Chapter 5
Selected multifactorial and multivariate methods

All models are wrong, but some are useful.
George E.P. Box

So far we have only been concerned with monofactorial methods, i.e., methods in which we investigated how maximally one independent variable is correlated with the behavior of one dependent variable. In many cases, proceeding like this is the beginning of the empirical quantitative study of a phenomenon. Nevertheless, such a view on phenomena is usually a simplification: we live in a multifactorial world in which probably no phenomenon is really monofactorial – probably just about everything is correlated with several things at the same time. This is especially true for language, one of the most complex phenomena resulting from human evolution. In this section, we will therefore discuss several multifactorial techniques, which can handle this kind of complexity better than the monofactorial methods discussed so far. You should know, however, each section's method below could easily fill courses for several quarters or semesters, which is why I can unfortunately not discuss every aspect or technicality of the methods and why I will have to give you a lot of references and recommendations for further study. Also, given the complexity of these methods, there will be no discussion of how to compute them manually.

Before we can begin to discuss multifactorial methods, however, there is a lot to discuss. On a very abstract level, this discussion involves the notions of *interaction* and *model (selection)* and will be the subject of Section 5.1. However, as you will see soon, these notions will quickly lead to a variety of interrelated concepts and, ultimately, important analytical strategies for the subsequent, more hands-on sections.

1. The notions of interaction and model (selection)

1.1. Interactions

As was mentioned at the beginning of the previous chapter, multifactorial methods involve a dependent variable and two or more independent varia-

bles, not just one as in all of Chapter 4. This presence of more than one independent variable brings about potentially interesting findings, but also raises the question of how the two or more independent variables jointly relate to the dependent variable.

There are basically two different ways in which several independent and dependent variables may be related, which we will explore on the basis of the example involving constituent lengths from Chapter 1. Let us again assume you wished to study whether the lengths of constituents – captured in the dependent variable LENGTH – are correlated with two independent variables, the variable GRMRELATION (with the two levels *SUBJECT* and *OBJECT*) and the variable CLAUSETYPE (with the two levels *MAIN* and *SUBORDINATE*). Let us further assume you did a small a pilot study in which you investigated 120 constituents that are distributed as shown in Table 39.

Table 39. A fictitious data set of subjects and objects

	GRMRELATION: *SUBJ*	GRMRELATION: *OBJ*	Totals
CLAUSETYPE: *MAIN*	30	30	60
CLAUSETYPE: *SUBORD*	30	30	60
Totals	60	60	120

Let us finally assume you determined the syllabic lengths of all 120 constituents to compute the means for the variable level combinations – subjects in main clauses, subjects in subordinate clauses, objects in main clauses, objects in subordinate clauses – and obtained the following results:

- the average length of all subjects (i.e., across main and subordinate clauses) is less than that of all direct objects;
- the average length of all constituents (i.e., across subjects and objects) in main clauses is less than that of constituents in subordinate clauses.

The interesting thing is that these monofactorial results – recall from Section 3.2.2.2 that these are often referred to as *main effects* – can come in different forms. On the one hand, the effects of the two independent variables can be *additive*. That means the combination of the two variables has the effect you would expect from each main effect. Since subjects are short(er), as are constituents in main clauses, additivity predicts that main clause subjects should be the shortest constituents, and subordinate clause objects should be longest. This result, which is what H_0 would predict, is represented in Figure 57: black and grey dots indicate mean lengths of ob-

jects and subjects respectively in the two grammatical relations, but also averaged across both, and the "m" and the "s" represent the means of main and subordinate clause constituents across the two grammatical relations.

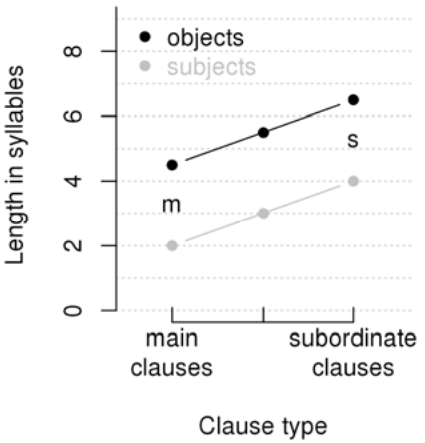

Figure 57. Interaction plot for LENGTH ~ GRMRELATION * CLAUSETYPE 1

This result is in fact perfectly additive because the two lines are perfectly parallel. That means, if I tell you that

- the difference main clause subject length minus main clause object length is -2.5 syllables;
- the difference main clause subject length minus subordinate clause subject length is -2 syllables;
- the average main clause subject length is 2 syllables,

then you can perfectly predict the average subordinate clause object length: 2 + 2.5 + 2 = 6.5.

However, with the exact same kinds of main effects, it is also possible that the two independent variables *interact*. Two or more variables interact if their joint effect on the dependent variable is not predictable from their individual effects on the same dependent variable. One such scenario is represented in Figure 58. Consider first the left panel. You can see that there are still the same kinds of main effect of GRMRELATION (subjects are again shorter than objects) and CLAUSETYPE (main clause constituents are again shorter than subordinate clause constituents), but now the lines are not parallel anymore but intersect.

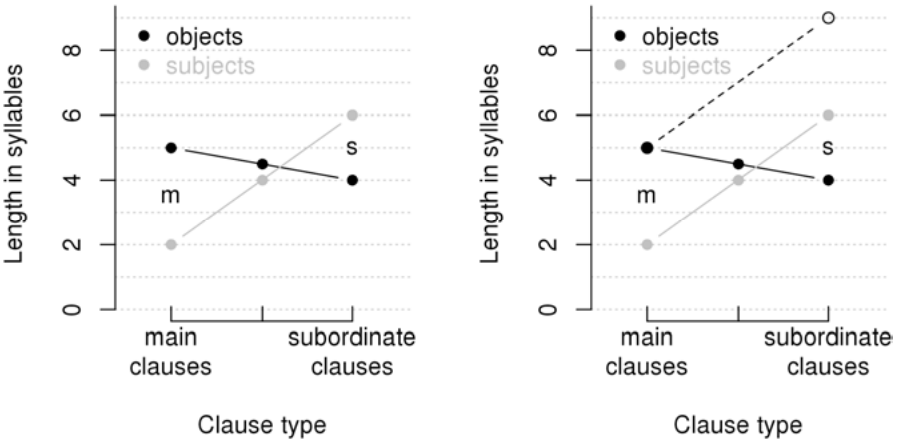

Figure 58. Interaction plot for LENGTH ~ GRMRELATION * CLAUSETYPE 2

What does that mean? It means, if I tell you that

- the difference main clause subject length minus main clause object length is -3 syllables;
- the difference main clause subject length minus subordinate clause subject length is -4 syllables;
- the average main clause subject length is 2 syllables,

then you can absolutely *not* predict the average subordinate clause object length: you would predict 2 + 3 + 4 = 9 syllables (as indicated in the right panel with the dashed line ending in a circle, which is parallel to the grey one), whereas the real average subordinate clause object length in the data is 4 syllables. *That* is an interaction: you cannot predict the average subordinate clause object length using the two main effects but need an additional interaction term that 'corrects down' the prediction from your predicted 9 to the real 4; a test of that interaction term would test whether that term is significantly different from zero or not.

Yet another kind of interaction is shown in Figure 59. Again, we have the by now familiar main effects but even though the lines do not intersect, this is still an interaction for the same reason as above. If I tell you that

- the difference main clause subject length minus main clause object length is -2 syllables;
- the difference main clause subject length minus subordinate clause sub-

ject length is -2 syllables;
– the average main clause subject length is 2 syllables,

then you can again *not* predict the average subordinate clause object length: you would predict 2 + 2 + 2 = 6 syllables (as again indicated in the right panel with the dashed line, which is parallel to the grey one), whereas the real average subordinate clause object length in the data is 8 syllables.

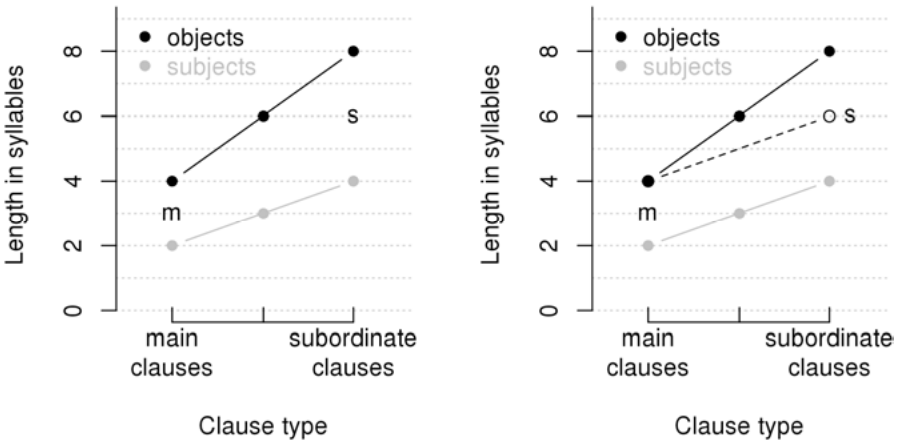

Figure 59. Interaction plot for LENGTH ~ GRMRELATION * CLAUSETYPE 3

Again an interaction: you cannot predict the average subordinate clause object length using the two main effects but need an additional interaction term that corrects up the prediction from your predicted 6 to the real 8, which again may be a significant interaction effect.

Before we move on, let me very briefly give a second example of an interaction, one that you are actually already familiar with, even if you may not have thought about it like this. The above example involved means, this one involves frequencies. Imagine you do a corpus study of 60 *of*-. vs. 80 *s*-genitives in which you try to determine whether the genitive choice is correlated with the animacy of the possessor NP (e.g., *John* in *John's car*). Imagine now you presented your results to a colleague in an overview table, but you leave out the main body of the table, as in Table 40. If you now asked your colleague to complete Table 40 without assuming anything particular going on in the data, that colleague should – maybe implicity – assume H_0 and adopt the logic of the chi-squared test and compute frequencies expected from H_0, as in Table 41.

Table 40. A fictitious data set of genitive choices (totals only)

	Animate possessor	Inanimate possessor	Totals
of-genitive			**60**
s-genitive			**80**
Totals	**70**	**70**	**140**

Table 41. A fictitious data set of genitive choices 1

	Animate possessor	Inanimate possessor	Totals
of-genitive	30	30	**60**
s-genitive	40	40	**80**
Totals	**70**	**70**	**140**

That's because if your colleague is explicitly told to not assume anything special, any deviation from Table 41 is really hard to motivate. Yes, your colleague could create something like Table 42 and say, "there's always a bit of chance variation", but … how could he possibly motivate Table 43 without assuming something special? That "something special" would be an interaction.

Table 42. A fictitious data set of genitive choices 2

	Animate possessor	Inanimate possessor	Totals
of-genitive	33	27	**60**
s-genitive	37	43	**80**
Totals	**70**	**70**	**140**

Table 43. A fictitious data set of genitive choices 3

	Animate possessor	Inanimate possessor	Totals
of-genitive	10	50	**60**
s-genitive	60	20	**80**
Totals	**70**	**70**	**140**

Thus, what in the chi-squared test scenario corresponds to the frequencies expected from H_0 are in fact the frequencies that result from assuming additive behavior of the two variables. And what a chi-squared test does is assess whether the data deviate from the distribution assuming *no* interaction so much that the *p*-value from the chi-squared test becomes < 0.05, which in turn means you will reject H_0 and assume there *is* an interaction.

This was probably a painstakingly detailed characterization but the notion of interaction is a very important one (and often misunderstood and/or

underutilized) so it is absolutely crucial you understand it. This is because the presence of a significant interaction means you cannot take the main effects of the independent variables in the interaction at face value! In Figure 58, while there is a main effect of objects being longer than subjects, the interaction shows that this is really only true in main clauses, but not in subordinate clauses. This property of significant interactions – that they qualify main effects – is one of the most important reasons for why their inclusion in a model is often essential, a topic to which we will turn now.

1.2. Model (selection)

The last section ended with a sentence using the word *model*, a word you also encountered when we discussed linear models and regression. I have used this word without a formal definition so far but you probably still had an intuitive understanding of what I meant. Now, more formally, I want to define a model as a formal characterization of the relationship between *predictors* – independent variables and their interactions – and one or more dependent variables. This 'characterization' typically comes in the form of a (regression) equation of the type you saw in Sections 3.2.3 and 4.4, and also schematically in the captions of Figure 57, Figure 58, and Figure 59, where the purpose of the regression equation is to quantify the relationship between predictors and dependent variable(s) and to generate predictions of the dependent variable(s). The development of an appropriate model, or regression equation, is called *modeling* or *model selection*, and different types of modeling are what's at the heart of most of this chapter.

One word of caution already: this chapter, as short as it is, will hopefully show that with multifactorial data, the cookbook-recipe type of approach used in Chapter 4 will not work: analyzing multifactorial data often requires leaving well-trodden paths and cherished distinctions (e.g., between exploratory and hypothesis-testing approaches). The analysis of a complex data set is much like detective work or peeling an onion, where at every step multiple avenues are possible, and I only wish I could claim I had all the solutions for all the data sets I ever explored … Ok, let's get to it!

1.2.1. Formulating the first model

The first step in model selection would seem to be the formulation of a first model, an equation that tries to model the relationship between *predictors*

and, for now, one dependent variable. However, there are a variety of threats to modeling that need to be taken into consideration. One of these has to do with something as mundane as recognizing the nature of the dependent variable: is it binary? categorical? numeric? numeric but only covering a particular range of discrete values (e.g., 0 and positive integers as with frequencies)? or just positive but with a floor as with reaction times?

We have only talked about modeling in a linear-modeling context (with the function `lm`), which is typically used when the dependent variable is numeric and spans a large range of values. However, since a linear regression will virtually always predict continuous values, it is not really well-suited to be applied to binary dependent variables (although this is still common) or categorical ones. Also, since a linear regression will virtually always predict negative values, it may not be well-suited to predict frequencies. Below, I will discuss different models for different dependent variables; thankfully, much of the logic of linear models, which you already know, can be applied to most of these cases.

A second threat is concerned with whether predictors are used on the most useful information value and scale. As for the former, there is still a lot of work out there in which continuous predictors are factorized. That means, instead of using the continuous predictor as is, researchers break it down into a categorical variable with only a few number of levels (maybe by using `cut`). This can not only lose a lot of information especially if the cutting is not done after a very careful analysis, but it also increases the *df* for the analysis, potentially making it harder to get significant results. If possible, keeping numeric variables numeric is probably a good idea.

As for the latter, the scale, it is important to realize, say, that not all numeric predictors should be entered into model as is. For example, frequency effects often operate on a logarithmic scale such that, even if $word_1$ is ten times as frequent as $word_2$, the effect of $word_1$ on the dependent variable, e.g. reaction time, may only be log (10) times as strong. Thus, what one should maybe put into the regression equation is log (frequency) (and to interpret results more easily, it may be good to use logs to the base of 2!

A third threat is concerned with the fact that probably most statistical modeling in linguistics is some sort of (generalized) linear modeling in which the effect of a predictor can be summarized with a straight regression line (in some numerical space). However, relations between predictors may differ with regard to how they are best characterized, as the two panels in Figure 60 exemplify. It's not a good idea to just force a straight regression line through the data in the right panel …

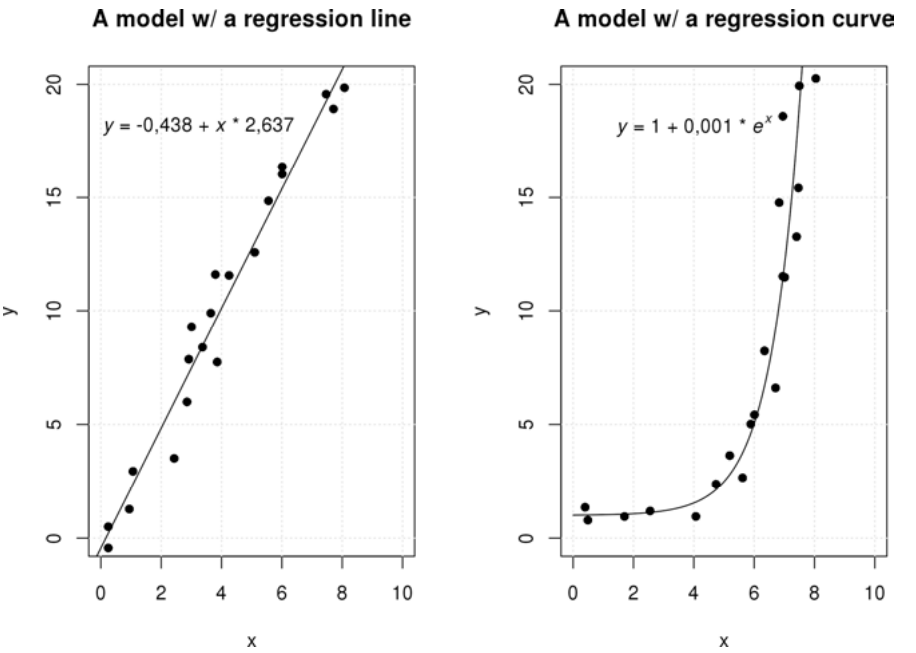

Figure 60. Models involing a regression line and a regression curve

Sometimes, data also exhibit interrupted trends that are best characterized with two or more regression lines/curves, etc. Again, just fitting one straight line through such data is risky, to say the least. The good part about the above three threats is that, if you proceed along the lines of Chapter 4, you won't usually make such mistakes. One reason why nearly every section in Chapter 4 involved some visualization was to hammer into your brain the fact that exploratory visualization should be an integral part and at the beginning of any statistical analysis, and proper visualization will reveal logarithmic relations, curvilinear trends, interrupted trends, and so on.

In addtition to these three risks I want to mention two others. One is rather trivial, one less so. The former is that your model is going to do a worse job at accounting for the data (the predictive aspect of the modeling process) and at allowing you to explain the data (the explanatory aspect of the modeling process) if you leave out important predictors. The latter is less trivial and brings us back to the notion of interaction, more specifically, to the question of whether or not to include interactions in your models. One can probably distinguish three different positions on this matter.

One is that interactions between independent variables should be included right from the start. This is because (i) if you do not include interac-

tions in the model equation, they do not get tested and you don't know whether the interactions(s) would in fact help account for the data much better, and (ii) if you included only interactions for which there was a clear theoretical motivation, it would become harder to find unexpected things; this would be a (not uncontroversial) way in which exploratory work seeps into what is usually a hypothesis-testing approach.

A second position is that you only include interactions you can motivate theoretically *a priori*. This has the above disadvantage, but the advanatage that this makes it hard to fish for something in the data.

The third position may still be the most frequent one: interactions are not included because the importance of the concept is not clear to the user or, just as bad, because the software that is being used makes including interactions hard (Varbrul is a case in point).

This issue of whether or not to include interactions is important enough to merit a short example (which you may recognize as a previous exercise). Let's assume 80 students (L1 speakers of German from two school classes A and B of 40 students each, a predictor called CLASS) had participated in one dictation in their L1 German and one in an L2 they are learning, English. Then, the numbers of mistakes in English (ENGLISH) and German (GERMAN) were counted to determine whether one can predict the numbers of mistakes made in the L2 on the basis of the numbers of mistakes in the L1 and the class the students attended. Two multifactorial models might be fitted to the data, one with the interaction between GERMAN and CLASS, one without (recall from Section 3.2.2 the two models in (58) are mere notational variants):[29]

(57) ENGLISH ~ GERMAN + CLASS

(58) a. ENGLISH ~ GERMAN + CLASS + GERMAN:CLASS

b. ENGLISH ~ GERMAN * CLASS

The results of the models in (57) and (58) are shown in Table 44 and Table 45 respectively. Both models are highly significant and explain the data really well: look at the huge and significant R^2-values. However, there are several important and interrelated problems with the model without the interaction (in (57)). First, this model does a worse job at accounting for the data than the model with it (in (58)): the bold figures in the rows called "Residual var(iance)" show how much variability in the data the models

29. I do not provide the data here but you will see this example again in one of the exercises for Chapter 5.

leave unaccounted for and you can see that that value is much higher in Table 44; a significance test would show that it is in fact significantly higher, which is another way of saying that the model in (57) is significantly worse than the one in (58).

Table 44. The results of the linear model in (57)

	SumSq	Estimate	Std. error	*t*	*p*
Intercept	23.61	2.75	1.52	1.8	0.08
GERMAN	2931.69	**1.75**	0.09	**20.1**	<0.001
CLASS	3010.30	**-8.72**	0.43	-20.37	<0.001
Residual var.	**558.68**				
overall R^2 / *p*	mult. R^2= 0.974	adj. R^2= 0.973		$F_{2, 77}$= 1416	*p*<0.001

Table 45. The results of the linear model in (58)

	SumSq	Estimate	Std. error	*t*	*p*
Intercept	24.9	2.82	1.15	2.44	0.017
GERMAN	2461.42	**1.64**	0.07	**24.29**	<0.001
CLASS	0.25	**-0.28**	1.15	-0.25	0.807
GERMAN:CLASS	241.73	-0.515	0.07	-7.61	<0.001
Residual var.	**316.95**				
overall R^2 / *p*	mult. R^2= 0.985	adj. R^2= 0.984		$F_{3, 76}$= 1661	*p*<0.001

Second, the *p*-values for the regression coefficients, or estimates, are very different. The main and crucial difference is that the model that explains the data better ((58)) says CLASS is not significant on its own but only in the interaction whereas the one that explains the data worse (in (57)) says CLASS is a significant main effect. This is not an unimportant technicality: CLASS is a binary variable, which means that, if it is a significant, its coefficient is a difference in means between the two classes, and GERMAN is a numeric variable, which means that, if it is significant, its coefficient is a slope of a regression line. Thus, what the model in (57) leads you to believe is this: students from the two classes are differently good on average (differing by 8.72 mistakes), but you can use one and the same slope for both classes to predict ENGLISH from GERMAN. This is represented in the left panel of Figure 61, with GERMAN and ENGLISH on the *x* and *y*-axis respectively, and CLASS is indicated by the letters.

However, the model in (58) says something very different, namely that there is no difference in *means* between the two classes (the *p*-value of

CLASS is huge). However, GERMAN:CLASS is significant – but what does that mean? It means that the *slope* between GERMAN and ENGLISH differs significantly across classes, which is represented in the right panel of Figure 61, and even if we did not already know from the first comment above that this model is better, the fit of the two regression lines with their separate slopes certainly seems better. Thus, the two models say very different things about what CLASS does …

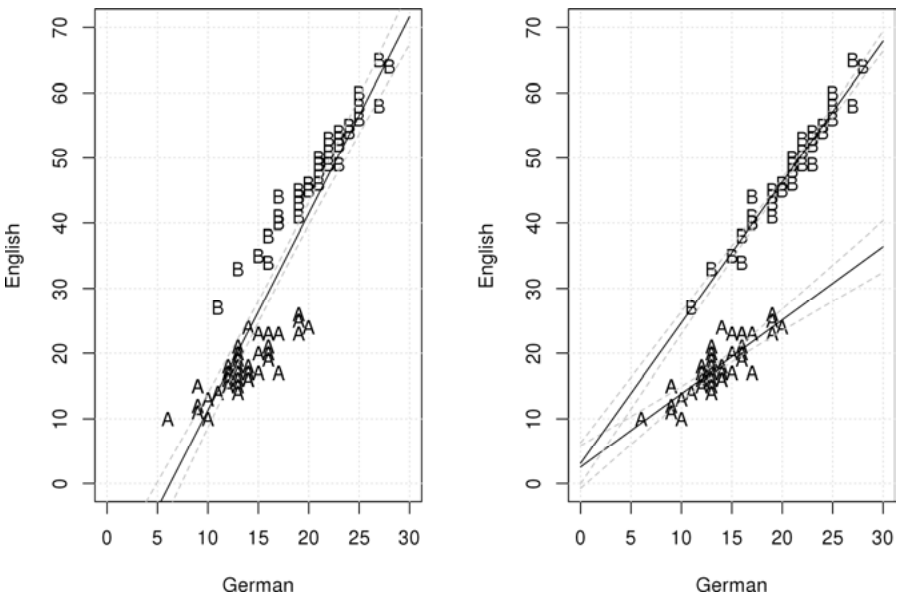

Figure 61. ENGLISH ~ GERMAN + CLASS (+ GERMAN:CLASS)

Finally and related to the previous point, the coefficients in the two models, and thus their predictions, differ a lot. The residuals of the worse model are on average more than 36% higher than those of the better one.

In sum, in this case, leaving out the interaction would leave you with a model that looks great on the surface (large R^2 and highly significant) but that is significantly worse than the model with the interaction, which tells a very different explanatory story about the data, and which is much worse at 'predicting' the data points. Against this background, it is amazing how often interactions are still not explored (properly).

One thing I have seen is that researchers seem aware of such issues but that their tools are not equipped to handle interactions (or continuous data) well; again, Varbrul is a case in point. So how might they then try to ad-

dress interactions? By fitting a separate model for each class:

(59) $\text{ENGLISH}_{\text{CLASS A}} \sim \text{GERMAN}_{\text{CLASS A}}$
(60) $\text{ENGLISH}_{\text{CLASS B}} \sim \text{GERMAN}_{\text{CLASS B}}$

If one does that, one does indeed get two significant simple regressions and the correct slopes of 1.13 for class A and 2.16 for class B. But why is this still a bad idea?

THINK
BREAK

Well, with this approach how do you know whether the difference between these two slopes is significant or not? The interaction does not show up in either model in (59) and (60) so the slopes never get compared to each other so you don't get a *p*-value so you don't know whether that is a significant difference or not. I have seen plenaries and papers and more where Varbrul weights for different time periods were compared to each other without any test of whether the difference between the Varbrul weights of different time periods were significant and thus indicative of change over time or not … You either have to include the interaction and get a *p*-value for it, or you have to at least check the confidence intervals of the slopes for whether they do not overlap (which, here, they do not).

In sum, I advise you to consider carefully the nature of the variables involved,to spend a considerable amount of time exploring your data (especially visually) before you start doing anything else, and to be very aware of the potential importance of interactions.

1.2.2. Selecting a final model

Once the above issues have been considered, a first model is formulated, and often this model is what is called a *maximal model*, i.e. a model including all independent variables, all of their interactions (often only up until interactions of three independent variables, because interactions of an even higher order are extremely difficult to understand). However, this is usually only the starting point since the maximal model usually contains predictors that do not contribute enough to the model, and since the famous dictum

called Occam's razor (*entia non sunt multiplicanda praeter necessitatem*) essentially requires you to discard predictors that don't pull their own weight or, more formally, do not contribute enough to the model's success.

Model selection is then influenced by two parameters: the *direction* of model selection and the *criterion* determining whether or not a predictor gets to be in the model. As for the former, there are three approaches:

- *backward selection*: here, you start with the maximal model as outlined above and successively test whether you have to discard predictors which do not contribute enough to the model. The selection process ends when no predictor can be discarded anymore with making the model too much worse or when no predictors are left in the model. The elimination of predictors begins with the highest level of interactivity and proceeds downwards in the direction of main effects, and you cannot discard a predictor that participates in a required higher-order interaction. That means, you cannot delete even an insignificant predictor B if the interaction A:B is significant.
- *forward selection*: here, you start with a very small model (maybe even just one that consists of the overall mean) and successively test whether you can add predictors. The selection process ends when no addition of a predictor improves the model enough anymore or when all available predictors are already in the model. The addition of predictors begins with main effects and moves up to higher-order interactions (if their main effects have already been included, as above).
- *bidirectional*: here, you start with some model and allow a usually automatic algorithm to add and subtract predictors as warranted.

I think the first approach is most widely used in linguistics but there are also good arguments not to do model selection at all (cf. Harrell 2001: Section 4.3 or Faraway 2005: Section 8.2).

As for the latter, I have been intentionally vague above when it came to describing when predictors are added or discarded: I always just said "good enough." This is because there are again at least two possible ways (who would want life to be easy …):

- a *significance-based approach*, according to which a predictor can be added to a model if it makes the model significantly better, and according to which a predictor should be discarded if its deletion does not make the model significantly worse.
- a *criterion-based approach*: the *AIC* (Akaike Information Criterion), for

instance, is one measure that relates the quality of a model to the number of predictors it contains (and thus operationalizes Occam's razor). If two models explain data equally well, then the model with fewer predictors will have a smaller *AIC*. Thus, in this approach, a predictor can be added to, or deleted from, a model if that lower *AIC*.

Once the model selection process has been completed, you have what is sometimes called the minimal adequate model, which can then be explored in terms of (i) whether the model as a whole is significant or not and how well it accounts for the data and (ii) what each predictor in that model contributes to the model: is it significant, what is the direction of its effect(s), and what is the strength of its effect(s). After this lengthy, but necessary theoretical introduction, the following sections will discuss all these matters – (different types of) regression models, main effects, interactions, model selection. prediction accuracy etc. – on the basis of many practical examples. Section 5.2 discusses linear models for (multiple) linear regression, ANOVAs, and ANCOVAs.

Recommendation(s) for further study
– Good and Hardin (2012: Part III) and Crawley (2007: Ch. 9)

2. Linear models

In Sections 3.2.3 and 4.4.1, we looked at how to compute and evaluate the correlation between an independent ratio-scaled variable and a dependent ratio-scaled variable using the Pearson product-moment correlation coefficient *r* and linear regression. In this section, we will extend this to the case of multiple independent variables. The data we will explore involve the question of how to predict speakers' reaction times to nouns in a lexical decision task and involves the following variables:[30]

– a dependent variable, namely the reaction time (RT) to words in a lexical decision task REACTTIME, whose correlation with the following in-

30. The words (but not the reaction times) are borrowed from a data set from Baayen's comprehensive (2008) book; the other characteristics of these words were taken from, or made up / modified based on, the MRC Psycholinguistic Database; cf. <http://www.psy.uwa.edu.au/ mrcdatabase/mrc2.html> for more detailed explanations regarding the variables in general.

dependent variables you are interested in (in this case, the dependent variable is an average of reaction times, which would not normally be the case; this is for expository reasons only and doesn't matter here);
- an independent numeric variable FREQUENCY, which corresponds to their logged frequency (according to Kučera and Francis 1967);
- an independent categorical variable FAMILIARITY, which is an index summarizing subjects' rated familiarity with the referent of the word;
- an independent binary variable IMAGEABILITY, which is an index summarizing subjects' rated imageability of the referent of the word;
- an independent numeric variable MEANINGFULNESS, which indicates subjects' average meaningfulness rating of the stimulus word.

This is the overall procedure of the linear modeling process we will use:

Procedure
- Formulating the hypotheses
- Loading the data, preparing them for modeling, and exploring them
- Computing, selecting, and interpreting a linear model
  - obtaining *p*-values for all predictors and for the model as a whole
  - interpreting the regression coefficients/estimates on the basis of (i) predicted values and (ii) plots of observed and/or predicted values
- Testing the main assumption(s) of the test:[31]
  - the variances of the residuals are homogeneous and normally distributed in the populations from which the samples were taken or, at least, in the samples themselves
  - the residuals are normally distributed (with a mean of 0) in the populations the samples come from or, at least, in the samples themselves

As you can see, in this section, we will test some assumptions of the linear modeling only after we have fit a model, which is because you can only check residuals when you have a model from which they can be computed. Also, this section is quite different from those in Chapter 4. It has been my experience – both in teaching and in my own research – that one of the greatest difficulties in linear modeling is not to get a significant result, but to understand what the regression coefficients (or estimates, I will use these terms interchangeably) in the results mean, a problem aggravated by the

31. There are other requirements – e.g., independence of residuals and absence of collinearity (!) – but for reasons of space I cannot discuss them all. See Fox and Weisberg (2011: Ch. 6) and Field, Miles, and Field (2012: Section 7.7) for exhaustive discussion.

fact that few books (i) say very explicitly in a language that beginners understand what the output means and (ii) discuss what the output means for two different ways to do linear modeling. In order to address both of these issues, this section will walk you through six fairly simple linear models that differ in terms of the predictors they involve to show you exactly – both numerically and visually – what regression coefficients mean. In addition, each of these linear models will be computed in two ways. One is a frequently-used standard in some commercial software applications that are unfortunately still in wide use, the other is the standard way in R.

Before we begin with the modeling, it is important to you to realize that, if this was a real study, you would not *run* many different models on the data to test different but overlapping hypotheses as I will do here. I will walk you through these models only so that you see how these are fit, interpreted, and visualized – what *you* would do if this was a real study is a model selection process of the type discussed in Section 5.2.7.

Let's begin by formulating the hypotheses, which will be applicable to all linear models in this section. We use an extension of the coefficient of determination r^2, namely its multiple regression equivalent multiple R^2:

H_0: There is no correlation between REACTTIME on the one hand and the predictors (independent variables and their interactions) on the other hand: multiple $R^2 = 0$.

H_1: There is a correlation between REACTTIME on the one hand and the predictors (independent variables and their interactions) on the other hand: multiple $R^2 > 0$.

Let us now load the data (from <_inputfiles/05-2_reactiontimes.csv>) such that the words for which have data become the row names, which is useful for some plots (output not shown):

```
> RTs<-read.delim(file.choose(), row.names=1)¶
> summary(RTs)¶
```

The summary shows you that this is a very small data set – a real study would better be based on more data. In addition, we find something that is only too realistic, namely that some variables have missing data, marked as NA, as they should be. For now, we will adopt a quick and dirty solution and make use of the fact that R's linear modeling function `lm` will automatically discard those cases of variables in the model that have missing data.

Before we begin with the modeling, there are two ways in which data

can often be prepared for better analysis. One of these is that it is sometimes useful to *z*-standardize numeric variables (with `scale`, recall Section 3.1.4), which may help with the problem of collinearity (the undesirable phenomenon that several of your predictors are highly correlated) and which may help with interpreting the results because the mean of standardized predictors is zero, which.e.g., makes intercepts and regression coefficients easy to understand. (On the other hand, it can also make results harder to understand because we lose the original units of the scale.) We will therefore not use this here, but it's a good thing to keep in mind for later.

The second thing we are going to do has to do with factors (and now you will see why I talked about them so much above). If you look at the summary output, you will see that the factor `FAMILIARITY` has levels that are ordered alphabetically but that that order is not compatible with the ordinal information that the levels communicate. We would want either lo, med, and hi, or hi, med, and lo, but not hi, lo, med. Thus, for both `FAMILIARITY` and `IMAGEABILITY`, we reorder their levels in an ordinally reasonable and homogeneous way:

```
> RTs$FAMILIARITY< factor(RTs$FAMILIARITY, levels=
   levels(RTs$FAMILIARITY)[c(2, 3, 1)])¶
> RTs$IMAGEABILITY<-factor(RTs$IMAGEABILITY, levels=
   levels(RTs$IMAGEABILITY)[c(2, 1)])¶
> summary(RTs)¶
```

Since graphical or tabular exploration (e.g., with boxplots or ecdf plots). which I strongly recommend you always do on data, does not really yield anything else in need of correction/preparation, we can now attach `RTs` and load a few packages we will use. Finally, the code file defines a few functions we will use a few times – `se.mean`, `ci.mean`, and `error.bar` – so just copy and paste that code into R so that you can use these functions below.

```
> attach(RTs)¶
> library(aod); library(car); library(effects); library(gvlma);
   library(multcomp); library(rgl)¶
```

2.1. A linear model with a binary predictor

Although this first linear model is the simplest of all, this section will be a bit longer because all the things having to do with linear models will show up for the first time. So, don't despair, everything else later will be shorter. To test whether IMAGEABILITY is correlated with REACTTIME, we fit what

is about the simplest possible linear model. However, to get results that are comparable with what some commercial software outputs, we first set the way R computes contrasts as shown here (more on that in a while), then we fit a linear model (where we also tell R which data frame the variables are from with the `data` argument), and then we inspect the output:

```
> options(contrasts=c("contr.sum", "contr.poly"))¶
> model.01<-lm(RT~IMAGEABILITY, data=RTs)¶
> summary(model.01)¶
                                                                    […]
Residuals:
    Min      1Q  Median      3Q     Max
-84.629 -40.016   2.145  26.975 160.799

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    620.666      6.998  88.693   <2e-16 ***
IMAGEABILITY1   12.987      6.998   1.856   0.0693 .
---                                                                 […]
Residual standard error: 50.5 on 51 degrees of freedom
  (24 observations deleted due to missingness)
Multiple R-squared: 0.06326,     Adjusted R-squared:  0.0449
F-statistic: 3.444 on 1 and 51 DF,  p-value: 0.06925
```

Let's start at the bottom: the model as a whole is not significant, as the *p*-value shows, which in turn is computed from the *F*-value at *df*=1, 51 (here it is: `pf(3.444, 1, 51, lower.tail=FALSE)¶`). The multiple correlation between IMAGEABILITY and REACTTIME, multiple R^2, ranges theoretically from 0 to 1 and quantifies the variability accounted for, so a value of 0.06326 is really small. In addition, the value usually reported is the adjusted R^2. This R^2-value is adjusted such that you incur a slight penalty for every predictor included in your model. Thus, if, in a desperate attempt to explain more variability, you were to add a useless variable into the model, then it is very likely that whatever little bit of random variation that useless variable accounts for will be eaten up by the penalty. Thus, this adjustment brings Occam's razor into modeling. Obviously, adjusted R^2 is also really small. Then, there is a warning that R deleted 24 observations because these cases had NA in the variables in the model.

We ignore the residual standard error and briefly skip to the top of the output,[32] where we get a summary output regarding the residuals and we can already see that these are hardly normally distributed – whatever we learn here must be interpreted cautiously (We'll get back to this.)

32. The residual standard error is the root of the quotient of the residual sums of squares divided by the residual *df* (in R: `sqrt(sum(residuals(model.01)^2)/51)`.

While we have not talked about what the coefficients mean, let me already point out the obvious: they are just estimates, which is how R labels them, which means you can get confidence intervals for them, and the fact that the confidence interval for IMAGEABILITY includes 0 already suggests that, whatever it is – to be discussed in a moment – it's not significant:

```
> confint(model.01)¶
                   2.5 %    97.5 %
(Intercept)   606.617138 634.71498
IMAGEABILITY1  -1.061602  27.03624
```

Before we turn to the coefficients and their *p*-values, let us run two more lines of code, which are very useful for predictors with more than one *df*, i.e. predictors that are neither binary nor numeric (i.e., this does not apply here, I mention it here anyway for the sake of consistency).

```
> drop1(model.01, test="F")¶
Single term deletions
Model:
RT ~ IMAGEABILITY
             Df Sum of Sq    RSS    AIC F value  Pr(>F)
<none>                    130059 417.69
IMAGEABILITY  1    8783.6 138843 419.15  3.4443 0.06925 .
---                                                    […]
> Anova(model.01, type="III")¶
Anova Table (Type III tests)
Response: RT
               Sum Sq Df   F value  Pr(>F)
(Intercept)  20060844  1 7866.4268 < 2e-16 ***
IMAGEABILITY     8784  1    3.4443 0.06925 .
Residuals      130059 51                                […]
```

These functions are important ways to get *p*-values for predictors. The first, `drop1`, looks at all the predictors in the model and checks which predictor could theoretically be deleted from the model at this stage in the model selection process, and for the predictors that could be deleted at this point, it returns a *p*-value for the test of the original model, `model.01`, against the model that you would get without that predictor. The second, `Anova`, is available from the library `car`. It computes a *p*-value for predictors that is the same as commercial software returns by default.[33] As you

33. The issue of sums of squares (the `type="III"` argument) is hotly debated. I will not engage in the discussion here which approach is better but use `type="III"` for reasons of comparability with other software even if `type="II"` may often be more useful; see Crawley (2002: Ch. 18, 2007: 368ff.), Larson-Hall (2010: 311-313), Fox and Weisberg (2011: Sections 4.4, 4.6), Field, Miles, and Field (2012: 475f.) and the R-help list.

can see, both return the already known *p*-value for the only predictor.

With this output, let us now turn to the coefficients. First the simpler part, the *p*-values, then, second, the coefficients. The *p*-value for the intercept is usually disregarded: it tests the H_0 that the intercept is 0, but there are few applications where that is relevant. More interesting is the *p*-value for the predictor IMAGEABILITY. (In fact, R writes `IMAGEABILITY1`, I will explain that in a moment.) In this simplest of cases, where our model only has one binary predictor, the *p*-value there is the same as the *p*-value of the whole model, and the same of that predictor in the `drop1` and in the `Anova` output: 0.06925. So, the predictor does not have a significant effect and, in a sense, the output of `drop1` says that most intuitively because what `drop1` is essentially saying is "if you drop IMAGEABILITY from `model.01`, then the resulting model is not significantly worse (p=0.06925)." A different way to view this is as showing that the regression coefficient is not significantly different from 0. All this is identical to what you get from a *t*-test.

While this model/predictor is not significant, we will proceed with the discussion and plotting as if it were, because at this point I want to show you how such a model output is interpreted and plotted; a more realistic model selection process follows in Section 5.2.7.

So – finally – what do the estimates mean, the 620.666 of `(Intercept)` and the 12.987 for `IMAGEABILITY1`? I recommend to approach this question on the basis of the values that the model predicts as in Section 4.4.1:

```
> preds.hyp<-expand.grid(IMAGEABILITY=levels(IMAGEABILITY));
   preds.hyp[c("PREDICTIONS", "LOWER", "UPPER")]<-predict(
   model.01, newdata= preds.hyp, interval="confidence");
   preds.hyp¶
  IMAGEABILITY PREDICTIONS    LOWER    UPPER
1           lo    633.6534 612.5138 654.7929
2           hi    607.6787 589.1691 626.1884
```

Thus, `model.01` predicts that, when the word is of low imageability, then people's reaction times will be about 26 ms slower than when the word is of high imageability. Just to make this clear: this means the model makes only two different predictions: when IMAGEABILITY is low, it always predicts an RT of 633.6534, and when IMAGEABILITY is high, it always predicts an RT of 607.6787, and these two predicted values are also the observed means: try `tapply(RT, IMAGEABILITY, mean)¶`. Note also how much the confidence intervals of the two predictions overlap.

If we look at `preds.hyp`, you may already suspect what the regression estimates mean. When you compute the linear model as we did here, i.e. with sum contrasts!, then these two values mean the following:

- the intercept, 620.666, is the unweighted (!) mean of the means of the dependent variable, when it is grouped by the independent variable. That is, 620.666 is the mean of of 633.6534 and 607.6787, and that is an unweighted mean because it does not take into consideration that the two levels of IMAGEABILITY are not equally frequent.
- the coefficient for `IMAGEABILITY1`, 12.987, is what you have to add to the intercept to get the predicted RT for the first level of IMAGEABILITY (hence the 1): 620.666 + 12.987 = 633.653. (And since the intercept is the mean of means, if you subtract the coefficient from the intercept, you get the predicted RT for the second level of IMAGEABILITY: 620.666 - 12.987 = 607.679.)

This is visually represented in Figure 62.

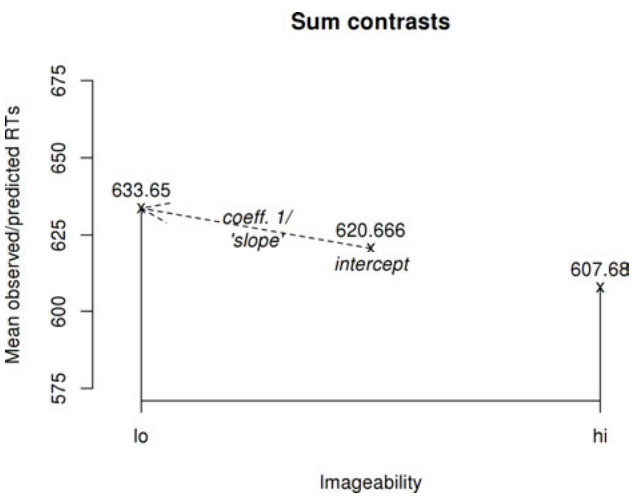

Figure 62. The regression estimates of `model.01` with sum contrasts

Now, you may wonder why it says "slope" in Figure 62. This is because you can conceptualize the intercept as an *x*-axis value of 0 and IMAGEABILITY:1 as an *x*-axis value of 1, which is pretty much what linear modeling does under the hood: For numeric variables, effects are given as slopes which represent how much the predicted *y*-value changes for every unit change on the *x*-axis anyway, but with the above perspective you can also understand coefficients for factor levels (e.g., 12.987) as slopes.

Finally, while this particular model is so simple that the coefficients etc. can be understood without any visualization, this can quickly change so I

will even here present two ways in which the data can be visualized. The code to generate the plots in Figure 63 is in the code file. The left is an ordinary barplot of means, the only thing I added are the confidence intervals for the means; the right plot is a very easy-to-generate effect plot.

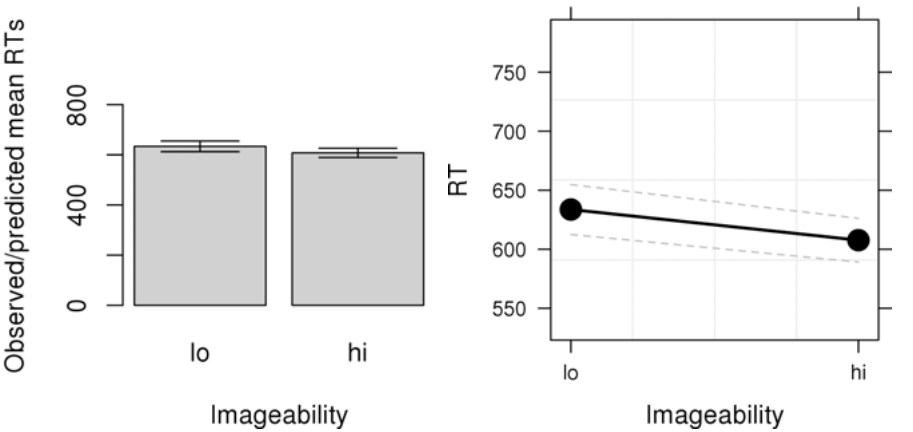

Figure 63. The effects of `model.01`: barplot with observed/predicted means and their 95% confidence-interval bars (left panel); effects plot from the library `effects` (right panel)

All the above was how much commercial software would report the results. However, the standard way in R is actually a bit different, thankfully it is really only a bit … Since I want you to know R's standard approach and since that approach will help you understand logistic regression later, I will now discuss it very briefly. The only real difference in execution for this second, R's standard approach, is that you now use R's default contrasts, treatment contrasts. If you then generate the model again, the R^2-values, the overall *p*-value, most is the same but not the coefficients:

```
> options(contrasts=c("contr.treatment", "contr.poly"))¶
> model.01<-lm(RT~IMAGEABILITY, data=RTs)¶
> summary(model.01)¶
                                                                […]
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)      633.65      10.53  60.177   <2e-16 ***
IMAGEABILITYhi   -25.97      14.00  -1.856   0.0693 .
---                                                             […]
> confint(model.01)¶
```

After what we have done above, you probably immediately see what the

intercept and the coefficient for `IMAGEABILITYhi` represent:

- the intercept, 633.65, is the observed/predicted mean of the dependent variable, when the independent variable IMAGEABILITY is its first level, *LO*.
- the coefficient for `IMAGEABILITYhi`, -25.97 is what you add to the intercept to get the predicted RT for the second level of IMAGEABILITY (hence the *HI*): 633.65 + -25.97 = 607.68; the *p*-value shows that the difference between the intercept (representing IMAGEABILITY: *LO*) and this predicted RT for IMAGEABILITY:*HI* is not significant.

This is also represented in Figure 64, where, as discussed above, the annotation of $x = 0$ and $x = 1$ motivate the use of the word *slope* in the plot.

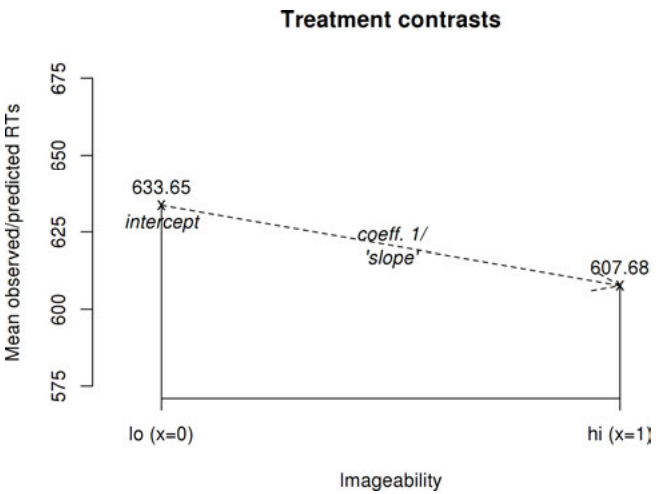

Figure 64. The regression estimates of `model.01` with treatment contrasts

As you can see, in this simple case both approaches yield different coefficients, but they amount to the same significance tests (with `drop1` again, see the code file) and the same predictions (in the new `preds.hyp`; see the code file). Also, note that I provide some extra code to get *p*-values for coefficients using `wald.test` and `glht` in the code file. You should always run that, too, since it will be very useful later; much later you may want to explore Bretz, Hothorn, and Westfall (2011).

You can summarize the results as follows: "A linear model was fit with REACTTIME as the dependent variable and IMAGEABILITY (low vs. high) as

the independent variable. The model was not significant ($F = 3.444$, $df_1 = 1$, $df_2 = 51$, p=0.069). There was only a marginally significant tendency such that low and high imageability correlated with slower and faster reaction times respectively. [Show graph(s)]."

2.2. A linear model with a categorical predictor

In this section, we still cover only one predictor – so actually, we are still not doing multifactorial analysis – but we make the model a bit more complex by studying a predictor with three levels (FAMILIARITY), which means you could not do a *t*-test anymore.[34] First again the approach using sum contrasts (from now on, I will not show all the output anymore):

```
> options(contrasts=c("contr.sum", "contr.poly"))¶
> model.01<-lm(RT~FAMILIARITY, data=RTs)¶
> summary(model.01)¶
> confint(model.01)¶
```

This model is significant: the overall p-value is < 0.001. Since this is also a model with only one predictor, you know that this is now also the p-value for that one predictor. However, if you look at the table of coefficients, you don't find it there. Instead you have an intercept and then two quite different p-values. How do you get a p-value for FAMILIARITY other than by looking at the overall p-value (e.g., when you have more than one predictor)? This is a case where `drop1` and `Anova` are needed because – remember from above – here the (only) predictor has more than 1 *df* because it is neither binary nor numeric. Thus you use `drop1` and `Anova`:

```
> drop1(model.01, test="F")¶
> Anova(model.01, type="III")¶
```

There's the p-value for FAMILIARITY, and this time you can see how the significance-based and the criterion-based approach agree: FAMILIARITY is significant and taking it out increases *AIC* considerably.

34. Incidentally, this section as well as the previous cover linear models that some would refer to as ANOVAs, analyses of variance. However, since the underlying approach between linear regressions with only numerical independent variables, ANOVAs with only categorical independent variables, and ANCOVAs with both categorical and numeric independent variables is the same – in R they are all fit with `lm` – I will not topicalize the differences between these methods but rather focus on their commonalities.

So, what do the estimates mean? Again, we approach this via the predicted values. It turns out that there is a nice ordinal effect: as FAMILIARITY increases, RTs go down, which makes sense.

```
> preds.hyp<-expand.grid(FAMILIARITY=levels(FAMILIARITY));
   preds.hyp[c("PREDICTIONS", "LOWER", "UPPER")]<-
   predict(model.01, newdata=preds.hyp,
   interval="confidence"); preds.hyp¶
```

From `preds.hyp`, you can again guess what the estimates mean, and this is also visualized again in Figure 65:

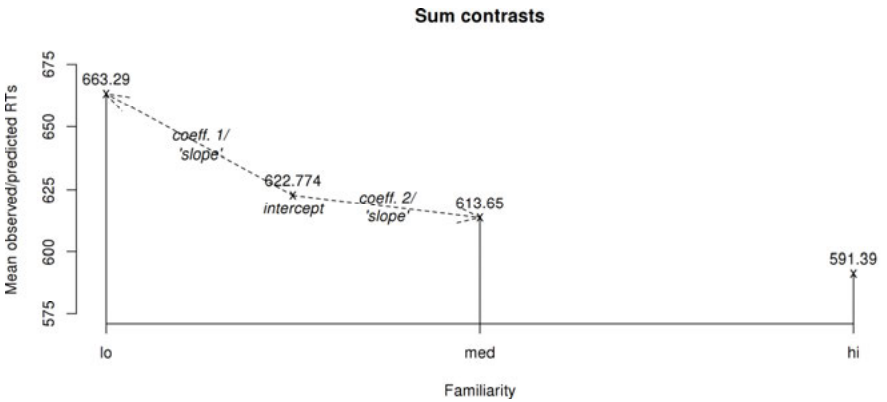

Figure 65. The regression estimates of `model.01` with sum contrasts

- the intercept, 622.774, is the unweighted (!) mean of the means of the dependent variable, when it is grouped by the independent variable. That is, 622.774 is the mean of 663.2880, 613.6471, and 591.3879, and that is an unweighted mean because it does not take into consideration that the levels of FAMILIARITY are not all equally frequent;
- the coefficient for `FAMILIARITY1`, 40.514, is what you add to the intercept to predict the RT for the first level of FAMILIARITY (hence the 1);
- the coefficient for `FAMILIARITY2`, -9.127, is what you add to the intercept to predict the RT for the second level of FAMILIARITY;
- and if you subtract both coefficients for FAMILIARITY from the intercept, you get the predicted RT for the third level of FAMILIARITY.

Note that you do not get *p*-values for *all* differences between the intercept and the levels, and sometimes you may want to run a variety of tests

on differences between means. One (rather conservative) way to approach this question involves the function `TukeyHSD`.

```
> TukeyHSD(aov(model.01), ordered=TRUE)¶
```

The main argument of this function is an object created by the function `aov` (an alternative to `anova`), which in turn requires the relevant linear model as an argument. As a result, you get a table for all three comparisons you can make between three means. You get the differences between the means, the lower and the upper confidence intervals for the differences, and *p*-values that have been adjusted for the fact that you are suddenly performing three significance tests on the same data set. Why would *p*-values have to be adjusted for that?

THINK
BREAK

The point of a significance level was to make sure that, if you accept an H_1, your probability to do that *in*correctly was < 0.05. Now, if you reject two independent H_0 at each $p = 0.05$, what is the probability that you do so correctly both times? It's 0.9025, i.e. 90.25% Why? Well, the probability you are right in rejecting the first H_0 is 0.95. But the probability that you are *always* right when you reject H_0 on two independent trials is $0.95^2 = 0.9025$. This is the same logic as if you were asked for the probability to get two sixes when you simultaneously roll two dice: $^1/_6{}^2 = {}^1/_{36}$. The probability that you are *always* right when you reject H_0 on three independent trials is $0.95^2 = 0.857375$. In fact if you look at 13 H_0s, then the probability that you do not err once if you reject all of them is in fact dangerously close to 0.5: $0.95^{13} \approx 0.5133$, a.k.a. pretty far away from 0.95. Thus, the probability of error you use to evaluate each of *n* H_0s should not be 0.05 – it should be smaller so that when you perform all *n* tests, your overall probability to be always right is 0.95. Thus, if you want to test *n* H_0s, you must use $p = 1\text{-}0.95^{(1/n)}$. For 13, that means $p \approx 0.00394$. Then, the probability that you are right on any one rejection is 1-0.00394 = 0.99606, and the probability that you are right with all 13 rejections is $0.99606^{13} \approx 0.95$. A shorter heuristic that is just as conservative (actually, too conservative) is the Bonferroni correction. It consists of just dividing the desired significance level – i.e., usually 0.05 – by the number of tests – here 13. You get $^{0.05}/_{13} \approx$

0.003846154, which is close (enough) to the exact probability of 0.00394 computed above. Thus, if you do multiple *post hoc* tests on a dataset, you usually adjust the significance level, which makes it harder for you to get significant results just by fishing around in your data, which should motivate you to formulate reasonable H_1s beforehand rather than excessive *post hoc* testing.

Back to the data at hand: we can see that the difference between medium and high levels of FAMILIARITY is not significant, but the other two differences are. What does that mean?

THINK BREAK

It means that Occams razor would require that you now test whether you need to uphold the difference between medium and high familiarity or whether you must conflate the two, and we will do this in Section 5.2.7.

As a last step for this model, you can generate some plots again, and the code file will show you how to generate plots like Figure 63 for this model.

Now, let us very briefly explore this same model, but now with R's default of treatment contrasts again:

```
> options(contrasts=c("contr.treatment", "contr.poly"))¶
> model.01<-lm(RT~FAMILIARITY, data=RTs)¶
> summary(model.01)¶
                                                                    […]
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)       663.29      13.23  50.118  < 2e-16 ***
FAMILIARITYmed    -49.64      15.59  -3.185 0.002449 **
FAMILIARITYhi     -71.90      18.72  -3.842 0.000334 ***
---                                                                 […]
> confint(model.01)¶
```

After what we have done above, the estimates are probably clear:

- the intercept, 663.29, is the observed/predicted mean when the independent variable FAMILIARITY is its first level, lo.
- the coefficient for `FAMILIARITYmed`, -49.64 is what you add to the intercept to predict the RT for the second level of FAMILIARITY.
- the coefficient for `FAMILIARITYhi`, -71.90 is what you add to the intercept to predict the RT for the third level of FAMILIARITY.

This is also represented in Figure 66, where, as discussed above, the annotation of $x = 0$ and $x = 1$ (two times, one for each estimate) help motivate the use of the word *slope* in the plot. Thus, in some sense, it's all the same as before in Section 5.2.1 and you can summarize this section's model along the lines of the one above.

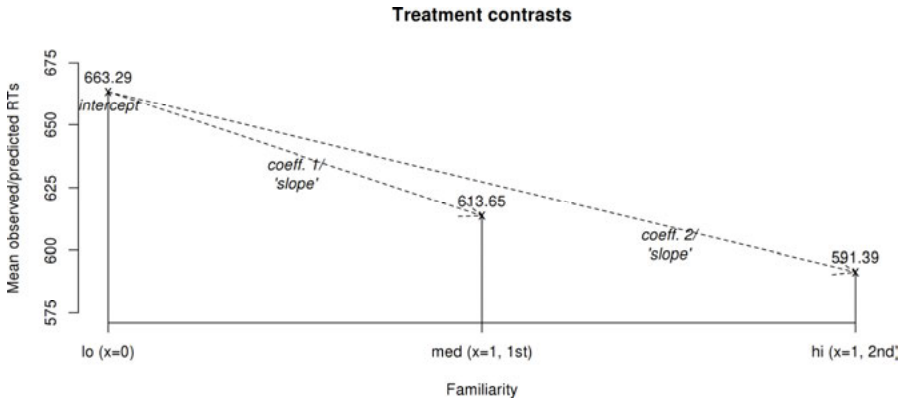

Figure 66. The regression estimates of `model.01` with treatment contrasts

2.3. A linear model with a numeric predictor

We are still only preparing for multifactorial models. In the last two monofactorial ones, the only predictor was a (binary or categorical) factor and, correspondingly, its effects were differences between means. However, we also began to approach that as a slope, by conceptualizing differences between means as slopes from the y-value at a reference level (at $x = 0$) to a y-value at a level defined as $x = 1$. In this section, we will very briefly revisit the case of a numeric predictor, i.e., what we discussed in Section 4.4.1. One nice thing is that, with just an interval-scaled predictor, we do not have to cover two types of contrasts. We are going to look at the correlation between FREQUENCY and REACTTIME.

```
> options(contrasts=c("contr.sum", "contr.poly"))¶
> model.01<-lm(RT~FREQUENCY, data=RTs)¶
> summary(model.01)¶
> confint(model.01)¶
```

By now we have studied such cases both with `cor.test` and `lm` already so I won't go over all the results in detail again. Suffice it to say, that the

model is significant because its only predictor is, etc. Since the predictor is numeric, we do not really need the following two lines, but just to entrench them in your mind, here they are again, and they return the same *p*-value.

```
> drop1(model.01, test="F")¶
> Anova(model.01, type="III")¶
```

To determine what the estimates mean, we follow the same strategy as before and compute predictions for values from the attested range:

```
> preds.hyp<-expand.grid(FREQUENCY=floor(min(FREQUENCY)):
   ceiling(max(FREQUENCY))); preds.hyp[c("PREDICTIONS",
   "LOWER", "UPPER")]<-predict(model.01, newdata=
   preds.hyp, interval="confidence"); preds.hyp¶
```

You can recognize what you hopefully already guessed from above:

- the intercept, 667.03, is the predicted RT when the independent variable FREQUENCY is 0.
- the coefficient for FREQUENCY, -24.266 the increase in the predicted RT (i.e., given the minus, a decrease) for each unit increase of FREQUENCY.

As usual, you should plot the data to get an impression of the fit, and the code file provides a few examples of how you could do that.

Now that we have covered the basics, we can finally move on to multifactorial linear models. While this introductory part may have seemed long, having covered everything in that much detail will make things easier now.

2.4. A linear model with a two categorical predictors

We begin with a model in which we try to predict REACTTIME on the basis of two independent categorical variables, IMAGEABILITY and FAMILIARITY, and their interaction, IMAGEABILITY:FAMILIARITY. As before, we begin with a model based on sum contrasts. Recall the notation using the asterisk to say 'all these main effects and their interactions':

```
> options(contrasts=c("contr.sum", "contr.poly"))¶
> model.01<-lm(RT~IMAGEABILITY*FAMILIARITY, data=RTs)¶
> summary(model.01)¶
> confint(model.01)¶
```

The output becomes more complex … The model as a whole is significant ($p = 0.01138$), we can see that IMAGEABILITY is not significant, but we don't have individual *p*-values for FAMILIARITY and the interaction IMAGEABILITY:FAMILIARITY. Thus, before we try to understand the coefficients/estimates, a quick look at `drop1` and `Anova`:

```
> drop1(model.01, test="F")¶
> Anova(model.01, type="III")¶
```

This time, the output of the two is differently comprehensive. The output of `drop1` follows the above logic of backwards model selection and only returns *p*-values for those predictors that could be dropped *at this time*. Since there is an interaction of two variables and nothing more complex than that in the model, you can drop that interaction, but you cannot at this stage drop any of the variables from the interaction as long as the interaction is still in the model. Thus, `drop1` only returns the *p*-value for the model with vs. without the interaction and since the interaction is not significant, one should drop it (following Occam's razor).

The output of `Anova` is more comprehensive and returns *p*-values for all predictors in the model; you can recognize the *p*-values for IMAGEABILITY from the `summary(lm())` output, and the one for the interaction from the `drop1` output. We will not drop the interaction now because at this point I want to show you how such a model output is interpreted and plotted; again, the more realistic model selection process follows in Section 5.2.7.

Now to the predictors and their estimates:

```
> preds.hyp<-expand.grid(IMAGEABILITY=levels(IMAGEABILITY),
   FAMILIARITY=levels(FAMILIARITY)); preds.hyp[
   c("PREDICTIONS", "LOWER", "UPPER")]<-predict(model.01,
   newdata=preds.hyp, interval="confidence"); preds.hyp¶
```

I will not explain every coefficient in detail here –see the code file for painfully detailed definitions of each estimate – for two reasons. First, to save space: you will see how long and convoluted the definition of the estimates in the code file can become. Second, the whole point of generating `preds.hyp` is that we don't *have* to look at the coefficients that much. Of course you should still understand the explanation in the code file but in actual practice understanding the coefficients of a model with, say, five significant 3-way interactions and 10 other predictors on the basis of the coefficients is pretty much impossible. Thus, read the explanation of the coefficients in the code file carefully, run the code there to verify my ex-

planations, and try to recognize their effects in the data, but for now we will explore the model on the basis of its predictions, which show that

- the observed/predicted means don't do much as IMAGEABILITY changes (averaging across FAMILIARITY);
- the observed/predicted means decrease as FAMILIARITY increases (averaging across IMAGEABILITY)'
- there is a hint of an interaction (but we know from above it is not significant) because, when FAMILIARITY is *LO* or *MED*, then a change from IMAGEABILITY *LO* to *HI* speeds up reaction times, but has the opposite effect when FAMILIARITY is *HI*.

The Tukey test shows that, with a very conservative post-hoc testing approach, there is hardly anything significant in the data. But let us visualize the data. The code file shows you different kinds of plots, interaction plots using lines, a bar plot of means and confidence intervals, dot charts of means, and a (too?) colorful boxplot of the observed medians and their notches as well as means and their confidence intervals. Finally, the last one is an effect plot, which again shows clearly that this interaction is not significant: the lines for the means are nearly parallel.

Now, what about the same analysis with treatment contrasts?

```
> options(contrasts=c("contr.treatment", "contr.poly"))¶
> model.01<-lm(RT~IMAGEABILITY*FAMILIARITY, data=RTs)¶
> summary(model.01)¶
> confint(model.01)¶
> drop1(model.01, test="F")¶
```

Again, everything is the same as above except for the estimates and I explain what they mean in detail in the code file. However, since understanding treatment contrasts will be *very* important for logistic regressions, I want to comment on them here as well. There are two central rules that, once internalized, help you understand all treatment contrast results easily:

(61) Each coefficient/estimate for a predictor *X* (main effect, interaction, or factor level) is the value you must add to the intercept to,
   a. in the case of categorical variables, predict the value for the level of *X* you are looking at;
   b. in the case of numeric variables, predict the value that results from a one-unit change of *X*;

   while, and this is the crucial point, all categorical predictors not

mentioned in *X* are set to their first level (usually the alphabetically first level, but it can also be the one you set first, as in this case), and all numeric predictors not mentioned in *X* are 0.

The second rule is just a special case of (61), namely the intercept:

(62) Therefore, the intercept, where *no* predictor is mentioned, is the predicted value when
   a. *all* categorical variables in the model equation are set to their first level;
   and/or (!)
   b. *all* numerical variables are set to zero (which, if you centered or *z*-standardized them, corresponds to their mean)

Thus,

- the intercept is the predicted RT when both predictors are set to their first level (*LO*);
- the second coefficient is what you add to the intercept to predict the RT for when the predictor mentioned changes to the level mentioned here (i.e., IMAGEABILITY changes from *LO* to *HI*) and when the predictor not mentioned here stays at the level from the intercept (i.e., FAMILIARITY remains *LO*);
- the third coefficient is what you add to the intercept to predict the RT for when the predictor mentioned changes to the level mentioned here (i.e., FAMILIARITY changes from *LO* to *MED*) and when the predictor not mentioned here stays at the level from the intercept (i.e., IMAGEABILITY remains *LO*), similarly for the fourth coefficient;
- the fifth coefficient is for a predictor that is an interaction. Thus, to use it for a prediction, you do not just add this estimate to the intercept, but also the estimates for the main effects that are part of it. Thus, to predict the RT for when IMAGEABILITY is *HI* and FAMILIARITY is *MED*, you add to the intercept the second coefficient (for when IMAGEABILITY is *HI*), the third coefficient (for when FAMILIARITY is *MED*), and this fifth one (for the interaction):

```
> 676.30 + -26.11 + -58.94 + 18.91¶
[1] 610.16
```

Compare that to `preds.hyp[4,]`: the result is the same, and the same logic applies to the sixth coefficient. It is probably obvious by now why inspecting `preds.hyp` and plotting predicted values is easier than ploughing through the table of coefficients, especially since `preds.hyp` is the basis for the plotting, which you have done above.

You could now summarize `model.01` as before: overall model statistics, predictors and their *p*-values, and a plot.

2.5. A linear model with a categorical and a numeric predictor

In the last section, both variables were categorical so all effects were (adjustments to) means. Now we turn to mixed variables: one variable is categorical (FAMILIARITY), one is numeric (FREQUENCY). First, sum contrasts:

```
> options(contrasts=c("contr.sum", "contr.poly"))¶
> model.01<-lm(RT~FAMILIARITY*FREQUENCY, data=RTs)¶
> summary(model.01)¶
> confint(model.01)¶
```

The model is very significant ($p = 0.001728$), but as before you do not get all *p*-values for all predictors: you can see FREQUENCY is significant, but you do not get one *p*-value for FAMILIARITY and the interaction. Thus:

```
> drop1(model.01, test="F")¶
> Anova(model.01, type="III")¶
```

Both show that the interaction is not significant. On to the estimates:

```
> preds.hyp<-expand.grid(FAMILIARITY=levels(FAMILIARITY),
   FREQUENCY=floor(min(FREQUENCY)):ceiling(max(FREQUENCY)));
   preds.hyp[c("PREDICTIONS", "LOWER", "UPPER")]<-
   predict(model.01, newdata=preds.hyp, interval=
   "confidence"); preds.hyp¶
```

This data frame is not easy to process. (Given the length of this table, I show how to create a version that is easier to process in the code file.) One can see generally that, as FREQUENCY goes up, predicted RTs go down, but really what is needed is a graph. But a question first: What does the interaction represent and, therefore, how does this have to be plotted?

THINK
BREAK

As in Section 5.1.2, the interaction of a categorical and a numeric variable means that the there is not one slope for the effect of the numeric variable in the model but as many slopes as there are levels of that categorical variable. That is, the interaction reflects adjustments to slopes. Hence, we plot a graph that has different regression lines for FREQUENCY for each level of FAMILIARITY; the levels of FAMILIARITY are represented by their first letters. The plot here is quite minimalist (e.g., by not including the original data points), but the code file provides a variety of alternatives; the simplest one to do is, as usual, the effect plot.

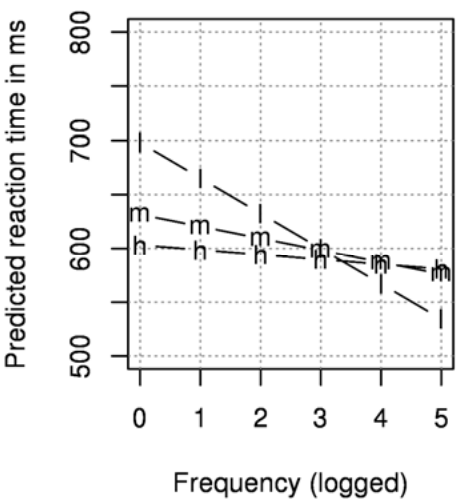

Figure 67. The interaction FAMILIARITY:FREQUENCY in `model.01`

The plot shows the results more efficiently than anything else (esp. when confidence intervals are added to show that interaction is not significant). The model shows that, on the whole, FREQUENCY speeds subjects up but especially when FAMILIARITY is *LO* compared to when it is not. This, together with the *p*-values etc., should be in your summary of the model.

Now again a quick glance at treatment contrasts:

```
> options(contrasts=c("contr.treatment", "contr.poly"))¶
> model.01<-lm(RT~FAMILIARITY*FREQUENCY, data=RTs)¶
```

```
> summary(model.01)¶
> confint(model.01)¶
> drop1(model.01, test="F")¶
```

As usual, it's the coefficients that change, and they change in accordance with the rules in (61) and (62):

- the intercept is the predicted RT when all predictors are set to their first level or 0, i.e. when FAMILIARITY is *LO* and FREQUENCY is 0;
- the second coefficient is what you add to the intercept to predict the RT for when the predictor mentioned changes to the level mentioned here (i.e., FAMILIARITY changes from *LO* to *MED*) and when the predictor not mentioned here stays at the level from the intercept (i..e. FREQUENCY remains 0), similarly for the third coefficient;
- the fourth coefficient is what you add to the intercept to predict the RT for when the predictor mentioned increases by one unit (since FREQUENCY is numeric, it changes from 0 to 1) and when the predictor not mentioned here stays at the level from the intercept (i.e., FAMILIARITY remains *LO*);
- the fifth coefficient is for a predictor that is an interaction. Thus, to use it for a prediction, you do not just add this estimate to the intercept, but also the estimates for the main effects that are part of it. Thus, to predict the RT for when FAMILIARITY is *MED*, and FAMILIARITY increases by 1, you add to the intercept the second coefficient (for when FAMILIARITY is *MED*), the fourth coefficient (for when FAMILIARITY increases by 1), and this one (for the interaction):

```
> 697.96 + -66.40 + -32.73 + 21.65¶
[1] 620.48
```

Compare that to `preds.hyp[5,]`; same for the sixth coefficient.

2.6. A linear model with two numeric predictors

Now we are getting serious, enough fun and games. We are going to model REACTTIME as a function of two numeric variables, FREQUENCY and MEANINGFULNESS, and their interaction. This is somewhat tricky because of the interaction. An interaction between two categorical variables reflects adjustments to means, an interaction between a categorical variable and a numeric variable reflects adjustments to slopes – but what is an interaction

between two numeric variables? As you will see, it is that one numeric variable's slope effect changes across the range of the other numeric variable, which also means we will sometimes have to consider three-dimensional plots: one predictor on the *x*-axis, the other predictor on the *y*-axis, the prediction on the *z*-axis.

With only two numeric predictors, we need not distinguish between sum and treatment contrasts so let's get started. (You may also paste the `drop1` and `Anova` lines, but they are unnecessary: every predictor has 1 *df*.)

```
> options(contrasts=c("contr.sum", "contr.poly"))¶
> model.01<-lm(RT~MEANINGFULNESS*FREQUENCY, data=RTs)¶
> summary(model.01)¶
> confint(model.01)¶
```

A just about significant model – although no predictor is significant, which is somewhat rare. We generate `preds.hyp`, which this time is a bit more cumbersome. Since we have two numeric variables, we generate ranges of values for both of them. For FREQUENCY we do this as before, for MEANINGFULNESS I do not just use eight values (an arbitrary choice, it could also be 20) from the attested range, but also 0 and 1 (so I can explain the coefficients).

```
> preds.hyp<-expand.grid(MEANINGFULNESS=c(0:1,
   seq(floor(min(MEANINGFULNESS, na.rm=TRUE)),
   ceiling(max(MEANINGFULNESS, na.rm=TRUE)), length.out=8)),
   FREQUENCY=floor(min(FREQUENCY)):ceiling(max(FREQUENCY)))¶
> preds.hyp[c("PREDICTIONS", "LOWER", "UPPER")]<-predict(
   model.01, newdata=preds.hyp, interval="confidence")¶
> preds.hyp¶
```

In fact, the coefficients mean what they always mean; cf. (61) and (62):

- the intercept is the predicted RT when both MEANINGFULNESS and FREQUENCY is 0;
- the second coefficient is what you add to the intercept to predict the RT for when the predictor mentioned increases by one unit (i.e., when MEANINGFULNESS increases from 0 to 1) and when the predictor not mentioned here stays at the level from the intercept (i.e., FREQUENCY remains 0);
- the third coefficient is what you add to the intercept to predict the RT for when FREQUENCY increases from 0 to 1 and when MEANINGFULNESS stays at the level from the intercept (i.e., remains 0);

– the fourth coefficient is for a predictor that is an interaction. Thus, to use it for a prediction, you do not just add this estimate to the intercept, but also the estimates for the main effects that are part of it. Thus, to predict the RT for when MEANINGFULNESS is 1 and FREQUENCY is 1, you add to the intercept all coefficients.

Now, in actual work you would not have added to the predictions values that are based on MEANINGFULNESS values as far away from the real values, which also affects the plotting. We therefore generate a data frame `preds.hyp.for.plot` with a huge number of predictions, namely all predictions based on all combinations of 100 MEANINGFULNESS and 100 FREQUENCY values, as shown in the code file (note the use of `seq(…, length.out=…)`. and the use of `na.rm=TRUE` to make sure that `min` and `max` don't have problems with the missing data.

Now you have several possibilities. The first two shown in the code involve something I cannot really demonstrate well in a book: The function `plot3d` generates rotatable 3-dimensional plots – you can click onto the plot and move the mouse to turn the coordinate system – and the `col` argument uses the function `grey` (see `?grey`) to make the darkness of the points dependent on the height of the predicted value. Usually, you have quite some turning of the plot to do before you can see what's happening in the data – I do recommend, however, to let the predicted values be on the vertical axis most of the time. (An alternative plot shows that you can use any color scaling you want.)

While this is very useful to interpret the data, you cannot usually publish such graphs. Thus, sometimes you can represent the predicted values not in a third dimension but using color or plotting symbols. The following plot is a scatterplot with MEANINGFULNESS and FREQUENCY on the *x*- and *y*-axis respectively, and the size of the predicted value is represented by the lightness: the lighter the grey, the slower subjects are predicted to be.

On the whole, but especially when MEANINGFULNESS is low, as FREQUENCY increases, predicted RT decreases – see how in the left half of the plot, the grey gets darker as you go up. Also on the whole, but especially when FREQUENCY is low, as MEANINGFULNESS increases, predicted RT decreases – see how in the lower half of the plot, the grey gets darker as you go to the right. However, and this is the slight hint at an interaction (and indicated by the slight bow upwards in the 3-dimensional plot), when both MEANINGFULNESS and FREQUENCY become very high, we do *not* get the fastest RTs:

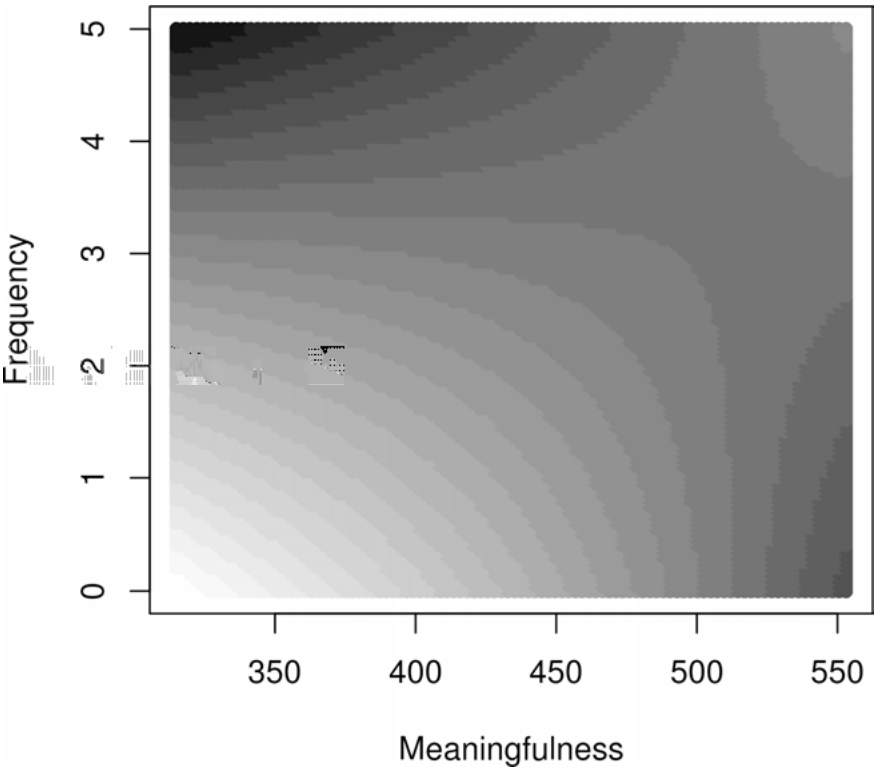

Figure 68. The interaction MEANINGFULNESS:FREQUENCY in `model.01`

In the upper right corner, the points are *not* the darkest. But, `model.01` showed that this bit of an interaction is in fact not significant, which you would also have to say in your results/discussion section.

Other graphical possibilities to play around with are exemplified in the code file including an effects plot. One of these uses numbers as plotting symbols and shows nicely how predictions change in the space spanned by MEANINGFULNESS and FREQUENCY.

2.7. A linear model selection process with multiple predictors

So far, we have ignored two things. First, for expository reasons we have ignored Occam's razor: when a predictor – a main effect or an interaction – was not significant, we left it in the model and plotted it anyway. In this section, we will look at how to do a backwards model selection process.

Second, we have ignored tests of the regression assumptions so we will also talk about this a bit at the end. The maximal model we will explore here involves all the independent variables you have seen so far and including all their interactions up till (and including) 3-way interactions; let me note in passing that this can only be a didactic example since the number of predictors is too high compared to the small number of data points:

```
> options(contrasts=c("contr.sum", "contr.poly"))¶
> model.01<-lm(RT~(FREQUENCY+FAMILIARITY+IMAGEABILITY+
   MEANINGFULNESS)^3, data=RTs[complete.cases(RTs),])¶
> summary(model.01)¶
```

Note how we define the `data` argument to make sure only complete cases are entered into the process. Also note the syntax to say that we want to include main effects, 2-way, and 3-way interactions: variables are parenthesized and then we say `^3`.

The results show an overall insignicant model with some significant but many insignificant predictors in it. Part of the reason why the overall model is significant is because the large number of (insignificant predictors) increases the degrees of freedom, which makes it harder to get a significant result; note in this connection the huge difference between multiple R^2 and adjusted R^2. Also, we have a problem that is quite common especially with naturalistic data: one cell in our design has only one observation – the combination of FAMILIARITY:*HI* and IMAGEABILITY:*LO* – which leads to NAs in the coefficients, which in turn makes the `Anova` function not work.

```
> Anova(model.01, type="III")¶
```

There are three ways to handle this. The probably best one is to use `drop1`, which, as usual, will test for all predictors that could be omitted at this stage whether their deletion would make the model significantly worse:

```
> drop1(model.01, test="F")¶
```

As you can see, just as discussed in Section 5.1.2.2, `drop1` tests only the highest-order interactions and the one with the highest *p*-value would be the best one to be deleted first: FREQUENCY:FAMILIARITY:IMAGEABILITY.

A second possibility is to add an argument to `Anova`, which provides the same result and conclusion regarding which interaction to delete first:

```
> Anova(model.01, type="III", singular.ok=TRUE)¶
```

The final possibility would be the most laborious one. It involves identifying the four interactions that could be deleted, computing four models each of which differs from `model.01` only by missing one of these interactions – that is, the smaller model is a sub-model of the larger! – and then doing a model comparison to see how much worse the smaller model is. After this is done for all four candidate interactions to be deleted, you delete the one for which the largest non-significant *p*-value was obtained.

The first of these steps, generating a sub-model, is best done with the function `update`. The first argument is the model which you want to change, followed by `~.`, followed by what you want to do, e.g. here subtract a predictor: (I only show the first two updates; you should also explore the help for `update`, which can be used in other useful ways.)

```
> model.02a<-update(model.01, ~. -
   FREQUENCY:FAMILIARITY:IMAGEABILITY)¶
> model.02b<-update(model.01, ~. -
   FREQUENCY:FAMILIARITY:MEANINGFULNESS)¶
```

Then you compare the first model with everything to these sub-models using the function `anova` (small a!): (Again I only show the first two.)

```
> anova(model.01, model.02a)¶
> anova(model.01, model.02b)¶
```

You end up with the interaction to be deleted first. To now delete that interaction you again use `update` and now define `model.02` as `model.01` without FREQUENCY:FAMILIARITY:IMAGEABILITY:

```
> model.02<-update(model.01, ~. -
   FREQUENCY:FAMILIARITY:IMAGEABILITY)¶
```

This process is now repeated as often as needed and as shown in the code file. You of course only need to run one of the alternatives shown there. One comment: `drop1` will sometimes already return *p*-values for the deletion of predictors of a lower degree of interactivity than the one you are currently checking. We will stick to the above and only go to a lower level of interactivity, or to lower-order interactions, if no higher-order interactions is left to delete; cf. the sequence in the code file.

After quite some testing, you arrive at `model.14`, which, following Occam's razor, contains only FAMILIARITY as a predictor – everything else had to be thrown out.

```
> summary(model.14)¶
                                                                [...]
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   619.049      7.145  86.636  < 2e-16 ***
FAMILIARITY1   33.201     11.619   2.858  0.00644 **
FAMILIARITY2   -5.541      8.445  -0.656  0.51512
---                                                             [...]
Multiple R-squared: 0.1711, Adjusted R-squared: 0.1343
F-statistic: 4.645 on 2 and 45 DF,  p-value: 0.01465
```

But we are not done. FAMILIARITY has three levels, but maybe we don't need all of them, something which was above suggested already by `TukeyHSD(aov(…))`. We therefore continue with model comparison – not anymore by testing to discard variables, but now variable levels. Following the logic of Crawley (2007: 563), we create two new factors, each of which conflates two adjacent levels and add them to our data frame (to make sure we test the same number of cases), and then we compute two new models, one with each conflated version of FAMILIARITY, and then we do `anova` model comparisons:

```
> FAMILIARITY.conflat1<-FAMILIARITY.conflat2<-FAMILIARITY¶
> levels(FAMILIARITY.conflat1)<-c("lo", "med-hi", "med-hi")¶
> levels(FAMILIARITY.conflat2)<-c("lo-med", "lo-med", "hi")¶
> RTs<-cbind(RTs, FAMILIARITY.conflat1=FAMILIARITY.conflat1,
   FAMILIARITY.conflat2=FAMILIARITY.conflat2)¶
```

```
> model.15a<-lm(RT~FAMILIARITY.conflat1, data=
   RTs[complete.cases(RTs),])¶
> model.15b<-lm(RT~FAMILIARITY.conflat2, data=
   RTs[complete.cases(RTs),])¶
> anova(model.14, model.15a)¶
> anova(model.14, model.15b)¶
```

The results show that the first conflation – the one that also had the higher *p*-value in the TukeyHSD test – does not make the model significantly worse whereas the second one does. So, now Figure 69 is how the final model can be summarized (see the code and `?plotmath` for how the main heading can feature italics, superscripts, etc.):

Let me at this point briefly interrupt the discussion of this model and return to a more general point. In this case, we only have a significant main effect, and in the sections above we discussed how to plot interactions between two variables. Sometimes, users then raise the question, "ok, but I have a significant interaction of three variables – how do I plot that one?"

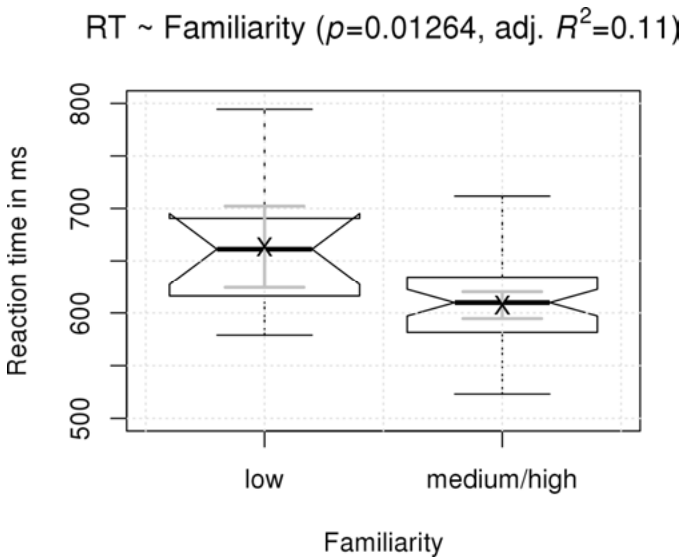

Figure 69. The final only significant effect from `model.15`

The usual answer is that you should plot these on the basis of the above plots. For example, if you have an interaction of three categorical variables X, Y and Z, then you do plots for X:Y as discussed in Section 5.2.4 for each level of Z (and ideally you try out different configurations to see which graph is easiest to interpret). For example, imagine an interaction of two categorical variables X and Y and one numeric variable Z. In that case, you might plot X:Z as discussed in Section 5.2.5 for every level of Y (or Y:Z for every level of X), etc. That is, you just take the plots discussed above and use them as building blocks for higher-order interactions.

A related and very important question is how to get something like `preds.hyp` for a predictor X in a model when X is not the only predictor left in the model. In such scenarios, the approach with `preds.hyp` from above is not ideal when another predictor in the model, say Y, has levels whose frequencies differ wildly, which often happens with observational data. For example, `model.10` in the model selection process involves the following formula:

```
> formula(model.10)¶
RT ~ FREQUENCY + FAMILIARITY + IMAGEABILITY + MEANINGFULNESS
   + IMAGEABILITY:MEANINGFULNESS
```

If you want to extract the predicted values for FAMILIARITY, then you

can use the function `effect` to create a list called, say, `fam`:

```
> fam<-effect("FAMILIARITY", model.10); fam¶
 FAMILIARITY effect
FAMILIARITY
      lo      med       hi
653.7191 613.6080 606.0162
```

While this output is exactly what you would need, getting these numbers out of there (maybe even with confidence intervals) is not as easy as it seems. You have to know that

- the predictor variables we created with `expand.grid` are in `fam$x`;
- the predicted values are now in `fam$fit`;
- the lower bounds of the confidence interval are in `fam$lower`;
- the upper bounds of the confidence interval are in `fam$upper`.

How do you then use this to create something like `preds.hyp` for the interaction? Check out the code file to see how it's done.

Back to `model.15`. The final thing to be done before you explain the model selection process you have done and summarize the results is to check the model assumptions. This can be done in many ways but two practical ones are the following. First, you can inspect some model-diagnostic graphs; second, you can use the function `gvlma` from the package with the same name to get a quick overview.

```
> par(mfrow=c(2, 2))¶
> plot(model.15)¶
> par(mfrow=c(1, 1))¶
```

The two left graphs test the assumptions that the variances of the residuals are constant. Both show the ratio of the fitted/predicted values on the *x*-axis to kinds of residuals on the *y*-axis. Ideally, both graphs would show a scattercloud without much structure; here we have only two fitted values (one for each level of `FAMILIARITY.conflat1`), but no structure such that the dispersion of the values increases or decreases from left to right: here, these graphs look ok.[35] Several words are marked as potential outliers. Also, the plot on the top left shows that the residuals are distributed well around the desired mean of 0.

35. You can also use `ncvTest` from the library `car`: `ncvTest(model.15)¶`, which returns the desired non-significant result.

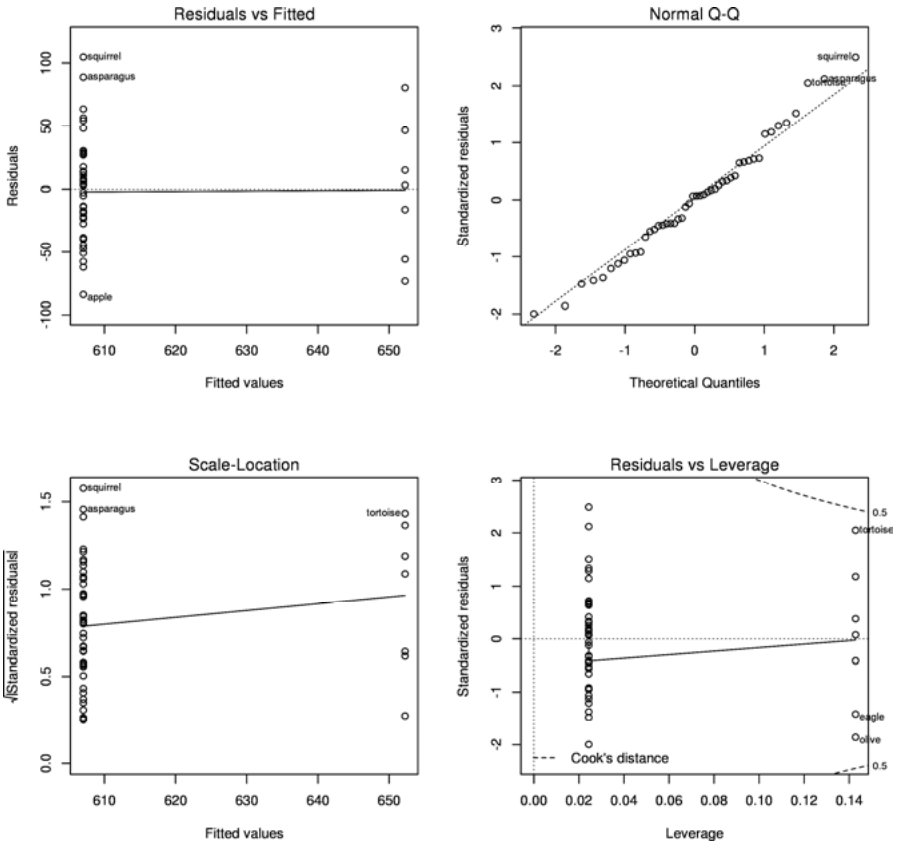

Figure 70. Model diagnostics for `model.15`

The assumption that the residuals are distributed normally also seems met: The points in the top right graph should be rather close to the dashed line, which they are; again, three words are marked as potential outliers. But you can of course also do a Shapiro-Wilk test on the residuals, which also yields the result hoped for.

Finally, the bottom right plot plots the standardized residuals against the so-called leverage. Leverage is a measure of how much a data point may influence a model (because it is far away from the center of the relevant independent variable). As you can see, there are a few words with a larger leverage, and these are all cases of FAMILIARITY:*LO*, which in this toy data set is a much smaller number of data points. Let me briefly also show one example of model-diagnostic plots pointing to violations of the model assumptions. Figure 71 below shows the upper two model plots I once found

when exploring the data of a student who had been advised (by a stats consultant!) to apply an ANOVA-like linear model to her data. In the left panel, you can clearly see how the range of residuals increases from left to right. In the right panel, you can see how strongly the points deviate from the dashed line especially in the upper right part of the coordinate system. Such plots are a clear warning (and the function `gvlma` mentioned above showed that four out of five tested assumptions were violated!). One possible follow-up would be to see whether one can justifiably ignore the outliers indicated; see Fox and Weisberg (2011: Chapter 6) for discussion.

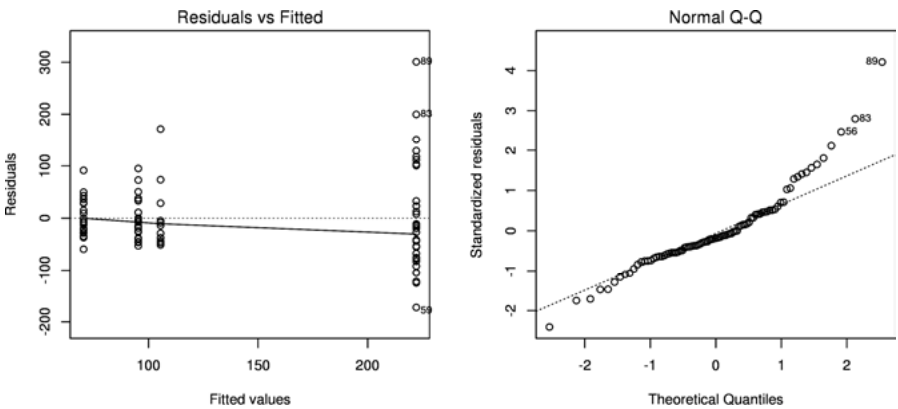

Figure 71. Problematic model diagnostics

Recommendation(s) for further study

- for model selection:
  - the function `step`, to have R perform model selection automatically based on *AIC* (by default) and the function `stepAIC` (from the library `MASS`). Warning: automatic model selection processes can be dangerous: different algorithms can result in very different results
- on model diagnostics:
  - the functions `residualPlots` and `marginalModelPlots` and `vif` from the library `car`, the former two as alternatives to `plot(model)`, the latter to test for collinearity, which is very important to explore; it is the threat posed by highly intercorrelated predictor variables; cf. Faraway (2005: Sections 5.3 and 9.3, Fox and Weisberg 2011: Chapter 6)
  - the functions `influence.measures` and other functions mentioned in this help file (esp. `dfbeta`) to identify leverage points and outliers
- the functions `oneway.test` and `kruskal.test` as alternatives to mono-

factorial ANOVAs
- the libraries `robust`, `MASS`, and `nls` for robust as well as nonlinear regressions; cf. esp. Crawley (2002, 2005) and Faraway (2005, 2006)
- the function `rpart` from the library `rpart`, and the function `ctree` from the library `party`, to compute classification and regression trees as alternatives to (generalized) linear models
- Harrell (2001), Crawley (2002: Ch. 13-15, 17), Faraway (2005: Ch. 14), Crawley (2007: Ch. 10-12, 14, 16), Gelman and Hill (2007: Ch. 3-4), Baayen (2008: 4.4, Ch. 6-7), Johnson (2008: Section 2.4, 3.2, Ch. 4), Zuur et al. (2009: Ch. 2-4, 6-7), Fox and Weisberg (2011), Baguley (2012: Ch. 5, 12-15)

3. Binary logistic regression models

In the last section, we dealt with linear methods, in which the dependent variable is interval-/ratio-scaled and covers a wide range of values. However, in many situations the dependent variable is binary, categorical, or numeric but maybe only ≥ 0 and/or discrete (as for frequencies) or … Since the 'normal' linear model discussed above predicts values between $-\infty$ and $+\infty$, it will predict values that do not make much sense for such dependent variables – what would a predicted value of -3.65 mean when you try to predict frequencies of something? For situations like these, other models are used, some falling under the heading of *generalized linear models*, leading to types of regression such as:

- binary logistic regression for binary dependent variables;
- ordinal logistic regression and multinomial regression for ordinal and categorical dependent variables respectively;
- Poisson/count regression for frequencies as dependent variables.

To be able to apply a linear modeling approach to such data, the dependent variable is transformed with a so-called link function, which transforms the predicted range of values of a linear model ($-\infty$ to $+\infty$) to a range more appropriate for the dependent variable. For binary logistic regression, for example, the inverse logit transformation in (63a) transforms values from the range of $-\infty$ to $+\infty$ to into values ranging from 0 to 1, which can then be interpreted as probabilities of a predicted event. For Poisson regression, the exponential transformation in (64a) transforms values from the range of $-\infty$ to $+\infty$ to into values ranging from 0 to $+\infty$.; the functions in

(63b) and (64b) transform in the opposite direction:

(63) a. inverse logit of x: $\frac{1}{1+e^{x}}$

b logit of x: $\log\frac{x}{1-x}$

(64) a. exponential function of x: e^{x}

b. logarithmic function of x: $\log_{natural}\, x$

Thus, there is good news and bad news … The bad news is that binary logistic regression is not easy to understand, because of how the link function transforms the dependent variable and because, as you will see, there are three different ways in which one can report results of such a regression, which makes it difficult to understand how textbooks or papers explain methods/results. The good news is that, once you have abstracted away from the link function, everything else is pretty much the same as above, and in this section we can work with R's default treatment contrasts all the time – no need for two types of contrasts.

The data set we will explore involves the question how a main and a subordinate clause in a sentence are ordered. It involves these variables:

- a dependent binary variable, namely ORDER: *MC-SC* vs. *SC-MC* indicating whether or not the main clause precedes the subordinate clause;
- an independent binary variable SUBORDTYPE: *CAUS* vs. *TEMP* indicating whether the subordinate clause is a causal or a temporal one;
- two independent numeric variables LENGTHMC and LENTHSC, representing the number of words of the main and the subordinate clause;
- an independent numeric variable LENGTHDIFF, which represents the difference main clause length minus subordinate clause length; that is, negative values indicate the main clause is shorter;
- a categorical independent variable CONJ, which represents the conjunction used in the subordinate clause. Since these data are from the study involving parallel corpus data, these are the levels: *ALS/WHEN*, *BEVOR/BEFORE*, *NACHDEM/AFTER*, and *WEIL/BECAUSE*;
- an independent binary variable MORETHAN2CL: *NO* vs. *YES* indicating whether or not there is more than just this main and subordinate clause in the sentence. This can be understood as a question of whether the sentence involves more complexity than just these two clauses.

A binary logistic regression involves the following procedure:

Procedure
- Formulating the hypotheses
- Loading the data, preparing them for modeling, and exploring them
- Computing, selecting, and interpreting a logistic regression model
  - obtaining *p*-values for all predictors and for the model as a whole
  - interpreting the regression coefficients/estimates on the basis of (i) predicted values and (ii) plots of predicted probabilities
- Testing the main assumption(s) of the test:
  - independence of data points and residuals, no overly influential data points, no multicollinearity, and no overdispersion
  - fewer than 95% of the model'2 absolute standardized residuals > 2
  - few if any of the absolute dfbetas of any case and predictor > 1

First, the hypotheses:

H_0: There is no correlation between ORDER and the predictors (independent variables and their interactions): Nagelkerke's $R^2 = 0$.

H_1: There is a correlation between ORDER and the predictors (independent variables and their interactions): Nagelkerke's $R^2 > 0$.

Then you load the data from <_inputfiles/05-3_clauseorders.csv>:

```
> CLAUSE.ORDERS<-read.delim(file=file.choose())¶
> summary(CLAUSE.ORDERS); attach(CLAUSE.ORDERS)¶
```

In this case, no further preparation of the data will be undertaken, which is why the data frame has already been attached. However, we do want to write two helper functions (and define `error.bar` again as above), load a few packages, and make sure we're using treatment contrasts:

```
> logit<-function(x) { log(x/(1-x)) }¶
> ilogit<-function(x) { 1/(1+exp(-x)) }¶
> options(contrasts=c("contr.treatment", "contr.poly"))¶
```

Exploration of the data with cross-tabulations and spineplots of variables against ORDER does not raise any red flags so let's go ahead. In this section, I will show relatively little code/plots in the book so do follow along with the code file!

3.1. A logistic regression with a binary predictor

In this section, we will consider whether ORDER is correlated with SUBORDTYPE. As before, this first section will be longer than the ones that follow to lay the groundwork for the more complex things later.

Just like a linear model with one binary predictor reduces to a simpler test we already know – the *t*-test – so does a logistic regression with a binary predictor relate to a simpler test: the chi-squared test, since we really just have two binary variables (as in Section 4.1.2.2):

```
> orders<-table(SUBORDTYPE, ORDER); orders¶
           ORDER
SUBORDTYPE mc-sc sc-mc
      caus   184    15
      temp    91   113
> chi.orders<-chisq.test(orders, correct=FALSE); chi.orders¶
 Pearson's Chi-squared test
data:  orders
X-squared = 106.4365, df = 1, p-value < 2.2e-16
```

Two brief comments: First, remember the notions of odds and odds ratios from Section 4.1.2.2. Here's how from this table you would compute the odds of *MC-SC* first with causal, then with temporal subordinate clauses: Plus, we also said that you can compute an odds ratio from that and that sometimes you will see a logged odds ratio

```
> (184/199) / (15/199)¶
[1] 12.26667
> (91/204) / (113/204)¶
[1] 0.8053097
> 12.26667/0.8053097¶
[1] 15.23224
> log(15.23224)¶
[1] 2.723414
```

Finally, you can of course express the fact that, obviously, causal subordinate clauses prefer to follow the main clause whereas temporal subordinate clauses prefer to precede the main clause with percentages: 92.46% of all causal subordinate clauses, but only 44.61% of the temporal subordinate clauses, follow the main clause. In other words, we have three different but of course related ways to talk about this result – odds, log odds, and percentages/probabilities – something I will come back to in a moment.

The second comment has to do with an alternative to χ^2. Logistic regression does not use a χ^2-value as computed in a χ^2-test but a so-called likeli-

hood ratio test that results in a *G*-value. *G* is also χ^2-distributed and, in a logistic regression involving only one binary/categorical variable, can be computed in a way that is similar to χ^2; cf. (65) and also the code file for a little demonstration showing how similar χ^2 and *G* are.

$$G = 2 \cdot \sum_{i=1}^{n} observed \cdot \log \frac{observed}{expected} \tag{65}$$

```
> 2*sum(orders*log(orders/chi.orders$expected))¶
[1] 116.9747
```

With all this in mind, let us now run a logistic regression. The main function is `glm`, for generalized linear model, and it takes a formula and a data argument as before, but now also an argument that allows R to infer you want to use a link function for binary logistic regression (I am simplifying a bit). Also as before, the coefficients the regression will return are estimates so we immediately request confidence intervals with `confint`:

```
> model.01<-glm(ORDER~SUBORDTYPE, data=CLAUSE.ORDERS,
   family=binomial)¶
> summary(model.01)¶
                                                              […]
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.2706  -0.3959  -0.3959   1.0870   2.2739

Coefficients:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)     -2.5069     0.2685  -9.336   <2e-16 ***
SUBORDTYPEtemp   2.7234     0.3032   8.982   <2e-16 ***
                                                              […]
    Null deviance: 503.80  on 402  degrees of freedom
Residual deviance: 386.82  on 401  degrees of freedom
AIC: 390.82
                                                              […]
> confint(model.01)¶
                   2.5 %    97.5 %
(Intercept)    -3.076455 -2.016328
SUBORDTYPEtemp  2.156967  3.352559
```

Similar to `lm` output, but also different. For example, you do not get an overall *p*-value. However, you can infer that the model is significant from the fact that the only predictor is significant (and that its confidence interval does not include 0). Also, at the bottom you find the so-called null deviance – informally speaking, the amount of overall variability in the data –

and the residual deviance – informally speaking, the amount of variability left in the data after the predictor has taken care of some of the variability – and the difference between the two is G. As mentioned above, G is χ^2-distributed with *df* as the difference between the *df*s of the deviances, i.e. 1. Thus, the model's overall p-value can be computed as follows:

```
> pchisq(503.80-386.82, 402-401, lower.tail=FALSE)¶
[1] 2.899771e-27
```

Again, we find summary statistics regarding the residuals at the top, and again, before we discuss the estimates, we run code that would help us to get p-values for predictors with more than one *df*:

```
> drop1(model.01, test="LR")¶
Single term deletions
Model:
ORDER ~ SUBORDTYPE
            Df Deviance    AIC    LRT  Pr(>Chi)
<none>           386.82 390.82
SUBORDTYPE  1   503.80 505.80 116.97 < 2.2e-16 ***
---                                                    […]
> anova(model.01, glm(ORDER~1, family=binomial), test="LR")¶
Analysis of Deviance Table
Model 1: ORDER ~ SUBORDTYPE
Model 2: ORDER ~ 1
  Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
1       401     386.82
2       402     503.80 -1  -116.97 < 2.2e-16 ***          […]
```

The only change for `drop1` is that we now don't do an F-test but the likelihood ratio test (with `LR`). The line with `anova` does the same kind of model comparison: it compares `model.01` against a minimal model where `ORDER` is only regressed onto an overall intercept (`1`) and returns the same likelihood ratio test. We can also use `Anova` again, we just need to switch to sum contrasts just for this one test, and again we get the familiar result:

```
> options(contrasts=c("contr.sum", "contr.poly"))¶
> Anova(model.01, type="III", test.statistic="LR")¶
Analysis of Deviance Table (Type III tests)
Response: ORDER
           LR Chisq Df Pr(>Chisq)
SUBORDTYPE   116.97  1  < 2.2e-16 ***                    […]
> options(contrasts=c("contr.treatment", "contr.poly"))¶
```

As before, in the code file I provide some extra code to get p-values for predictors/coefficients using `wald.test` and `glht`.

Now, finally, on to the coefficients and one last time we generate `preds.hyp` in the same way as with linear models. However, for generalized linear models, `predict` does unfortunately not return confidence intervals for the predicted values.

```
> preds.hyp<-expand.grid(SUBORDTYPE=levels(SUBORDTYPE));
   preds.hyp["PREDICTIONS"]<-predict(model.01, newdata=
   preds.hyp); preds.hyp¶
  SUBORDTYPE PREDICTIONS
1       caus  -2.5068856
2       temp   0.2165283
```

Now what does that mean? Obviously, this is neither an ordering choice nor 0 vs. 1 choice … To understand what these values mean, you have to (i) recollect the three different ways we talked about the data above: odds and odds ratios, log odds, and probabilities, and (ii) you have to realize that these predicted values are log odds for the predicted ordering, and by default R predicts the second level of the dependent variable, i.e. here *SC*-*MC*. Once you know that, you can use the above to also realize how the three ways to consider these data are related, which is represented in Figure 72.

This graph represents the three perspectives on the results next to each other and it represents the possible numerical ranges of the three ways on the *y*-axes: odds range from 0 to +∞, log odds from -∞ to +∞, and probabilities from 0 to 1. Each of these perspectives expresses preference, dispreference, and lack of effect in different ranges. In numerical odds space, no preference is 1, in log odds space it's 0, and for predicted probabilities it's of course 0.5 (since we have two options). For odds, preferences are reflected by odds greater than 1, by positive log odds, and by predicted probabilies of > 0.5, and the opposites reflect dispreferences.

Now, if the predictions above are log odds, we can transform them to help us recognize what they mean. Let me show `orders` again first.

```
> orders¶
             ORDER
SUBORDTYPE mc-sc sc-mc
      caus   184    15
      temp    91   113
```

If `preds.hyp` contains log odds, anti-logging/exponentiating them should give us odds (cf. again Figure 72 and the code file), and it does.

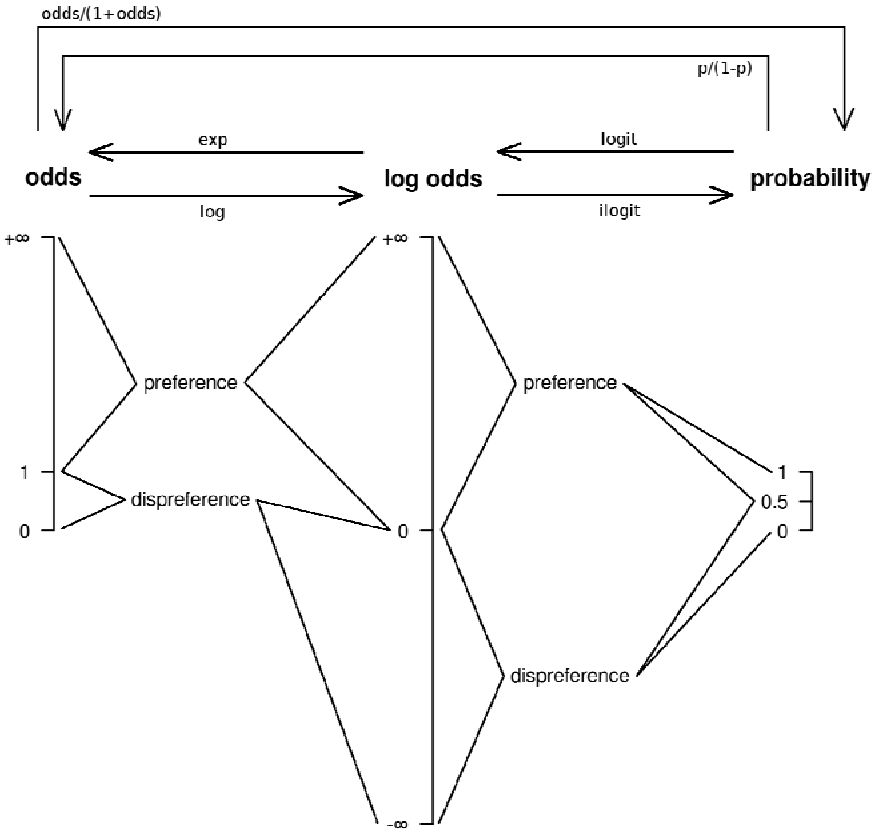

Figure 72. Three ways to look at results from a binary logistic regression

```
> exp(-2.5068856) # odds for SC-MC when SUBORDTYPE=="caus"¶
[1] 0.08152174
> exp(0.2165283) # odds for SC-MC when SUBORDTYPE=="temp"¶
[1] 1.241758
```

This also means, dividing the two gets us an odds ratio, which you also get from anti-logging the coefficient:

```
> exp(-2.5068856)/exp(0.2165283) # 1/odds ratio from above¶
[1] 0.06565025
> exp(0.2165283)/exp(-2.5068856) # odds ratio from above¶
[1] 15.23223
> exp(2.7234)¶
[1] 15.23202
```

Similarly, if `preds.hyp` contains log odds, applying `ilogit` should give us probabilities (cf. again Figure 72 and the code file), and it does:

```
> ilogit(-2.5068856) # prob. of sc-mc when SUBORDTYPE="caus"¶
[1] 0.07537688
> ilogit(0.2165283) # prob. of sc-mc when SUBORDTYPE="temp"¶
[1] 0.5539216
```

Finally, you can also get to these probabilities from the odds using the equation shown in Figure 72 and in (37) on p. 186:

```
> 0.08152174/(1+0.08152174)¶
[1] 0.07537689
> 1.241758/(1+1.241758)¶
[1] 0.5539215
```

A great part of what can be so confusing about logistic regression for beginners is that authors of papers or textbooks can and do use any one of these three perspectives: they are all right, but without something like Figure 72 it's hard to see how these map onto each other. The natural question now is, which of the three scales is best. As usual, people disagree, but I will tell you which one I am using in my own work and also here.

I myself don't like the odds scale on the left. The fact that the numerical space to express preference (of, say *SC*-*MC*) is from 1 to $+\infty$, but that the corresponding dispreferences are 'squeezed' into the range from 0 to 1 and the multiplicative nature of this scale make me disprefer it strongly. The log odds scale has attractive properties: it is additive and the numerical spaces for preference and dispreference are equally large and symmetric around 0. What I still do not like about the scale is that it is a scale of something as utterly unintuitive as log odds. Thus, I prefer the probability scale. I can think in terms of probabilities, and the numerical spaces for preference and dispreference are equally large and symmetric around 0.5. It may now seem that probabilities do not come with a disadvantage – but they do, which you will learn about in Section 5.3.3 (see p. 306f.), but I still prefer them. Thus, it is the rightmost scale that we will work with and plot here.

Let us get back to the predictions and re-work this example in a simpler way with probabilities and add confidence intervals, too. First, we generate a version of `preds.hyp` as above, but this time we immediately use the more powerful approach of the `effects` package: we generate an object with all the results for the relevant effect (`sot`) and extract all relevant info from it – the levels of the only predictor, the predicted values, and the confidence limits – and apply `ilogit` to the numeric results:

```
> sot<-effect("SUBORDTYPE", model.01)¶
> preds.hyp<-data.frame(sot$x, PREDICTIONS=ilogit(sot$fit),
   LOWER=ilogit(sot$lower), UPPER=ilogit(sot$upper))¶
> preds.hyp¶
  SUBORDTYPE PREDICTIONS     LOWER     UPPER
1       caus  0.07537688 0.0459497 0.1212547
2       temp  0.55392157 0.4851215 0.6207159
```

Again, these are the predicted probabilities of the second level of the dependent variable. Thus, the model predicts a low probability of *SC*-*MC* when the subordinate clause is causal and a much higher one when it is temporal. Since we have two options, it is only natural to make 0.5 the cutoff-point (as in Figure 72) and say when the predicted probability of *SC*-*MC* is < 0.5, then the model predicts *MC*-*SC* – otherwise the model predicts *SC*-*MC*. We can use this to determine how well the model is at predicting the ordering choices. We first generate a vector that contains a predicted probability of *SC*-*MC* for every data point in our data using `fitted`. Then we use `ifelse` to let R decide for each predicted probability which ordering it predicts. And then we tabulate the choices predicted by the model with the actual choices and compute how often the two were the same:[36]

```
> predictions.num<-fitted(model.01)¶
> predictions.cat<-ifelse(predictions.num>=0.5,
   "sc-mc", "mc-sc")¶
> table(ORDER, predictions.cat)¶
       predictions.cat
ORDER   mc-sc sc-mc
  mc-sc   184    91
  sc-mc    15   113
> (184+113)/length(predictions.cat)¶
[1] 0.7369727
```

Is that good? What do you compare that to?

THINK
BREAK

Unlike what you might think, you should not compare it to a chance accuracy of 0.5 (because you have two orderings). Why? Because the two orderings are not equally frequent. The more frequent ordering, *MC*-*SC*, accounts for 68.24% of all data, so just by always guessing that, you al-

36. Harrell (2001:248) cautions against using classification accuracy as a way to measure how good a model is. We will use a better measure in a moment.

ready get much more than 50% right. From that perspective (see the code file for another one), the present result is not great: SUBORDTYPE only improves our accuracy by about 5%.

Let us now also visualize the results. I present two graphs here, nicer versions of which you will see when you run the code in the code file. The left panel of Figure 73 shows a bar plot of the predicted probabilities of *SC-MC*; the right panel shows a line plot of those probabilities.

```
> barplot(preds.hyp$PREDICTIONS, ylim=c(0, 1),
   names.arg=preds.hyp$SUBORDTYPE)¶
> plot(sot, ylim=c(0, 1), rescale.axis=FALSE)¶
```

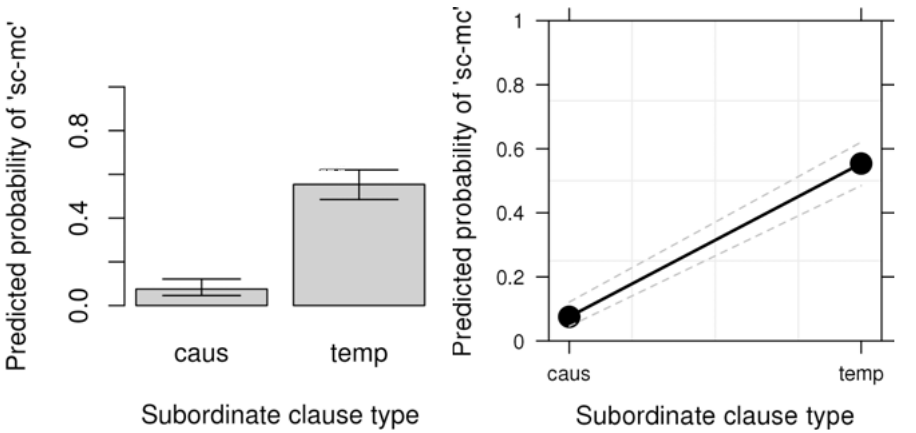

Figure 73. The effects of `model.01`: barplot with observed/predicted probabilities and their 95% confidence-interval bars (left panel); effects plot from the library `effects` (right panel)

Finally, I want to introduce a very useful tool for all sorts of regression modeling, the package `rms` and its function `lrm` (for logistic regression modeling). To use all that `lrm` has to offer, it is useful to first run the first line of code below so that functions from `rms` can access basic information about the ranges of variables etc. The second line uses the function `lrm` to fit the model with the formula from `model.01`. You do not have to specify `family`, but for many follow-up applications (later, when you become more proficient) several other arguments may be provided as shown:

```
> dd<-datadist(CLAUSE.ORDERS); options(datadist="dd")¶
> model.01.lrm<-lrm(formula(model.01), data=CLAUSE.ORDERS,
   x=TRUE, y=TRUE, linear.predictors=TRUE, se.fit=TRUE);
   model.01.lrm¶
```

```
                                                                    […]
                      Model Likelihood     Discrimination
                         Ratio Test           Indexes
Obs            403    LR chi2     116.97    R2        0.353
 mc-sc         275    d.f.             1    g         1.365
 sc-mc         128    Pr(> chi2) <0.0001    gr        3.915
max |deriv| 2e-09                          gp        0.240
                                           Brier     0.159

Rank Discrim.
   Indexes
C       0.776
Dxy     0.552
gamma   0.877
tau-a   0.240
Brier   0.159                                                       […]
```

I am not showing all the output but some advantages of `lrm` should be clear: You get the significance test of the model (see the likelihood ratio test), you get an R^2-value (often given as Nagelkerke's R^2 and, as usual, ranging from 0 to 1), and you get a *C*-value, which can be used as an indicator of the classification quality of the model. This value ranges from 0.5 to 1 and values above 0.8 are considered good, which we don't quite achieve here. (Note in passing, $C = 0.5+(D_{xy}/2)$.

To sum up: "A binary logistic regression shows there is a highly significant but weak correlation betwween the type of subordinate clause and the order of main and subordinate clause (G = 116.97; df = 1; $p < 0.001$; Nagelkerke's $R^2 = 0.353$, $C = 0.776$); 73.7% of the orderings are classified correctly (against a chance accuracy of 68.24%). The model predicts that causal subordinate clauses prefer to follow main clauses whereas temporal ones prefer to preceed main clauses. [add a graph and maybe coefficients]"

3.2. A logistic regression with a categorical predictor

As before, we will build up the complexity of the regression models in a stepwise fashion. We therefore now turn to a categorical predictor, CONJ. Again, I will show much less output from now on. The model is significant, with a likelihood ratio value of 123.32 at df =3 (see above). Since FAMILIARITY has more than 1 df, you use `drop1` (or other functions, see the code file) to get one p-value, and FAMILIARITY is highly significant.

```
> model.01<-glm(ORDER~CONJ, data=CLAUSE.ORDERS,
   family=binomial)¶
> summary(model.01)¶
```

```
> confint(model.01)¶
> drop1(model.01, test="LR")¶
```

To explore what the coefficients reveal, you turn to the predictions. These show that *als/when* and *nachdem/after* prefer *SC-MC*, whereas *bevor/before* and *weil/because* prefer *MC-SC*. I will not explain the meaning of the intercept and all coefficients in detail here but you will find all these explanations in the code file and should read them carefully! The logic and everything else is the same as explained above with (61) and (62), just that the coefficients now represent differences between the intercept – the first level of FAMILIARITY – and the other levels on the log odds scale.

```
> conj<-effect("CONJ", model.01)¶
> preds.hyp<-data.frame(conj$x, PREDICTIONS=ilogit(conj$fit),
   LOWER=ilogit(conj$lower), UPPER=ilogit(conj$upper))¶
> preds.hyp¶
           CONJ PREDICTIONS     LOWER     UPPER
1      als/when  0.60215054 0.4997995 0.6962850
2  bevor/before  0.39130435 0.2623180 0.5375019
3 nachdem/after  0.60000000 0.4773236 0.7112984
4  weil/because  0.07537688 0.0459497 0.1212547
```

Then we see how well the model classifies the orderings. We get an improvement over chance, but 76.18% does not seem like a huge step ahead.

```
> predictions.num<-fitted(model.01)¶
> predictions.cat<-ifelse(predictions.num>=0.5,
   "sc-mc", "mc-sc")¶
> table(ORDER, predictions.cat)¶
> (212+95)/length(predictions.cat)¶
```

Finally, we represent the data visually as before and generate a model with `lrm` to get R^2 and C, which are 0.369 and 0.798 respectively. With these plots and summary statistics, we can now summarize the result of our regression model in the same way as above on p. 304. Note that, given the overlap of the temporal conjunctions, one should strictly speaking also test whether the fine resolution of three temporal conjunctions is warranted …

```
> barplot(preds.hyp$PREDICTIONS, ylim=c(0, 1),
   names.arg=preds.hyp$CONJ)¶
> plot(allEffects(model.01), ask=FALSE, ylim=c(0, 1),
   rescale.axis=FALSE)¶
> dd<-datadist(CLAUSE.ORDERS); options(datadist="dd")¶
> model.01.lrm<-lrm(formula(model.01), data=CLAUSE.ORDERS);
   model.01.lrm¶
```

3.3. A logistic regression with a numeric predictor

As the final monofactorial logistic regression, we will now turn to a numeric predictor, LENGTH_DIFF.

```
> model.01<-glm(ORDER~LENGTH_DIFF, data=CLAUSE.ORDERS,
   family=binomial)¶
> summary(model.01)¶
> confint(model.01)¶
```

For the sake of consistency you also run `drop1` (and maybe `Anova`/`anova`) although LENGTH_DIFF has 1 *df* so it's not really necessary.

```
> drop1(model.01, test="LR")¶
```

Note there is a slight difference between the *p*-values from the `summary` output on the one hand and `drop1`, `anova`, and `Anova` on ther other hand; according to Fox and Weisberg (2011:239), the likelihood ratio test you get with `drop1` etc. may be more reliable.

The model is significant but the effect really seems quite weak. To understand the coefficients, we create the predictions, but this time around we have to discuss this in more detail, since it is here that a slight disadvantage of the probability perspective on logistic regression results manifests itself. In one sense at least, things are as before: the intercept still represents the probability of the predicted ordering when the independent variable is at its first level or, as here, 0, which you can see in `preds.hyp` and compare that to `ilogit(-0.77673)¶`:

```
> lendiff<-effect("LENGTH_DIFF", model.01,
   xlevels=list(LENGTH_DIFF=-max(abs(range(LENGTH_DIFF))):
   max(abs(range(LENGTH_DIFF))))); lendiff¶
> preds.hyp<-data.frame(lendiff$x, PREDICTIONS=
   ilogit(lendiff$fit), LOWER=ilogit(lendiff$lower),
   UPPER=ilogit(lendiff$upper)); preds.hyp¶
```

Similarly, the coefficient of LENGTH_DIFF still represents the change of the probability of the predicted order, *SC*-*MC*, for a unit change of LENGTH_DIFF. However, this change is linear/constant only on the log odds scale – once we check it on the probability scale, you can see that a change of 1 of LENGTH_DIFF does not bring about the same difference in probabilities: when you increase LENGTH_DIFF by 1

- from -20 to -19, this results in the predicted probability of *SC-MC* growing by 0.006019;
- from -10 to -9, this results in the predicted probability of *SC-MC* growing by 0.0078747;
- from 0 to 1, this results in the predicted probability of *SC-MC* growing by 0.0096109.

This is because the inverse logit transformation is not linear (in probability space). But let us now see how well this model predicts the orderings:

```
> predictions.num<-fitted(model.01)¶
> predictions.cat<-ifelse(predictions.num>=0.5,
   "sc-mc", "mc-sc")¶
> table(ORDER, predictions.cat)¶
> (272+2)/length(predictions.cat)¶
```

In some sense, the performance is abysmal: it is worse than chance even though the direction of the effect makes sense – you should make sure you recognize that it amounts to 'short before long'? Also, you can see from the table that the model hardly ever predicts *SC-MC* – only five times. To visualize the effect of LENGTH_DIFF and explore this bad performance, let us plot the predicted probabilities against length differences from `preds.hyp` (cf. Figure 74).

```
> plot(preds.hyp$LENGTH_DIFF, preds.hyp$PREDICTIONS,
   xlim=c(-35, 35), ylim=c(0, 1))¶
```

As you can see, in probability space you do not get a straight regression line but a curve. Thus, the change of LENGTH_DIFF by one word has different effects depending on where it happens and this is the disadvantage of the probability scale I alluded to earlier. However, given how we can nicely plot such curves in R, this is a disadvantage I am happy to live with (compared to those of the odds or log odds scales).

Figure 74 also helps understand the bad classification accuracy. The horizontal line at the cut-off point of $y = 0.5$ only applied to very few points (see the rugs). One way to try to force the regression to make somewhat more diverse predictions is to choose a cut-off point other than 0.5, and one possibility is to use the median of all predicted probabilities (0.3150244; cf. Hilbe 2009: Section 7.2.2 for discussion).

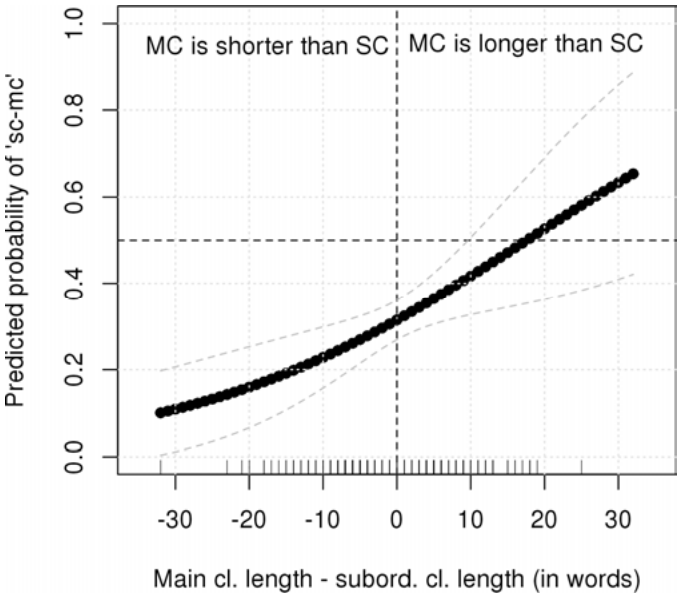

Figure 74. The effects of `model.01`: scatterplot with predicted probabilities and their 95% confidence-interval band

If you do that here, you do get more balanced frequencies of categorical predictions, but the accuracy decreases even further. The code file shows how, when you choose 0.42 as a cut-off point, you get a slightly more balanced frequency of predictions and still about 68% right; for now, we will use this value and I leave the topic of ROC curves and how they help identifying cut-off points for your future exploration Note, however, that if you do not choose the 'default' cut-off point of 0.5 for the categorical predictions, you should mention which one you chose and why. Finally, we do the regression again with `lrm` to get the really small R^2 (0.026) and C (0.603), and then we can summarize our results as above.

```
> dd<-datadist(CLAUSE.ORDERS); options(datadist="dd")¶
> model.01.lrm<-lrm(formula(model.01), data=CLAUSE.ORDERS);
   model.01.lrm¶
```

3.4. A logistic regression with two categorical predictors

We now move on to the first logistic regression with more than one predictor: we will explore whether CONJ and MORETHAN2CL and their interac-

tion affect the clause orders. Since one of the predictors has more than 1 *df*, we fit the model and immediately add `drop1` and `Anova`:

```
> model.01<-glm(ORDER~CONJ*MORETHAN2CL, data=CLAUSE.ORDERS,
   family=binomial)¶
> summary(model.01)¶
> confint(model.01)¶
> drop1(model.01, test="LR")¶
> options(contrasts=c("contr.sum", "contr.poly"))¶
> Anova(model.01, type="III", test.statistic="LR")¶
> options(contrasts=c("contr.treatment", "contr.poly"))¶
```

You can see that `drop1` now only returns the *p*-value for the non-significant interaction, which, in a normal model selection process, we would now omit. We leave it in here to explore how to understand and visualize the interaction – with `Anova`, however, we also get all other *p*-values; only CONJ seems significant.

On to the predictions using code I only show in the code file because there I have more space to explain both the code and the meanings of the coefficients, which as usual follow the rules in (61) and (62)). If you look at `preds.hyp`, *weil/because* sticks out, but the output also shows why the interaction is not significant: the confidence intervals are huge and, on the whole, the conjunctions seem to pattern alike across both levels of MORETHAN2CL. How good are these predictions?

```
> predictions.num<-fitted(model.01)¶
> predictions.cat<-ifelse(predictions.num>=0.5,
   "sc-mc", "mc-sc")¶
> table(ORDER, predictions.cat)¶
> (216+93)/length(predictions.cat)¶
```

We get nearly 77.7% right, which is at least above chance again. Finally, we represent the data visually as before and generate a model with `lrm` for the overall model test (likelihood ratio χ^2=132.06, *df*=7, $p < 0.001$), R^2 (0.392), and *C* (0.82).

```
> dd<-datadist(CLAUSE.ORDERS); options(datadist="dd")¶
> model.01.lrm<-lrm(formula(model.01), data=CLAUSE.ORDERS);
   model.01.lrm¶
```

Since the data now involve an interaction, the code can become a bit more involved (at least when you do not use the functions from the effects package) and I show it only in the code file. The non-significant interaction is reflected by the large overlap of the confidence intervals and the similar

(differences of) values of the predicted probabilities, which is, with everything else, what you would discuss in your results section.

3.5. A logistic regression with a categorical and a numeric predictor

As in Section 5.2.5, we now turn to a case with an interaction between a categorical and a numeric predictor, which means again that the interaction coefficients will reflect adjustments to the slope of the numeric predictor.

```
> model.01<-glm(ORDER~CONJ*LENGTH_DIFF, data=CLAUSE.ORDERS,
   family=binomial)¶
> summary(model.01)¶
> confint(model.01)¶
> drop1(model.01, test="LR")¶
> options(contrasts=c("contr.sum", "contr.poly"))¶
> Anova(model.01, type="III", test.statistic="LR")¶
> options(contrasts=c("contr.treatment", "contr.poly"))¶
```

This time, both main effects are significant, and the interaction is only just about not significant. Therefore, it would be warranted to at least look at the interaction (as we will for didactic purposes anyway).

We generate the predictions the usual way and it is again prudent to maybe also generate a slightly flatter table that can be inspected before we turn to plots, as shown in the code file:

```
> intact<-effect("CONJ:LENGTH_DIFF", model.01,
   xlevels=list(LENGTH_DIFF=seq(-32, 32, length.out=9)))¶
> preds.hyp<-data.frame(intact$x, PREDICTIONS=
   ilogit(intact$fit), LOWER=ilogit(intact$lower),
   UPPER=ilogit(intact$upper)); preds.hyp¶
```

Especially the flatter representation of `preds.hyp.2` is now easier to read. You can see for each conjunction how the predicted probability of *SC-MC* changes as LENGTH_DIFF changes. A plot will make this even more obvious in a moment. Again, read the code file in detail to understand how the coefficients result in these predictions! How good are the predictions?

```
> predictions.num<-predict(model.01, type="response")¶
> predictions.cat<-ifelse(predictions.num>=0.5,
   "sc-mc", "mc-sc")¶
> table(predictions.cat, ORDER)¶
> (228+83)/length(predictions.cat)¶
```

Approximately 77.2% are classified correctly, and a cut-off point of 0.57 results in a slightly better value (of about 78.2%). But now, what do the results amount to? In this case, where the *p*-values of all effects are at least < 0.07, we will do some nice plotting, which will be partly based on `preds.hyp.2`, as it has separate columns for the conjunctions. Figure 75 is what we want to create (in the lower panel, the letters are the first letters of the German conjunctions). Remember, though: the interaction is significant so that is what you should focus on – not the main effects! The code file shows how exactly this is done; I know it's a lot of lines, but you should invest the time to see what every line is doing because, once you get it, you will be able to use this logic for many examples in your own work.

The first main effect is somewhat familiar from above. The three temporal conjunctions prefer *SC*-*MC* whereas *weil/because* strongly prefers *MC*-*SC*. The second main effect is also familiar: short-before-long. Now the interaction is interesting. Three observations can be made:

- *als/when* and *nachdem/after* exhibit similar average preferences for *SC*-*MC* and both react to LENGTH_DIFF in a way that is (more) compatible with short-before-long than the average;
- *bevor/before* not only has less of a preference for *SC*-*MC* but also the opposite tendency compared to the other two temporal conjunctions: when the main clause becomes longer, it want to precede the subordinate clause;
- *weil/because* subordinate clauses are pretty much completely immune to considerations of length: the want to come second no matter what.

(The code file also contains code for the interaction with confidence intervals, but those make the plot harder to read, defeating its purpose; the effects plot is more useful in that regard.) All the above, together with the *p*-values for the predictors, the *p*-value for the overall model (likelihood ratio χ^2=135.21, *df*=7, $p < 0.001$), R^2 (0.399) and *C* (0.818) from the corresponding model with `lrm` (see below) would be part of your results section.

3.6. A logistic regression with two numeric predictors

The final logistic regression, as before with two numeric predictors. We are going to check whether the lengths of the two clauses affect the ordering choice. First a question: how is this different from checking whether LENGTH_DIFF is significant?

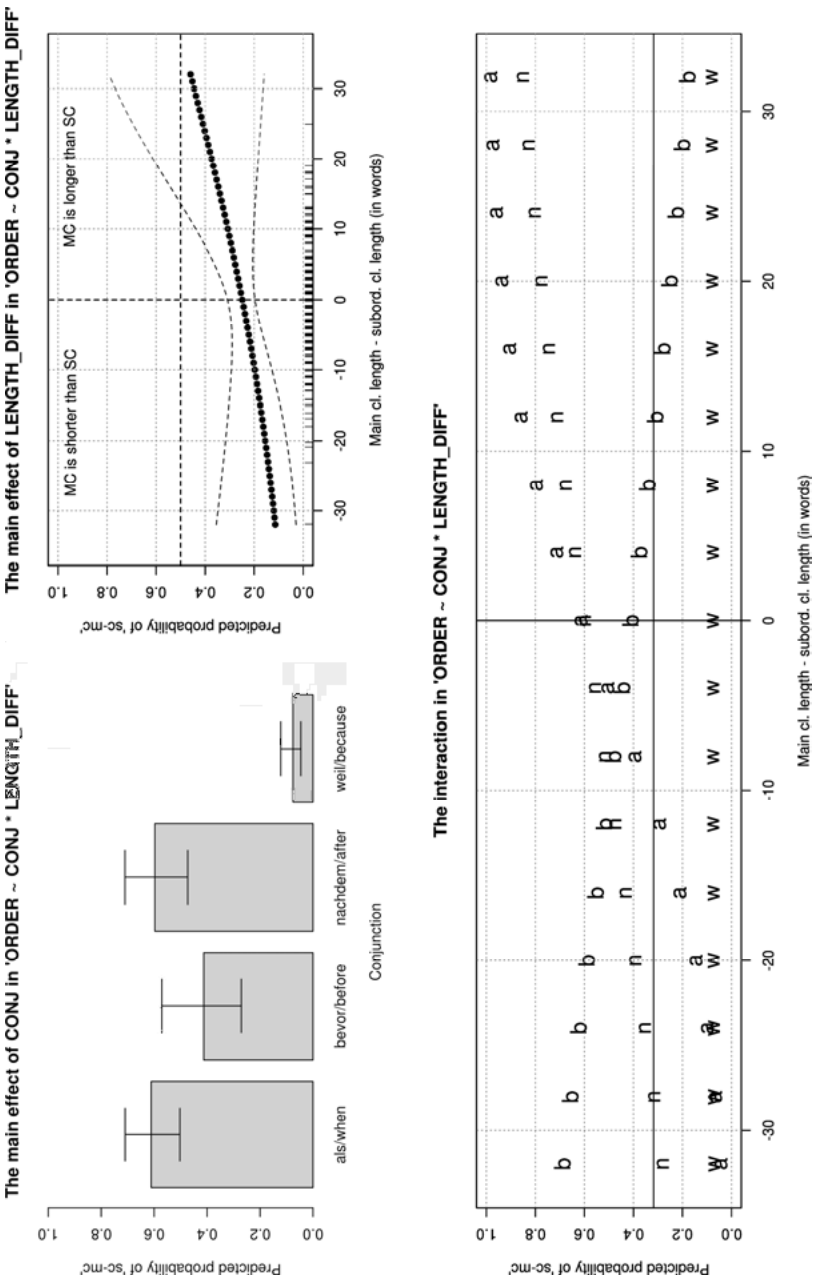

Figure 75. All effects in `model.01` (remember, however: if the interaction is significant, your discussion should focus on it, not the main effects)

THINK BREAK

It is different because it takes into account where a particular length difference may be observed: If LENGTH_DIFF is 1, that value does not reveal whether it arises from the main and the subordinate clause containing 10 and 9 or 20 and 19 words respectively. As usual, we fit the model; since both variables are numeric, `drop1` and `Anova` are not really necessary:

```
> model.01<-glm(ORDER~LEN_MC*LEN_SC, data=CLAUSE.ORDERS,
   family=binomial)¶
> summary(model.01)¶
> confint(model.01)¶
```

This is all as non-significant as it gets; it is only for didactic reasons that we proceed with the predictions, and with two numeric predictors spanning a wide range of values, both `preds.hyp` and the flatter `preds.hyp.2` (see the code file) are not easy to process. (Don't forget to read the code file's explanation of the coefficients.) Next, we check the cross-classification table, immediately using the median of the predicted probabilities as the cut-off point:

```
> predictions.num<-predict(model.01, type="response")¶
> predictions.cat<-ifelse(predictions.num>=
   median(predictions.num), "sc-mc", "mc-sc")¶
> table(predictions.cat, ORDER)¶
> (151+82)/length(predictions.cat)¶
```

The accuracy is quite low, even beyond chance, which is not surprising given the *p*-values of the predictors. But what do the effects look like? As before, you have basically two possibilities. First, you can again generate a 3-dimensional rotatable plot (with `plot3d`), and the code file shows an, I think, nice version where different colors and different letters (the first letter of the first clause) represent which ordering is predicted for which combination of lengths. The fact that the interaction is insignificant is reflected by the fact that the letters nearly form a straight plane in 3-dimensional space. The more publication-ready version is shown in Figure 76.

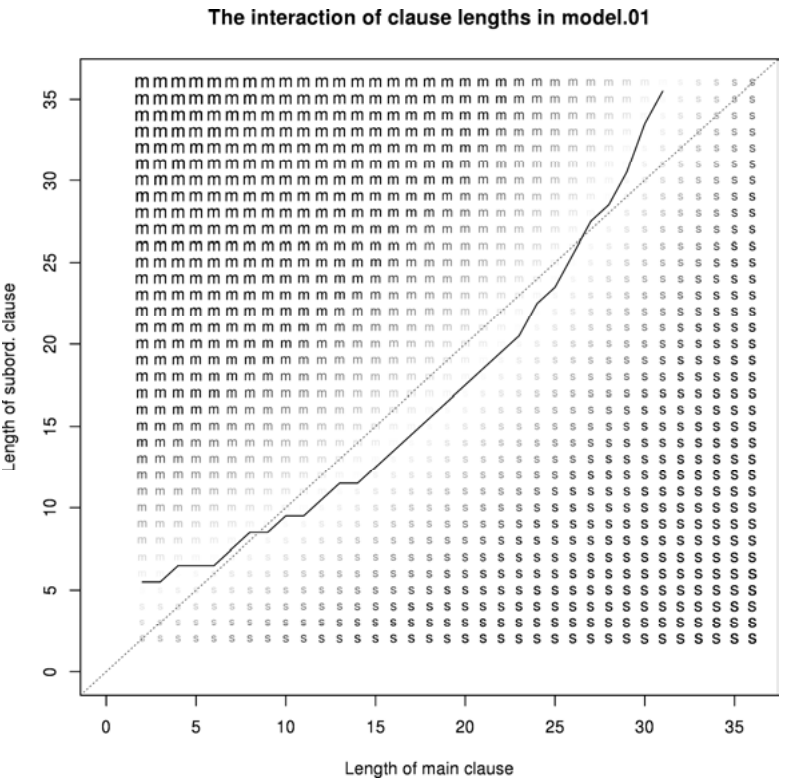

Figure 76. The interaction LENGTHMC:LENTHSC in `model.01`

LENGTHMC and LENGTHSC are on the *x*- and *y*-axis respectively. When the categorical prediction is *MC*-*SC*, I plot an *m* otherwise an *s*. The larger and the darker the letter, the more 'certain' is the model about the prediction in the sense that the prediction is based on a probability further away from the cut-off point. The straight grey line is the main diagonal where both clauses are equally long, and the curved black line indicates for every main clause length where the prediction flips to the other clause order, which is why there the letters are so light. While the interaction is not significant, we see short-before-long again: when the main clause is short (the left of the plot), then as the subordinate clause becomes longer, it wants to go in the hind position, and the same the other way round for subordinate clauses. The effects plots show the same kind of result, but by splitting up the subordinate clause lengths into ten different ranges and then plotting regression lines and their confidence intervals. While easier to generate in terms of code, I find that graph less easy to interpret than Figure 76.

Finally, you would generate the model with `lrm` and sum it all up; here of course, all predictors are weak and non-significant and R^2 as well as *C* are really low (0.028 and 0.606 respectively).

We have now completed the overview of the different logistic regression models. Again, as mentioned above on p. 263, you would of course not have done all these models separately, but a model selection process like the one in Section 5.2.7. One of the exercises for this chapter will have you do this for the present data, and you will find that the results shed some more light on the unexpected behavior of *bevor/before* in Section 5.3.5.

What remains do be covered, however, is again how to test whether the assumptions of the regression model are met. Above I mentioned three different criteria (but see Fox and Weisberg (2011: Ch. 6)). You already know about inspecting residuals but overdispersion is new. It requires that you look at the ratio of your model's residual deviance and its residual *df*s, which should not be much larger than 1. In this case, it is $^{495.58}/_{399}$=1.242. Several references just say that, if you get a value that is much larger than 1, e.g. > 2, then you would run the `glm` analysis again with the argument `family=quasibinomial` and take it from there. Baayen (2008: 199) uses as an approximation a chi-square test of the residual deviance at the residual *df*:

```
> pchisq(495.58, 399, lower.tail=FALSE)¶
[1] 0.0006880771
```

Thus, if this was a real analysis with a significant result, one might want to follow that advice. The other criteria I mentioned were concerned with the absolute values of the standardized residuals of the model and of the dfbetas. The former are a type of corrected residuals (see Fox and Weisberg 2011: 286f.) and Field, Miles, and Field (2012: Section 8.6.7.3) suggest that no more than 5% should be > 2 or < -2. This is easy to test:

```
> prop.table(table(abs(rstandard(model.01))>2))¶
     FALSE       TRUE
0.99751861 0.00248139
```

In this case not even 1% is > 2 or < -2. Similarly straightforward is the test of the dfbetas, which reflect how much a regression coefficient changes when each case is removed from the data. Again, testing this in R is simple:

```
> summary(dfbetas<-abs(dfbeta(model.01)))¶
```

The output (not shown here) indicates that in fact no absolute dfbeta is greater than 0.1 so this criterion also poses no problems to our model. Checking diagnostics carefully is an important component of model checking and R in general, and the library `car` in particular, has many useful functions for this purpose.

Recommendation(s) for further study

- just like in Section 5.2, it can also help interpreting the regression coefficients when the input variables are centered
- the function `hoslem.test` from the library `ResourceSelection` for the Hosmer-Lemeshow test (see Hilbe 2009: Section 7.2) (you want to see a non-significant result)
- Field, Miles, and Field (2012: Section 8.8.2) on the assumption of the linearity of the logit
- Pampel (2000), Jaccard (2001), Crawley 2005: Ch. 16), Crawley (2007: Ch. 17), Faraway (2006: Ch. 2, 6) , Zuur, Ieno, and Smith (2007: Section 6.1), Gelman and Hill (2007: Ch. 5), Baayen (2008: Section 6.3), Baguley (2012: Ch. 17)

4. Other regression models

The above two types of regression models have been the most widely-used ones in linguistics. In this section, I will introduce a variety of other regression models that are not that widespread yet, but which are bound to become used more in the near future: *ordinal logistic regression* (where the dependent variable is ordinal), *multinomial regression* (where the dependent variable is categorical with 3+ levels), and *Poisson regression* (where the dependent variable consists of frequencies). The logic of the exposition will be as above, but – for reasons of space – much abbreviated. Specifically, after a short introduction to each section and its data, I will only discuss one example for each regression in the book, namely the case of two independent variables, one categorical and one numeric. However, the code file will discuss six regression models for each, just like before, so that you get a nice homogeneous treatment of all models. I therefore recommend that you load the data, read the chapter in the book, and follow along with the fifth of the six examples in the code file, and then explore the other examples based on the code file as well.

4.1. An ordinal logistic regression with a categorical and a numeric predictor

The example we will explore to approach ordinal logistic regression is concerned with which of a set of independent variables allows us to predict which of three different end-of-term exams or assignments foreign-language learners of English will choose. It involves these variables:

- a dependent ordinal variable, namely ASSIGNMENT: *ORALEXAM* vs. *LABREPORT* vs. *THESIS*; crucially, these are ordered in ascending order of difficulty (based on a previous study);
- an independent binary variable SEX: *FEMALE* vs. *MALE* indicating the sex of the student whose choice has been recorded;
- a categorical independent variable REGION, which represents the geographic region where the student comes from: *CENTRAL-EUROPEAN*, *HISPANIC*, and *MIDDLE-EASTERN*;
- an independent numeric variable WORKHOURS, representing the numbers of hours/month the students claimed to have invested into the class;
- an independent numeric variable MISTAKES, which represents the numbers of mistakes the students made in their last assignment for this class.

Here are the steps of an ordinal logistic regression that we will follow:

Procedure
- Formulating the hypotheses
- Loading the data, preparing them for modeling, and exploring them
- Computing, selecting, and interpreting a logistic regression model
  - obtaining *p*-values for all predictors and for the model as a whole
  - interpreting the regression coefficients/estimates on the basis of (i) predicted values and (ii) plots of predicted probabilities
- Testing the main assumption(s) of the test:
  - the usual suspects: independence of data points and residuals, no overly influential data points, no multicollinearity
  - the dependent variable "behaves in an ordinal fashion with respect to each predictor" (Harrell 2001:332)

First, the hypotheses:

H_0: There is no correlation between ASSIGNMENT and the predictors

(independent variables and their interactions): $R^2 = 0$.

H_1: There is a correlation between ASSIGNMENT and the predictors (independent variables and their interactions): $R^2 > 0$.

Then you load the data from <_inputfiles/05-4-1_assignments.csv> as well as the library `rms`, whose function `lrm` we will use here:

```
> rm(list=ls(all=TRUE)); library(rms)¶
> ASSIGNS<-read.delim(file=file.choose()); str(ASSIGNS)¶
```

If you inspect the summary provided by `str`, you will see that the levels of the factor `ASSIGNMENT` are not in the right order, and that that factor is not even an ordered factor, which means R treats it as a categorical variable (as all factors in this book so far), not as the desired ordinal variable. Thus, we change this (check `str` again), and then we can `attach` and, since we will use the `lrm` function again, create the required `datadist` object:

```
> ASSIGNS$ASSIGNMENT<-factor(ASSIGNS$ASSIGNMENT, ordered=
   TRUE, levels=levels(ASSIGNS$ASSIGNMENT)[c(2,1,3)])¶
> str(ASSIGNS); attach(ASSIGNS)¶
> ddist<-datadist(ASSIGNS); options(datadist="ddist")¶
```

As mentioned before, I will now skip the first four models discussed in the code file and go directly to the fifth one, where we explore the joint influence of REGION and WORKHOURS on ASSIGNMENT:

```
> model.01<-lrm(ASSIGNMENT ~ REGION*WORKHOURS, data=ASSIGNS,
   x=TRUE, y=TRUE, linear.predictors=TRUE, se.fit=TRUE)¶
> model.01¶
> anova.rms(model.01)¶
```

The model is highly significant (Likelihood ratio χ^2=493.09, df=5, p < 0.001) and shows there is a very strong correlation: R^2=0.908, C=0.938. The `anova.rms` output is a bit different. Rather than giving you a p-value for each main effect and each interaction (as `Anova` from the libray `car` did), you get two p-values for what each main effect does alone together with what it does in the interaction, and you get a p-value for the interaction. Since the interaction is nearly significant, we will focus on that. But what is its nature? The coefficients are now quite different from what we have seen before: there is more than one intercept. I explain the meanings of each coefficient in detail in the code file, but the easiest way to understand the results is again via predicted probabilities, which we will generate

using the same logic but slightly different code (the `effect` function does not work with `lrm` objects but you can sometimes use `polr` from the package MASS). Here are the first two lines:

```
> preds.hyp<-expand.grid(REGION=levels(REGION),
   WORKHOURS=c(0, 1, floor(min(WORKHOURS)):
   ceiling(max(WORKHOURS))))¶
> preds.hyp<-data.frame(preds.hyp, predict(model.01,
   newdata=preds.hyp, type="fitted.ind"))¶
```

This generates a data frame `preds.hyp` again, which contains for each combination of `REGION` and a large number of values of `WORKHOURS` the predicted probability of each kind of assignment. For example, when the student is from the Hispanic region and puts in 26 workhours, he is strongly predicted to choose the lab report:

```
> preds.hyp[preds.hyp$REGION=="hispanic" & preds.hyp$
   WORKHOURS==26,]¶
```

But we want it even nicer: we do not just want the predicted probabilities, but immediately also for each row what the categorical prediction is. As you can see, the `predict` function combined the name of the dependent variable, `ASSIGNMENT`, with the predicted levels by inserting a period between them. We do not want to see that so we use the following:

```
> preds.hyp<-data.frame(preds.hyp, ASSIGNMENT.pred=
   sub("^.*?\\.", "", names(preds.hyp)[-(1:2)][
   max.col(preds.hyp[,-(1:2)])]))¶
> preds.hyp[38:42,]¶
```

The function `sub` takes three arguments: what to look for (and the argument `".*\\."` means 'characters up to and including a period'), what to replace it with (and `""` means 'nothing', i.e., 'delete'), and where to do all this (in the three column names of `preds.hyp` that are not the first two). And then, these levels are subset with the vector of numbers that results from R checking for each row where the maximal predicted probability is (always excluding the first two columns of `preds.hyp`, which contain the independent variables!). Verify this by looking at these five lines of output.

We now first remove the first rows of `preds.hyp` because these were only included to explain the coefficients but were unrepresentative of the real values of `WORKHOURS`. Then, we check the classification accuracy:

```
> preds.hyp<-preds.hyp[-(1:6),]¶
> predictions.num<-predict(model.01, type="fitted.ind")¶
> predictions.cat<-sub("^.*?[\\.=]", "", colnames(
   predictions.num)[max.col(predictions.num)])¶
> table(predictions.cat, ASSIGNMENT)¶
```

As you can see, we achieve a good accuracy, nearly 85%, which is highly significantly better than the chance level of 33% (since the three assignments are equally frequent). Then we plot the predicted probabilities, which, given the multitude of results these types of regressions yield, becomes a bit more involved. One way to represent these results is shown in Figure 77. There is one panel for each region, the workhours are on each *x*-axis, the predicted probabilities on each *y*-axis, and the three assignments are represented by lines and their first letters. On the whole, there is a very strong effect of WORKHOURS: students who self-reported lower workhours are strongly predicted to choose the easiest exam/assignment, the oral exam. Those who report an intermediate number of workhours are strongly predicted to use the intermediately difficult exam/assignment, the lab report, and those who report the largest numbers of workhours are predicted to go with the thesis. The nearly significant interaction, however, indicates that this behavior is not completely uniform across the three regions: For example, the Hispanic students choose the more difficult exams/assignments with smaller numbers of workhours than the Middle Easterners. The Central Europeans stick more to the oral exams even if they work a number of hours where the other students have already begun to prefer the lab report, and only the most industrious Middle Easterners choose the thesis. (See the code file for other graphs.)

Let us finally check some assumptions of this type of regression: The first five plots represent the residuals and those are mostly quite close to 0, as required. The ordinality assumption looks a bit more problematic, though so this requires some more attention, which is beyond the scope of this book; see the recommendations for further study.

```
> par(mfrow=c(2, 4))¶
> residuals(model.01, type="score.binary", pl=TRUE)¶
> plot.xmean.ordinaly(ASSIGNMENT ~ REGION*WORKHOURS)¶
> par(mfrow=c(1, 1))¶
```

Leaving this issue aside for now, you would now be able to summarize the regression results numerically (Likelihood ratio χ^2, df, p, R^2, C) and discuss the graph and its implications along the lines discussed above.

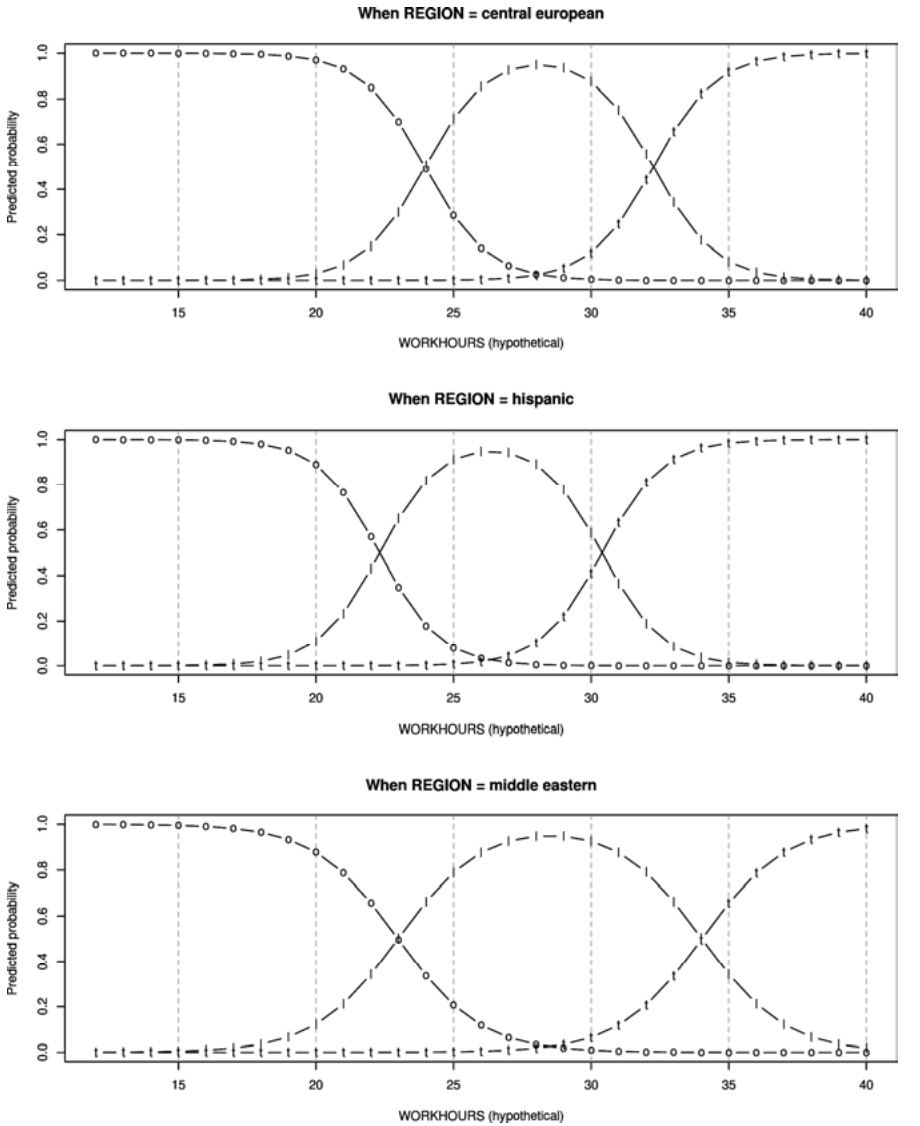

Figure 77. The interaction REGION:WORKHOURS in `model.01`

Recommendation(s) for further study

- the function `polr` from the library `MASS`, for ordinal logistic regressions
- Harrell (2001: Ch. 13-14), Baayen (2008: Section 6.3.2), Hilbe (2009: Ch. 10), Agresti (2010), Fox and Weisberg (2011: Section 5.9)

4.2. A multinomial regression with a categorical and a numeric predictor

After having discussed ordinal logistic regression, we now turn to multinomial regression. For the sake of simplicity, we will use the same data set and just *not* consider ASSIGNMENT an ordinal variable (and hence an ordered factor) but a categorical variable and hence a 'regular' unordered factor. This is the procedure we will follow:

Procedure
- Formulating the hypotheses
- Loading the data, preparing them for modeling, and exploring them
- Computing, selecting, and interpreting a multinomial regression model
  - obtaining *p*-values for all predictors and for the model as a whole
  - interpreting the regression coefficients/estimates on the basis of (i) predicted values and (ii) plots of predicted probabilities
- Testing the main assumption(s) of the test:
  - the usual suspects: independence of data points and residuals, no overly influential data points, no multicollinearity
  - independence of irrelevant alternatives, a non-significant Hasuman-McFadden test (which I will not discuss, see the references below)

Given that we're using the same data set, the hypotheses stay the same, too, plus you can load the file, change the levels of ASSIGNMENT as above, (without changing it to an ordered factor though), and we load a number of libraries. Then we fit a multinomial regression model as follows:

```
> model.01<-multinom(ASSIGNMENT ~ REGION*WORKHOURS,
   data=ASSIGNS)¶
> summary(model.01, Wald=TRUE)¶
> mlogit.display(model.01)¶
> confint(model.01)¶
```

The output of `summary` is a bit overwhelming because we get again get multiple intercepts and coefficients for all but the first level of the dependent variable. These represent in a somewhat complicated way the differences between the first level of the dependent variable and each of the others; in a way, multinomial regressions are series of binary logistic regressions. We also get Wald statistics, which are, as usual, the coefficients divided by their standard errors.

Let us check the significance of the predictors. We can unfortunately

not use `drop1`, but we can do something that is pretty much equivalent to it: an anova comparison of `model.01` to a model without the interaction, plus we can use `Anova` in the by now familiar way. Both reveal that the interaction is not significant at all: $p > 0.9$.

```
> anova(model.01, multinom(ASSIGNMENT ~ REGION+WORKHOURS))¶
> options(contrasts=c("contr.sum", "contr.poly"))¶
>    Anova(model.01, type="III")¶
> options(contrasts=c("contr.treatment", "contr.poly"))¶
```

Now what do the coefficients mean? I am nearly tempted to say, "you don't want to know …" The explanations of the coefficients are even more evidence why trying to understand the results in terms of coefficients is often not the best/most intuitive strategy. You will find detailed explanations of them in the code file; suffice it to say here that, when you exponentiate them, you get ratios between different predicted probabilities. One visual representation we might use is the type exemplified by Figure 77 above and the code file shows you how you can generate that graph as well as two others. On the whole, the results are comparable to those of Figure 77: low numbers of work hours lead to oral exams, intermediate ones lead to lab reports, and high ones are more associated with theses, and these preferences are, with some small differences, obtained for all regions.

To determine the classification accuracy, we could proceed the usual way, or we can take take things to the next level. Again we use function `model.statistics` from Antti Arppe's nice package `polytomous`:

```
> model.statistics(ASSIGNMENT, predictions.cat,
   predictions.num)¶
```

This provides an immensely useful set of summary statistics: Log-likelihood statistics and deviances for our `model.01` (-329.5837 and 143.4688) and for a model consisting of just the intercept (-71.73441 and 143.4688), the classification accuracy (0.8733), and, as in the excurses before, Nagelkerke R^2 (0.9233), everything one would want to know …

Recommendation(s) for further study

- Gries (2009: Section 5.1) on (hierarchical) configural frequency analysis (and the script `hcfa` with which it can be computed interactively) and Field, Miles, and Field (2012: Sections 18.7-18.12) on loglinear analysis; also see the functions `loglin` and the function `loglm` (from the library `MASS`) to compute loglinear analyses

- the function `hmftest` from the library `mlogit` to compute the Hausman-McFadden test
- Agresti (2002: Ch. 7), Faraway (2006: Ch. 5), Fox and Weisberg (2011: Section 5.7), Field, Miles, and Field (2012: Section 8.9)

4.3. A Poisson regression with a categorical and a numeric predictor

In this section, I will discuss another type of generalized linear model, namely Poisson regression, which is used to model counts/frequencies. As discussed above on p. 294, just like binary logistic regression this approach also requires a link function – this time the exponential function – to make sure that a linear-model type of approach can be applied to a dependent variable that is never negative. As in the last two sections, I will only discuss one regression with a categorical and a numeric predictor here in the book and encourage you to then explore the other five examples in the code file. The example I will use to explain Poisson regression is concerned with factors that lead to in-/decreased numbers of disfluencies in conversations of bilingual and/or highly advanced non-native speakers and involves the following variables:

- a dependent variable DISFLUENCY, which represents the numbers of disfluencies 300 speakers each produced in 20 minutes of conversation;
- an independent binary variable SEX: *FEMALE* vs. SEX: *MALE*, the speaker's sex;
- a categorical independent variable MOVEDWHEN, which indicates when the speaker moved to the U.S.A.: as an *ADULT*, during *HIGH SCHOOL*, or during *PRIMARY SCHOOL*;
- an independent numeric variable REALITYTV, representing the numbers of hours/month the speakers self-reports to watch reality TV shows;
- an independent numeric variable SOCIALNETWORK, which represents the numbers of hours/week the speakers self-reports to spend time on social networks.

Here are the steps of a Poisson regression that we will follow:

Procedure

- Formulating the hypotheses

- Loading the data, preparing them for modeling, and exploring them
- Computing, selecting, and interpreting a Poisson regression model
  - obtaining *p*-values for all predictors and for the model as a whole
  - interpreting the regression coefficients/estimates on the basis of (i) predicted values and (ii) plots of predicted probabilities
- Testing the main assumption(s) of the test:
  - the usual suspects: independence of data points and residuals, no overly influential data points, no multicollinearity
  - the model does not suffer from overdispersion

First, the hypotheses, then we load some libraries (see the code file) and also the data (from <_inputfiles/05-4-3_disfluencies.csv>).

H_0: There is no correlation between DISFLUENCY and the predictors (independent variables and their interactions): $R^2 = 0$.

H_1: There is a correlation between DISFLUENCY and the predictors (independent variables and their interactions): $R^2 > 0$.

```
> DISFL<-read.delim(file=file.choose())¶
> str(DISFL); attach(DISFL)¶
```

The model we will discuss here tests the hypothesis that the frequency of disfluencies is correlated with the point of time when the speaker moved to the U.S. and the amount of time spent on social networks:

```
> summary(model.01<-glm(FREQDISFL ~ MOVEDWHEN*SOCNETWORK,
   data=DISFL, family=poisson))¶
```

The output of this model already indicates a first problem: overdispersion. The ratio of the residual deviance (3532.6) and the residual degrees of freedom (294) is much much larger than one and significant (`pchisq(3532.6, 294, lower.tail=FALSE)`¶), which is why we fit the model again with `family=quasipoisson`, which corrects the predictors' standard errors and, thus, the *p*-values, and we compute what has been proposed as an R^2:

```
> summary(model.01<-glm(FREQDISFL ~ MOVEDWHEN*SOCNETWORK,
   data=DISFL, family=quasipoisson))¶
> 1-(model.01$deviance/model.01$null.deviance)¶
[1] 0.2558909
```

Since we're using `glm`, much of the code for logistic regressions also applies here. For example, `drop1` and `Anova` get us *p*-values for predictors. Obviously, the interaction is not significant at all, so we would normally update the model by deleting it, and obviously MOVEDWHEN does not seem to play a role whereas SOCIALNETWORK does.

```
> drop1(model.01, test="LR")¶
Single term deletions
Model:
FREQDISFL ~ MOVEDWHEN * SOCNETWORK
                    Df Deviance scaled dev. Pr(>Chi)
<none>                   3532.6
MOVEDWHEN:SOCNETWORK  2   3538.0     0.46628    0.792

> options(contrasts=c("contr.sum", "contr.poly"))¶
>    Anova(model.01, type="III", test="LR")¶
Analysis of Deviance Table (Type III tests)
Response: FREQDISFL
                    LR Chisq Df Pr(>Chisq)
MOVEDWHEN             2.7684  2     0.2505
SOCNETWORK           19.3469  1   1.09e-05 ***
MOVEDWHEN:SOCNETWORK  0.4663  2     0.7920
                                                    […]
> options(contrasts=c("contr.treatment", "contr.poly"))¶
```

For expository purposes only, we continue with the interaction. Above, we used the function `effect` to obtain predicted probabilities – here, we're using it to obtain predicted frequencies and we can really re-use a lot of what we know about using `effect` from before. The only real difference is that, above we applied `ilogit` to `effect`'s output, because the binary logistic regression uses `logit` as a link function – since the Poisson regression uses `log` as a link function, we now apply `exp`. In the code file, I again explain the meanings of the coefficients and how they give rise to the predicted frequencies in much detail. We therefore proceed to the plot. Figure 78 plots DISFLUENCY against the SOCIALNETWORK and then adds three regression lines, one for each level of MOVEDWHEN. (I omitted the confidence bands here, which clutter up the graph unless one can use colors.)

It is plain to see why the interaction is not significant. The positive correlation between DISFLUENCY and SOCIALNETWORK is the same for each level of MOVEDWHEN. That positive correlation as a main effect is significant, but then the differences between the different levels of MOVEDWHEN – the intercepts – also do not reach standard levels of significance. Thus, since here we do not remove the interaction (again, just for expository reasons), we could wrap up the results: "On the whole, there is a highly significant (L.R. χ^2=1214.8; df = 5; $p < 0.001$) [see page 298 on how to compute

this] but not particularly strong correlation (R^2= 0.26). This correlation is due to the fact that the number of hours spent on social networks is significantly positively correlated with the numbers of disfluencies produced (L.R. χ^2=19.35; df = 1; $p < 0.001$) whereas the age of moving to the U.S.A. is not ($p > 0.25$), and neither is their interaction ($p > 0.79$)."

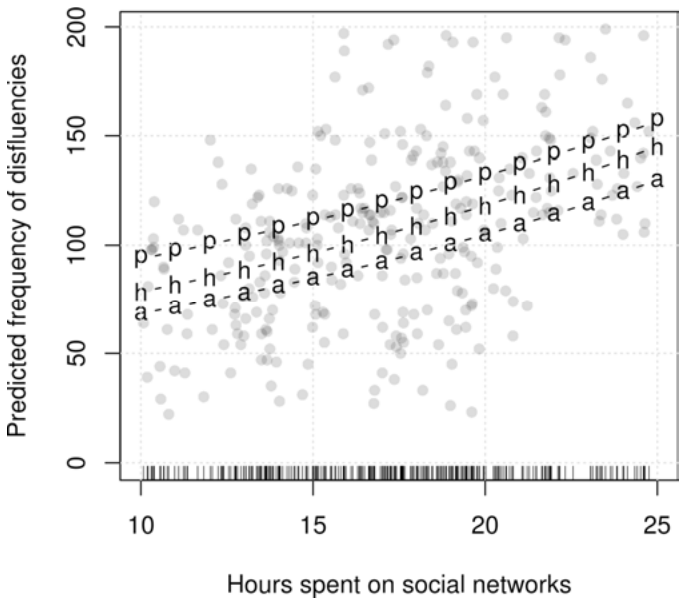

Figure 78. The interaction MOVEDWHEN:SOCIALNETWORK in `model.01`

Recommendation(s) for further study

- Gries (2009: Section 5.1) on (hierarchical) configural frequency analysis (and a script to compute this interactively) and Field, Miles, and Field (2012: Sections 18.7-18.12) on loglinear analysis; also see `loglin` and `loglm` (from the package `MASS`) for loglinear analyses
- Agresti (2002: Ch. 4, 8-9), Faraway (2006: Ch. 3-4), Zuur, Ieno, and Smith (2007: Section 6.1), Zuur et al. (2009: Ch. 8-9, 11), Hilbe (2009: Ch. 11), Fox and Weisberg (2011: Section 5.5-5.6)

5. Repeated measurements: a primer

The final section in this part on regression modeling is devoted to a type of scenario that differs from all previous ones. All models discussed so far

shared one and the same assumption, that the data points (and their residuals) are independent of each other. For instance, in the section on linear regressions, the average reaction time to each word was considered independent of the average reaction time of any of the other words. This scenario, while frequent, is not the only possible one – as you know from page 159 above, groups/samples can be dependent, which means that data points *are* related to each other. The most common ways in which data points are related to each other involve the following scenarios:

- in experimental settings, you obtain more than one response per subject (i.e., you do *repeated measurements* on each subject), which means that the characteristics of any one subject affect more than one data point;
- in experimental settings, you obtain more than one response per, say, a lexical item which you test in some stimulus, which means that the characteristics of any one lexical item affect more than one data point;
- in corpus data, you obtain more than one data point per speaker (often approximated by corpus file), which means …, you get the picture.

If your data involve such related data points but you ignore that in your statistical analysis, you run several risks. First, you run the risk of what is called "losing power", which means you may stick to H_0 although H_1 is true in the population (what is called a *type II error*, a *type I error* is to accept H_1 although H_0 is true in the popluation). Second, you risk obtaining inaccurate results because your statistical analysis doesn't take all the known structure in the data into consideration and will return – in the context of regression modeling – coefficients that are not as precise as they should be.

In this section, I will talk about methods that are used in such cases. However, this section will only be very brief because, while the methods that are used in such cases are quite important and powerful, they also involve considerable complexity and require much more space than I can devote to them here. (See below for some excellent references for follow-up study, in particular Girden (1992), which inspired some of the discussion here, and Field, Miles, and Field (2012).) Also, while the overall logic of repeated measurements applies to many different kinds and configurations of independent and dependent variables, I will only discuss cases that could be considered repeated-measures ANOVAs, i.e. cases in which the dependent variable is interval-/ratio-scaled (i.e., not categorical) and in which the independent variables involved are treated as categorical.

5.1. One independent variable nested into subjects/items

By way of introduction, I will begin my discussion here with a brief example of three different ways in which the simplest possible dependent-samples type of data can be analyzed. In Section 4.3.2.2, we dealt with such a case when we explored the question whether translations of 16 texts were longer than the originals. That scenario involved dependent samples because one could connect every original to its translation and we, therefore, computed a *t*-test for dependent samples. Let us clear memory, load the package `ez`, reload those data (now from <_inputfiles/05-5-1_textlengths.csv>) and then revisit this scenario, here for expository reasons as a two-tailed hypothesis (that the mean lengths from originals and translations differ). Also, before we attach the data frame, we convert the column `TEXT`, which simply numbers the texts, to a factor: this variable is really only categorical since the numbers do not do anything but identify which text a length belongs to – the sizes of the numbers do not matter.

```
> Texts<-read.delim(file.choose()¶
> Texts$TEXT<-factor(Texts$TEXT)¶
> str(Texts); attach(Texts)¶
```

We already know from above that the differences between the originals' and the translations' lengths are normally distributed so we immediately compute the *t*-test for dependent samples (again, here a two-tailed one) and obtain the familiar *t*- and *df*-values as well as a now only marginally significant *p*-value.

```
> t.test(LENGTH~TEXTSOURCE, paired=TRUE)¶
```

Above, we saw that a linear model with one binary predictor is essentially equivalent to a *t*-test for independent samples (recall p. 266f.). It may therefore not be a big surprise that a repeated-measures ANOVA with one binary predictor is essentially equivalent to a *t*-test for dependent samples. The two ANOVAs differ, however, in how the variability in the data is divvied up in the analysis. An independent-measures linear model with one binary or categorical predictor divides the variability in the data up into variability that is attributed to the levels of the independent variable and variability that is attributed to random variation (random noise, residual variability, or error). The effect of the independent variable is then assessed by comparing the two amounts of variability, and the more variability the independent variable accounts for compared to the residual variability, the

more likely it is the independent variable's effect will be significant.

In a repeated-measures ANOVA, the variability is divided up differently. First, there is variability between typically different subjects or here, different texts. But then there is also variability within different subjects (or here, different texts), and a part of that variability is due to the independent variable (here, TEXTSOURCE: *ORIGINAL* vs. TEXTSOURCE: *TRANSLATION*) and the remainder is random error / residual variation. Since in repeated-measures ANOVAs the effect of the independent variable is nested within subjects or, here, texts, we therefore compare the amount of within-subject/text variability that is attributed to the independent variable not to the overall remaining variability, but to the remaining amount of within-subject/text variability, and again the more within-subject variability is accounted for by the independent variable compared to residual within-subject variability, the more likely it is the result will be significant. And this is why dependent-samples / repeated-measurements studies can be more precise: the effects of independent variables are compared to a smaller amount of residual (within-subject/text) variability.

How do we do this in R? We use the function `aov` (for analysis of variance) and tell it (i) that we want a model in which `LENGTH` is modeled as a function of `TEXTSOURCE` (`LENGTH ~ TEXTSOURCE`, no surprises here) and (ii) what the relevant source of error/residual variability (`ERROR(…)`) is by stating that the independent variable `TEXTSOURCE` is nested into, i.e. repeated within, each element of `TEXT` (`TEXT/TEXTSOURCE`):

```
> model.01.aov<-aov(LENGTH ~ TEXTSOURCE +
   Error(TEXT/TEXTSOURCE))¶
> summary(model.01.aov)¶

Error: TEXT
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals 15 210479   14032

Error: TEXT:TEXTSOURCE
           Df Sum Sq Mean Sq F value Pr(>F)
TEXTSOURCE  1  51040   51040   3.717  0.073 .
Residuals  15 205991   13733
```

As discussed above, the output divides the overall variability into that between subjects/texts (i.e., the upper part labeled `Errror: TEXT`) and the one within the subjects/texts, which in turn is either due to the independent variable `TEXTSOURCE` (mean square: 51,040) or random/residual noise (mean square: 13733). The *F*-value is then the ratio of the two mean squares at levels of the independent variable minus 1 and subjects/texts

minus 1 degrees of freedom. As you can see, this result is then identical to the *t*-test: The *F*-value is t^2, the *F*-value's residual *df* are the *t*-tests *df*, the *p*-values are identical, and, obviously, so is the conclusion you would write up: With a two-tailed hypothesis, the means (`model.tables( model.01.aov, "means")¶`) do not differ from each other significantly.

A very attractive alternative way to conduct a repeated-measures ANOVA involves the very useful function `ezANOVA` from the library `ez`. The first argument (`data`) is the data frame containing the data, the second (`dv`) specifies the dependent variable, the third (`wid`) specifies the subjects/text identifier, and the fourth (`within`) defines the independent variable nested within the identifier. You get an ANOVA table with the same *F*-value, its two *df*s, the same *p*-value, and a measure of effect size in the column labeled *ges*. (Plus, explore the code with `ezPlot` and look at `?ezStats¶`.)

```
> ezANOVA(data=Texts, dv=.(LENGTH), wid=.(TEXT),
   within=.(TEXTSOURCE))¶
```

5.2. Two independent variables nested into subjects/items

How do we extend the above approach to more complex data such as cases where two variables are nested into a subject or an item? Consider a hypothetical case where five subjects are asked to provide as many synonyms as they can to eight stimuli (different for each subject, so there is no repetition of items), which result from crossing words with positive or negative connotations (a variable called MEANING) and words from four different parts of speech (a variable called POS). Let's assume we wanted to know whether the numbers of synonyms subjects named in 30 seconds differed as a function of these independent variables (and for the sake of simplicity we are treating these frequencies as interval-/ratio-scaled data). We load the data from <_inputfiles/05-5-2_synonyms.csv>:

```
> Syns<-read.delim(file.choose())¶
> str(Syns); attach(Syns)¶
```

In this case, no monofactorial test is available for comparison so we immediately do the repeated-measures ANOVA. The logic is actually not different from above: we want to study the effects of both independent variables and their interaction but both these variables are nested into `SUBJECT`. Thus, we either use `aov` …

```
> model.01.aov<-aov(SYNONYMS ~ MEANING*POS +
   Error(SUBJECT/(MEANING*POS)))
> summary(model.01.aov)¶
```

… or ezANOVA:

```
> ezANOVA(data=Syns, dv=.(SYNONYMS), wid=.(SUBJECT),
   within=.(MEANING, POS))¶
```

As before, both return the same results: The only effect reaching standard levels of significance is POS, and the output of `model.tables` shows that nouns and verbs resulted in high numbers of synonyms, whereas adjectives and adverbs only yielded medium and lower numbers respectively. The output of ezANOVA also returns a test for sphericity, a very important assumption of repeated-measures ANOVAs (see the recommendations for further study). In this case, all the *p*-values are > 0.05 so sphericity is not violated and we can rely on the results of our *F*-tests.

5.3. Two independent variables, one between, one within subjects/items

One final example, in which I show how to handle cases where you have two independent variables, but only one of them is nested into subjects – the other varies between subjects. Imagine you had 10 non-native speaker subjects, each of whom participated in four proficiency tests, or tasks: an oral exam, an in-class grammar test, an essay written in class, and an essay written at home. This is the variable nested into the subjects. However, you also suspect that the sexes of the speakers play a role and, guess what, those are not nested into subjects … This is the between-subjects variable. Let's load the data from <_inputfiles/05-5-3_mistakes.csv>.

```
> Mistakes<-read.delim(file.choose())¶
> str(Mistakes); attach(Mistakes)¶
```

It should be clear what to do: for aov, you specify the formula with all independent variables and tell it that only TASK is nested into SUBJECTS.

```
> model.01.aov<-aov(MISTAKES ~ SEX*TASK +
   Error(SUBJECT/TASK))¶
> summary(model.01.aov)¶
```

For ezANOVA, you use the argument `between` to tell the function that the

independent variables SEX does not vary within, but between subjects:

```
> ezANOVA(data=Mistakes, dv=.(MISTAKES), wid=.(SUBJECT),
   within=.(TASK), between=.(SEX))¶
```

Unfortunately, while we get significant results for both TASK and its interaction with SEX – explore the means with `model.tables` again – this time around the sphericity tests are cause for alarm. `ezANOVA` suggests two corrections for violations of sphericity, both of which still return significant values for TASK and TASK:SEX, but this is beyond the scope of this book, see the recommendations for further study below and the next section.

5.4. Mixed-effects / multi-level models

The above has already indicated that repeated-measures ANOVAs are not always as straightforward to use as the above may have made you expect. First, repeated-measures ANOVAs as discussed above only involve categorical independent variables, but you may often have interval-/ratio-scaled variables and may not want to factorize them (given the loss of information and power that may come with that, see Baayen 2010). Second, many variables you may wish to include are not *fixed effects* (i.e., variables whose levels in the study cover all possible levels in the population) but are *random effects* (i.e., variables whose levels in the study do not cover all possible levels in the population, such as SUBJECT, ITEM, …, see Gelman and Hill 2007: 245f.). Third, repeated-measures ANOVA requires a balanced design and may therefore be problematic with missing data in experiments and unbalanced observational data. Finally, violations of sphericity are not always easy and uncontroversial to address; (see Baguley 2012: Section 18.2.2 for more discussion).

A strategy to handle data with dependent/related data points and random effects that is currently very hot in linguistics is the use of mixed-effects models, or multi-level models. With much simplification, these are regression models that can handle fixed and random effects as well as repeated measurements, unbalanced data, and hierarchical/nested data. They do this by simultaneously modeling different sources of variability by, for example, instead of simply fitting one regression line over many subjects through a point cloud in a coordinate system, they allow the analyst to model the dependent variable with a different regression line for each subject or item, where the different regression lines may have, say, subject-

specific or item-specific different intercepts (called *random intercepts*, because they are modeled as a normally-distributed random variable) and/or slopes (called *random slopes*). For reasons of space and others to be discussed below, I will not discuss these highly complex models here in detail, but I want to give one or two brief examples. For the first of these, I will return to the *t*-test for dependent samples again:

```
> rm(list=ls(all=TRUE)); library(effects); library(nlme)¶
> Texts<-read.delim(file.choose()¶
> Texts$TEXT<-factor(Texts$TEXT)¶
> str(Texts); attach(Texts)¶
```

The package `nlme` (as well as the newer package `lme4`) allows you to fit a large variety of mixed-effects models. This is one way of applying these to the *t*-test data. The function for linear mixed effects is `lme`, and here it takes two arguments: First, the argument `fixed`, which defines the fixed-effects structure of the model, and our only fixed-effect independent variable is `TEXTSOURCE`. Second, the argument `random` describes the random-effects structure of the model, and the notation means we want the intercept (`1`) to be able to vary by `TEXT` (`|TEXT`), which is just another way of capturing text-specific variability as we did in the repeated-measures ANOVA.

```
> model.01.lme<-lme(fixed = LENGTH~TEXTSOURCE, random= ~
   1|TEXT)¶
> summary(model.01.lme)¶
```

The output we get contains a lot of information but we will only focus on the random-effects and the fixed-effects output. The former (in the section "Random effects") contains an estimate of the variability of the 16 random intercepts for the 16 texts, namely a standard deviation of 12.23172. The latter (in the section "Fixed effects: …") contains the familiar kind of table of coefficients, standard errors, *t*-values, and *p*-values. The *t*-value (1.92787), its *df* (15), and the *p*-value (0.073) should look very familiar, since they correspond to the above results for the same data. And you can even create the familiar kind of effects plot for this result because the function `effect` does accept `lme` models as input:

```
> plot(effect("TEXTSOURCE", model.01.lme))¶
```

Some other applications of repeated-measures ANOVAs can be explored similarly. For example, the above data on the mistakes can be studied with this function call:

```
> summary(model.01.lme<-lme(fixed = MISTAKES ~ SEX*TASK,
   random= ~ 1|SUBJECT))¶
```

You can allow intercepts to vary across subjects in the same ways as above (you can also allow slopes to vary, but I will not discuss that here), you can plot the main effects or the interactions of such models with `effect`, and you can even apply `Anova(model)` to `lme` models to get *p*-values for the fixed-effect predictors.

Seems simple, doesn't it? Why isn't there a whole section on this, explaining and exemplifying it all in detail as for the other models in this chapter. Well, unfortunately, things are very far from being that simple. In fact, mixed-effects modeling is one of the most fascinating but also among the most complex statistical techniques I have seen. Right now, it actually seems to be seen as the best thing since sliced bread, and indeed the potential of this approach is immense and far-reaching. Having said that, I must admit that I sometimes think that some of the hype about this method is a bit premature simply because so many things are still unclear. Ask any two or three experts on how to do X with multi-level models, and you often get very different responses. Pick any two to three references on mixed-effects modeling and you will see that not only is there very little agreement on some seemingly central questions, but also that some types of problems are not even mentioned very widely. For example,

- it seems we're not even close to a somewhat widely accepted view on what a *model selection process* or even just a *maximal model* would look like. Some sources recommend a model selection process where we begin with no fixed effects but first explore random effects; others recommend starting with a full-fledged fixed-effects maximal model; some recommend beginning with a simple random-effects structure (just intercepts), others recommend beginning with a maximal random-effect structure with random intercepts and slopes for everything (and then simulations suggest that these models do no converge even if they are given the right model structure) …;
- it is not clear yet how *predictors should be selected* for retention in, or deletion from a model: some use *p*-values (based on *t*- or *F*-values, but then it's debated how to choose the residual *df*s), some use MCMC sampling (which is not easily available for some types of dependent variables); some use information criteria (such as *AIC* or *BIC* or even *DIC*) for the whole process; some use likelihood ratio tests, which require attention to whether the models have been fit with ML or REML, …;

- many references do not discuss how to handle the *intercorrelations of random intercepts and slopes*;
- many references say practically nothing about how to decide on a *covariance structure of the data*; I think I have seen only one reference discussing this in a somewhat accessible fashion;
- you have seen that *centering variables* can be useful in regression modeling but how to do this best in mixed-effects models is again often not discussed well – when do we center around an overall mean, when around group means? And the list goes on, boundary effects, how to compute R^2s, …

None of the above is to deny that mixed-effects modeling is very powerful and has the potential to help us very much in analyzing our data … once the field has developed a bit more of some common thoughts on how they should be applied to the various kinds of data out there. The fact that now some journals already *require* mixed-effects modeling for particular data sets seems a bit overeager, given how many open questions remain. However, once some standards regarding the many open questions begin to emerge and once some libraries and functions are developed that make tackling some of these questions more easily (Baayen's `pvals.fnc` is one case in point), then the discipline will benefit from mixed-effects models in innumerable ways. Till that happens, here are some, I think, very good references (of varying degrees of technicality) that will hopefully get you started beyond this little primer …

Recommendation(s) for further study

- for repeated-measures ANOVAs: Girden (1992), Johnson (2008: Sections 4.3-4.4), and especially Miles, Field, and Miles (2012: Ch. 13-14)
- for mixed-effects/multi-level models: Twisk (2006), Gelman and Hill (2007: Ch. 11-15), Zuur, Ieno, and Smith (2007: Ch. 8), Baayen (2008: Ch. 7), Baayen, Davidson, and Bates (2008), Johnson (2008: Sections 7.3, 7.4), Zuur et al. (2009, in particular Ch. 5), Miles, Field, and Miles (2012: Ch. 19), Baguley (2012: Ch. 18); also see Baayen (2011)

6. Hierarchical agglomerative cluster analysis

We have so far only concerned ourselves with methods in which independent and dependent variables were clearly separated and where we already

had at least an expectation and a hypothesis prior to the data collection. Such methods are sometimes referred to as *hypothesis-testing statistics*, and we used statistics and *p*-values to decide whether or not to reject a H_0. The method called hierarchical agglomerative cluster analysis that we deal with in this section is a so-called *exploratory*, or *hypothesis-generating*, method or, more precisely, a family of methods. It is normally used to divide a set of elements into clusters, or groups, such that the members of one group are very similar to each other and at the same time very dissimilar to members of other groups. An obvious reason to use cluster analyses to this end is that this method can handle larger amounts of data and be at the same time more objective than humans eyeballing huge tables.

To get a first understanding of what cluster analyses do, let us look at a fictitious example of a cluster analysis based on similarity judgments of English consonant phonemes. Let's assume you wanted to determine how English native speakers distinguish the following consonant phonemes: /b/, /d/, /f/, /g/, /l/, /m/, /n/, /p/, /s/, /t/, and /v/. You asked 20 subjects to rate the similarities of all $^{(11 \cdot 10)}/_{2} = 55$ pairs of consonants on a scale from 0 ('completely different') to 1 ('completely identical'). As a result, you obtained 20 similarity ratings for each pair and could compute an average rating for each pair. It would now be possible to compute a cluster analysis on the basis of these average similarity judgments to determine (i) which consonants and consonant groups the subjects appear to distinguish and (ii) how these groups can perhaps be explained. Figure 79 shows the result that such a cluster analysis might produce – how would you interpret it?

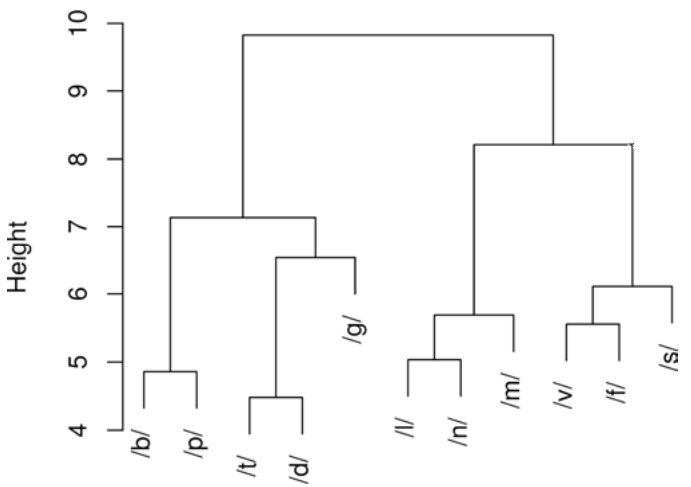

Figure 79. Fictitious results of a cluster analysis of English consonants

THINK
BREAK

The 'result' suggests that the subjects' judgments were probably strongly influenced by the consonants' manner of articulation: on a very general level, there are two clusters, one with /b/, /p/, /t/, /d/, and /g/, and one with /l/, /n/, /m/, /v/, /f/, and /s/. It is immediately obvious that the first cluster contains all and only all plosives (i.e., consonants whose production involves a momentary *complete* obstruction of the airflow) that were included whereas the second cluster contains all and only all nasals, liquids, and fricatives (i.e., consonants whose production involves only a momentary *partial* obstruction of the airflow).

There is more to the results, however. The first of these two clusters has a noteworthy internal structure of two 'subclusters'. The first subcluster, as it were, contains all and only all bilabial phonemes whereas the second subcluster groups both alveolars together followed by a velar sound.

The second of the two big clusters also has some internal structure with two subclusters. The first of these contains all and only all nasals and liquids (i.e., phonemes that are sometimes classified as between clearcut vowels and clearcut consonants), and again the phonemes with the same place of articulation are grouped together first (the two alveolar sounds). The same is true of the second subcluster, which contains all and only all fricatives and has the labiodental fricatives merged first.

The above comments were only concerned with which elements are members of which clusters. Further attempts at interpretation may focus on how many of the clusters in Figure 79 differ from each other strongly enough to be considered clusters in their own right. Such discussion is ideally based on follow-up tests which are too complex to be discussed here, but as a quick and dirty heuristic you can look at the lengths of the vertical lines in such a tree diagram, or dendrogram. Long vertical lines indicate more autonomous subclusters. For example, the subcluster {/b/ /p/} is rather different from the remaining plosives since the vertical line leading upwards from it to the merging with {{/t/ /d/} /g/} is rather long.[37]

Unfortunately, cluster analyses do not usually yield such a perfectly interpretable output but such dendrograms are often surprisingly interesting

37. For a similar but authentic example (based on data on vowel formants), cf. Kornai (1998).

and revealing. Cluster analyses are often used in semantic, cognitive-linguistic, psycholinguistic, and computational-linguistic studies (cf. Miller 1971, Sandra and Rice 1995, Rice 1996, and Manning and Schütze 1999: Ch. 14 for some examples) and are often an ideal means to detect patterns in large and seemingly noisy/chaotic data sets. You must realize, however, that even if cluster analyses as such allow for an objective identification of groups, the analyst must still make at least three potentially subjective decisions. The first two of these influence how exactly the dendrogram will look like; the third you have already seen: one must decide what it is the dendrogram reflects. In what follows, I will show you how to do such an analysis with R yourself. Hierarchical agglomerative cluster analyses typically involve the following steps:

Procedure

Tabulating the data

- Computing a similarity/dissimilarity matrix on the basis of a user-defined similarity/dissimilarity metric
- Computing a cluster structure on the basis of a user-defined amalgamation rule
- Representing the cluster structure in a dendrogram and interpreting it
- (Post-hoc exploration (such as average silhouette widths)

The example we are going to discuss is from the domain of corpus/computational linguistics. In both disciplines, the degree of semantic similarity of two words is often approximated on the basis of the number and frequency of shared collocates. A very loose definition of a 'collocates of a word *w*' are the words that occur frequently in *w*'s environment, where *environment* in turn is often defined as 'in the same sentence' or within a 4- or 5-word window around *w*. For example: if you find the word *car* in a text, then very often words such as *driver*, *motor*, *gas*, and/or *accident* are relatively nearby whereas words such as *flour*, *peace treaty*, *dictatorial*, and *cactus collection* are probably not particularly frequent. In other words, the more collocates two words *x* and *y* share, the more likely there is a semantic relationship between the two (cf. Oakes 1998: Ch. 3, Manning and Schütze 2000: Section 14.1 and 15.2 as well as Gries 2009a for how to obtain collocates in the first place).

In the present example, we look at the seven English words *bronze*, *gold*, *silver*, *bar*, *cafe*, *menu*, and *restaurant*. Of course, I did not choose these words at random – I chose them because they intuitively fall into two clusters with bar (and thus constitute a good test case). One cluster consists

of three co-hyponyms of the *metal*, the other consists of three co-hyponyms of *gastronomical establishment* as well as a word from the same semantic field. Let us assume you extracted from the British National Corpus (BNC) all occurrences of these words and their content word collocates (i.e., nouns, verbs, adjectives, and adverbs). For each collocate that occurred with at least one of the seven words, you determined how often it occurred with each of the seven words. Table 46 is a schematic representation of the first six rows of such a table. The first collocate, here referred to as *X*, co-occurred only with bar (three times); the second collocate, *Y*, co-occurred 11 times with *gold* and once with *restaurant*, etc.

Table 46. Schematic co-occurrence frequencies of seven English words in the BNC

Collocate	*bronze*	*gold*	*silver*	*bar*	*cafe*	*menu*	*restaurant*
X	0	0	0	3	0	0	0
Y	0	11	0	0	0	0	1
Z	0	1	1	0	0	0	1
A	0	0	0	1	0	2	0
B	1	0	0	1	0	0	0
C	0	0	0	1	0	0	1
...	...	...	...	...	...	...	...

We are now asking the question which words are more similar to each other than to others. That is, just like in the example above, you want to group elements – above, phonemes, here, words – on the basis of properties – above, average similarity judgments, here, co-occurrence frequencies. First you need a data set such as Table 46, which you can load from the file <_inputfiles/05-6_collocates.RData>, which contains a large table of co-occurrence data – seven columns and approximately 31,000 rows.

```
> load(file.choose ()) # load the data frame¶
> ls() # check what was loaded¶
[1] "collocates"
> str(collocates)¶
'data.frame':   30936 obs. of  7 variables:
 $ bronze    : num  0 0 0 0 1 0 0 0 0 0 0 ...
 $ gold      : num  0 11 1 0 0 0 0 1 0 0 ...
 $ silver    : num  0 0 1 0 0 0 0 0 0 0 ...
 $ bar       : num  3 0 0 1 1 1 1 0 1 0 ...
 $ cafe      : num  0 0 0 0 0 0 0 0 0 1 ...
 $ menu      : num  0 0 0 2 0 0 0 0 0 0 ...
> attach(collocates)¶
 $ restaurant: num  0 1 0 0 0 0 0 0 0 0 ...
```

Alternatively, you could load those data with `read.table(…)` from the file <_inputfiles/05-6_collocates.csv>. If your data contain *missing data*, you should disregard those. There are no missing data, but the function is still useful to know (cf. the recommendation at the end of Chapter 2):

```
> collocates<-na.omit(collocates)¶
```

Next, you must generate a similarity/dissimilarity matrix for the seven words. Here, you have to make the first possibly subjective decision, deciding on a similarity/dissimilarity measure. You need to consider two aspects: the level of measurement of the variables in point and the definition of similarity to be used. With regard to the former, we will only distinguish between binary/nominal and ratio-scaled variables. I will discuss similarity/dissimilarity measures for both kinds of variables, but will then focus on ratio-scaled variables.

In the case of nominal variables, there are four possibilities how two elements can be similar or dissimilar to each other, which are represented in Table 47. On the basis of Table 47, the similarity of two elements is typically quantified using formula (66), in which w_1 and w_2 are defined by the analyst:

$$(66) \qquad \frac{a + w_1 \cdot d}{(a + w_1 \cdot d) + (w_2 \cdot (b + c))}$$

Table 47. Feature combinations of two binary elements

	Element 2 exhibits characteristic *x*	Element 2 does not exhibit characteristic *x*
Element 1 exhibits characteristic *x*	*a*	*b*
Element 1 does not exhibit characteristic *x*	*c*	*d*

Three similarity measures are worth mentioning here:

- the Jaccard coefficient: $w_1 = 0$ and $w_2 = 1$;
- the Simple Matching coefficient: $w_1 = 1$ and $w_2 = 1$;
- the Dice coefficient: $w_1 = 0$ and $w_2 = 0.5$.

What are their pairwise similarity coefficients of these three vectors?

```
> aa<-c(1, 1, 1, 1, 0, 0, 1, 0, 0, 0)¶
> bb<-c(1, 1, 0, 1, 0, 1, 0, 1, 0, 1)¶
> cc<-c(1, 0, 1, 1, 1, 1, 1, 1, 1, 0)¶
```

THINK BREAK

- Jaccard coefficient: for `aa` and `bb`: 0.375, for `aa` and `cc` 0.444, for `bb` and `cc` 0.4;
- Simple Matching coefficient: for `aa` and `bb`: 0.5, for `aa` and `cc` 0.5, for `bb` and `cc` 0.4;
- Dice coefficient: for `aa` and `bb`: 0.545, for `aa` and `cc` 0.615, for `bb` and `cc` 0.571 (see the code file for a function that computes these).

But when do you use which of the three? One rule of thumb is that when the presence of a characteristic is as informative as its absence, then you should use the Simple Matching coefficient, otherwise choose the Jaccard coefficient or the Dice coefficient. The reason for that is that, as you can see in formula (66) and the measures' definitions above, only the Simple Matching coefficient fully includes the cases where both elements exhibit or do not exhibit the characteristic in questions.

For ratio-scaled variables, there are (many) other measures, not all of which I can discuss here. I will focus on (i) a set of distance or dissimilarity measures (i.e., measures where large values represent large degrees of dissimilarity) and (ii) a set of similarity measures (i.e., measures where large values represent large degrees of similarity). Many distance measures are again based on one formula and then differ in terms of parameter settings. This basic formula is the so-called Minkowski metric represented in (67).

$$(67) \qquad \left(\sum_{i-1}^{n} \left| x_{qi} - x_{ri} \right|^{y} \right)^{1/y}$$

When y is set to 2, you get the so-called Euclidean distance.[38] If you in-

38. The Euclidean distance of two vectors of length n is the direct spatial distance between two points within an n-dimensional space. This may sound complex, but for the simplest case of a two-dimensional coordinate system this is merely the distance you would measure with a ruler.

sert $y = 2$ into (67) to compute the Euclidean distance of the vectors `aa` and `bb`, you obtain:

```
> sqrt(sum((aa-bb)^2))¶
[1] 2.236068
```

When y is set to 1, you get the so-called Manhattan- or City-Block distance of the above vectors. For `aa` and `bb`, you obtain:

```
> sum(abs(aa-bb))¶
[1] 5
```

The similarity measures are correlational measures. One of these you know already: the Pearson product-moment correlation coefficient r. A similar measure often used in computational linguistics is the cosine (cf. Manning and Schütze 1999: 299–303). The cosine and all other measures for ratio-scaled are available from the function `Dist` from the library `amap`.[39] This function requires that (i) the data are available in the form of a matrix or a data frame and that (ii) the elements whose similarities you want are in the rows, not in the columns as usual. If the latter is not the case, you can often just transpose a data structure (with `t`):

```
> library(amap)¶
> collocates.t<-t(collocates)¶
```

You can then apply the function `Dist` to the transposed data structure. This function takes the following arguments:

- `x`: the matrix or the data frame for which you want your measures;
- `method="euclidean"` for the Euclidean distance; `method="manhattan"` for the City-Block metric; `method="correlation"` for the product-moment correlation r (but see below!); `method="pearson"` for the cosine (but see below!) (there are some more measures available which I won't discuss here);
- `diag=FALSE` (the default) or `diag=TRUE`, depending on whether the distance matrix should contain its main diagonal or not;
- `upper=FALSE` (the default) or `upper=TRUE`, depending on whether the distance matrix should contain only the lower left half or both halves.

39. The function `dist` from the standard installation of R also allows you to compute several similarity/dissimilarity measures, but fewer than `Dist` from the library `amap`.

Thus, if you want to generate a distance matrix based on Euclidean distances for our collocate dataset you simply enter this:

```
> Dist(collocates.t, method="euclidean", diag=TRUE,
       upper=TRUE)¶
```

As you can see, you get a (symmetric) distance matrix in which the distance of each word to itself is of course 0. This matrix now tells you which word is most similar to which other word. For example, the word *silver* is most similar to is *cafe* because the distance of *silver* to *cafe* (2385.566) is the smallest distance that *silver* has to any word other than itself.

The following computes a distance matrix using the City-Block metric:

```
> Dist(collocates.t, method="manhattan", diag=TRUE,
       upper=TRUE)¶
```

To get a similarity matrix with product-moment correlations or cosines, you must compute the difference 1 minus the values in the matrix. To get a similarity matrix with correlation coefficients, you therefore enter this:

```
> 1-Dist(collocates.t, method="correlation", diag=TRUE,
       upper=TRUE)¶
           bronze   gold silver    bar   cafe   menu restaurant
bronze     0.0000 0.1342 0.1706 0.0537 0.0570 0.0462     0.0531
gold       0.1342 0.0000 0.3103 0.0565 0.0542 0.0458     0.0522
silver     0.1706 0.3103 0.0000 0.0642 0.0599 0.0511     0.0578
bar        0.0537 0.0565 0.0642 0.0000 0.1474 0.1197     0.2254
cafe       0.0570 0.0542 0.0599 0.1474 0.0000 0.0811     0.1751
menu       0.0462 0.0458 0.0511 0.1197 0.0811 0.0000     0.1733
restaurant 0.0531 0.0522 0.0578 0.2254 0.1751 0.1733     0.0000
```

You can check the results by comparing this output with the one you get from `cor(collocates)¶`. For a similarity matrix with cosines, you enter:

```
> 1-Dist(collocates.t, method="pearson", diag=TRUE,
       upper=TRUE)¶
```

There are also statistics programs that use 1-*r* as a distance measure. They change the similarity measure *r* (values close to zero mean low similarity) into a distance measure (values close to zero mean high similarity).

If you compare the matrix with Euclidean distances with the matrix with *r*, you might notice something that strikes you as strange …

THINK
BREAK

In the distance matrix, small values indicate high similarity and the smallest value in the column *bronze* is in the row for *cafe* (1734.509). In the similarity matrix, large values indicate high similarity and the largest value in the column *bronze* is in the row for *silver* (ca. 0.1706). How can that be? This difference shows that even a cluster algorithmic approach is influenced by subjective though hopefully motivated decisions. The choice for a particular metric influences the results because there are different ways in which vectors can be similar to each other. Consider as an example the following data set, which is also represented graphically in Figure 80.

```
> y1<-1:10; y2<-11:20; y3<-c(6, 6, 6, 5, 5, 5, 4, 4, 4, 3)¶
> y<-t(data.frame(y1, y2, y3))¶
```

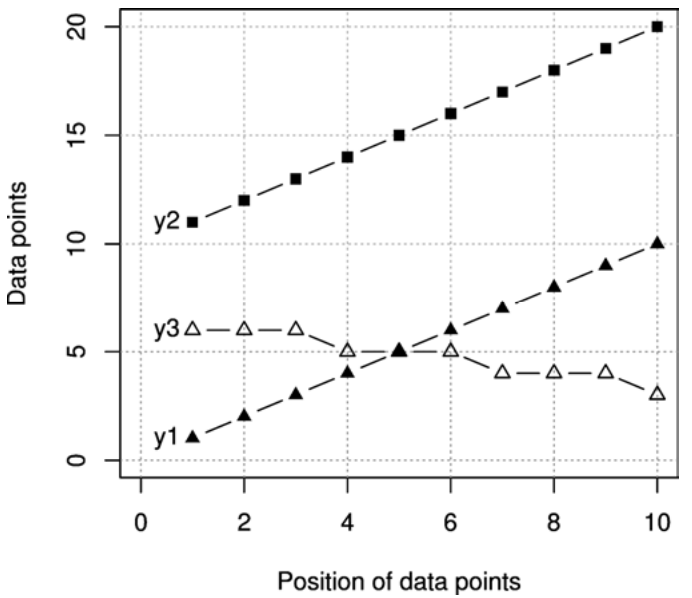

Figure 80. Three fictitious vectors

The question is, how similar is `y1` to `y2` and to `y3`? There are two obvious ways of considering similarity. On the one hand, `y1` and `y2` are perfectly parallel, but they are far away from each other (as much as one can say

that about a diagram whose dimensions are not defined). On the other hand, y1 and y3 are not parallel to each other at all, but they are close to each other. The two approaches I discussed above are based on these different perspectives. The distance measures I mentioned (such as the Euclidean distance) are based on the spatial distance between vectors, which is small between y1 and y3 but large between y1 and y2. The similarity measures I discussed (such as the cosine) are based on the similarity of the curvature of the vectors, which is small between y1 and y3, but large between y1 and y2. You can see this quickly from the actual numerical values:

```
> Dist(y, method="euclidean", diag=TRUE, upper=TRUE)¶
          y1       y2       y3
y1  0.00000 31.62278 12.28821
y2 31.62278  0.00000 35.93049
y3 12.28821 35.93049  0.00000
> 1-Dist(y, method="pearson", diag=TRUE, upper=TRUE)¶
           y1        y2        y3
y1 0.0000000 0.9559123 0.7796728
y2 0.9559123 0.0000000 0.9284325
y3 0.7796728 0.9284325 0.0000000
```

According to the Euclidean distance, y1 is more similar to y3 than to y2 – 12.288 < 31.623 – but the reverse is true for the cosine: y1 is more similar to y2 – 0.956 > 0.78. The two measures are based on different concepts of similarity. The analyst must decide what is more relevant: low spatial distances or similar curvatures. For now, we assume you want to adopt a curvature-based approach and use 1-*r* as a measure; in your own studies, you of course must state which similarity/distance measure you used, too.[40]

```
> dist.matrix<-Dist(collocates.t, method="correlation",
      diag=TRUE, upper=TRUE)¶
> round(dist.matrix, 4)¶
           bronze   gold silver    bar   cafe   menu restaurant
bronze     0.0000 0.8658 0.8294 0.9463 0.9430 0.9538     0.9469
gold       0.8658 0.0000 0.6897 0.9435 0.9458 0.9542     0.9478
silver     0.8294 0.6897 0.0000 0.9358 0.9401 0.9489     0.9422
bar        0.9463 0.9435 0.9358 0.0000 0.8526 0.8803     0.7746
cafe       0.9430 0.9458 0.9401 0.8526 0.0000 0.9189     0.8249
menu       0.9538 0.9542 0.9489 0.8803 0.9189 0.0000     0.8267
restaurant 0.9469 0.9478 0.9422 0.7746 0.8249 0.8267     0.0000
```

The next step is to compute a cluster structure from this similarity ma-

40. I am simplifying a lot here: the frequencies are neither normalized nor logged/dampened etc. (cf. above, Manning and Schütze 1999: Section 15.2.2, or Jurafsky and Martin 2008: Ch. 20).

trix. You do this with the function `hclust`, which can take up to three arguments of which I will discuss two. The first is a similarity/distance matrix, the second chooses an amalgamation rule that defines how the elements in that matrix get merged into clusters. This choice is the second potentially subjective decision and there are again several possibilities.

The choice `method="single"` uses the so-called *single-linkage-* or *nearest-neighbor* method. In this method, the similarity of elements *x* and *y* – where *x* and *y* may be elements such as individual consonants or subclusters such as {/b/, /p/} in Figure 79 – is defined as the *minimal* distance between any one element of *x* and any one element of *y*. In the present example this means that in the first amalgamation step *gold* and *silver* would be merged since their distance is the smallest in the whole matrix (1-*r* = 0.6897). Then, *bar* gets joined with *restaurant* (1-*r* = 0.7746). Then, and now comes the interesting part, {*bar restaurant*} gets joined with *cafe* because the smallest remaining distance is that which *restaurant* exhibits to *cafe*: 1-*r* = 0.8249. And so on. This amalgamation method is good at identifying outliers in data, but tends to produce long chains of clusters and is, therefore, often not particularly discriminatory.

The choice `method="complete"` uses the so-called *complete-linkage-* or *furthest-neighbor* method. Contrary to the single-linkage method, here the similarity of *x* and *y* is defined as the *maximal* distance between any one element of *x* and any one element of *y*. First, *gold* and *silver* are joined as before, then *bar* and *restaurant*. In the third step, {*bar restaurant*} gets joined with *cafe*, but the difference to the single linkage method is that the distance between the two is now 0.8526, not 0.8249, because this time the algorithm considers the maximal distances, of which the smallest is chosen for joining. This approach tends to form smaller homogeneous groups and is a good method if you suspect there are many smaller groups in your data.

Finally, the choice `method="ward"` uses a method whose logic is similar to that of ANOVAs because it joins those elements whose joining increases the error sum of squares least. For every possible amalgamation, the method computes the sums of squared differences/deviations from the mean of the potential cluster, and then the clustering with the smallest sum of squared deviations is chosen. This method is known to generate smaller clusters that are often similar in size and has proven to be quite useful in many applications. We will use it here, too, and again in your own studies, you must explicitly state which amalgamation rule you used. Now you can compute the cluster structure and plot it.

```
> clust.ana<-hclust(dist.matrix, method="ward")¶
```

```
> plot(clust.ana)¶
> rect.hclust(clust.ana, 2) # red boxes around clusters¶
```

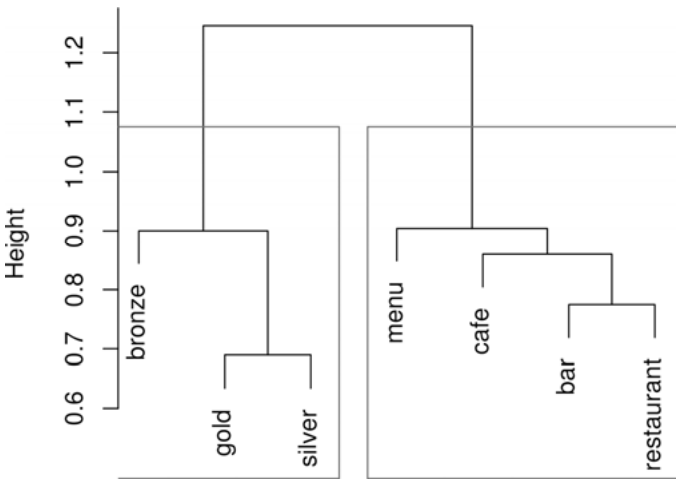

Figure 81. Dendrogram of seven English words

This is an uncharacteristically clearly interpretable result. As one would have hoped for, the seven words fall exactly into the two main expected clusters: one with the 'metals' and one with the gastronomy-related words. The former has a substructure in which *bronze* is somewhat less similar to the other two metals, and the latter very little substructure but groups the three co-hyponyms together before *menu* is added. With the following line you can have R show you for each element which cluster it belongs to when you assume two clusters.

```
> cutree(clust.ana, 2)¶
bronze     gold     silver      bar     cafe     menu restaurant
     1        1          1        2        2        2          2
```

While I can't discuss the method in detail, I want to briefly give you at least a glimpse of how more difficult cluster structures can be explored. As you will remember, Figure 79 was a much less clear-cut case in terms of how many clusters should be distinguished: any number between 2 and 5 seems defensible. The function cluster.stats from the library fpc offers a variety of validation statistics, which can help to narrow down the number of clusters best distinguished. One of these involves the notion of average silhouette widths, which quantifies how similar elements are to the clusters which they are in relative to how similar elements are to other clusters. It is

then possible to compute average silhouette widths for all possible cluster solutions and pick the one with the highest average silhouette widths. If we apply this logic to Figure 79, we get Figure 82. It shows why the decision for any one number of clusters is so difficult – many solutions fare nearly equally well – but why, if anything, four clusters sould be distinguished: with four clusters, the average silhouette width is highest: 0.14.

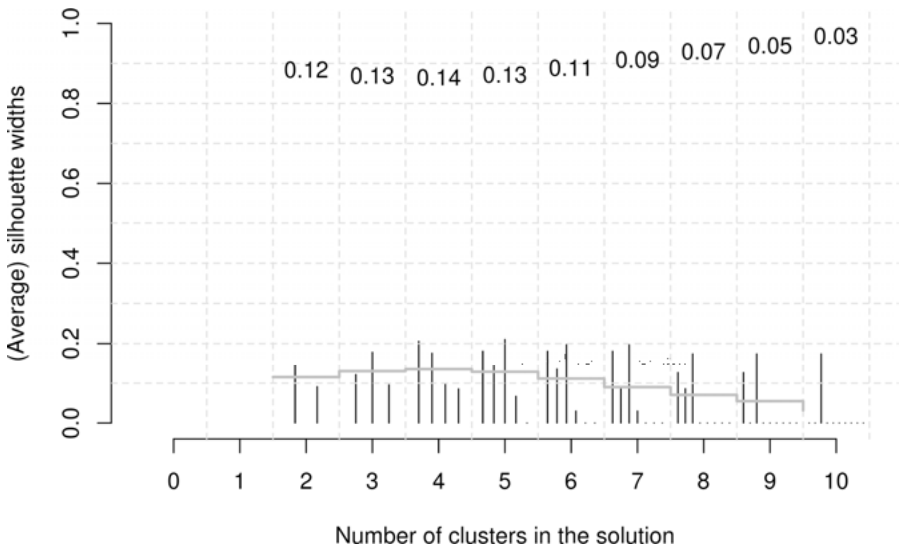

Figure 82. Average silhouette widths for all cluster solutions of Figure 79

Now you should do the exercises for Chapter 5 …

Recommendation(s) for further study

- the function `daisy` (from the library `cluster`) to compute distance matrices for dataset with variables from different levels of measurement
- the function `kmeans` to do cluster analyses where you provide the number of clusters beforehand
- the function `pvclust` (from the library `pvclust`) to obtain *p*-values for clusters based on resampling methods; cf. also `pvrect` and `pvpick` (from the same library)
- the function `varclus` (from the library `Hmisc`) to do variable clustering
- the function `nj` (from the library `ape`) to perform neighbor clustering and phylogenetic cluster analyses
- Crawley (2007: Ch. 23), Baayen (2008: Ch. 5), Johnson (2008: Ch. 6)