# 13 Choice Modeling

Much of the data we observe in marketing describes customers purchasing products. For example, as we discussed in Chap. 12, retailers now regularly record the transactions of their customers. In that chapter, we discussed analyzing retail transaction records to determine which products tend to occur together in the same shopping basket. In this chapter we discuss how to analyze customers' product choices within a category to understand how features and price affect which product a customer will choose. For example, if a customer comes into the store and purchases a 30 oz. jar of Hellman's brand canola mayonnaise for $3.98, we can conceptualize this as the customer choosing that particular type of mayonnaise among all the other mayonnaise available at that store. This data on customers' choices can be analyzed to determine which features of a product (e.g., package size, brand, or flavor) are most attractive to customers and how they trade off desirable features against price.

On the surface, this may sound quite similar to what we discussed in Chap. 7, where we cover how to use linear models to identify drivers of outcomes. It is similar, except that product choice data doesn't fit well into the linear modeling framework, because the outcome we observe is not a number or a rating for each product. Instead, we observe that the customer makes a *choice* among several options, each of which has its own set of attributes. To accommodate this unique data structure, marketers have adopted choice models, which are well suited to understanding the relationship between the attributes of products and customers' choices among sets of products. In this chapter, we focus on the *multinomial logit model*, the most frequently used choice model in marketing.

While choice models are often used to analyze retail purchase data, there are some settings where it is more difficult to collect data on customers' product choices. For example, when people shop for a car, they typically gather information from many sources over several months, so it is more difficult to reconstruct the set of products that they considered and the features and prices of those products. In these settings, marketers turn to choice-based *conjoint analysis*, which is a survey method where

C. Chapman, E.M. Feit, *R for Marketing Research and Analytics*, Use R!,
DOI 10.1007/978-3-319-14436-8_13

**Which of the following minivans would you buy?**
Assume all three minivans are identical other than the features listed below.

| | **Option 1** | **Option 2** | **Option 3** |
|---|---|---|---|
| | 6 passengers<br>2 ft. cargo area<br>gas engine<br>$35,000 | 8 passengers<br>3 ft. cargo area<br>hybrid engine<br>$30,000 | 6 passengers<br>3 ft. cargo area<br>gas engine<br>$30,000 |
| **I prefer (check one):** | ☐ | ☐ | [✓] |

**Fig. 13.1.** An example choice-based conjoint survey question.

customers are asked to make choices among products with varying features and prices. We analyze these survey choices using the multinomial logit model just as we might analyze real purchases. In this chapter, our example focuses on choice-based conjoint analysis, but the methods we describe could be applied to retail purchase data as well.

## 13.1 Choice-Based Conjoint Analysis Surveys

Suppose an automotive company such as Toyota or Ford is designing a new line of minivans and is trying to determine how large the minivan should be and what type of engine it should have. To inform this decision it would be helpful to understand how customers value those different features. Do customers like or dislike hybrid engines? If they like them, how much more would they be willing to pay for a hybrid engine? Are there segments of customers who like hybrid engines more than other customers?

Conjoint surveys give marketers information about how customers choose products by asking respondents to answer survey questions like the one shown in Fig. 13.1. In this question, respondents are asked to choose from three product profiles, each with a specific passenger capacity, cargo capacity, engine type, and price. The product options in the survey are called *alternatives* and the product features are called *attributes*. This conjoint analysis study has three alternatives in each question, described by four attributes. Each attribute occurs at some *level*. For example, the possible levels for cargo capacity in our example survey are 2 ft. and 3 ft.

In a typical choice-based conjoint survey, we ask respondents who are likely buyers of minivans to answer a number of questions similar to the one in Fig. 13.1. Each question has the same structure, but varies the levels of the attributes for the alternatives.

In the next section, we generate hypothetical data from a conjoint survey where each respondent answers 15 questions like the one in Fig. 13.1. Each question offers the respondent three alternatives to choose from, so each respondent sees a total of $15 \times 3 = 45$ product profiles. Conjoint surveys often include more attributes, more questions, and more alternatives in each question; a typical study might have 5–10 attributes and include 10–20 questions for each respondent.

You may recall that we also discussed “conjoint analysis” in Chap. 9. In that chapter we asked respondents to rate single products instead of having them choose among sets of products. This is called “ratings-based conjoint” or “metric conjoint” and is analyzed with linear models as we did in Chap. 9. While asking respondents to give ratings allows us to use a linear model instead of a choice model, rating profiles is a more difficult task for the respondent. When was the last time you considered whether a product was a 7 or an 8 on a 10-point scale? Choosing products as in Fig. 13.1 is a natural task that consumers do every day. For this reason choice-based conjoint surveys have become a standard tool in the arsenal of marketing researchers. When marketers say “conjoint,” they often mean choice-based conjoint analysis.

The key difference between choice-based conjoint and metric conjoint is the structure of the data you collect. In the minivan choice-based conjoint survey, each observation is a choice among three alternatives with varying levels of the product attributes. The goal of our analysis is to relate the choice to the product attributes. To do this we use a choice model, which is tailored to this unusual data structure.

The next section, where we simulate conjoint data, is written for readers who have some knowledge of choice models already. We encourage those of you who are new to choice modeling to download the data using the commands below and skip ahead to Sect. 13.3. Those who are familiar with choice modeling might wish to work through Sect. 13.2 to see how choice data is structured in R.

```
> cbc.df <- read.csv("http://goo.gl/5xQObB",
+                     colClasses = c(seat = "factor", price = "factor"))
> summary(cbc.df)
    resp.id           ques          alt      carpool     seat     cargo
 Min.   :  1.00   Min.   : 1   Min.   :1   no :6345   6:3024   2ft:4501
 1st Qu.: 50.75   1st Qu.: 4   1st Qu.:1   yes:2655   7:2993   3ft:4499
 Median :100.50   Median : 8   Median :2              8:2983
 Mean   :100.50   Mean   : 8   Mean   :2
 3rd Qu.:150.25   3rd Qu.:12   3rd Qu.:3
 Max.   :200.00   Max.   :15   Max.   :3
...
```

## 13.2 Simulating Choice Data*

If you loaded the data above, you can skip this optional section and go to Sect. 13.3.

The first step in creating any conjoint survey is to decide on which product attributes to include in the survey. Since this study focuses on size and engine type, we include four attributes: number of seats in the minivan, the cargo capacity (measured by the depth of the cargo area), the engine type, and the price. We create a list in R called `attrib` to store the attributes:

```
> attrib <- list(seat = c("6", "7", "8"),
+                cargo = c("2ft", "3ft"),
+                eng = c("gas", "hyb", "elec"),
+                price = c("30", "35", "40"))
```

Each element of this list is a character vector indicating levels of the attribute to include in the survey.

The next step is to generate *part worths* for the attributes. Part worths are conceived to be latent values a customer places on levels of an attribute when making choices. Each attribute in the choice model is treated like a factor in a linear model. As we discussed in Chap. 7, when we include a factor as a predictor in any model, that factor has to be coded. In this chapter, we use dummy coding, so that one level of the factor is considered the base level and the model includes coefficients that describe the part worth or value of that factor *over the base level*. If this is puzzling, you might review Sect. 7.4 on including factors as predictors in a linear model.

We designate the first level of each attribute to be the base level. We create names for the coefficients by looping over the attribute list, dropping the first level of the attribute, and then concatenating the name of the attribute and the level designation:

```
> coef.names <- NULL
> for (a in seq_along(attrib)) {
+   coef.names <- c(coef.names,
+                   paste(names(attrib)[a], attrib[[a]][-1], sep=""))
+ }
> coef.names
[1] "seat7"    "seat8"    "cargo3ft" "enghyb"   "engelec"  "price35"
[7] "price40"
```

Now we have a vector of seven coefficient names.

To generate the simulated data we assume that the average part worths in the population are:

```
> mu <- c(-1, -1, 0.5, -1, -2, -1, -2)
> names(mu) <- coef.names
> mu
   seat7    seat8 cargo3ft   enghyb  engelec  price35  price40
    -1.0     -1.0      0.5     -1.0     -2.0     -1.0     -2.0
```

You can see that we've given names to the elements of `mu`. While this isn't absolutely necessary, by keeping everything labeled using R's built-in `names`, the output is easier to read.

We assume that each respondent has his or her own unique part worth coefficients and that these follow a multivariate normal distribution in the population with a covariance matrix `Sigma`:

```
> Sigma <- diag(c(0.3, 1, 0.1, 0.3, 1, 0.2, 0.3))
> dimnames(Sigma) <- list(coef.names, coef.names)
> Sigma["enghyb", "engelec"] <- Sigma["engelec", "enghyb"] <- 0.3
```

The last line above creates a correlation between the part worth for `engelec` (electric engine) and `enghyb` (hybrid engine), so respondents who have a stronger preference for `engelec` over `enggas` will also have a stronger preference for `enghyb` over `enggas`.

With `mu` and `Sigma` in hand, we generate each respondent's part worth coefficients using the `mvrnorm` function from the `MASS` package. We create a vector of respondent IDs for the 200 respondents (`resp.id`) and a factor variable indicating whether each respondent intends to use the minivan to carpool (`carpool`).

```
> set.seed(33040)
> resp.id <- 1:200 # respondent ids
> carpool <- sample(c("yes", "no"), size=length(resp.id), replace=TRUE,
+                   prob=c(0.3, 0.7))
> library(MASS)
> coefs <- mvrnorm(length(resp.id), mu=mu, Sigma=Sigma)
> colnames(coefs) <- coef.names
```

Finally, we adjust the part worths for respondents who use the minivan to carpool:

```
> coefs[carpool=="yes", "seat8"] <- coefs[carpool=="yes", "seat8"] + 2
> coefs[carpool=="yes", "seat7"] <- coefs[carpool=="yes", "seat7"] + 1.5
```

`coefs` is now a matrix where each row contains the part worths for each respondent. To get a better sense of what we have done, you could type `head(coefs)` or, better yet, `head(cbind(carpool, coefs))`. You can also use the `by()` command (Sect. 3.4.5) to compute mean part worths for those who do and do not carpool. Just keep in mind that these coefficients are parameters of the model we plan to *simulate* from, not the final *observed* data.

With `coefs` in hand, we are ready to generate our survey questions and the observed responses. Our survey includes 15 questions that each ask the respondent to choose from three alternative minivans. We set the number of questions and alternatives as variables (`nques` and `nalt`) so that we might easily change the size of the survey in the future:

```
> nques <- 15
> nalt <- 3
```

Next, we create a master list of all possible minivan profiles by passing the `attrib` list to `expand.grid()`, which we discuss below:

```
> profiles <- expand.grid(attrib)
> nrow(profiles)
[1] 54
> head(profiles)
  seat cargo eng price
1    6   2ft gas    30
2    7   2ft gas    30
3    8   2ft gas    30
4    6   3ft gas    30
5    7   3ft gas    30
6    8   3ft gas    30
```

As you can see, profiles has 54 rows, representing all possible combinations of three levels of seating capacity, two levels of cargo capacity, three levels of engine, and three levels of price ($3 \times 2 \times 3 \times 3 = 54$). We can convert profiles to dummy coding using model.matrix().

```
> profiles.coded <- model.matrix(~ seat + cargo + eng + price,
+                                  data=profiles)[ , -1]
> head(profiles.coded)
  seat7 seat8 cargo3ft enghyb engelec price35 price40
1     0     0        0      0       0       0       0
2     1     0        0      0       0       0       0
3     0     1        0      0       0       0       0
4     0     0        1      0       0       0       0
5     1     0        1      0       0       0       0
6     0     1        1      0       0       0       0
```

profiles.coded now contains 54 rows, one for each possible combination of features, that are coded using the dummy coding scheme.

We haven't yet reviewed expand.grid() and model.matrix(); they are utility functions for handling factor variables. They are used under the hood in linear modeling routines such as lm() and predict() (Chap. 7). Because choice models are a variant of linear models, we can use these to generate our choice data. The only adjustment we need to make is to remove the intercept from the result of model.matrix() because choice models typically do not include intercepts.

Now that the respondent part worth coefficients are in coefs and the set of all possible minivan profiles is in profiles.coded, we are ready to generate our hypothetical survey questions and responses and store them in a data frame called cbc.df. For each of the 200 respondents, we choose nques*nalt (or $15 \times 3 = 45$) profiles at random from the list of all possible profiles in profiles.coded. The profiles indicated by the vector profiles.i are the profiles that we show respondent *i* in the survey: the first three profiles are the alternatives shown in choice 1, the next three profiles are the alternatives for choice 2, and so forth.

We compute each respondent's expected utility for each profile by multiplying the respondent's coefs by the coded product profile. (This happens in the line utility <- profiles.coded[profiles.i, ]%*%coefs[i, ] in the code below. We then compute choice probabilities for each alternative in the question according to the multinomial logit probabilities, computed as `probs <- exp(wide.util)/rowSums(exp(wide.util))`. We then take a random draw to determine which of the `nalt` products the customer chooses `choice <- apply(probs, 1, function(x) sample(1:nalt, size=1, prob=x))`. Finally, we append the choices and profiles to the `cbc.df` data frame. All of these steps are repeated for each respondent.

```
> cbc.df <- data.frame(NULL)
> for (i in seq_along(resp.id)) {
+   profiles.i <- sample(1:nrow(profiles), size=nques*nalt)
+   utility <- profiles.coded[profiles.i, ] %*% coefs[i, ]
+   wide.util <- matrix(data=utility, ncol=nalt, byrow=TRUE)
+   probs <- exp(wide.util) / rowSums(exp(wide.util))
+   choice <- apply(probs, 1, function(x) sample(1:nalt, size=1, prob=x))
+   choice <- rep(choice, each=nalt)==rep(1:nalt, nques)
+   conjoint.i <- data.frame(resp.id=rep(i, nques),
+                            ques = rep(1:nques, each=nalt),
+                            alt = rep(1:nalt, nques),
+                            carpool = rep(carpool[i], nques),
+                            profiles[profiles.i, ],
+                            choice = as.numeric(choice))
+   cbc.df <- rbind(cbc.df, conjoint.i)
# Tidy up, keeping only cbc.df and attrib
> rm(a, i, resp.id, carpool, mu, Sigma, coefs, coef.names,
+    conjoint.i, profiles, profiles.i, profiles.coded, utility,
+    wide.util, probs, choice, nalt, nques)
+ }
```

The code above leverages R's vector and matrix operations quite extensively. Going through it carefully and figuring out how each step works may take some time, but it will help you understand R's matrix computations and give you a clearer understanding of the assumptions of the multinomial logit model. At the core, this model is very similar to a linear model; the equation for `utility` is, in fact, a linear model. What makes a choice model distinct is that the `utility` is not observed directly; we only observe which product the respondent chooses. This is why we haven't stored the `utility` in our synthetic data in `cbc.df`.

In the code above, we have generated data from a choice model called a *hierarchical multinomial logit model*. *Hierarchical* refers to the fact that there is a different set of coefficients for each respondent and that those coefficients follow an "upper level" model for the population. In our code, the parameters of the upper level model are `mu`, `Sigma` and the adjustments we made for people who use their minivan to carpool. At the "lower level," the choices of an individual consumer follow a multinomial logit. The *hierarchical multinomial logit model* has become the workhorse of choice-based conjoint and is incorporated into commercial software for conjoint analysis such as *Sawtooth Software*. In this chapter, we begin by analyzing the data using the simpler multinomial logit model in Sect. 13.3, and then estimate the *hierarchical* multinomial logit model in Sects. 13.4 and 13.5.

## 13.3 Fitting a Choice Model

The simulated choice-based conjoint data is in the `cbc.df` data frame.

```
> head(cbc.df)
   resp.id ques alt carpool seat cargo  eng price choice
19       1    1   1     yes    6   2ft  gas    35      0
12       1    1   2     yes    8   3ft  hyb    30      0
4        1    1   3     yes    6   3ft  gas    30      1
1        1    2   1     yes    6   2ft  gas    30      0
23       1    2   2     yes    7   3ft  gas    35      1
31       1    2   3     yes    6   2ft elec    35      0
```

The first three rows in `cbc.df` describe the first question that was asked of respondent `1`, which is the question shown in Fig. 13.1. The `choice` column shows that this respondent chose the third alternative, which was a 6-passenger gas engine minivan with 3 ft of cargo capacity at a price of $30,000 (represented in $1,000s as "`30`"). `resp.id` indicates which respondent answered this question, `ques` indicates that these first three rows were the profiles in the first question and `alt` indicates that the first row was alternative 1, the second was alternative 2, and the third was alternative 3. (The row numbers all the way to the left in the output are not very meaningful. They indicate the profile number from our master list of 54 profiles that were used to generate the question; R carried this information over when we generated the data.) The variable `choice` indicates which alternative the respondent chose; it takes the value of `1` for the profile in each choice question that was indicated as the preferred alternative.

The `cbc.df` data frame organizes the data in what is sometimes called "long" format, where each profile is on its own line and there is a column that indicates which question the profile was displayed in. This is generally our preferred format for choice data, since it allows you to have a different number of profiles in each question by including additional rows. However, there are several other popular formats including a "wide" format, where each row corresponds to a different question and another format where the profiles are stored separately from the choices.

Because there is no standard format for choice data, when you work with different R packages or use data collected with other software systems, you need to pay close attention to how the package you are using expects the data to be formatted. Fortunately, there are R functions that can be helpful when reformatting data including base functions such as `reshape()`. You should never have to resort to tedious manual reformatting using a spreadsheet tool. Often someone else has written reliable R code to do the reformatting. For example, `Rcbc` [25] provides a helpful set of utilities for converting from the format used by Sawtooth Software into the format used by the `ChoiceModelR` package.

### 13.3.1 Inspecting Choice Data

Once you have your data properly formatted, it is tempting to estimate a complete choice model immediately. Popular choice modeling software packages make it easy to fit a model without even doing basic descriptives on the data. Don't fall into this trap! As with any other modeling, it is important to first get an understanding of the data using basic descriptives. We start with `summary`:

```
> summary(cbc.df)
    resp.id            ques          alt       carpool     seat     cargo
 Min.   :  1.00   Min.   : 1   Min.   :1   yes:2655   6:3024   2ft:4501
 1st Qu.: 50.75   1st Qu.: 4   1st Qu.:1   no :6345   7:2993   3ft:4499
 Median :100.50   Median : 8   Median :2              8:2983
 Mean   :100.50   Mean   : 8   Mean   :2
 3rd Qu.:150.25   3rd Qu.:12   3rd Qu.:3
 Max.   :200.00   Max.   :15   Max.   :3
...
```

We see how many times each level of each attribute appeared in the questions (about 3,000 times for three-level attributes and about 4,500 times for two-level attributes). However, a more informative way to summarize choice data is to compute choice *counts*, which are cross tabs on the number of times respondents chose an alternative at each feature level. We can do this easily using `xtabs()`, covered in Chap. 5:

```
> xtabs(choice ~ price, data=cbc.df)
price
  30   35   40
1486  956  558
```

Respondents chose a minivan at the $30 K price point much more often than they chose minivans priced at $35 K or $40 K. If we compute counts for the `cargo` attribute, we find that the choices were more balanced between the two options, suggesting that `cargo` was not as important to customers as `price`:

```
> xtabs(choice ~ cargo, data=cbc.df)
cargo
 2ft  3ft
1312 1688
```

We encourage you to compute choice counts for each attribute before estimating a choice model. If you find that your model's estimates or predicted shares are not consistent with the raw counts, consider whether there could be a mistake in the data formatting. Many times, a junior analyst has come to one of us saying, "The predictions from my choice model don't make sense to the client," and our first question is always,"Have you looked at the raw choice counts?"

Often this reveals a mistake, but when there is no mistake, it can be helpful to show the clients the raw choice counts to help them understand that your model predictions are based on how people responded in the survey. With that warning, we can now estimate our first choice model. By fitting a choice model, we can get a precise measurement of how much each attribute is associated with respondents' choices.

### 13.3.2 Fitting Choice Models with `mlogit()`

We use the `mlogit` package, which you may need to install with `install.packages()`. `mlogit` estimates the most basic and commonly used choice model, the *multinomial logit* model. This model is also called the *conditional logit*.

`mlogit` requires the choice data to be in a special data format created using the `mlogit.data()` function. You pass your choice data to `mlogit.data`, along with a few parameters telling it how the data is organized. `mlogit.data` accepts data in either a "long" or a "wide" format and you tell it which you have using the `shape` parameter. The `choice`, `varying` and `id.var` parameters indicate which columns contain the response data, the attributes and the respondent ids, respectively.

```
> library(mlogit)
> cbc.mlogit <- mlogit.data(data=cbc.df, choice="choice", shape="long",
+                           varying=3:6, alt.levels=paste("pos",1:3),
+                           id.var="resp.id")
```

The resulting `cbc.mlogit` is an `mlogit.data` object that can be used to estimate a model with `mlogit()`. The syntax for `mlogit` uses formula notation similarly to other functions for regression models in R:

```
> m1 <- mlogit(choice ~ 0 + seat + cargo + eng + price, data = cbc.mlogit)
> summary(m1)
...
Frequencies of alternatives:
  pos 1   pos 2   pos 3
0.32700 0.33467 0.33833
...
Coefficients :
          Estimate Std. Error  t-value  Pr(>|t|)
seat7    -0.535280   0.062360  -8.5837 < 2.2e-16 ***
seat8    -0.305840   0.061129  -5.0032 5.638e-07 ***
cargo3ft  0.477449   0.050888   9.3824 < 2.2e-16 ***
enghyb   -0.811282   0.060130 -13.4921 < 2.2e-16 ***
engelec  -1.530762   0.067456 -22.6926 < 2.2e-16 ***
price35  -0.913656   0.060601 -15.0765 < 2.2e-16 ***
price40  -1.725851   0.069631 -24.7856 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Log-Likelihood: -2581.6
```

The output also looks quite similar to what we have seen for other models. At the bottom of the output is a table of the estimated part worth coefficients for the population. The `Estimate` lists the mean values for each level; these must be interpreted relative to the base levels of each attribute. For example, the estimate for `seat7` measures the attractiveness of 7 passenger minivans *relative to 6 passenger minivans*. The negative sign tells us that, on average, our simulated customers preferred 6 seat minivans to 7 seat minivans. Estimates that are larger in magnitude indicate stronger preferences, so we can see that customers strongly disliked electric engines (relative to the base level, which is gas) and disliked the $40 K price (relative to the base level price of $30). These parameter estimates are on the logit scale (Sect. 9.2.1) and typically range between $-2$ and 2.

The `Std. Error` column gives a sense of how precise the estimate is, given the data, along with a statistical test of whether the coefficient is different than zero. A non-significant test result indicates that there is no detectible difference in preference for that level relative to the base level. Just as with any statistical model, the more data you have in you conjoint study (for a given set of attributes), the smaller the standard errors will be. Similarly, if there are many attributes and levels in a study (for a fixed number of respondents answering a survey of a given length), the part worth estimates will be very imprecise. We'll discuss more what this means for an analysis in Sect. 13.6.

You may have wondered why we included `0 +` in the formula for `m1`, indicating that we did not want an intercept included in our model. We could estimate a model with an intercept:

```
> m2 <- mlogit(choice ~ seat + cargo + eng + price, data = cbc.mlogit)
> summary(m2)
...
Coefficients :
                    Estimate Std. Error  t-value  Pr(>|t|)
pos 2:(intercept)   0.028980   0.051277   0.5652    0.5720
pos 3:(intercept)   0.041271   0.051384   0.8032    0.4219
seat7              -0.535369   0.062369  -8.5840 < 2.2e-16 ***
seat8              -0.304369   0.061164  -4.9763 6.481e-07 ***
cargo3ft            0.477705   0.050899   9.3854 < 2.2e-16 ***
enghyb             -0.811494   0.060130 -13.4956 < 2.2e-16 ***
engelec            -1.529423   0.067471 -22.6677 < 2.2e-16 ***
price35            -0.913777   0.060608 -15.0769 < 2.2e-16 ***
price40            -1.726878   0.069654 -24.7922 < 2.2e-16 ***
...
Log-Likelihood: -2581.3
McFadden R^2:  0.21674
Likelihood ratio test : chisq = 1428.5 (p.value = < 2.22e-16)
```

When we include the intercept, `mlogit` adds two additional parameters that indicate preference for the different positions in the question (left, right, or middle in Fig. 13.1): `pos2:(intercept)` indicates the relative preference of the second position in the question (versus the first) and `pos3:(intercept)` indicates the preference for the third position (versus the first.) These are sometimes called *alternative specific constants* or ASC's to differentiate them from the single intercept in a linear model.

In a typical conjoint analysis study, we don't expect that people will choose a minivan because it is on the left or the right in a survey question! For that reason, we would not expect the estimated alternative specific constants to differ from zero. If we found one of these parameters to be significant, that might indicate that some respondents are simply choosing the first or the last option without considering the question.

In this model, the intercept parameter estimates are non-significant and close to zero. This suggests that it was reasonable to leave them out of our first model, but we can test this formally using `lrtest()`:

```
> lrtest(m1, m2)
Likelihood ratio test

Model 1: choice ~ 0 + seat + cargo + eng + price
Model 2: choice ~ seat + cargo + eng + price
  #Df  LogLik Df  Chisq Pr(>Chisq)
1   7 -2581.6
2   9 -2581.3  2 0.6789     0.7122
```

This function performs a statistical test called a likelihood ratio test, which can be used to compare two choice models where one model has a subset of the parameters of another model. Comparing `m1` to `m2` results in a p-value (`Pr(>Chisq)`) of `0.7122`. Since the p-value is much greater than 0.05, we can conclude that `m1` and `m2` fit the data equally well. This suggests that we don't need the alternative specific constants to fit the present data.

There are a few occasions where alternative specific constants do make sense. In some conjoint studies, the respondent is presented with several "fixed" alternatives. Option 1 might be a salad, option 2 might be a sandwich, and option 3 might be a soup. In each question, the attributes of those options vary, but the respondent is always asked to chose from one salad, one sandwich and one soup. Similarly, there might be a study of commuters' choice of transportation alternatives where alternative 1 is always a bus, alternative 2 is always a train, and alternative 3 is always driving. In such cases, you should include the alternative specific constants, but in the majority of conjoint analysis surveys in marketing, alternative specific constants aren't used.

You don't have to treat every attribute in a conjoint study as a factor. As with linear models, some predictors may be factors while others are numeric. For example, we can include `price` as a numeric predictor with a simple change to the model formula. In the model formula, we convert `price` to character vector using `as.character` and then to a number using `as.numeric`. (If you use `as.numeric` without `as.character` first, `price` will be converted to the values `1`, `2`, and `3` due to the way R stores factors internally. Converting to a character first results in values of `30`, `35`, and `40`.)

```
> m3 <- mlogit(choice ~ 0 + seat + cargo + eng
+                        + as.numeric(as.character(price)),
+              data = cbc.mlogit)
> summary(m3)
...
Coefficients :
                                  Estimate Std. Error  t-value  Pr(>|t|)
seat7                           -0.5345392  0.0623518  -8.5730 < 2.2e-16 ***
seat8                           -0.3061074  0.0611184  -5.0084 5.488e-07 ***
cargo3ft                         0.4766936  0.0508632   9.3721 < 2.2e-16 ***
enghyb                          -0.8107339  0.0601149 -13.4864 < 2.2e-16 ***
engelec                         -1.5291247  0.0673982 -22.6879 < 2.2e-16 ***
as.numeric(as.character(price)) -0.1733053  0.0069398 -24.9726 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.'0.1 ' ' 1

Log-Likelihood: -2582.1
```

The output now shows a single parameter for price. The estimate is negative indicating that people prefer lower prices to higher prices. A quick likelihood ratio test suggests that the model with a single price parameter fits just as well as our first model.

```
> lrtest(m1, m3)
Likelihood ratio test

Model 1: choice ~ 0 + seat + cargo + eng + price
Model 2: choice ~ 0 + seat + cargo + eng + as.numeric(as.character(price))
  #Df  LogLik Df  Chisq Pr(>Chisq)
1   7 -2581.6
2   6 -2582.1 -1 0.9054     0.3413
```

Given this finding, we choose `m3` as our preferred model because it has fewer parameters.

### 13.3.3 Reporting Choice Model Findings

It is often difficult, even for those with training in choice models, to interpret choice model part worth estimates directly. The coefficients are on an unfamiliar scale and they measure relative preference for the levels, which can make them difficult to understand. So, instead of presenting the coefficients, most choice modelers prefer to focus on using the model to make *choice share predictions* or to compute *willingness-to-pay* for each attribute.

#### 13.3.3.1 Willingness-to-Pay

In a model like `m3` where we estimate a single parameter for price, we can compute the average willingness-to-pay for a particular level of an attribute by dividing the coefficient for that level by the price coefficient.

```
> coef(m3)["cargo3ft"]/(-coef(m3)["as.numeric(as.character(price))"]/1000)
cargo3ft
2750.601
```

The result is a number measured in dollars, $2750.60 in this case. (We divide by 1000 because our prices were recorded in 1,000s of dollars.) Willingness-to-pay is a bit of a misnomer; the proper interpretation of this number is that, on average, customers would be equally divided between a minivan with 2 ft of cargo space and a minivan with 3 ft of cargo space that costs $2750.60 more. Another way to think of it is that $2750.60 is the price at which customers become indifferent between the two cargo capacity options. This same willingness to pay value can be computed for every attribute in the study and reported to decision makers to help them understand how much customers value various features.

#### 13.3.3.2 Simulating Choice Shares

While willingness-to-pay is more interpretable than attribute coefficients, it can still be difficult to understand. Many analysts prefer to focus exclusively on using the model to make share predictions. A share simulator allows you to define a number of different alternatives and then use the model to predict how customers would choose among those new alternatives. For example, you could use the model to predict choice share for the company's new minivan design against a set of key competitors. By varying the attributes of the planned minivan design, you can see how changes in the design affect the choice share.

Unfortunately, there isn't a handy `predict()` function for `mlogit` model objects, as there are for many other types of model objects. Luckily, it isn't too difficult to write our own:

```
> # Predicting shares
> predict.mnl <- function(model, data) {
+   # Function for predicting shares from a multinomial logit model
+   # model: mlogit object returned by mlogit()
+   # data: a data frame containing the set of designs for which you want to
+   #       predict shares.  Same format as the data used to estimate model.
+   data.model <- model.matrix(update(model$formula, 0 ~ .), data = data)[,-1]
+   utility <- data.model%*%model$coef
+   share <- exp(utility)/sum(exp(utility))
+   cbind(share, data)
+ }
```

In a moment, we'll walk through this code more carefully, but first let's see how it works. The comments tell us that the function takes two inputs: a model object returned from `mlogit()` and a data frame containing the set of designs for which you want to predict shares. We already have several model objects, so all we need to do is create new data. One way to do this is to create the full set of possible designs using `expand.grid()` and select the designs we want by row number:

```
> (new.data <- expand.grid(attrib)[c(8, 1, 3, 41, 49, 26), ])
   seat cargo  eng price
8     7   2ft  hyb    30
1     6   2ft  gas    30
3     8   2ft  gas    30
41    7   3ft  gas    40
49    6   2ft elec    40
26    7   2ft  hyb    35
```

We then pass these designs to `predict.mnl()` to determine what customers would choose if they had to pick among these six minivan alternatives:

```
> predict.mnl(m3, new.data)
        share seat cargo  eng price
8  0.11268892    7   2ft  hyb    30
1  0.43263922    6   2ft  gas    30
3  0.31855551    8   2ft  gas    30
41 0.07216867    7   3ft  gas    40
49 0.01657221    6   2ft elec    40
26 0.04737548    7   2ft  hyb    35
```

The model-predicted shares are shown in the column labeled `share` and we can see that among this set of products, we would expect respondents to choose the 7-seat hybrid engine minivan with 2 ft of cargo space at $30 K a little more than 11 % of the time. If a company was planning to launch a minivan like this, they could use the model to see how changing the attributes of this product would affect the choice shares. Note that these share predictions are always made relative to a particular set of competitors; the share for the first minivan would change if the competitive set were different.

For those who are new to choice models, we should caution against using share predictions based on survey data as a market share forecast. While these share predictions are typically a good representation of how respondents would behave if they were asked to choose among these six minivans in a new survey, that predicted survey response might not translate directly to sales in the marketplace. Customers might not be able to find the product in stores or they may react differently to the features when they see them in the showroom. We generally recommend that the analyst be careful to communicate this by labeling the predicted shares as "survey shares" or "preference shares" to alert others to this distinction. If you estimate a multinomial logit model using retail purchase data, as we discussed earlier, you would not need to make this caveat, as your predictions would be based on real-world purchases.

We could compute shares using model `m1`, which treated price as a factor rather than a continuous variable:

```
> predict.mnl(m1, new.data)
        share seat cargo  eng price
8  0.11273356    7   2ft  hyb    30
1  0.43336911    6   2ft  gas    30
3  0.31917819    8   2ft  gas    30
41 0.07281396    7   3ft  gas    40
49 0.01669280    6   2ft elec    40
26 0.04521237    7   2ft  hyb    35
```

We see that the predicted shares are almost identical, confirming our previous conclusion that `m3` is very similar to `m1`. (Comparing predicted shares is not the best way to compare two models. For a formal comparison of models, we recommend `lrtest()`.)

Now that we have seen how `predict.mnl()` works, let's take a closer look at the code for the function. Ignoring the comments, the code is just four lines. We repeat them here so that we can discuss each line, but you don't need to type them into the console again. On the first line, we convert the data, which is stored as factors, to a coded matrix:

```
data.model <- model.matrix(update(model$formula, 0 ~ .), data = data)[,-1]
```

We do this using two functions from base R for working with formulas. The function `model.matrix`, which we saw earlier in the chapter, converts the data from factors to coded effects. It requires the right-hand side of the formula from `model`, which we obtain using the `update` function for formulas. We also have to remove the first column of the result of `model.matrix`, because our choice model doesn't have an intercept. On the next line, we compute the utility for each product by multiplying the coded data by the model coefficients using matrix multiplication:

```
utility <- data.model %*% model$coef
```

The result is a utility value for each product in the set based on its attributes. Finally, we convert that to shares using the multinomial logit equation:

```
share <- exp(utility) / sum(exp(utility))
```

The function then returns the shares along with the product design data. (Experienced choice modelers will notice that we are slightly abusing terminology when we call this `utility`. More precisely this should be called the deterministic portion of the utility, since it doesn't include the error term. We do not include a stochastic component in the share simulator, because we want to report the expected average shares across many choices.)

#### 13.3.3.3 Sensitivity Plots

Often a product design team has a particular product design in mind and wants to know how share would change if they were to change their design. For example, suppose the minivan designers plan to build a 7-passenger hybrid minivan with 2 ft. of cargo space and sell it at $30 K. The model can be used to predict how share would change if different levels of the attributes were included (while keeping the competitive set fixed.) The plot in Fig. 13.2 shows how share would change if we changed each of the attributes of the design, one at a time. We see that changing the planned 7-seat design to a 6-seat design would increase share by just under 0.07. Increasing the price to $35 K would decrease share by about 0.06. This gives the design team an at-a-glance picture of how changes in their design affect choice share.

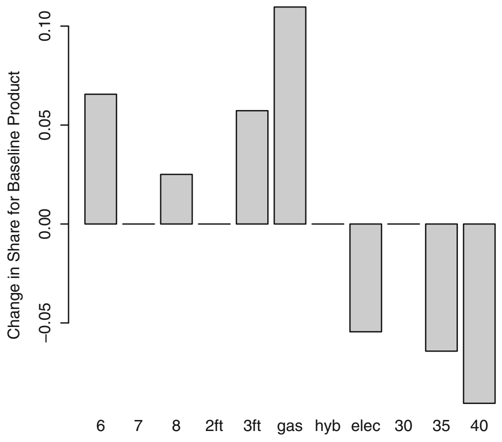

**Fig. 13.2.** Sensitivity plot showing how share for the planned design changes as we change each attribute, relative to a set of competing designs. The planned design is a 7-passenger hybrid minivan with 2 ft. of cargo space offered at $30,000.

Producing this plot using R is relatively simple: we just need to loop through all the attribute levels, compute a share prediction, and save the predicted share for the target design. Since this is an analysis we do regularly, we wrote a function to do it.

```
> sensitivity.mnl <- function(model, attrib, base.data, competitor.data) {
+   # Function for creating data for a share-sensitivity chart
+   # model: mlogit object returned by mlogit() function
+   # attrib: list of vectors with attribute levels to be used in sensitivity
+   # base.data: data frame containing baseline design of target product
+   # competitor.data: data frame containing design of competitive set
+   data <- rbind(base.data, competitor.data)
+   base.share <- predict.mnl(model, data)[1,1]
+   share <- NULL
+   for (a in seq_along(attrib)) {
+     for (i in attrib[[a]]) {
+       data[1,] <- base.data
+       data[1,a] <- i
+       share <- c(share, predict.mnl(model, data)[1,1])
+     }
+   }
+   data.frame(level=unlist(attrib), share=share, increase=share-base.share)
+ }
```

Using `sensitivity.mnl`, we create the plot in Fig. 13.2 with four commands:

```
> base.data <- expand.grid(attrib)[c(8), ]
> competitor.data <- expand.grid(attrib)[c(1, 3, 41, 49, 26), ]
> (tradeoff <- sensitivity.mnl(m1, attrib, base.data, competitor.data))
       level      share    increase
seat1      6 0.17831027  0.06557671
seat2      7 0.11273356  0.00000000
...
price3    40 0.02211862 -0.09061494
> barplot(tradeoff$increase, horiz=FALSE, names.arg=tradeoff$level,
+         ylab="Change in Share for Baseline Product")
```

### 13.3.4 Share Predictions for Identical Alternatives

Occasionally, you may want to predict shares for two designs that are identical in terms of the attributes that you've included in your conjoint study. For example, you might be planning to offer a design that is the same as a competitor. A naive analyst might include both designs in a set to estimate with `predict.mnl()` and there is nothing to stop one from doing that:

```
> new.data.2 <- expand.grid(attrib)[c(8, 8, 1, 3, 41, 49, 26), ]
> predict.mnl(m1, new.data.2)
         share seat cargo  eng price
8   0.10131227    7   2ft  hyb    30
8.1 0.10131227    7   2ft  hyb    30
1   0.38946350    6   2ft  gas    30
3   0.28684152    8   2ft  gas    30
41  0.06543701    7   3ft  gas    40
49  0.01500162    6   2ft elec    40
26  0.04063181    7   2ft  hyb    35
```

However, these share predictions may be considered unrealistic. When we estimate shares from `m1` with just one copy of design 8, we get a share of about 0.113. With two copies of the same design, each alternative is predicted to get a share of about 0.101 for a total of 0.202 between the two of them. It seems quite unreasonable that people would be more likely to choose a 7-passenger, 2 ft., hybrid at $30 K just because there are two of them in the choice set. Beyond that, the relative shares of all the other vehicles have remained the same, including the higher-priced 7-passenger hybrid (design 26 in the last row). Why wouldn't design 8 steal more share from design 26 than from the other non-hybrid vehicles?

While this is confusing, multinomial logit models make predictions in this way. Much has been written about this property of the multinomial logit model and there are many arguments about whether it is desirable. In fact, the property has been given a name: the *independence of irrelevant alternatives* or IIA property. It is also sometimes called the "red bus/blue bus problem" based on a classic example that involves predicting share for two different color buses that have otherwise identical features. Predictions from the multinomial logit model for two identical alternatives or even two nearly identical alternatives will exhibit this property.

More sophisticated hierarchical models, which we discuss in Sect. 13.5, relax this property somewhat, although they still may make predictions for similar or identical alternatives that seem unreasonable [36]. There are a number of proposed methods to estimate choice models that do not have the IIA property including nested logit, generalized logit, and multinomial probit. If you need to predict shares for nearly identical designs, we encourage you to review those alternatives. However, the majority of marketers today use either the multinomial logit or the hierarchical multinomial logit model, and—we hope—try to avoid including identical or nearly identical designs when estimating shares.

### 13.3.5 Planning the Sample Size for a Conjoint Study

A crucial issue in planning a successful conjoint analysis study is to decide how many respondents should complete the survey. To see how sample size affects the model estimates and share predictions, let's estimate a model using just the data from the first 25 respondents. We do this by creating `small.conjoint`, which is an `mlogit.data` object with just the first $25 \times 15 \times 3 = 1125$ rows of our original `cbc.df` data, corresponding to the survey responses for the first 25 respondents.

```
> small.cbc <- mlogit.data(data=cbc.df[1:(25*15*3),],
+                             choice="choice", shape="long",
+                             varying=3:6, alt.levels=paste("pos", 1:3),
+                             id.var="resp.id")
> m4 <- mlogit(choice ~ 0 + seat + cargo + eng + price, data = small.cbc)
```

If we take a look at the coefficient estimates for m4 and compare them to the coefficient estimates for m1 (above), we can see that the estimated coefficients for m4 are similar, but the standard errors for the coefficients are more than three times as big, reflecting the fact that with less data, our estimates of the model coefficients are less precise.

```
> summary(m4)  # larger standard errors
...
         Estimate Std. Error t-value  Pr(>|t|)
seat7    -0.74326    0.17767 -4.1833 2.873e-05 ***
seat8    -0.15180    0.16859 -0.9004 0.3679142
cargo3ft  0.45613    0.14459  3.1546 0.0016071 **
enghyb   -0.59674    0.16838 -3.5440 0.0003941 ***
engelec  -1.62677    0.19764 -8.2311 2.220e-16 ***
price35  -0.81508    0.17304 -4.7105 2.471e-06 ***
price40  -1.71390    0.20304 -8.4410 < 2.2e-16 ***
...
```

The standard errors are also higher for the attribute-levels that are least often chosen, including the engelec and price40 coefficients. In general, standard errors for less-frequently chosen attributes will be higher. One method for planning sample sizes focuses on reducing the standard errors of the estimates to acceptable levels. We discuss this further when we discuss the design of conjoint surveys in Sect. 13.6.

We can also compare predictions between m1 and m4:

```
> cbind(predict.mnl(m4, new.data), predict.mnl(m1, new.data))
        share seat cargo  eng price      share seat cargo  eng price
8  0.10876219    7   2ft  hyb    30 0.11273356    7   2ft  hyb    30
1  0.41536666    6   2ft  gas    30 0.43336911    6   2ft  gas    30
3  0.35686673    8   2ft  gas    30 0.31917819    8   2ft  gas    30
41 0.05615650    7   3ft  gas    40 0.07281396    7   3ft  gas    40
49 0.01470947    6   2ft elec    40 0.01669280    6   2ft elec    40
26 0.04813846    7   2ft  hyb    35 0.04521237    7   2ft  hyb    35
```

Here we find that the two models make similar share predictions. This illustrates the fact that comparing share predictions is not the ideal way to compare two different conjoint survey designs. When we look at the standard errors for the coefficients, we see the difference between m1 and m4 more clearly.

If we looked at the standard errors of the share predictions, they would be more precise for m1, but we can't see that here because there are no standard errors reported for shares. While it is possible to compute standard errors for share predictions, it requires using the "delta method" or a bootstrapping strategy, both of which are difficult to do outside of a programming environment like R. So, among those who do not use R, it is uncommon to report standard errors or estimates of uncertainty for share predictions.

This is unfortunate; decision makers often see only the point estimates for share predictions and are not informed about the confidence intervals of those shares. An ambitious reader might write code to produce intervals for share predictions from the multinomial logit model, but we will hold off on estimating intervals for share predictions until we review choice models in a Bayesian framework in Sect. 13.5.

## 13.4 Adding Consumer Heterogeneity to Choice Models

Up to this point, we have focused on the multinomial logit model, which estimates a single set of part worth coefficients for a whole sample. In this section, we look at a model that allows for each respondent to have his or her own coefficients. Different people have different preferences, and models that estimate individual-level coefficients can fit data better and make more accurate predictions than sample-level models (see [137]).

To estimate a model where each respondent has his or her own part worths, it is helpful to have multiple observations for each respondent. This is not a problem in a typical conjoint analysis study because each respondent answers multiple question. However, it can be a problem when estimating choice models using retail purchase data, because many people only make a single purchase. Most conjoint analysis practitioners routinely estimate heterogeneous choice models with conjoint survey data and it is easy to do this in R. In this section, we show how to do it using `mlogit`, which uses traditional frequentist statistical methods. In Sect. 13.5, we show how to estimate heterogeneous choice models using Bayesian methods. For either section, if you are not familiar with hierarchical models, you should review the basics in Sect. 9.3.

### 13.4.1 Estimating Mixed Logit Models with `mlogit()`

The statistical term for coefficients that vary across respondents (or customers) is *random coefficients* or random effects (see Sect. 9.3.1). To estimate a multinomial logit model with random coefficients using `mlogit`, we define a vector indicating which coefficients should vary across customers. `mlogit` requires a character vector the same length as the coefficient vector with a letter code indicating what distribution the random coefficients should follow across the respondents: 'n' for normal, 'l' for log normal, 't' for truncated normal, and 'u' for uniform. For this analysis, we assume that all the coefficients are normally distributed across the population and call our vector `m1.rpar`.

```
> m1.rpar <- rep("n", length=length(m1$coef))
> names(m1.rpar) <- names(m1$coef)
> m1.rpar
   seat7    seat8 cargo3ft   enghyb  engelec  price35  price40
     "n"      "n"      "n"      "n"      "n"      "n"      "n"
```

We pass this vector to `mlogit` as the `rpar` parameter, which is short for "*r*andom *par*ameters". In addition, we tell `mlogit` that we have multiple choice observations for each respondent (`panel=TRUE`) and whether we want to allow the random parameters to be correlated with each other. For this first run, we assume that we do not want random parameters to be correlated (`correlation=FALSE`), a setting we reconsider below.

```
> m1.hier <- mlogit(choice ~ 0 + seat + eng + cargo + price,
+                   data = cbc.mlogit,
+                   panel=TRUE, rpar = m1.rpar, correlation = FALSE)
```

The algorithm to estimate the heterogeneous logit model is computationally intensive, so it may take a few seconds to run. Once it finishes, you can look at the parameter estimates using `summary()`:

```
> summary(m1.hier)
...
Coefficients :
             Estimate Std. Error  t-value  Pr(>|t|)
seat7       -0.642241   0.070893  -9.0593 < 2.2e-16 ***
seat8       -0.390021   0.070460  -5.5353 3.106e-08 ***
enghyb      -0.926145   0.067456 -13.7296 < 2.2e-16 ***
engelec     -1.831864   0.083439 -21.9544 < 2.2e-16 ***
cargo3ft     0.550838   0.058459   9.4226 < 2.2e-16 ***
price35     -1.081310   0.070874 -15.2567 < 2.2e-16 ***
price40     -1.991787   0.085312 -23.3471 < 2.2e-16 ***
sd.seat7    -0.651807   0.101906  -6.3961 1.594e-10 ***
sd.seat8     0.995007   0.093397  10.6535 < 2.2e-16 ***
sd.enghyb    0.159495   0.137950   1.1562  0.247607
sd.engelec   0.973303   0.099850   9.7476 < 2.2e-16 ***
sd.cargo3ft  0.307194   0.131109   2.3430  0.019127 *
sd.price35  -0.260907   0.121369  -2.1497  0.031579 *
sd.price40   0.418148   0.128104   3.2641  0.001098 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Log-Likelihood: -2498.5

random coefficients
         Min.    1st Qu.     Median       Mean    3rd Qu. Max.
seat7    -Inf -1.0818780 -0.6422410 -0.6422410 -0.2026039  Inf
seat8    -Inf -1.0611428 -0.3900209 -0.3900209  0.2811010  Inf
cargo3ft -Inf  0.3436387  0.5508377  0.5508377  0.7580366  Inf
enghyb   -Inf -1.0337226 -0.9261449 -0.9261449 -0.8185673  Inf
engelec  -Inf -2.4883466 -1.8318636 -1.8318636 -1.1753805  Inf
price35  -Inf -1.2572890 -1.0813097 -1.0813097 -0.9053304  Inf
price40  -Inf -2.2738236 -1.9917870 -1.9917870 -1.7097505  Inf
```

The results show 14 estimated parameters, which is twice as many as we had in `m1`. These parameters describe the average part worth coefficients across the population of respondents (labeled `seat7`, `seat8`, etc.) as well as how those parameters vary across the population (reported as standard deviations and labeled `sd.seat7`, `sd.seat8`, etc.)

The standard deviation parameter estimates indicate that there is a lot of heterogeneity in preference for 7 or 8 seats over 6 seats. For example, the estimate of `sd.seat8` is about 0.995 larger than the mean estimate for the level of 0.39 which suggests that some people prefer 6 seats to 8, while others prefer 8. Another way to see this is in the output section labeled `random coefficients`, which shows the range of respondent-level coefficients. For `seat8`, the first quartile is $-1.06$ (indicating a preference for 6 seats) and the third quartile is 0.281 (indicating a preference for 8 seats). Because we specified the random coefficients as normally distributed, the model assumes that the majority of respondents are in the middle, slightly preferring 6 seats to 8. Given that there is a large fraction of respondents who prefer 8 seats, it may make sense for the company to offer a minivan with 6 seats *and* a minivan with 8 seats. To tell that for certain, you could make several share predictions and compare the potential increase in market share to the costs of offering both options.

There is one additional feature we can add to the random coefficients model using `mlogit()`. Model `m1` assumed that there were no correlations between the random coefficients, meaning that if one person prefers 8 seats over 6, we would not expect that they also prefer 7 seats over 6. Including correlations in the random coefficients allows us to determine, based on the data, whether people who like one attribute also tend to like another attribute. This is easily done by including `correlations = TRUE` as a parameter in the call to `mlogit` or by using the `update` function provided by `mlogit`.

```
> m2.hier <- update(m1.hier, correlation = TRUE)
> summary(m2.hier)
...
Coefficients :
                    Estimate Std. Error  t-value  Pr(>|t|)
seat7             -0.6571127  0.0730592  -8.9942 < 2.2e-16 ***
seat8             -0.4336405  0.0754669  -5.7461 9.132e-09 ***
enghyb            -0.9913358  0.0731532 -13.5515 < 2.2e-16 ***
engelec           -1.8613750  0.0855809 -21.7499 < 2.2e-16 ***
cargo3ft           0.6021314  0.0623728   9.6537 < 2.2e-16 ***
price35           -1.1819210  0.0770295 -15.3437 < 2.2e-16 ***
price40           -2.1749326  0.0960858 -22.6353 < 2.2e-16 ***
seat7.seat7        0.6830318  0.1046707   6.5255 6.776e-11 ***
seat7.seat8        1.0089934  0.1092730   9.2337 < 2.2e-16 ***
seat7.cargo3ft    -0.0624345  0.0962322  -0.6488 0.5164737
seat7.enghyb      -0.3517319  0.1146392  -3.0682 0.0021538 **
seat7.engelec     -0.1946944  0.0859581  -2.2650 0.0235131 *
seat7.price35      0.1318172  0.0973219   1.3544 0.1755947
...
```

Model `m2.hier` now includes many more parameters, so many that we have truncated the output. The additional parameters are the variance and covariance parameters between the random coefficients. `seat7.seat7` is the variance of the `seat7` random coefficient and `seat7.seat8` is our estimate of the covariance between preference for 7 seats and preference for 8 seats. The estimate is significant and positive, indicating that people who prefer 7 seats also tend to prefer 8 seats. To get a better sense of the strength of this association, we can extract the covariance matrix using `cov.mlogit` and then convert it to a correlation matrix using `cov2cor` from base R.

```
> cov2cor(cov.mlogit(m2.hier))
              seat7        seat8   cargo3ft       enghyb    engelec     price35
seat7     1.0000000  0.774540837 -0.1116303 -0.313095351 -0.4886197  0.24366546
seat8     0.7745408  1.000000000  0.1364164 -0.001877182 -0.2351177 -0.07335863
cargo3ft -0.1116303  0.136416377  1.0000000  0.498038677 -0.6257625 -0.03750020
enghyb   -0.3130954 -0.001877182  0.4980387  1.000000000  0.1096523  0.16249428
engelec  -0.4886197 -0.235117662 -0.6257625  0.109652292  1.0000000 -0.24671982
price35   0.2436655 -0.073358628 -0.0375002  0.162494277 -0.2467198  1.00000000
price40   0.1318966  0.034897989  0.3074710 -0.026541375 -0.4260211  0.54721196
...
```

This matrix shows that the correlation between the part worth for 7 seats and the part worth for 8 seats is 0.77, a strong association. In real data, it is common to find correlations between levels of the same attribute; if an attribute is important to a respondent, then he or she will likely have parameters with larger absolute values for all the levels of that attribute. For this reason, we strongly recommend that you include correlations in all random coefficients choice models. When you review the estimates of those models, you should review both the mean part worth coefficients, which represent the *average* value that respondents place on each attribute, *and* the variance and covariances in preferences across the population.

### 13.4.2 Share Prediction for Heterogeneous Choice Models

Reporting share predictions for heterogeneous choice models is largely the same as for standard choice models. The key difference is in how those share predictions are computed. The model assumes that there is a *population* of respondents, each with different part worth coefficients. So, when we compute shares, we need to compute the choice shares for many different respondents and then average over those to get our overall share predictions. You can see how we do this by comparing our prediction function for the hierarchical multinomial logit model to the prediction function we had for the standard multinomial logit.

```
> predict.hier.mnl <- function(model, data, nresp=1000) {
+   # Function for predicting shares of a hierarchical multinomial logit model
+   # model: mlogit object returned by mlogit()
+   # data: a data frame containing the set of designs for which you want to
+   #       predict shares.  Same format at the data used to estimate model.
+   # Note that this code assumes all model parameters are random
+   data.model <- model.matrix(update(model$formula, 0 ~ .), data = data)[,-1]
+   coef.Sigma <- cov.mlogit(model)
+   coef.mu <- m2.hier$coef[1:dim(coef.Sigma)[1]]
+   draws <- mvrnorm(n=nresp, coef.mu, coef.Sigma)
+   shares <- matrix(NA, nrow=nresp, ncol=nrow(data))
+   for (i in 1:nresp) {
+     utility <- data.model%*%draws[i,]
+     share = exp(utility)/sum(exp(utility))
+     shares[i,] <- share
+   }
+   cbind(colMeans(shares), data)
+ }
```

The key difference is that we now compute the shares for each of `nresp=1000` newly sampled, representative respondents. The part worths for these respondents are drawn from a multivariate normal distribution with mean set at our estimated value of `mu` and covariance equal to our estimated value of `Sigma` (`draws <- mvrnorm(n=nresp, coef.mu, coef.Sigma)`). The computation for each respondent is exactly the same as our computation in `predict.mnl`. Once we have the shares for all of the representative respondents, we average across respondents to get our overall share predictions.

We compute shares using `predict.hier.mnl` just as we did before with `predict.mnl`. It may take a moment, because we are doing 1,000 times more computation.

```
> predict.hier.mnl(m2.hier, data=new.data)
   colMeans(shares) seat cargo  eng price
8         0.08959674    7   2ft  hyb    30
1         0.46390066    6   2ft  gas    30
3         0.34231092    8   2ft  gas    30
41        0.05370156    7   3ft  gas    40
49        0.01797406    6   2ft elec    40
26        0.03251606    7   2ft  hyb    35
```

If you compare these share predictions to those we got with `predict.mnl(m1, data=new.data)`, you will see that they are similar, but not quite the same. For example, the electric minivan in the second-to-last row gets slightly more share with the heterogeneous model. Models that account for heterogeneity often predict that “niche” products attract a slightly larger share because the model accounts for the fact that there are a small number of respondents who find those “niche” designs very attractive. These models do not strictly follow the IIA property (see Sect. 13.3.4); if two similar products appeal to the same subset of customers, they will compete more closely with each other than with other products.

The share predictions produced by `predict.hier.mnl` are still based on the *point* estimates of `coef.Sigma` and `coef.mu`. So, while we have accounted for consumer heterogeneity in these predictions, we still haven’t accounted for our uncertainty in the parameter estimates. This makes it difficult to determine what would be a (statistically) meaningful difference in share for two alternative designs. While it is possible to estimate prediction intervals for these models in the frequentist framework, it is easier to do so in a Bayesian framework. We address prediction intervals for shares in the next section where we review Bayesian choice models.

## 13.5 Hierarchical Bayes Choice Models

In this section, we show how to estimate choice models with heterogeneity using Bayesian methods and point out advantages (and some disadvantages) of the Bayesian approach.

Moving into the Bayesian framework can be somewhat confusing, both because the Bayesian approach to estimation is different and because Bayesians often use different language to describe the same thing. Those who use the classical methods often refer to the model we estimated in the previous section as the "random-coefficients multinomial logit" or "mixed logit" model. Bayesians tend to refer to these same models (and some extensions of them) as hierarchical Bayes multinomial logit.

There are several available packages for estimating choice models using Bayesian methods. The `MCMCpack` package we used in Chap. 7 includes a function called `MCMCmnl` to estimate non-hierarchical multinomial choice models [109]. To estimate the hierarchical choice model here, we use the `ChoiceModelR` package [143], which builds on the `bayesm` package [136].

### 13.5.1 Estimating Hierarchical Bayes Choice Models with `ChoiceModelR`

Unfortunately, there isn't a universal standard for how choice data is stored and we have to reorganize our data slightly to use `ChoiceModelR`. `ChoiceModelR` requires the data to be stored in a "long" data frame where each row is an alternative (as we have already in `cbc.df`), but it requires the selected alternative to be stored as an integer number on the first row of each choice task, with zeros in the remaining rows. It turns out that it takes just a few lines of code to create the new choice data.

```
> choice <- rep(0, nrow(cbc.df))
> choice[cbc.df[,"alt"]==1] <- cbc.df[cbc.df[,"choice"]==1,"alt"]
> head(choice)
[1] 3 0 0 2 0 0
```

Since there are three alternatives in each question, the first element of `choice` indicates that the respondent chose the third alternative in the first choice task; the second and third elements for `choice` are left as zeros. Similarly, the fourth elements indicates that the respondent chose the second alternative in the second choice task, and the next two elements are zeros.

`ChoiceModelR` automatically codes factors but it uses a different scheme than `mlogit`. To be consistent with the models we've run before, we'll go ahead and code the factors manually ourselves using `model.matrix`.

```
> cbc.coded <- model.matrix(~ seat + eng + cargo + price, data = cbc.df)
> cbc.coded <- cbc.coded[, -1] # remove the intercept
```

Finally, we can create a new data frame that combines the coded attributes and the choice back together with the `resp.id`, `ques` and `alt` (which are the first three columns in `cbc.df`).

```
> choicemodelr.data <- cbind(cbc.df[,1:3], cbc.coded, choice)
> head(choicemodelr.data)
   resp.id ques alt seat8 enghyb engelec cargo3ft price35 price40 choice
19       1    1   1     0      0       0        0       1       0      3
12       1    1   2     1      1       0        1       0       0      0
4        1    1   3     0      0       0        1       0       0      0
1        1    2   1     0      0       0        0       0       0      2
23       1    2   2     0      0       0        1       1       0      0
31       1    2   3     0      0       1        0       1       0      0
```

The function we use to estimate the hierarchical Bayes choice model is `choicemodelr()`, which requires the data to be organized in exactly the format above: a number indicating which respondent answered the question, a number indicating which question the profile belongs to, and a number indicating which alternative this was, and then the attributes followed by the choice. The choice is stored as an integer number in the first row of each question.

A key advantage of the hierarchical Bayes framework is that it allows you relate a customer's part worths for the attributes to characteristics of the customer (sometimes called "demographics," although this is a very poor name as we discuss later.) In our data set, we happen to know whether each customer uses his or her car to carpool and it seems quite reasonable that people who carpool might have different part worths than people who don't carpool. To figure out whether this is true, we estimate a model where the part worths are a function of the respondent characteristics, following a linear model. Of course, this means we need to pass the data on the respondent characteristics to `choicemodelr()`, which expects this data to be formatted as a matrix with one row for each respondent and one column for each respondent characteristic.

```
> carpool <- cbc.df$carpool[cbc.df$ques==1 & cbc.df$alt==1]=="yes"
> carpool <- as.numeric(carpool)
> choicemodelr.demos <- as.matrix(carpool, nrow=length(carpool))
> str(choicemodelr.demos)
 num [1:200, 1] 1 0 0 0 1 0 0 1 0 0 ...
```

Note that each row in `choicemodelr.demos` represents a *respondent* and not a question or an alternative, and so we have 200 rows. A value of `1` indicates that the respondent does use their car to carpool and a value of `0` indicates that they don't.

With this bit of data re-organization done, we can call `choicemodelr()`:

```
> library(ChoiceModelR)
> hb.post <- choicemodelr(data=choicemodelr.data, xcoding=rep(1, 7),
+                         demos=choicemodelr.demos,
+                         mcmc=list(R=20000, use=10000),
+                         options=list(save=TRUE))
```

In addition to the `data` and the `demos`, there are a couple of additional parameters of `choicemodelr` that control the estimation routine. The `xcoding` parameter tells `choicemodelr` how you want the attributes coded; by setting this to a vector of `1`s, we indicate that we've already done the coding. The `mcmc` and `options` parameters control several aspects of the algorithm, which we discuss below. You can always type `?choicemodelr` for more details, although the help files might be more helpful to experienced Bayesian modelers than to novices.

While we recommend `ChoiceModelR`, there are a few aspects that make it less "R-like" than some packages. For example, `choicemodelr()` does not use R's formula notation or the base functions for coding factors. It relies on the order of the columns in the data frame, rather than using column names. The package does not include common utility functions like `summary()` and `predict()`. Because R is open source, it is up to each package development team to decide how they want to structure their functions and how consistent the package is with other R functions. While it sometimes requires a bit of work to figure out how a particular package works, it is difficult to complain too much, since the package was donated by the developers. If there is some functionality you'd like to see in a package, you can always write it yourself and then suggest to the package developers that they include your extension in their next release. (The name and email address of every package's maintainer are available in the package listing on CRAN.)

If you ran the code above, you probably noticed that it took a long time to run and produced a lot of output about its process. We omitted that output here. In the graphics window, you might have noticed something similar to Fig. 13.3.

What is Fig. 13.3? The focus of Bayesian inference is on generating a posterior distribution for the parameters of a model. The posterior distribution samples the likely values of a model's parameters, given the observed data. Most Bayesian routines like `choicemodelr` produce a set of random draws from the posterior distribution. Figure 13.3 is called a trace plot and it shows the posterior draws that have been produced so far by the estimation routine.

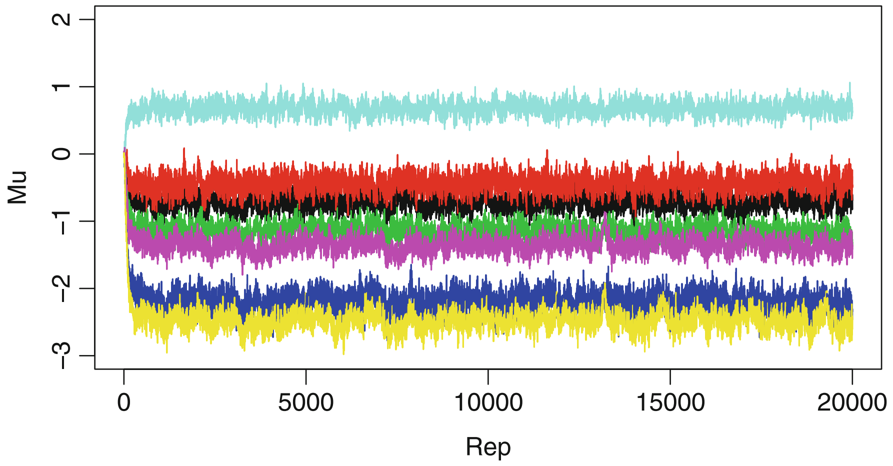

**Fig. 13.3.** Trace plot of posterior draws for a hierarchical Bayes choice model produced by `choicemodelr()`.

Without getting into the details of how and why these algorithms work, it is important to know that they don't always start out producing posterior draws. There is typically a *burn-in* period, where the algorithm settles into the posterior distribution. Before we use draws from the distribution, we have to throw out these initial burn-in draws. The trace plot allows us to see where the burn-in period ends. Judging from Fig. 13.3 the algorithm seems to have settled in after the first 1,000 draws, because that is where the lines plotting the estimated values settle into a stable, horizontal pattern (apart from noise).

We used two other arguments with `choicemodelr()`. The `mcmc=list (R=20000, use=10000)` argument tells `choicemodelr` that we want to produce 20,000 posterior draws and that we want to use only the last 10,000 (giving us a wide margin on the burn-in). The `options=list(save=TRUE)` argument tells `choicemodelr()` to save those last 10,000 posterior draws. By default, `choicemodelr()` saves every tenth draw, so it actually stores 1,000 posterior draws.

When `choicemodelr()` finishes, the posterior draws are saved to the object we specified, `hb.post`, which becomes a list with four elements:

```
> names(hb.post)
[1] "betadraw"  "deltadraw" "compdraw"  "loglike"
```

The key parameters of the model are the average and the variance of the part worths across the population. We can access one posterior draw of these parameters by selecting an element of `hb.post$compdraw`. We arbitrarily look at draw 567.

```
> hb.post$compdraw[[567]]$mu
[1] -0.6565015 -0.4263809 -1.1496282 -1.8733265  0.5620929 -1.2089470
[7] -2.5772394
```

These are the average population part worths and you can compare them to the parameters we estimated with `mlogit()`. The parameters above come in the same order as the first seven parameters estimated by `mlogit()`. For example, the average part worth for `seat7` was `-0.642` when we estimated it with `mlogit()` and for this posterior draw we get a value of `-0.657` from `choicemodelr()`.

There is one key difference between this model and the model we estimated with `mlogit()`. The parameters above represent the average part worth parameters among *respondents who do not use their car to carpool*. The hierarchical Bayes model also includes a set of "adjustments" for people who carpool; we can look at the 567th draw of these adjustment factors by looking at the appropriate row of `hb.post$deltadraw`.

```
> hb.post$deltadraw[567,]
[1]  1.63415698  1.78079508 -0.04400289 -0.12966126  0.02713614 -0.21926035
[7] -0.19518696
```

You can see that there are huge adjustments in the part worths for the first two parameters: `seat7` and `seat8`. The average part worth for 7 seats (versus the base level of 6) for people who carpool is $-0.657 + 1.634 = 0.977$. This means that on average people who carpool actually *prefer* 7 over 6 seats while people who don't carpool prefer 6 seats on average. This is a potentially critical insight for product designers that we completely missed when we used the mixed logit model with `mlogit`. You may not see a major difference in share predictions between these two models, but the insight you get from reviewing the parameters can be quite valuable.

We caution readers that this potential insight comes at a cost. We had to estimate seven additional parameters to describe the population. Adding a large number of additional parameters can make the burn-in period longer and it can add to uncertainty to the parameter estimates. We suggest you only include respondent characteristics that you believe should be related to the part worths. In this case, it seems reasonable that minivan preferences should be different for people who carpool.

In general, covariates that are directly related to product usage are ideal. There are also potential issues with the scaling of these respondent characteristics; binary indicators tend to work well as they avoid these scaling issues. It is generally a bad idea to include general demographic variables like age, race, or gender, just because you have them. Often demographic variables are not associated with product preferences [47]. For this reason, we avoid referring to covariates as "demographics."

We also have a set of parameters that describe the variance in part worths across the population. We can pull out the 567th draw from `hb.post`:

```
> hb.post$compdraw[[567]]$rooti
         [,1]        [,2]         [,3]       [,4]         [,5]         [,6]
[1,] 1.011972 -0.04367931 -0.045915749  0.1129225  0.005337426 -0.081773492
[2,] 0.000000  0.81711384  0.003180727  0.1535246  0.161135551 -0.192515134
[3,] 0.000000  0.00000000  1.011131267 -0.6173136  0.216005822 -0.011914137
[4,] 0.000000  0.00000000  0.000000000  0.8302175 -0.018093331 -0.002099283
[5,] 0.000000  0.00000000  0.000000000  0.0000000  1.146767704 -0.023986760
[6,] 0.000000  0.00000000  0.000000000  0.0000000  0.000000000  1.332875325
[7,] 0.000000  0.00000000  0.000000000  0.0000000  0.000000000  0.000000000
...
```

This set of parameters is actually stored as the Cholesky root of the covariance matrix. This is the matrix equivalent of a square root and we can recover the covariance matrix by "squaring" `rooti` with `crossprod()`:

```
> crossprod(hb.post$compdraw[[567]]$rooti)
             [,1]         [,2]         [,3]        [,4]         [,5]
[1,]  1.024087812 -0.044202250 -0.046465463  0.11427448  0.005401327
[2,] -0.044202250  0.669582906  0.004604584  0.12051469  0.131432954
[3,] -0.046465463  0.004604584  1.024504812 -0.62888172  0.218677697
[4,]  0.114274477  0.120514688 -0.628881721  1.10665848 -0.123023753
[5,]  0.005401327  0.131432954  0.218677697 -0.12302375  1.388055204
[6,] -0.082752504 -0.153734971 -0.008904404 -0.03317798 -0.061500274
[7,]  0.014590162  0.073588053 -0.233576644  0.47484770  0.281647756
...
```

The diagonals of this matrix describe the variance across the population in the part worths and if you compare them to the estimates we got with `mlogit`, you will find that variation across the population is generally smaller, particularly for the first two parameters that describe preferences for number of seats. The reason for this is that the new model accounts for some of the differences between individuals who carpool versus not, so the remaining unexplained variation between respondents is smaller.

In addition to population level parameters, we look at posterior draws of the individual-level parameters:

```
> head(hb.post$betadraw[,,567])
           [,1]       [,2]       [,3]        [,4]       [,5]       [,6]  ...
[1,]  1.0112255  0.6282393 -0.2210578 -0.01774596  0.8777881 -0.8889406 ...
[2,] -2.1737290  1.2036846 -2.2063721 -3.21025972  0.3277637 -2.8757266 ...
[3,] -2.4349625 -1.5172192 -0.7548992 -0.76935985 -0.2273173 -1.2754315 ...
...
```

Each row of this output represents the part worths for each person, which you can see vary widely. For example, for this posterior draw the first respondent really likes 7 seats over 6 or 8, since 1.011 is larger than 0 or 0.628. The second respondent prefers 8 seats over 6 or 7. You could plot histograms of these part worth values to get a sense for how preferences vary across the population (Sec. 9.4.3).

Up to this point, we've been talking about a single draw from the posterior (number 567). But if you look at `hb.post$betadraw`, you can see that there are 1,000 posterior draws of the seven part worths for each of 200 respondents.

```
> str(hb.post$betadraw)
 num [1:200, 1:7, 1:1000] 0.816 -1.083 -2.306 -0.91 2.043 ...
```

To fully characterize the posterior and our uncertainty about these parameters, we need summarize *all* of the posterior draws. Unfortunately, `choicemodelr` does not provide convenient summaries, but for the respondent-level `betadraws`, we can find the posterior means using `apply`.

```
> beta.post.mean <- apply(hb.post$betadraw, 1:2, mean)
> head(beta.post.mean)
           [,1]        [,2]       [,3]      [,4]      [,5]       [,6]
[1,]  0.6333957  0.26137739 -0.4483550 -1.426702 1.1568186 -0.4751817
[2,] -2.1134244  0.64602926 -1.0926311 -1.916450 0.9893599 -1.4853397
[3,] -1.9297260 -2.10100104 -0.9827285 -1.413800 0.4395638 -1.4189584
...
```

The values in `beta.post.mean` show our best estimate for each individual's part worths. While it is possible to obtain individual-level estimates using classical methods, it is much more common for Bayesian choice modelers to focus on individual-level parameters.

It is also important to recognize that with just 15 choice questions for each respondent, there is still a great deal of uncertainty about those individual-level part worths. We can get a sense for how much uncertainty there is by looking at the posterior quantiles of the part worths for each respondent. We compute the fifth and 95th quantiles of the individual `betadraws`, then display the mean and quantiles for the first respondent:

```
> beta.post.q05 <- apply(hb.post$betadraw, 1:2, quantile, probs=c(0.05))
> beta.post.q95 <- apply(hb.post$betadraw, 1:2, quantile, probs=c(0.95))
> rbind(q05=beta.post.q05[1,], mean=beta.post.mean[1,], q95=beta.post.q95[1,])
           [,1]       [,2]       [,3]         [,4]       [,5]       [,6]
q05  -0.5380902 -1.1223590 -1.5063476 -2.876466010 0.05096963 -1.5266892
mean  0.6333957  0.2613774 -0.4483550 -1.426702290 1.15681862 -0.4751817
q95   1.8404189  1.5819603  0.6093953  0.001986669 2.36650000  0.6431527
...
```

These numbers represent how much uncertainty we have in our estimates of the first respondent's part worth estimates. Roughly, the range of likely values for respondent 1's preference for 7 seats over 6 (the first parameter) is about $-0.538$ to 1.840. In other words, given this data, we can say that our best guess is that respondent

1 prefers 7 seats (i.e., has a positive coefficient), but it is quite possible that the respondent prefers 6 (i.e., has a negative coefficient). This is a huge amount of uncertainty that we need to account for when making share predictions.

### 13.5.2 Share Prediction for Hierarchical Bayes Choice Models

The 1,000 posterior draws in `hb.post$betadraw` give us a sense of the range of part worth values that each respondent might have and we can use these draws to figure out the likely range of shares that we might get for new vehicle designs. For each posterior draw, we can compute the shares for a new set of product designs based on the values of the part worth coefficients for that draw. Each time we do this, the shares we obtain represent a posterior draw for the shares. (Being able to compute posterior draws for any function of the parameters in this way is one of the great advantages of Bayesian MCMC.) We can compute the shares for a number of different posterior draws (selected at random from the draws that we produced when we called `choicemodelr`) and then analyze the range of shares that we get.

We create a function for computing shares that loops over both the respondents and the posterior draws:

```
> predict.hb.mnl <- function(betadraws, data) {
+   # Function for predicting shares from a hierarchical multinomial logit
    model
+   # betadraws: matrix of betadraws returned by ChoiceModelIR
+   # data: a data frame containing the set of designs for which you want to
+   #       predict shares.  Same format at the data used to estimate model.
+   data.model <- model.matrix(~ seat + eng + cargo + price, data = data)
+   data.model <- data.model[,-1] # remove the intercept
+   nresp <- dim(betadraws)[1]
+   ndraws <- dim(hb.post$betadraw)[3]
+   shares <- array(dim=c(nresp, nrow(data), ndraws))
+   for (d in 1:ndraws) {
+     for (i in 1:nresp) {
+       utility <- data.model%*%betadraws[i,,d]
+       shares[i,,d] = exp(utility)/sum(exp(utility))
+     }
+   }
+   shares.agg <- apply(shares, 2:3, mean)
+   cbind(share=apply(shares.agg, 1, mean),
+         pct=t(apply(shares.agg, 1, quantile, probs=c(0.05, 0.95))),
+         data)
+ }
```

The inner loop in this function for `i in 1:nresp` computes the shares for each respondent for a given posterior draw. The outer loop for `d in 1:ndraws` loops over the posterior draws. (If there were too many posterior draws, we could also use a random subset of them.) The function stores the share estimates for each user for each draw in `shares`. In the last few lines, the function averages the shares across respondents resulting in an estimate of the shares for each posterior draw. We then compute the mean as well as the quantiles of those posterior draws to get a sense for the likely range of shares.

When we compute the shares in this function, we use the estimated individual-level part worths for the respondents in our data, which is what most analysts do in practice. In contrast, when we computed share predictions in the previous section using the output from `mlogit()`, we sampled new representative respondents based on our estimates of the population mean and covariance. We should point out that it is possible to use the same approach with a Bayesian choice model, using the posterior draws of `mu`, `delta`, and `rooti` and sampling a new set of respondents from the multivariate normal distribution. This would require a relatively small change to `predict.hb.mnl`.

When we apply `predict.hb.mnl` to the designs in `new.data`, we get both point estimates and ranges of potential shares for each design:

```
> predict.hb.mnl(hb.post$betadraw, new.data)
        share       pct.5%    pct.95% seat cargo  eng price
8  0.09920353 0.086505556 0.11352010    7   2ft  hyb    30
1  0.45300946 0.428510287 0.47765666    6   2ft  gas    30
3  0.32986260 0.305608282 0.35368987    8   2ft  gas    30
41 0.06947448 0.056391010 0.08434871    7   3ft  gas    40
49 0.01357954 0.009832746 0.01800444    6   2ft elec    40
26 0.03487038 0.028179167 0.04215179    7   2ft  hyb    35
```

There is quite a bit of uncertainty in these share predictions. Our “best guess” estimate of the shares for the 6-passenger base engine minivan (number 1 in the second row) is 45.3 %, but it could be as low as 42.9 % or as high as 47.8 %. Understanding the uncertainty in our model predictions helps in interpreting differences in share. For example, if we make a minor change such that 6-passenger gas minivan share increases to 46.0 %, we would recognize that this change in share is well within the prediction error of our model, and we probably shouldn’t make strong statements that one design will do better than the other in the marketplace. But if we change the seating to 7 passenger for that vehicle, the predicted share is 30.2 % with a range of 28.2 % to 32.1 %. Because the prediction intervals do not overlap, we can say that the 7-passenger version of the design has significantly lower share, knowing that we are not over-interpreting the limited data that we have.

Given how easy it is to compute these share prediction ranges, we think it is surprising how rarely practitioners report prediction intervals of choice model shares. Many conjoint analysis studies unfortunately report only point estimates of share predictions. This leaves decision makers blind to the possibility that the share predictions they are relying on may not be very accurate. In extreme cases, when there are many attributes in the model and very few choice questions, one may find that the prediction intervals are extremely wide. This is an indication that there isn’t sufficient data to make precise predictions and suggests that one might wish to collect more or different data. In the next section, we discuss the design of choice-based conjoint surveys.

## 13.6 Design of Choice-Based Conjoint Surveys*

Once you start looking at parameter estimates and share predictions for your choice models, you may start to wonder how you can make your parameter estimates and prediction intervals tighter. The easiest way to do this is to increase the amount of data you collect, either by increasing the number of respondents or by increasing the number of questions that you ask each respondent. If you were to recreate data as we've used in this chapter with 1,000 respondents instead of 200, you would see that the standard deviations for the parameter estimates and the prediction intervals would be smaller (as is true for any model).

A good way to assess sample sizes before fielding a conjoint analysis project is to simulate data from known parameters, estimate the model from the synthetic data, and examine the resulting prediction intervals. This would only require a few changes to the code presented in this chapter. Such analysis can help you determine how many respondents you need for a given number of attributes and levels, or, as is more often the case, how many attribute and levels you can afford given your available budget for collecting data.

Beyond getting more data, choosing the right questions to ask can also result in more precise parameter estimates and share predictions. The selection of questions to include in the conjoint survey is example of an *experimental design* problem.

If you review the code in Sect. 13.2, you will notice that when we generated the conjoint questions we selected a different set of minivan profiles *at random* to create the choice questions for each respondent. This approach works well and is robust, as long as you can give a different set of questions to each respondent. If your survey platform is limited so that every respondent must answer the same questions, a random design will not be very efficient. There are several other approaches you can use that can improve upon selecting questions randomly. The main design approaches are the following.

- **Random designs** use a randomly selected set of profiles in each question.
- **Fractional factorial designs** are based on the assumption that there are no interactions between different attributes (e.g., there isn't some additional boost to having 8-seats *combined* with 3 ft of cargo space). Many of the advantages of fractional factorial designs, such as orthogonality, are only beneficial in the context of linear models and not choice models. However, fractional factorial designs are occasionally used in practice because they are readily available and were once the standard approach for conjoint analysis surveys. These designs are often constrained so that every respondent answers the same questions, or such that there are only a few survey versions.
- **Optimal designs** are created by selecting a set of questions that minimizes the standard error of the estimated parameters (called D-Optimal designs) or minimizes the standard error of share predictions (called G-Optimal designs). These designs are created by starting with an arbitrary design and then iteratively

changing questions and assessing whether those changes make the sampling error or posterior intervals smaller. The routines may not always produce the true optimal design, but they can often improve substantially on the starting design. These designs may also be constrained so that every respondent answers the same question.

- There are a number of **heuristic conjoint design strategies** that aren't based on a formal theory, but have produced good quality predictions in the past.
- **Adaptive designs** select successive choice questions based on the respondent's answers as he or she takes the survey. For instance, a survey might ask about preferred options and then focus on the features that a respondent identifies as important. One approach is called fast-polyhedral conjoint design [154]. Another method is Adaptive Choice-Based Conjoint (ACBC) from Sawtooth Software [142].

There is much debate in the conjoint analysis community about which of these methods is the best. While we can't answer that, we can say that each has worked well in at least some conditions. A common mistake when comparing these conjoint design methods is to look at whether different methods result in different point estimates of the share predictions (either in-sample or for holdout questions). Since many of these methods produce similar point estimates for shares, that is a poor way to compare different experimental design strategies. A better approach is to compare the prediction *range* between approaches. Better designs produce smaller prediction ranges, meaning that there will be less uncertainty in predictions.

Unfortunately, there aren't yet readily available tools for the design of choice experiments in R. The `AlgDesign` package can produce fractional factorial and optimal designs, but it isn't customized for choice models and the package is no longer being maintained. For our own work, we tend to use random designs (which are easy to produce in R) or use other software to create designs. JMP includes routines for creating D-Optimal designs for choice models. Sawtooth Software offers a variety of heuristic and adaptive design strategies for choice models.

## 13.7 Learning More*

In this chapter, we have given a brief overview of choice modeling in the context of conjoint analysis surveys, with examples of how to estimate choice models in R. For those who want to learn more about choice modeling, there are many additional resources, although no single text covers everything of importance.

For those who are interested strictly in conjoint surveys, Orme's *Getting Started with Conjoint Analysis* [122] offers an accessible introduction to how conjoint surveys are constructed and analyzed in practice while Louvier, Hensher, and Swait's *Stated Choice Methods* [106] provides a more extensive (but slightly dated) overview of the topic, including coverage of several variations on non-hierarchical multinomial logit models and how to create fractional factorial designs. Rossi, Allenby, McCulloch's *Bayesian Statistics and Marketing* provides technical coverage of the multinomial and hierarchical multinomial logit model from the Bayesian perspective and describes the `bayesm` package that `ChoiceModelR` uses heavily.

As we mentioned in the introduction, there are uses of choice models other than choice-based conjoint analysis surveys. One broad application area is the modeling of consumers' transportation choices. Kenneth Train (2009) offers a clear and concise overview of discrete choice methods [155] and their use in transportation economics, including coverage of both the mixed logit and hierarchical Bayes logit models; it is an ideal introduction for those using hierarchical choice models in nearly any context. Train also covers a number of alternative choice models including the nested logit model and the multinomial probit model. While there are advantages and disadvantages to each of these models, they are all based on the same premise that customers choose among a set of products based on part worths of the attributes of those products.

Another major application for choice models in marketing is to understand how consumers choose products in retail stores, such as grocery stores. Using data collected by grocery store scanners where customers are tracked over multiple visits (called *scanner panel* data), one can assemble observations that are nearly identical in structure to conjoint data [67]. Many marketing academics have used choice models with such data to assess the relationship between marketing actions such as price, promotion, and in-store display, and customers' product and brand choices. Much work has been published on extending these models to accommodate different types of consumer behavior such as stockpiling goods, learning about products, strategically trying new products, and changes in preferences over time.

## 13.8 Key Points

- Choice models are used to understand how product attributes drive customers' choices. The most popular choice model in practice is the multinomial logit model. This model can be estimated using frequentist methods with `mlogit` or using Bayesian methods with `MCMCmnl` (Sect. 13.3).
- Choice data can be stored in "long" or "wide" formats and there is no universal standard for how the data should be organized. Before you use any choice modeling package, read the documentation carefully to understand how the package expects the data to be formatted (Sect. 13.3).
- Before analyzing any choice data, it is useful to compute raw choice counts for each attribute. This can be done very easily using `xtabs`. (Sect. 13.3.1)
- Estimating a choice model is similar to estimating simpler linear models. The key output of the estimation is a set of parameters that describe how much each attribute is associated with the observed choices.

- Choice models can include both factors and numeric attributes in the choice alternatives. When you use a factor as a predictor, the factor has to be dummy coded, just as it would for a linear model. With dummy coding, the estimates are interpreted as the preference for a particular level of the attribute *relative* to the base level of the attribute (Sect. 13.3).
- Most choice models do not include intercepts. When a choice models does include intercepts, there is an intercept for each alternative in the choice questions; these are called alternative specific constants or ASCs (Sect. 13.3).
- When reporting choice models, it is best to focus on reporting *share predictions* from the model because parameter estimates are difficult for non-experts to interpret. If you model price as a numeric predictor, you can also report the willingness to pay for each attribute (Sect. 13.3.3).
- Heterogeneous choice models allow each respondent to have individually estimated part worths. This may result in share predictions that are slightly (and appropriately) higher for "niche" products (Sect. 13.4.2).
- Hierarchical choice models can be estimated using frequentist methods with `mlogit` and with Bayesian methods using `choicemodelr` (Sects. 13.4 and 13.5).
- Bayesian methods produce draws from the posterior distribution of the parameters. To understand the uncertainty in the parameters (given the data), examine the range of the posterior draws. To find the uncertainty in predicted shares, compute the share values for each posterior draw of the estimated parameters. The range of share estimates indicates the uncertainty in share predictions (Sect. 13.5).
- Bayesian methods allow you to incorporate an upper level model that relates respondent characteristics to attribute preferences. Good candidates for respondent characteristics are binary variables that describe product usage. (Sect. 13.5)
- In general, if you collect more data, your estimates of the parameters will be more precise and your prediction intervals will be smaller. Prediction intervals can also be made smaller by selecting better choice questions. There are several alternative approaches to choosing profiles to include in choice questions (Sect. 13.6).