Homework-Categorization-PartB

April 22, 2024

1 Homework - Categorization and Model Comparison Part B (40/110 points)

by *Todd Gureckis* and *Brenden Lake*
Computational Cognitive Modeling
NYU class webpage: https://brendenlake.github.io/CCM-site/

This homework is due before midnight on April 22 2024.

---

[33]:
```
from IPython.display import display
import string as str
import os
import numpy as np
import seaborn as sns
import pandas as pd
import math
from random import random, randint, shuffle, uniform
from scipy.optimize import fmin, brute
from scipy.special import comb  # gets the combinations function
from exemplarproto import *  # this grabs much of the code from Part A of the
  ↪homework
```

1.1 Fitting the models using maximum likelihood

As mentioned in the lecture, RMSE is not always an ideal mechanism for fitting models. One reason is that it is insensitive to the number of observations that define each data point. For example, remember in our experiment that participants saw the prototype item four times at test. In contrast, there were 20 different "new" patterns. This means there are five times as many trials contributing to the "new" bar in this graph as for the prototype patterns. Since RMSE measures the raw deviation of the average model predictions from those of the model it doesn't take into account these issues. Thus, we would like to also evaluate these two models using maximum likelihood.

The key to this is going to be the provided function below which computes the likelihood of a particular set of data under a binomial probability model.:

[34]:
```
##################################
# computeLogLikelihood
# N = number of observations
# S = number of "successes" (i.e., endorsements)
# p = predicted probability of successes by the model
##################################
def computeLogLikelihood(N, S, p):
    p = p if p > 0.0 else 0.0 + 1e-10
    p = p if p < 1.0 else 1.0 - 1e-10
    try:
        result = math.log(comb(N, S)) + (S * math.log(p) + (N - S) * math.log(1.
 ↪0 - p))
    except:
        print(N, S, p)  # this shouldn't happen but just in case
        result = 0
    return result


def pandas_ll(row):
    return computeLogLikelihood(
        row["Total"], row["N_Yes"], row["Probability of Endorsement"]
    )
```

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 the above `computeLogLikelihood()` 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. For this homework will we focus on fitting the group data rather than to individuals.

To get the data formatted into an appropriate shape for fitting likelihoods we provide a function `get_human_results_ll()` which returns a Pandas data frame containing the number of times a pattern of a particular type was endorsed and the number of times it was presented for each subject.

[16]:
```
human_res = get_human_results_ll()
human_res
```

[16]:
```
          Subject Condition Stimulus Type  N_Yes  Total
0    ./data/4.dat       cat     Prototype      4      4
1    ./data/4.dat       cat           Low      4      4
2    ./data/4.dat       cat          High     10     10
3    ./data/4.dat       cat        Random      5     20
4    ./data/4.dat       cat           Old     18     20
..            …         …             …      …      …
```

```
0   ./data/1.dat       rec     Prototype      3      4
1   ./data/1.dat       rec           Low      1      4
2   ./data/1.dat       rec          High      2     10
3   ./data/1.dat       rec        Random      0     20
4   ./data/1.dat       rec           Old     11     20

[70 rows x 5 columns]
```

This reorganizes the data per condition.

```
[17]: human_data = human_res.groupby(["Condition", "Stimulus Type"]).sum()
      human_data
```

```
[17]:                                                                Subject  \
Condition Stimulus Type
cat       High           ./data/4.dat./data/8.dat./data/12.dat./data/6…
          Low            ./data/4.dat./data/8.dat./data/12.dat./data/6…
          Old            ./data/4.dat./data/8.dat./data/12.dat./data/6…
          Prototype      ./data/4.dat./data/8.dat./data/12.dat./data/6…
          Random         ./data/4.dat./data/8.dat./data/12.dat./data/6…
rec       High           ./data/9.dat./data/7.dat./data/13.dat./data/11…
          Low            ./data/9.dat./data/7.dat./data/13.dat./data/11…
          Old            ./data/9.dat./data/7.dat./data/13.dat./data/11…
          Prototype      ./data/9.dat./data/7.dat./data/13.dat./data/11…
          Random         ./data/9.dat./data/7.dat./data/13.dat./data/11…

                         N_Yes  Total
Condition Stimulus Type
cat       High              49     70
          Low               26     28
          Old              110    140
          Prototype         25     28
          Random            37    140
rec       High              20     70
          Low               16     28
          Old               89    140
          Prototype         18     28
          Random             7    140
```

Finally these function allow us to compute the negative log likelihood of the data given the model.

```
[27]: def fit_exemplar_model_nll(params, human_results):
          [c_cat, k_cat, c_rec, k_rec] = params
          k_cat = k_cat if k_cat > 0.0 else 0.0
          k_rec = k_rec if k_rec > 0.0 else 0.0
          predictions = get_exemplar_results(c_cat, k_cat, c_rec, k_rec)
          model = predictions.groupby(["Condition", "Stimulus Type"], as_index=False).
       ↪mean(numeric_only=True)
```

```
    fitted_data = pd.merge(model, human_results)
    return -1.0 * fitted_data.apply(pandas_ll, axis=1).sum()


def fit_prototype_model_nll(params, human_results):
    [c_cat, k_cat, c_rec, k_rec] = params
    k_cat = k_cat if k_cat > 0.0 else 0.0
    k_rec = k_rec if k_rec > 0.0 else 0.0
    predictions = get_prototype_results(c_cat, k_cat, c_rec, k_rec)
    model = predictions.groupby(["Condition", "Stimulus Type"], as_index=False).
 ↪mean(numeric_only=True)
    fitted_data = pd.merge(model, human_results)
    return -1.0 * fitted_data.apply(pandas_ll, axis=1).sum()
```

Problem 6 (20 points)

The cell blocks below allow you to fit the exemplar model and the prototype model to the dataset we considered in Part A of the homework. Make sure you understand and follow the code provided above and in the provided library (exemplarproto.py). Next, try altering the parameters to minimize the negative log likelihood score. When you think you have found the best fit parameters for both the exemplar and prototype models report your final parameter values along with the plot of the resulting model predictions. In a markdown cell describe which model you believe fits better. Is this conclusion the same or different from what you considered in Part 4 of the homework? If the fit looks different, why?

Exemplar model

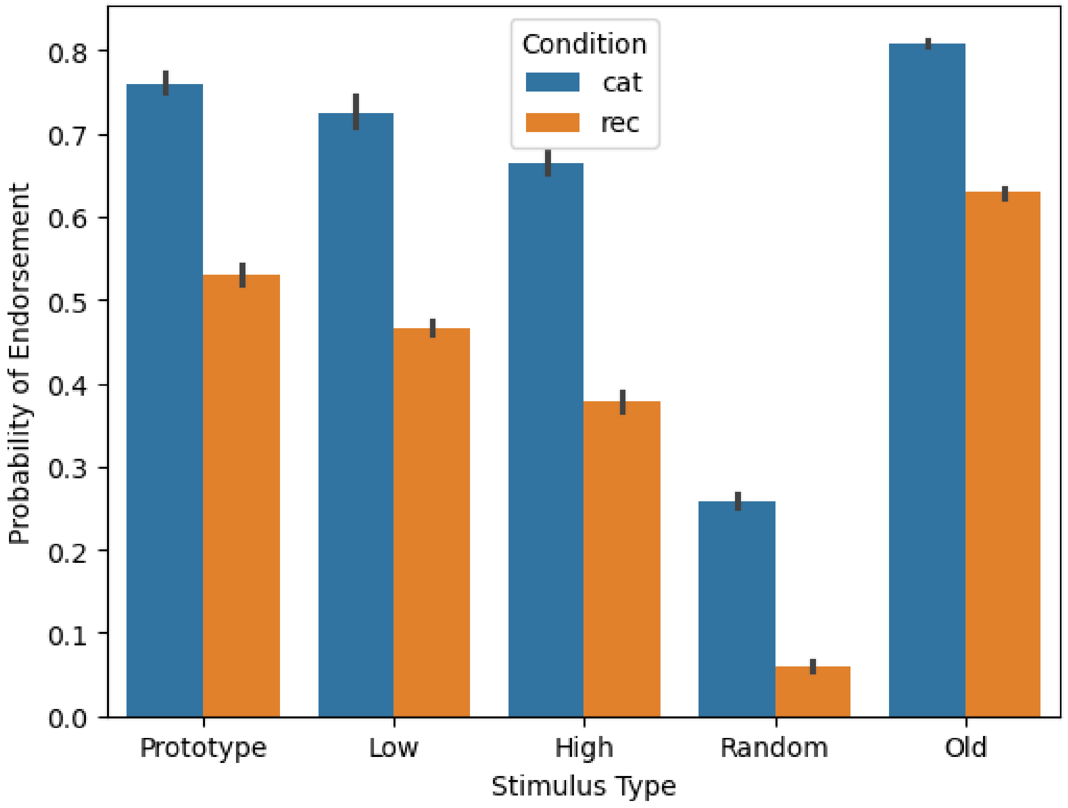
[45]:
```
human = human_res.groupby(["Condition", "Stimulus Type"], as_index=False).sum()

params = [1.5, 0.4, 1.8, 0.9]
nllfit = fit_exemplar_model_nll(params, human)
print(f"The negative log score is {nllfit}")

# now plot the data
c_cat, k_cat, c_rec, k_rec = params
res = get_exemplar_results(c_cat, k_cat, c_rec, k_rec)
sns.barplot(
    x="Stimulus Type", y="Probability of Endorsement", hue="Condition", data=res
)
```

```
The negative log score is 29.20235417108696
```

```
/opt/conda/envs/ccm/lib/python3.12/site-packages/seaborn/_base.py:949:
FutureWarning: When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass `(name,)`
instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/opt/conda/envs/ccm/lib/python3.12/site-packages/seaborn/_base.py:949:
FutureWarning: When grouping with a length-1 list-like, you will need to pass a
```

```
length-1 tuple to get_group in a future version of pandas. Pass `(name,)`
instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
```

```
[45]: <Axes: xlabel='Stimulus Type', ylabel='Probability of Endorsement'>
```



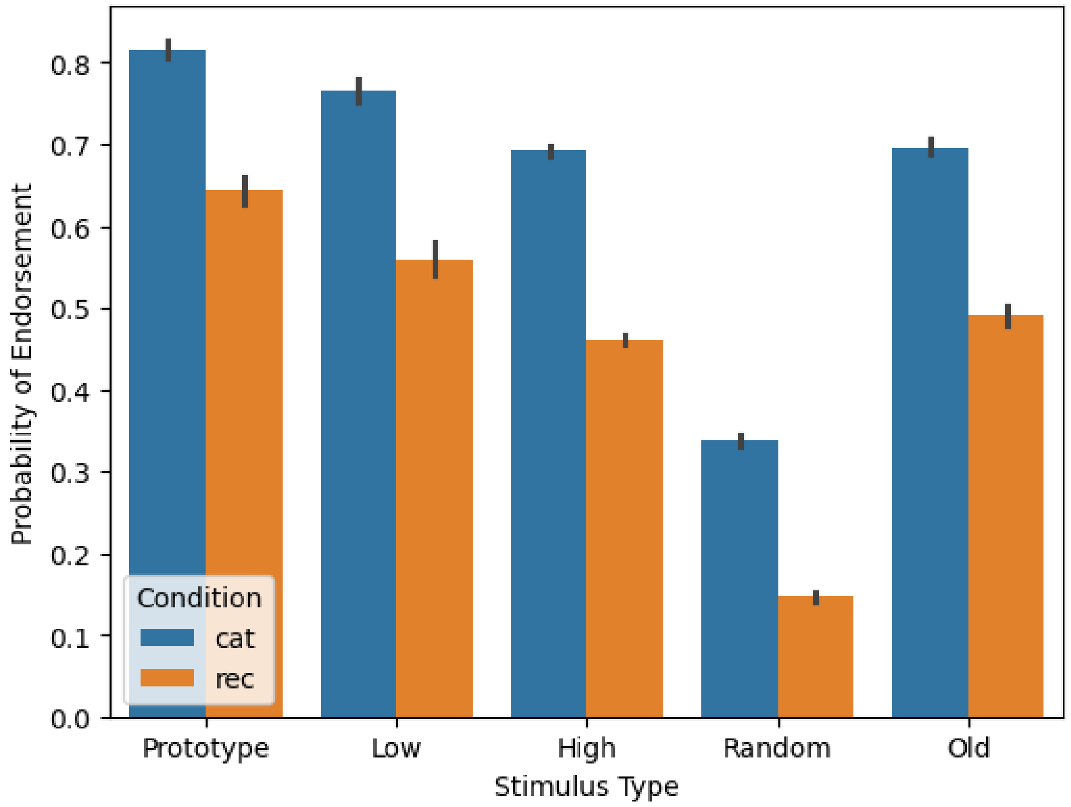
Prototype Model

```
[118]: human = human_res.groupby(["Condition", "Stimulus Type"], as_index=False).sum()

params = [1.05, 0.1, 1.1, 0.25]
nllfit = fit_prototype_model_nll(params, human)
print(f"The negative log score is {nllfit}")

# now plot the data
c_cat, k_cat, c_rec, k_rec = params
res = get_prototype_results(c_cat, k_cat, c_rec, k_rec)
sns.barplot(
    x="Stimulus Type", y="Probability of Endorsement", hue="Condition", data=res
)
```

```
The negative log score is 45.76231118845733
```

```
/opt/conda/envs/ccm/lib/python3.12/site-packages/seaborn/_base.py:949:
FutureWarning: When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass `(name,)`
instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/opt/conda/envs/ccm/lib/python3.12/site-packages/seaborn/_base.py:949:
FutureWarning: When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass `(name,)`
instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
```

```
[118]: <Axes: xlabel='Stimulus Type', ylabel='Probability of Endorsement'>
```

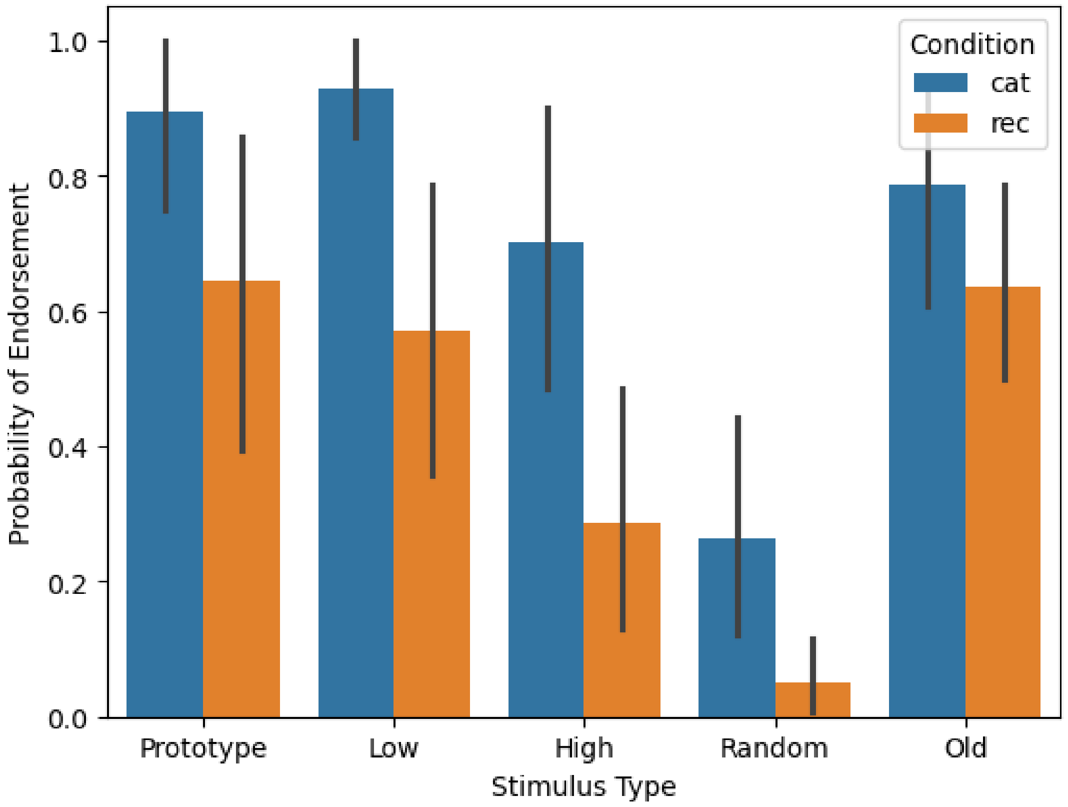


Human data again for reference

```
[32]: sns.barplot(
          x="Stimulus Type",
          y="Probability of Endorsement",
          hue="Condition",
          data=get_human_results(),
      )
```

```
/opt/conda/envs/ccm/lib/python3.12/site-packages/seaborn/_base.py:949:
FutureWarning: When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass `(name,)`
instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/opt/conda/envs/ccm/lib/python3.12/site-packages/seaborn/_base.py:949:
FutureWarning: When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass `(name,)`
instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
```

```
[32]: <Axes: xlabel='Stimulus Type', ylabel='Probability of Endorsement'>
```



For the exemplar model, the parameters that I found that best align the model with the human data are c_cat, k_cat, c_rec, k_rec = [1.5, 0.4, 1.8, 0.9] with an nll of 29.20235417108696. However, for the prototype model, the parameters that I found that best align the model with the human data are c_cat, k_cat, c_rec, k_rec = [1.05, 0.1, 1.1, 0.25] with an nll of 45.76231118845733. Thus, the exemplar model seems to fit better (like it did in Part 4 of the homework), although, we do not carry out the fmin algorithm as we did in Part A of the homework assignment, so we cannot know for sure. It could be that they both perform quite similarly like they did in Part A, as well.

Problem 7 (10 points)

A famous saying is the “All models are wrong, but some are useful” (George Box). Do you think the exemplar or prototype model provides the best account of the data? Refer to particular patterns in the data that you believe the different models do a better job with.

In both Part A and B of our homework assignment, it seems as though the exemplar model outperforms the prototype model when comparing the models with the human data. This makes sense, given that the model is able to retain individual memories of each exemplar, allowing it to predict the likelihood of endorsement for each new stimulus by comparing it to multiple stored exemplars instead of just comparing it with a single prototype. The prototype model may not account for certain patterns in the data as well as the exemplar model because it cannot account for greater deviances from the prototype in the High and Low distortion stimuli. The exemplar model also seems to be more in line with cognitive processes, given we can refer to multiple exemplars in memory when we come across a new stimulus (Like the birds example described in Part A). To address a particular pattern that supports this, we can examine the bar graphs, and more specifically, the Old stimuli.

Problem 8 (5 points)

Thinking about how these models work explain why both the exemplar and prototype models have relatively high endorsement for the prototype item even though it was never presented during the training phase. In addition, explain in your own words why the models are able to explain the high endorsement rates for the old items.

Problem 9 (5 points)

Are the exemplar model and the prototype model we considered nested? Would we compare them using AIC, BIC, or the G^2 statistic (or something else)?

```
[ ]:
```