Homework #2

Intro. to Computational Modeling

Spring 09

In this homework, we are going to dive into some issues of model fitting and verification. The goal is that by doing the homework you would develop some useful code that would let you more or less plug in[1] a model that you might come across in your research, fit it to data, and verify that the fits are good, etc... The plan is to stick with the exemplar and prototype models we have looked at so far as they are simple models to evaluate and understand, but this exercise applies equally well to a different model with more parameters, etc..

1 Generating Fake Data

To start with, we are going to need to generate fake choice data for our experiment. Thus, step 1 involves writing up the code that is necessary for this. The code I gave you provides two python function **exemplarmodel()** and **prototypemodel()**. If you call these functions you get a list of predicted response probabilities for each stimulus type (in order from left to right: [prototype, low, high, random, old]). We want to generate fake subject data on the basis of these probabilities. To do this we need to know something about the structure of the experiment. Remember from lecture one our "fake experiment" had the following number of test trials for each stimulus class:

- 4 prototypes
  2 low distortions
- 10 high distortions
- 20 random items
- 10 old items presented twice each

Thus, in order to make fake data we want to use the probabilities generated by the model, and create data vectors of fake data with a realistic number of trials. Here's a hint:

[1] Although modeling should never be done so blindly!!

```
from numpy import *
from random import random

def generatedata(probs, ns):
        subjectdata = zeros(range(len(probs)))
        for stim in range(len(probs)):
                for trial in range(ns[stim]):
                        if random() < probs[stim]: # flip the biased coin
                                subjectdata[stim]+=1
        return subjectdata[:]
```

Ok, so a little more than a hint, but this should get your started in making fake data from a set of probabilities according to the number of trials in the experiment. You'll want to use this function to generate lots of data from subjects for later on. You can save these in a big python list or write to disc, your choice.

2 Basic Fitting

The code that I gave you applies the exemplar model to a given set of human data and can return a goodness-of-fit measure based on RMSE (root-mean squared error). Your second step should be to implement the same thing for fitting the prototype model using RMSE as the fit metric. This should be a simple extension of the functions I provided (like 1 line).

3 Verify the fitting method to the "group" data

The first sanity check when doing parameter fits with a model is to make sure that you can effectively recover the parameters in an identifiable way. So, for this step, choose some parameters for both the prototype and exemplar models and use the functions exemplarmodel() and prototypemodel() to get the predicted "group" response curves for that particular set of parameters (you don't need the code from step 1 for this, but will use it later).

Next, use that data as your human data and see if you can fit the prototype-/exemplar models and recover the same parameters you used to generate the data (using the same model that generated the data to recover the parameters). In other words, use the prototype and exemplar model function to generate a fake data curve, then, using that data, refit it using the example *fmin* code I provided to see how closely you can get back your parameter. You should try this for a number of different parameter values to evaluate the recoverability within different regions of the parameter space. You can do this programmatically by creating a grid of values along each parameter (c, k):

```
cgrid = arange(0,6.0,0.01)
kgrid = arange(0,2.0,0.01)
```

```
for c in cgrid:
        for k in kgrid:
                # generate date and then refit it
```

then evaluate how closely you can recover the parameters for each combination of (c,k) you try from above. One way to summarize how close you are would be something like the RMSE between the actually values used to generate the data and the best-fit recovered values. However, you may think of a better way (if you do please explain what you did). You should make a 3-d plot which shows the space of this function over the entire range (where values of zero show clear identifiability of the model parameters and large values mean you were unable to recover the parameters). This could take a long time to run (depending on how small your grid is, my suggestion is to start with a coarsely sampled grid and get finer until you are happy). Letting things run over night is probably the most time you want to spend waiting. Results: two plots, one for prototype, one for exemplar model showing the recoverability of parameters over a grid of different c,k values. *Hint:* check 'matplotlib' for matlab-style plotting within python, or use whatever plotting program is your favorite.

4 Computing Maximum Likelihood Fits

As discussed in class, contemporary ways to evaluate models tend to use more sophisticated methods (than RMSE) to calculate a measure of goodness of fit which take into account the likelihood of the data given the model for a given set of parameters. For this step, you should implement a way of performing a maximum likelihood fit to the choice data. I have provided a function called **computeLogLikelihood(N, S, p)** which computes the likelihood of a particular set of data under a binomial probability model.

A short explanation may be in order: the models predictions take the form of probabilities of endorsement for each of the prototype, low, high, random, and old items. If you find out that people endorse the prototype on 2 out of 2 trials how likely is this outcome given that the model (for a particular set of parameters) predicts an endorsement of p=0.8? Three numbers are required to do this for each data point N, the number of trials/presentations within the stimulus class, S the number of successes observed (S<=N), and p the predicted probability. Then you can turn the crank on my loglikelihood function which returns the probability that you would get S successes in N trials if the true probability was p (make sure you understand what is happening in computeLogLikelihood). You can sum these log likelihoods for each stimulus class (prototype, low, high, random, old) to compute a total log(likelihood) of the data for any given model with any set of parameters. If you are fitting individuals you can also sum these up for each individual to get a total fit measure (if you are fitting group data see below).

Setting this up will require a bit more code on your part (and reference to the design parameters above). Thus, this part is where the code from step 1 will come in handy. In other words, when you make fake data for this you are

going to have to actually make fake trials and count the number of yes responses predicted. You can use the probabilities given by the model to make individual data this way (i.e., if p = 0.8, flip a coin N times where N=4 for the prototype, etc...) and count the number of successes, etc.. After you have your data, you want to try to fit it using scipys fmin function to find the parameters that minimize the negative of the loglikelihood (we want to maximizing likelihood, so minimizing negative of it will get us in the right direction).

For the same parameters and simulated data you used in step 1, see if you can recover the parameters using loglikelihood instead. How identifiable are the parameters now? Does you ability to figure out which parameters generated the data change between the two fit metrics?

Expected output: your results should be a simple 3d plot that shows how recoverability changes across the parameter space of both the prototype and exemplar models for ML (maximum likelihood, step 4) that compliment the ones you made in step 3[2]. You can fit individual or group data (or both), just be sure to explain what you did.

5 Changing the number of trials

- 10 presentations of the prototype at test

  10 low distortions

- 10 high distortions

- 30 random items

- 40 old items presented twice each

Generate fake data corresponding to this situation and repeat the ML fits above. How does this effect the relative findings from RMSE and ML fits above? Which method is more sensitive to aspects of the experiment design? This should require a very small change to your code above and a quick re-run.

6 Model identifiability

In the final step, we want to know how well we can figure out which model generated the data when we know what the actual generating process is. For example, if you take some fake data generated from the prototype or exemplar model how well can you figure out which model actually generated it? You can do this for the grid analysis you did in step 2 & 3 very simply. Instead of fitting

[2]You'll have to make a choice about how to do the ML fits: to the group or to individuals. To evaluate the group, you basically do what was provided in Step 1 for RMSE (just fit the model to the group means). However, to do this for the ML it is a little less clear what do to. One approach (the one you should adopt here) is to set N and S using the number of trials and successes summed across all the individuals (i.e., if you created 10 individuals and there were 4 trials per person N=40). This is the equivalent of fitting the group means using RMSE.

the same model to the data that generated it and finding the qualitative results, fit both models this time. Compare the fit quality of each model using RMSE or ML. If one model give a better RMSE it win, or vice versa (since the models have the same number of parameters) if the ML is lower for one model than the other it fits better. The results should be a 3d plot for the entire parameter space that shows the better fitting model for data generated originally for the prototype model and for the exemplar model.

7 Summary

The principal results should be a new script (email to me) along with a short lab report that explains what you found, what you did, and shows your simulation results. Note that the code for this project is actually quite simple, but depending on how fine-grained a grid you use for the parameter searches you may find it takes a while to run. Your report should have a number of 3-d plots showing the parameter identifiability and model selection information for the entire parameter space for both the exemplar and prototype model. The results should be a set of additions to the exemplar/prototype script I provided along with a set of plots that show:

- the identifiability of parameters for the prototype and exemplar model across a grid of parameters

  how this function changes for ML (maximum likelihood fits)

- how this function changes for a larger number of individual observation (experiment factors)

- two plots showing the best fit model for each set of parameters for data generated from the exemplar/prototype model respectively

You have a couple weeks to get this together, so feel free to let things run overnight (typical for modelers!), but don't go overboard: we just want to get a qualitative sense of how these things can influence our model fits, as well as getting a handle on what these metrics mean.