Class 3: Lab; model selection

To do for tomorrow (Friday)
- Read Anttila 1997
  - Write and turn in answers to study question(s) posted on course webpage (under "Readings").

Overview: Lab to practice regression and contingency tables; model selection: how can you choose between two different sets of predictors?

1 Getting set up

- Download and install R (http://www.r-project.org/). When the computer asks you where to install it, choose your user folder or the desktop.
- Open R. Try some commands on the command line.
- It is very tedious to re-type your commands every time you want to make a change. So we will use a **script file**.
  - `File > New script` should make a new, blank window appear.
  - Type a **comment** at the top like `##This is my script for our first in-class lab`
  - Save the file. Give it the extension `.R`
  - Now type some commands in the script file, one per line. Nothing happens yet.
  - To run a command in your script, click anywhere on that line, and press (in Windows) `CTRL-r`
  - The command now appears in the original console window and is executed. The cursor moves to the next line of your script.
  - You can also use the mouse to select part of the line, or multiple lines, and press `CTRL-r` to run what you selected. Try it.

Now we're ready to do some linear regression!

2 Linear regression: setup

- Go to the course webpage and follow the link to download the Hayes/White acceptability judgments data set.
- Take a look at the file. You might want to open it in Excel.
- Follow the other link on the course webpage to Bruce Hayes's webpage about the paper that these data come from, to see what the columns in the file mean.
- Find out where you stored the file. On my computer, it's `C:\Users\Kie\Documents\My Dropbox\TEACHING_SNUSeminar_2012\03_HayesWhite_SubjectData.txt`
- To read in the data, you need to add this command to your script:
  - `hayes_white = read.table("`**`C:/Users/Kie/Documents/My\ Dropbox/TEACHING_SNUSeminar_2012/03_HayesWhite_SubjectData.txt`**`",header=TRUE)`
  - Except, replace the bold part with your own path
  - Notice that I had to change all the "\" to "/"
  - Because my path had a " " (space) in it, I had to add a "\" before it
  - "`header=TRUE`" tells R that the first line of the file is the names of the columns
  - Now the contents of the file are in a table called `hayes_white`.
- To look at the `hayes_white` table, type `hayes_white`.
- To see just the first few lines, try `head(hayes_white)`
- Now try `summary(hayes_white)` and see if you can understand the summary of each column
- Try `hayes_white$LogResponse`. This is how you access the column `LogResponse` of the table.

3 Linear regression: plotting

- Let's try plotting `LogResponse` as a function of `Albright2009`:
  - `plot(hayes_white$Albright2009, hayes_white$LogResponse)`
  - Try plotting `LogResponse` as a function of some other variables

Especially if you have just one independent variable, you should always plot your data. It will help you see if a linear model is suitable

- maybe there is a strong trend, but it's not shaped like a line
- maybe there are some extreme outlying points
- maybe the points are much more or less spread out on certain parts of the *x*-axis

4 Linear regression: one independent variable

- Now let's try making a regression model where `Albright2009` predicts `LogResponse`:
  - `hayes_white.lm = lm(hayes_white$LogResponse ~ hayes_white$Albright2009)`
  - `summary(hayes_white.lm)`
  - `lm()` is a function that makes a **l**inear **m**odel
  - Based on your results, write out the regression equation.
  - Notice that we didn't have to mention the intercept in our original `lm()` command. R includes an intercept automatically.
  - Try plugging in some different values of `Albright2009` to see the predicted values of `LogResponse`. It might be easier to do it in Excel than with a calculator.

- We can add the regression line to the plot:
  - `plot(hayes_white$Albright2009, hayes_white$LogResponse)`
  - `abline(hayes_white.lm, col="blue")`
- Try it with different predicting factors (independent variables)
- Try it with an independent variable that is binary (use `summary(hayes_white)` again to find a binary variable). What does the resulting model mean?
  - The binary variables in this data set have values like `natural` and `unnatural` instead of 0 and 1. R automatically assigns 0 to the value that comes first in alphabetical order (`natural`) and 1 to the other one (`unnatural`)
  - Write out the regression equation and plug in different values for your independent variable.

5 Linear regression: multiple independent variables

- Now try a model with more than one predictor:

  -
    ```
    hayes_white.lm = lm(hayes_white$LogResponse ~
            hayes_white$TestOrControl +
            hayes_white$Naturalness)
    ```
  - `summary(hayes_white.lm)`
  - Can you guess why there is a "+" in the R command?
  - Write out the regression equation (this may help you understand the "+") and plug in a couple of different values for both independent variables.
  - You can even try three or four independent variables—play with it.

6 Linear regression: interactions

- Now try an **interaction**.
  - Instead of asking the computer to find the best values of *a, b, c* for $y = a + bx_1 + cx_2$,
  - ...we ask it to find the best values of *a,b,c,d* for $y = a + bx_1 + cx_2 + dx_1{*}x_2$

- Here is how to do it in R: replace "+" with "*":

  -
    ```
    hayes_white.lm = lm(hayes_white$LogResponse ~
        hayes_white$TestOrControl *
        hayes_white$Naturalness)
    ```
  - `summary(hayes_white.lm)`
- This might seem like we are asking for just $y = a + dx_1{*}x_2$, but R automatically includes bx_1 and cx_2.
- Write out the resulting regression equation and plug in some different values to help you understand what $dx_1{*}x_2$, the **interaction term**, does.

- Play with including other factors and other interactions

7 Logistic regression: setup

- Download the Hungarian dataset from the course web page. It's partly based on Hayes & al. 2009, and partly on the Hungarian WebCorpus (Halácsy & al. 2004, Kornai & al. 2006)

- Discovered in class: your computer might not like the Hungarian accented vowels
  - If you have problems, open the file again in Excel (open Excel, then use CTRL-O to open the file; **don't** drag the file into Excel).
  - Delete columns of words or vowels; keep just the columns of numbers.
  - Save under different name and use this new file in your read.table() command in R.

- Look at it in Excel.
- These are Hungarian noun stems of 2-4 syllables that end in a back vowel followed by 1 or 2 "neutral" vowels (front unrounded), and occur at least 5 times in the dative in the Hungarian National Corpus.
- Words like this can take either the back allomorph of the dative suffix, *-nak*, or the front allomorph, *-nek*.
  - The dependent variable is in the last column
  - I simplified by saying a word takes *–nek* (0) if it takes it 67% of the time or more; I said it takes *–nak* (1) if it takes it 67% of the time or more. The few words in the middle were deleted.
- There are several independent variables that could matter: number of neutral Vs, height and length of each neutral V, height and length of back V, frequency of word.
- Use `hungarian = read.table(...)` to read in the data, and `summary()` to have another look at it.

8 Logistic regression

- Set up a simple, one-factor logistic regression:

  -
    ```
    hungarian.glm = glm(hungarian$is_output_nak ~
        log(hungarian$total_frequency),
        family=binomial)
    ```
  - `summary(hungarian.glm)`
- The function `glm()` stands for **g**eneralized **l**inear **m**odel.
- `family=binomial` says we want a logistic regression.
- Notice that we didn't have to calculate the log frequency in our original data file—R can do it in the regression command.
- Write out the regression equation (remember it will be a **logistic** function: you may want to look at last time's handout), and plug in some values of log-frequency to see the predicted probability of taking *-nak* (even though the *p* values suggest that this is not a very good model).

- Now explore on your own for a while:
  - try some other independent variables and try to get a model with multiple significant factors
  - try some interactions
  - discuss with your neighbor and compare your models

If you get only this far, it's OK! But if you finish the above, below are some other things for you to try.

9 Contingency tables

- If you try to plot a binary variable vs. a binary variable you'll get a strange picture:
  - `plot(hungarian$number_of_neutral_Vs, hungarian$is_output_nak)`
- We can make a table instead
  - `table(hungarian$is_output_nak, hungarian$number_of_neutral_Vs)`
  - This counts up how many words with 1 neutral V have *–nek(0)* or *–nak(1)* and how many words with 2 neutral Vs have *–nek(0)* or *–nak(1).*
- To view the table in graphical form, use this command to make a **mosaic plot**
  - `mosaicplot(hungarian$number_of_neutral_Vs ~ hungarian$is_output_nak)`
  - Can you see how the mosaic plot relates to the table?
- The table seems to be skewed: we see more *–nak* when there is just one neutral vowel.
  - But is it significantly or surprisingly skewed?
- You can try making an observed/expected comparison first by hand in Excel or with a calculator.
  - fill in the lightly shaded cells with the observed values and totals

	1 neutral vowel	2 neutral vowels	*total*
nek (0)			
nak (1)			
total			*grand total:*

  - use the above to fill in the lightly shaded cells with the observed percentages

	1 neutral vowel	2 neutral vowels	*% of grand total*
nek (0)	■	■	
nak (1)	■	■	
% of grand total			■

  - get the expected percentages in the lightly shaded cells by multiplying the row and column percentages:

	1 neutral vowel	2 neutral vowels
nek (0)		
nak (1)		

  - multiply the expected percentages by the grand total to get the expected values:

	1 neutral vowel	2 neutral vowels
nek (0)		
nak (1)		

  - divide the observed values by the expected values to get observed/expected:

	1 neutral vowel	2 neutral vowels
nek (0)		
nak (1)		

- Significance? We could just do a logistic regression (try it!), but we can also do a "chi-squared" analysis, as long as all of our expected values are at least 5
  - `hungarian.chisq = chisq.test(hungarian$number_of_neutral_Vs, hungarian$is_output_nak)`
  - `hungarian.chisq`
  - What R did: sum up `(observed-expected)`^{2}`/expected` for each cell of the table, and look up the resulting number (along with "degrees of freedom", which in this case depends on how many rows and columns the table has) in a table to find the *p*-value
- `hungarian.chisq` is an **object** within the R programming language, so it has various properties of its own that we can examine. Try these and compare them to your tables above
  - `hungarian.chisq$observed`
  - `hungarian.chisq$expected`
  - and some arithmetic: `hungarian.chisq$observed / hungarian.chisq$expected`

10 Return to the Lieberman & al. irregular-verb data

- You can download the file through the link on the course web page (Class 2).
- Open it in Excel and replace any spaces (" ") with "_"
- We modeled whether verbs are irregular in modern English—try Middle English and Old English instead.
- Try including behavior at the previous stage as an independent variable—for example, predict modern English irregularity as a function of word frequency and Middle English irregularity.
- Remember you need to do logistic regression because the dependent variable is binary

11 Your own data?

- Do you have data with independent and dependent variables?
  - Use Excel to get it into an orderly spreadsheet, where each row in one observation and each column is one variable (dependent or independent).
  - Get rid of any spaces, question marks, parentheses, or other potentially strange symbols
- Save it as a .txt file and try opening it in R!
  - You'll probably get some error messages; I can come over in help
- Try a regression model

Part II of today's class: Model selection

12 Model selection: the problem

- Suppose we make two different regression models for Hungarian (I deleted some lines):

Three independent variables:

```
glm(formula = hungarian$is_output_nak ~ hungarian$last_V_height +
    hungarian$last_V_length + hungarian$B_height, family = binomial)

                        Estimate Std. Error z value Pr(>|z|)
(Intercept)               3.2014     0.5044   6.347  2.2e-10 ***
hungarian$last_V_height  -2.2333     0.2121 -10.528  < 2e-16 ***
hungarian$last_V_length   1.0714     0.3228   3.319 0.000902 ***
hungarian$B_height        0.3735     0.1687   2.214 0.026801 *
---
AIC: 295.36
```

Leaving out height of back vowel:

```
glm(formula = hungarian$is_output_nak ~ hungarian$last_V_height +
    hungarian$last_V_length, family = binomial)

                        Estimate Std. Error z value Pr(>|z|)
(Intercept)               4.0247     0.3786  10.629  < 2e-16 ***
hungarian$last_V_height  -2.1884     0.2070 -10.572  < 2e-16 ***
hungarian$last_V_length   1.1618     0.3190   3.642 0.000271 ***
---
AIC: 298.4
```

13 How can we choose between them?

- On the one hand, the model with more variables does a better job of **fitting the data**
  - it makes more-accurate predictions for our existing data, because it takes into account more information about each word.
  - Let's think about what would happen if we added to the model a factor that was totally uninformative. Could the fit get worse?
- But on the other hand, there is a danger of **overfitting**. I'm going to draw on the board for the next several minutes because it's hard to do this part on paper.
  - To summarize what just happened on the board: a model that fits the *existing* data too well, by adding too much complexity, could make worse predictions about *new* data.
- We need a measure that balances how well the model fits the existing data against how complex the model is—in this case, how many independent variables it has.

14 Statistical approach I: AIC

- The AIC you see in the regression results above (Akaike Information Criterion) is one such measure
  - You don't need to know this, but AIC = $2k - 2\ln(L)$
  - where k is the number of parameters (2 vs. 3 for our example)—complexity
  - and L ("likelihood") is how probable the observed data are in the model—goodness of fit
  - Which is better, a low value or a high value of AIC?
- Drawback: it just counts the number of independent variables. But some independent variables might be able to add a lot of complexity to the model and others little.

15 Statistical approach II: compare 2 models with ANOVA

- If one model's independent variables are a subset of the other's...
- If L_2 is the more-complex model's L and L_1 is the simpler model's, then take $2\ln(L_2 / L_1)$
  - This number can then be looked up in the chi-squared table, using the difference in number of independent variables as the "degrees of freedom", to obtain a p-value.
  - Interpretation of p-value: if the simpler model was correct, how probable is it to get such a big L_2 / L_1 by chance?
  - So a small p-value is evidence against the simpler model.
  - Again, we are only counting independent variables, not considering how powerful each one is.
- Fortunately, R can do all this with just one command:
  - `anova(hungarian.glm2,hungarian.glm1, test="Chisq")`

Again, you could take a whole course in model comparison in statistics! This is only the very basics.

16 Machine-learning approach: training set vs. development set vs. test set

- Now we depart from traditional statistics. If you already know statistics and have been bored by yesterday and today, here is something new!
- I got attracted to this approach by Andrew Ng's wonderful free online course in machine learning. I've included a link on the course web page.
- The idea is to test how well the model does on new data by *holding back some data* and then treating it as new.

Training set:

- Randomly select part of the data, say 60%. Fit each regression model to these data.

Development set:

- Of the remaining 40%, randomly select some, say half. See how well each regression model fits *those* data.

- Repeat until you've chosen your final model.

Test set:

- See how well your final model fits the remaining 20% of the data. Report this as your accuracy.

- In engineering applications, such as natural language processing, the last step (test set) is very important because we need to estimate the true accuracy.
- In theoretical linguistics, we might be able to skip it (and thus use more data for training and development), since we're less interested in how well the model performs and more interested in what is the "right" model.

- Especially if you have not very much data, people sometimes repeat the training-development steps:
  - E.g., divide your training-and-development data into 10 (or some other number) equal subsets.
    - Train on sets 1-9, evaluate on 10
    - Train on sets 1-8 and 10, evaluate on 9
    - Train on sets 1-7 and 9-10, evaluate on 8
    - etc.
  - Choose the model that does the best in overall on the 10 evaluations.
  - Or, for every single data item, train on the rest and then test on it.

- What I like about this approach is that it doesn't matter what kind of model we use: regression, stochastic OT, MaxEnt OT...
  - all we need is a measure of how well a model's predictions fit the data
  - for a continuous dependent variable, we can sum the squared errors (difference between actual and predicted)
  - for a binary dependent variable, number of incorrect classifications (model says 0 when it should be 1 or vice-versa) could work

17 Cross-validation for regression in R

- Fortunately, there are some packages in R that will automate cross-validate for us, at least for regression.
- One is the `boot` package (Davison & Hinkley 1997, Canty & Ripley 2012)
  - `Packages > Install packages...`
  - Choose Austria (or elsewhere) from the menu that appears
  - Choose `boot` from the menu that appears
  - Add this command to your script: `library(boot)`

    - For examples, type `help("cv.glm")`
- I had some code to show you for Hungarian but there seems to be a bug (I'm getting the same number for every model), so I'll show you on-screen but I don't want to put it in the handout because I don't want to infect you with my mistake.

References

Canty, Angel, and Brian Ripley (2012). boot: Bootstrap R (S-Plus) Functions. R package version 1.3-4.

Davison, A. C. & Hinkley, D. V. (1997) Bootstrap Methods and Their Applications. Cambridge University Press, Cambridge.

Halácsy, Péter, András Kornai, László Németh, András Rung, István Szakadát & Viktor Trón. 2004. Creating open language resources for Hungarian. Proceedings of the 4th international conference on Language Resources and Evaluation (LREC2004).

Hayes, Bruce and James White (2013/to appear). Phonological naturalness and phonotactic learning. *Linguistic Inquiry*.

Hayes, Bruce, Kie Zuraw, Péter Siptár, and Zsuzsa Londe. Natural and unnatural constraints in Hungarian vowel harmony. *Language* 85: 822-863.

Kornai, András, Péter Halácsy, V Nagy, Cs Oravecz, Viktor Trón & D Varga. 2006. Web-based frequency dictionaries for medium density languages.. In Adam Kilgarriff & Marco Baroni (eds.), Proceedings of the 2nd International Workshop on Web as Corpus ACL-6, 1–9.

Maindonald, John and W. John Braun (2012). DAAG: Data Analysis And Graphics data and functions. R package version 1.15. http://CRAN.R-project.org/package=DAAG