar plots as we've just done.

### Visualisation of one quantitative variable

Here we will concentrate on the question of how to visualise a distribution of a quantitative variable like "LengthOfRecipient". Let's start with the well-known histogram:

```
# Histogram of recipient length
hist(dative$LengthOfRecipient, main = "Histogram of recipient length",
     xlab = "Length of recipient", ylim = c(0, 3000),
     breaks = max(dative$LengthOfRecipient))
grid()
```

The argument "*breaks*" specifies the number of bins (i.e., bars) to use. Here I chose to have one bin per unit to display the count for each possible recipient length. This allows to see that in most constructions, the recipient is composed of a single word. But depending on the choice of the number of bins, the representation of the distribution would change.

**Questions:** Try different values for "*breaks*" to see how this affect the look of the distribution. Also, compare the histograms of recipient length with that of theme length (plot them in the same window by using par(mfrow = c(1,2))).

The box plot is another way to represent a distribution. It is basically a way to show the summary statistics that we've covered previously such as the median and the first and third quartiles. Let's create a box plot for the log-transformed variable Length of theme.

```
# Box plot of theme length
boxplot(log(dative$LengthOfTheme), main = "Box plot of theme lengths",
        ylab = "log of theme length")
grid()
```

The box plots display information about several statistical measures:

- Median: tick line inside the box (about 1.1)
- First quartile: lower boundary of the box (about 0.7)
- Third quartile: upper boundary of the box (about 1.6)
- Interquartile range (IQR): length of the box (about 0.9)
- Outliers: values that are further away from the boundaries of the box by 1.5 times the IQR.

### Visualisation of two quantitative variables

Box plots make it easier to compare the distribution of multiple quantitative variables. For example, we can compare the distributions of the log-transformed variables "*LengthOfRecipient*" and "*LengthOfTheme*" as follows:

```
# Comparing LengthOfTheme and LengthOfRecipient
boxplot(log(dative$LengthOfTheme), log(dative$LengthOfRecipient),
        main = "Box plot of theme length and recipient length",
        names = c("Theme length","Recipient length"), ylab = "log length")
```

It seems that constructions generally have longer themes than recipients, but is this is difference significant? To answer this question, we need to decide:

- Whether we have two independent or dependent (i.e., repeated measurements) samples. Here both the length of theme and of recipient are extracted from the same construction, so we have a repeated measures design.
- Whether we should apply a parametric statistical test, namely a paired t-test (since we have a repeated measure design) or a non-parametric one like the Wilcoxon-test. The advantage of parametric methods is that they are simple to use but come at the cost of making strong assumptions about the data. For a t-test, perhaps the most important assumption to verify is that of normality. Concretely, the distribution of the differences between the two variables/groups should be approximately normally distributed.

Let's then test the normality hypothesis. This can be done graphically using a QQ-plot.

```
# QQ-plot to test the normality assumption
qqnorm(log(dative$LengthOfTheme)-log(dative$LengthOfRecipient))
qqline(log(dative$LengthOfTheme)-log(dative$LengthOfRecipient))
```

A normally distributed variable would result in points perfectly lying on the solid diagonal line. The plot seems to indicate that there is no reason to reject the normality assumption (you can also run a Shapiro-Wilk test to support the graphical conclusion using the function *shapiro.test*). We will therefore run a paired t-test on our data:

```
# paired-test
t.test(log(dative$LengthOfTheme), log(dative$LengthOfRecipient), paired = TRUE)
```

Since the p-value is very small ($< 2.10^{-16}$), there is strong evidence that themes are longer than recipients. The confidence interval also confirms this conclusion, showing that, with 95% probability, the interval [0.71, 0.77] will contain the true mean of the difference of log-lengths.

**Question:** Use Wilcoxon-test to see if you will still get a significance result for the difference between log lengths of theme and recipient.

Sometimes, you may be interested more in studying the relationship/correlation between two variables instead of comparing them. Quantitatively, this can be achieved by computing their correlation. For example, the correlation between "LengthOfTheme" and "LengthOfRecipient" is given by:

```
# Correlation between LengthOfTheme and LengthOfRecipient
cor(dative$LengthOfTheme, dative$LengthOfRecipient)
```

The correlation is very low (the closer the absolute value of correlation to 1, the highly correlated are the two variables), suggesting that there is no relation between the two predictors. We can also assess this relationship visually using a scatter plot (*plot*):

```
# Scatterplot bewteen two variables
plot(dative$LengthOfRecipient, dative$LengthOfTheme,
     main = "Relationship between length of recipient and of theme",
     xlab = "Recipient length", ylab = "Theme length")
lines(lowess(dative$LengthOfRecipient, dative$LengthOfTheme), col = "red")
```

where the function *lowess()* was used to add the red curve, which highlights the main trend in the data.

## Logistic regression

After exploring the data, the next step in a statistical analysis is often to fit a model to the data to answer more complicated research questions. In our case, we will use a model called logistic regression to answer the following questions:

1. Is there a relationship between the choice of the alternative dative structure and the difference between the theme length and the recipient length?
2. Which linguistic variables contribute to the choice of the dative structure?
3. How accurately can we predict the dative structure of the constructions in our data set?

### Preparing the data

Before applying logistic regression to answer the above questions, we will start by making some modifications to the dative data set to match (as closely as possible) the data that Bresnan et al. (2007) used in their paper.

The first modification consists of discarding written constructions and keeping only spoken ones. Secondly, we will combine the categories "accessible" and "new" in accessibility of theme and recipient into a single

category, called "nongiven". Finally, we will create a new variable "*ldiff*", which is the difference between the log of "*LengthOfTheme*" and "*LengthOfRecipient*".

```
## Important the dataset
data(dative)
### Modification1: Keep only spoken dative constructions as in the paper
dative.sp = dative[(dative$Modality=="spoken"), ]
row.names(dative.sp) = 1:nrow(dative.sp)

### Modification2: Combine the categories "accessible" and "new" into"nongiven"
# For AccessOfRec
levels(dative.sp$AccessOfRec)[(levels(dative.sp$AccessOfRec)=="accessible")] = "nongiven"
levels(dative.sp$AccessOfRec)[(levels(dative.sp$AccessOfRec)=="new")] = "nongiven"
# For AccessOfTheme
levels(dative.sp$AccessOfTheme)[(levels(dative.sp$AccessOfTheme)=="accessible")] = "nongiven"
levels(dative.sp$AccessOfTheme)[(levels(dative.sp$AccessOfTheme)=="new")] = "nongiven"

### Modification3: Create a new variable that encodes the difference between the
#length of theme and that of recipient
dative.sp$LogLengthDiff =  log(dative.sp$LengthOfTheme) - log(dative.sp$LengthOfRecipient)
```

**Simple logistic regression: Is there a relationship between the choice of the alternative dative structure and the difference between the theme length and the recipient length?**

Logistic regression models the relationship between a categorical response variable and some predictors. The response variable can have two or more categories (although here we will concentrate only on binary responses variables). The predictors can be quantitative, categorical or a combination of both. In this part you will deal with the case of one quantitative predictor, and try to answer the first question in our list—that is, whether there is a relationship between the choice of the alternative dative structure and the difference between the theme length and the recipient length.

The logistic regression model has a very similar structure to that of the linear regression model, except that it models the logit (or log-odds) of the outcome instead of modelling the outcome directly. The logit is a value that compares the probability of an outcome of interest with the probability of the opposite outcome, and it has a one-to-one relationship with the probability of observing the outcome of interest. Formally, if you have one predictor $X$ (say the length of theme) and an outcome $Y$ (the relalisation of the recipient), taking on values 0 or 1 (e.g., 0 could represent the category "NP" and 1 the category "PP"), the model assumes that the relationship between $Y$ and $X$ is given by the following formula:

$$logit(Y) = b_0 + b_1 X \tag{1}$$

where the coefficient $b_0$ is called the intercept, and represents the probability of observing the outcome "1", when X is equal to 0. The coeffcient $b_1$ represents the effect of the predictor X. In other words, if the predictor X increases by one unit, the log odds of the outcome "1" increases or decreases by $b_1$, which also means that the probability of observing the outcome "1" is $exp(b_1)$ higher/lower than that of the outcome "0". For more conceptual details about the model, see Levshina (2015), Baayen (2008) or James et al. (2013).

Let's now apply simple logistic regression to model the relationship between the realization of the recipient and the log difference between theme length and recipient length (i.e., LogLengthDiff).

```
# Fit a simple logistic regression model
logReg.fit = glm(RealizationOfRecipient ~ LogLengthDiff, data = dative.sp,
                 family = binomial)
summary(logReg.fit) # Model outputs
```

The *glm()* function returns several outputs. For now, all you need to know is that R has estimated the coefficients of our logistic regression model and that we can use these coefficients to compute the probability of observing a certain outcome given any value of LogLengthDiff. This is what you will cover next.

### Coefficient estimates

To display only the coefficient estimates, you will need to use:

```
# Estimates of the coeffs
coef(logReg.fit)
```

the estimate of the intercept is $-0.55$, which is the log odds of the outcome "1" (i.e., $log[P(outcome = 1)/1 - P(outcome = 1)]$) when the predictor "LogLengthDiff" is equal to 0 (i.e., when the length of the recipient is equal to the length of the theme). Here "outcome = 0" corresponds to PP and "outcome = 1" corresponds to NP. The coding of the response variable is automatically chosen by the mode. You can check the coding used by the model by typing:

```
# Coding for the the response variable "RealizationOfRecipient"
contrasts(dative.sp$RealizationOfRecipient)
```

To be able to interpret the coefficient directly in terms of odds, you need to exponentiate the coefficient:

```
# Simple odds of the outcome RealizationOfRecipient = "PP""
exp(coef(logReg.fit))
```

The intercept coefficient shows that the chances of "PP" are 0.58 times greater than those of "NP". In other words, the chances of "PP" are 1.7 (=1/0.58) times lower than those of "NP". The predictor coefficient indicates that the odds of "PP" (outcome=1) are 5.3 (1/0.19) times lower when the log of Length difference increases by one unit.

### Predicting the probability of choosing the prepositional dative (PP)

The following code produces the estimated probabilities of choosing the prepositional dative (PP) for all constructions in our data set, according to the model.

```
# estimating the probability of selecting the PP dative for all constructions in # our dataset
logReg.probs = predict(logReg.fit, type="response")
logReg.probs[1:10] # probability of selecting the PP dative for the first 10 constructions
```

We can also plot the probability of the "PP" structure as a function of the predictor "LogLengthDiff"

```
# Ploting the probability of response as a function of the predictor
RealOfRec.coded = rep(0, nrow(dative.sp))
RealOfRec.coded[(dative.sp$RealizationOfRecipient=="PP")] = 1
plot(dative.sp$LogLengthDiff, RealOfRec.coded,
     xlab="length difference (log scale)", ylab="Probability of prepositional dative")
xpoints = seq(min(dative.sp$LogLengthDiff), max(dative.sp$LogLengthDiff), length.out = 1000)
ypoints = predict(logReg.fit, data.frame(LogLengthDiff = xpoints), type="resp")
lines(xpoints, ypoints, lwd = 4, lty = 1, type = "l", col="red")
```

The small circles represent the observed outcomes. 0 corresponds to "NP" structures, whereas 1 corresponds to "PP" structures.

### Assessing the goodness of fit of the model

The other outputs that we obtained using the summary function (i.e., summary(logReg.fit)) basically show how well the model fit the data, or what statisticians would call "the goodness of fit" of the model.

1. Significance of the coefficient estimates

The *glm()* function also tests whether each of the coefficients is non null—that is, whether the effect of the corresponding predictor is real or can be ignored. This is diplayed in the part of the output table entitled "Coefficients". To display just the coefficient estimates along with the results of the significance tests, you need to use:

```
# displaying the coefficients along with p-values
summary(logReg.fit)$coef
```

Since both p-values (4th column) are very small, there is strong evidence that the two coefficents are not equal to 0, and we can say that the length difference has an effect on the choice of the dative structure.

2. Deviance: a measure of the unexplained variation

The deviance shows how well the response variable is predicted by a model. The lower the deviance, the better the fit. The summary function produced two types of deviances: the null deviance and the residual deviance. The residual deviance shows how well our model predicts the response variable. You can obtain it separately by typing:

```
# deviance of the simple logistic regression model
logReg.fit$deviance
```

Compare it with the null deviance, which shows how well a model containing only the intercept fits the data.

```
# deviance of the intercept model
logReg.fit$null.deviance
```

A reduction of almost 600 in deviance by adding the predictor "LogLengthDiff", which supports the previous result that this predictor has a significant effect on the response variable.

3. Akaike information criterion (AIC)

AIC is another measure of goodness of fit. As the deviance, this measure is used mainly to compare different models—that is, to be able to say which of several models is the best, with the additional advantage that this measure takes into account the number of coefficients in our model (models with many coefficient might be good at fitting the data in hand but perform poorly on new data). What you need to know is that the lower the AIC value, the better is the model. Our model has an AIC value of 1862, which doesn't say much per se, but we will later use it to compare this model with other models.

```
# Akaike information criterion (AIC):
logReg.fit$aic
```

4. Classification table: comparing the true and predicted outcomes

Remember that we've already predicted the probabilities of observing a "PP" structure for all constructions in our data set. However, what is more interesting to us is to predict the dative outcome for each construction. To do that, we consider that a construction is "PP" if its estimated probability is higher than 0.5, and is "NP" if its estimated probability is lower than 0.5.

```
# Predicting the class labels
logReg.pred = rep("NP", 2360)
logReg.pred[logReg.probs >.5 ] = "PP"
logReg.pred[1:10] # Predicted outcomes for the first 10 constructions
```

The first 10 constructions are predicted by the model to have all a double object dative structure ("NP"). Next, we will construct what we call a classification (or confusion) matrix (similar to Table 1 in Bresnan's paper). It basically compares predicted and true responses.

```
# Confusion matrix to determine how many obs were incorrectly classified
table(dative.sp$RealizationOfRecipient, logReg.pred)
```

The elements on the diagonal represent the outcomes that were correctly predicted, whereas those outside of it are the incorrect outcome predictions. To get the proportion of correct responses, you could do:

```
# Accuracy rate of classification
(1822+182)/2360
```

Or:

```
# Another way to compute the accuracy rate of classification
mean(logReg.pred == dative.sp$RealizationOfRecipient)
```

This seems impressive, but in our data we have 1859 "NP" constructions and only 501 "PP" constructions. So by constructing a simple model that always predicts "NP", we could achieve an accuracy of 0.79.

```
# Correct from always guessing NP NP (=0)
mean((dative.sp$RealizationOfRecipient=="NP"))
```

Let's see if we can improve the accuracy by adding the other linguistic predictors.

### Multiple logistic regression: Which linguistic variables contribute to the choice of the dative structure?

Now that you have learned about simple logistic regression, we will try to model the dative choice using all the linguistic variables in our dative data set. When we include more than one predictor in a logistic regression model, the resulting model is called multiple logistic regression. The structure of this model is almost the same as the simple version, except that we also add new terms for the additional predictors, that is, if we have $X_1$, $X_2$, ..., $X_p$ predictors, the model is written as:

$$logit(Y) = b_0 + b_1X_1 + b_2X_2 + ... + b_pX_p \tag{2}$$

where $b_0$ still represents the intercept, and $b_1$, $b_2$,..., $b_p$ are the coefficients of the predictors $X_1$, $X_2$, ..., $X_p$.

Running a multiple logistic regression on our data set with the same linguistic predictors that Bresnan et al. (2007) used gives the following results:

```
# Fit
MultiLogReg.fit = glm(RealizationOfRecipient ~ ., data = subset(dative.sp,
                    select=c(-Speaker, -Modality, -Verb, -LengthOfTheme,
                             -LengthOfRecipient)), family = binomial)
summary(MultiLogReg.fit) # Model outputs
```

Let's start with the deviance and AIC measures. Adding the other linguistic variables decreased both quantities substantially. The deviance went down from 1858 to 1050, while the AIC decreased from 1862 to 1078.

Concerning the significance tests, all predictors have a significant effect except for the difference between the semantic class f and a. The classification table is given by:

```
# Accuracy
# Predicting the probability of choosing the prepositional dative (PP)
MultiLogReg.probs = predict(MultiLogReg.fit, type="response")
# Predicting the class labels
MultiLogReg.pred = rep("NP", 2360)
MultiLogReg.pred[MultiLogReg.probs >.5 ] = "PP"
# Accuracy table
table(dative.sp$RealizationOfRecipient, MultiLogReg.pred)
```

Note that the resulting classification table is slightly different from the accuracy table of Bresnan et al. (2007) (Table 1 - page 79). This is because they included three other predictors (number of recipient, number of theme). However, our model achieves the same accuracy level as theirs (0.92).

```
# Accuracy rate of classification
mean(MultiLogReg.pred == dative.sp$RealizationOfRecipient)
```

**Question:** Write down the formula of the model as was done in Figure 4 in Bresnan et al. (2007)'s paper.

## References


Baayen, R. H. 2008. Analyzing linguistic data: A practical introduction to statistics using r. Cambridge University Press.

Bresnan, J., A. Cueni, T. Nikitina, R. H. Baayen, and others. 2007. Predicting the dative alternation. Cognitive foundations of interpretation. 69–94.

James, G., D. Witten, T. Hastie, and R. Tibshirani. 2013. An introduction to statistical learning. Springer.

Levshina, N. 2015. How to do linguistics with r: Data exploration and statistical analysis. John Benjamins Publishing Company.