Homework-Categorization-PartA

April 22, 2024

1 Homework - Categorization and Model Comparison Part A (70/110 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.

---

[2]:
```python
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
import matplotlib.pyplot as plt
```

2 Background and Theory

In this homework we explore the cognitive mechanisms that support unsupervised pattern categorization in humans. In addition, we use this as an example of testing and comparing between models.

2.1 A simple (classic) unsupervised categorization experiment

In Posner and Keele (1968) report a now classic categorization experiment with humans. In the task participants viewed visual stimuli that are clouds of points (known as dot patterns) similar to a scatter plot of data on a graph. An examples of the stimuli is shown here:

Experiment Design

The experiment was divided into a training and test phase. During the training phase, for each subject a single random dot pattern was generated and considered to be the underlying "prototype" structure. A prototype is like a common template or reference pattern. The key is that participants never get to see the "prototype" pattern directly during training. Instead they see what are known as “distortions” of the prototype. A distortion of a pattern is made by adding random spatial noise to each point in a pattern to kind of “wiggle” the points away from their original position.

For example, here is a random prototype (top) and a bunch of random distortions of the prototype made by adding or subtracting small random values from the `<x,y>` value of each point in the pattern.

Posner and Keele created distortions that added more or less random noise. For example, “high” distortions add a lot of randomness to the underlying template pattern whereas “low” distortions add only a little bit of noise.

Training Phase

In the training phase of the experiment subjects view 10 training examples one at a time which are “high” distortions of a randomly generated prototype. The instructions are that subjects should look at these patterns, and that they come from a single category similar to if you viewed a series of pictures of dogs they would all come from the category `dog`. Subjects were try to figure out the pattern that related the different images to one another. Try it for yourself by looking at each of the “distortions” patterns above one by one and trying to detected the common structure.

Test Phase

During the test phase, participants view a series of dot patterns one at a time and have to judge: **Does the given pattern come from the same general category or family you studied earlier or is it a new pattern that is different?** This is an unsupervised categorization task because the subject has to abstract what the common structure is from the given patterns and then use that information to make classification decisions about new patterns.

Unknown to participants the set of test items varied in a specific way with respect the training patterns. In particular, there were five particular types of patterns presented during test.

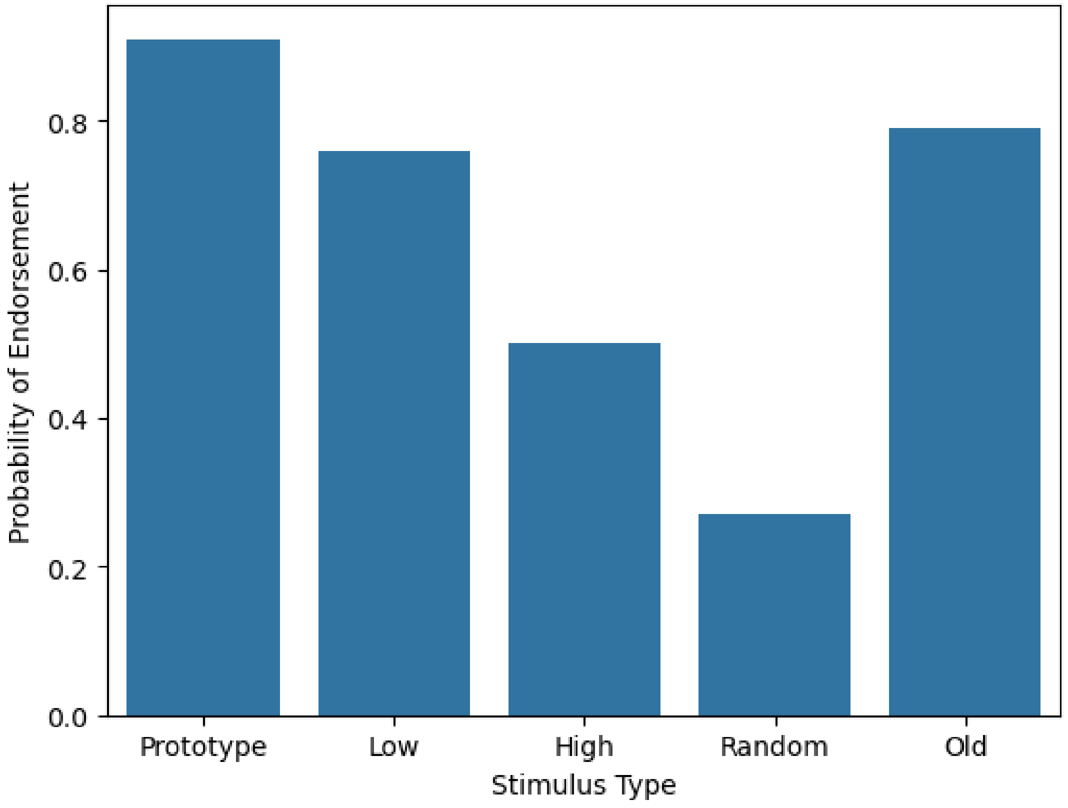
- The first type were “old” patterns which were identical to those presented during the training phase.
- The second type were “random” patterns which were from a complete new randomly generated prototype (thus had nothing to do with the items presented during training).
- The third type were new “high distortions” of the underlying prototype that was used to create the study set. These are thus similar to the “old” items but do not match exactly.
- The fourth type were “low distortions” of the underlying prototype that was used to crete the study set. These are more similar to the prototype pattern than the “high” distortions are.
- Finally the actual prototype used to generate the items during training was presented. This pattern is interesting because the prototype pattern was never seen exactly during training. However, people saw many high distortions of this item during training and given the instructions to detect what the common structure of the training patterns is, they may have learned something about this latent or hidden pattern.

Typical Results:

This graph show example results that are typical for an experiment like this:

[3]:
```
df = pd.DataFrame(
    {
        "Stimulus Type": ["Prototype", "Low", "High", "Random", "Old"],
        "Probability of Endorsement": [0.91, 0.76, 0.5, 0.27, 0.79],
    }
)
sns.barplot(x="Stimulus Type", y="Probability of Endorsement", data=df)
```

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



The height of the bars indicates the probability of endorsing a pattern as a member of the category during the test phase (high values mean that at test a subject is more likely to agree "yes this pattern fits with the one I learned"). Notice that the "old" items (the exact patterns studied during learning are endorsed at a relatively high rate. In contrast the "random" patterns (those coming from a completely different underlying pattern) are endorsed at a very low rate. The low and high distortions are endorsed at intermediate rates (with the low distoritions preferred). Interestingly, the prototype pattern is endorsed most strongly even though it was never presented during the study period! It is like during learning people figured out the underlying pattern that generated the stimuli!

2.2 Categorization: Memory for examples or abstractions?

The dot pattern experiments are interesting and have been replicated in various ways perhaps hundreds of time. How do people make these judgments? What information to people store during the study phase that would predict their performance in the test phase? What rule do they use for combining information from memory in order to make these kind of judgements? We are going to build up a model of categorization in a few simple steps.

2.3 Step 1: How are dot patterns represented in the mind?

Our first question concerns how people represent the dot patterns and the similarity between different dot patterns. A variety of work has suggested that the psychological similarity between pairs of dot patterns follows roughly a logarithmic transform of the average euclidean distance between the pairs of points (plus one). This was established by having people view pairs of dot patterns and rate how similar they seem. For example here is a plot from a paper by Smith and Minda showing a strong contgruence between dissimiliarty ratings between pairs of stimuli and `log(distance)`

In light of this lets define the similarity between two dot patterns i and j as s_{ij} and let it equal the following equation:

$$s_{ij} = log(1 + \frac{\sqrt{\sum_d [(i_{d_x} - j_{d_x})^2 + (i_{d_y} - j_{d_y})^2]}}{9})$$

where i_{d_x} is the x position of the d-th dot in pattern i and j_{d_x} is the x position for pattern j (likewise for i_{d_y}). There are 9 dots total, so we divide by 9 to get the average. Because it can sometimes be ambiguous which dot aligns which which one in a pattern we choose the dots which are closest in the two patterns to compute this score.

2.4 Step 2: What is stored in memory?

The next consideration is what people actually store in memory during the training phase of the experiment. There are of course many alternatives. People could store an "average" of the points seen so far, or they could store each individual pattern that they have seen, or they could store nothing and try to figure it out at test, or they could store some verbal description of what the shapes "look like", or the shape of the outer edge of the dot-cloud (the "convex null"), etc...

There are, however, two leading theories which have attracted considerable debate in the cognitive science literature: the prototype and exemplar theory.

2.4.1 Exemplar models

Exemplar models are a general class of psychological models related to nearest neighbor algorithms. The most important feature of these models is the idea that people have what appears to be a nearly infitite memory for the past and as a result you can store all past experiences or examples in memory. This seems crazy as we are forgetting things all the time but actually psychology is unclear about if we actually forget things or if we simply lose the ability to retreive a memory (i.e., more like losing the pointer to the memory rather than decay).

As mentioned in lecture, nearset neighbor classifiers use a similarity function (similar to the ones described above) to retrieve from memory the nearest labeled example and to predict the category membership based on the label for this item. This nearest neighbor algorithm can be relaxed slightly to consider k-nearest neighbors. According to this algorithm you find the k neighbor examples (with $k > 1$) to the current pattern and response based on what the majority of these examples say.

Now we can go a bit further and say that you compute the similarity to all past examples but *weight* their vote according to their similarity. So instead of picking the label of the closest or k-closest examples we compare the current pattern using a global match to all examples in the memory and weight their predictions based on similarity. Pretty neat!

Ok, but how does this help us in the case of **unsupervised** categorization such as in the dot pattern case? Here what we will assume is that we compute this similarity of the to-be-categorized item (the test item) to all the examples stored in memory and compare it to some criterion value. If the sum of the similarity to all the examples falls below this criterion then we assume the pattern is new and doesn't match what we learned. If it is above the criterion we judge the item is a good example of the category.

In the example model we will consider the probability of endorsing an item is going to be determined by the following equation:

$P(A|i) = \frac{\sum_j e^{-c \cdot s_{ij}}}{\sum_j e^{-c \cdot s_{ij}} + k}$

where $P(A|i)$ is the probability of endorsing pattern i as a member of the category seen during study. s_{ij} is the similarity between pattern i and pattern j which is an example stored in memory during the study phase. k is the criterion against which the summed similarity is being compared. If k is zero then you endorse the item as a member of the category all the time irrespecitive of the similiary and if k gets large you become more and more less likely to endorse the item (i.e., more likely to say no).

The sum is with respect to an exponential sum which has some deeper relation to research on categorization that we do not have time to discuss. However, it is basically the idea that very close matches ($s_{ij} = 0$) are especially strong and things that are less similar count less. You can think of it as the the particular weighted nearest neighbor algorithm we think the mind uses. c is a free parameter that controls that weighting function and is often fitted to data.

2.4.2 Prototype models

The prototype model is different than the exemplar model because it assumes that instead of storing each of the training patterns in memory exactly, instead people store a single summary representation. For example, people might store a mentally computed "average" pattern. When you think about how you would perform the task you might think that you kind of compare the training patterns to one another and then compute some summary.

In the case of the dot pattern stimuli one way to do this is to store a special trace in memory called the prototype which is the average of all the patterns seen so far (averaging the $< x, y >$ position of each point to find an average dot location.

According to the prototype model the probabililty of endorsing a test item pattern as a member of the category that was studied during training is:

$P(A|i) = \frac{e^{-c \cdot s_{ip}}}{e^{-c \cdot s_{ip}} + k}$

Note that nearly everything about this equation is the same except there is no longer a sum! Instead we simply compute the similarity between the test pattern and this special "prototype" pattern (p) which has been averaged from the training examples.

2.4.3 Parameters

k and c are "free parameters" in both the exemplar and prototype model which are assumed to modulate or alter the core psychological processes. These parameters might vary between subjects and as a function of condition. Thus, in order to assess the ability of the model to account for the data we often "fit" these parameters to our data.

3 Model Comparison

With these idea in mind, in this homework you are going to compare the exemplar and prototype model to account for some data from an actual dot pattern categorization task collected with human subjects. The goal is that by doing the homework you would develop some useful code that would let you more or less plug in a model that you might come across in your research, fit it to data, and verify that the fits are good, etc…

3.1 Reading in some data

The `data/` folder that comes with this homework contains data from 14 human subjects who participated in a dot pattern classification task. The data describing each subject is in a text file (`.dat`) indexed by subject number (e.g., `1.dat`, `2.dat`, etc…).

The organization of these files is as follows:

The first 44 lines of the file contain a description of the stimulus that the subject saw on a given trial including the x, y coordinate of each dot. The first columns of these 44 lines is the number of the pattern (`1-44`). The second column is the type of pattern using the following codes:

- `1` = prototype
- `2` = 10 "high distortions" of the prototype that were used as study patterns during learning
- `3` = 10 new "high distortions" of the prototype presented during test
- `4` = 4 "low distortions" of the prototype that were presented during test
- `5` = 20 random items that come from different prototypes that were presented at test

The next 18 values of each row are the coordinates of the dots (with the x, y coordinates in sequence). So `[x1, y1, x2, y2, x3, y3, ...]`.

The following 40 lines of the file show the sequence of items presented during the study phase. This is not all that important for our purposes, but basically the last column is which pattern was displayed (indexed from the patterns just described. Each of 10 "high distortions" were presented four times each during study in a random order.

Finally the remaining lines of the file report the results of the test phase. The first column is the subject number, the second columns is the condition number, the next is the trial number in the experiment, the other columns worth mentioning are the last column (the pattern number from the beginning of the file), the second to last column (the type of stimulus it is according to the codes described above), and the reaction time in milliseconds.

In addition, participants in this experiment were assigned to one of two conditions: a recognition condition and a categorization condition. These conditions differed only in the instructions given to participants at the start of the test phase. In the recognition condition participants were told they would view a series of patterns and they should respond "yes" only if the patterns was **exactly** one they say in the previous study phase. In the categorization condition, participants were asked to respond “yes” only if the pattern belonged to the same general category or pattern that they observed in the training phase.

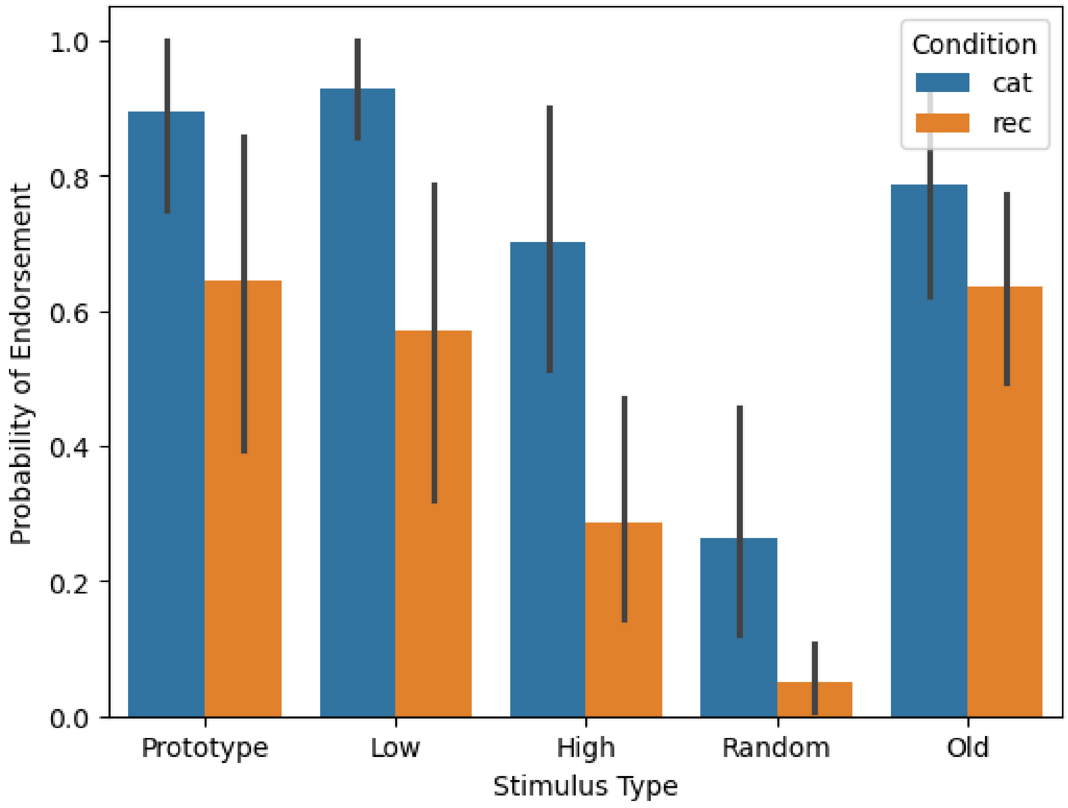
The following graph computes the probability of endorsement in the data set as a function of stimulus type and condition (Cat or Rec instructions).

[4]:
```python
################################
# getcurve
################################
def getcurve(filename):
    prototypes = []
    low = []
    old = []
    high = []
    random = []
    mydata = readfile(filename)
    cond = mydata[-1][1]
    for line in mydata:
        if line[4] == 2 and len(line) == 9:
            if line[7] == 1:
                prototypes.append(line[5])
            if line[7] == 2:
                old.append(line[5])
            if line[7] == 3:
                high.append(line[5])
            if line[7] == 4:
                low.append(line[5])
            if line[7] == 5:
                random.append(line[5])

    # print([len(prototypes), len(low), len(high), len(random), len(old)])
    # print(prototypes)
    # print(low)
    # print(high)
    # print(random)
    # print(old)
    return [
        np.average(prototypes),
        np.average(low),
        np.average(high),
        np.average(random),
        np.average(old),
        filename,
        cond,
    ]


def readfile(filename):
    results = []
    fp = open(filename, "r")
    for line in fp.readlines():
        myline = list(map(int, line.split(" ")[:-1]))
        results.append(myline[:])
    fp.close()
    return results


def get_all_filenames(directoryname):
    files = filter(
        lambda x: x[-4:] == ".dat" and x[0] != ".",
        os.listdir(os.path.join(".", directoryname)),
    )
    fn = map(lambda x: os.path.join(".", directoryname, x), files)
    # process each file and drop last 5 trials
    return list(fn)


def create_df(subjnum, cond, pattern):
    nobs = len(pattern)
    df = pd.DataFrame(
        {
            "Subject": [subjnum] * nobs,
            "Condition": [cond] * nobs,
            "Stimulus Type": ["Prototype", "Low", "High", "Random", "Old"],
            "Probability of Endorsement": pattern,
        }
    )
    return df


def get_human_results():
    allres = map(getcurve, get_all_filenames("data"))
    cat = []
    rec = []
    for patt in allres:
        if patt[-1] == 0:
            cat.append(create_df(patt[-2], "cat", patt[:-2]))
        else:
            rec.append(create_df(patt[-2], "rec", patt[:-2]))
    cat, rec = pd.concat(cat), pd.concat(rec)
    return pd.concat([cat, rec])
```

[5]:
```
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)
```

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



Problem 1 (20 points)

Using your own words explain the data pattern you see in the above figure. What is different between the conditions and stimulus type? Why do you suspect these patterns exist? Your answer will need to consider the nature of the experiments, what is manipulated, and even your intuitive psychological theory about what might be going on. Your response should take 3-4 sentences and appear in a cell below. Is any feature of this data surprising to you?

In the above figure, we can see that there is a lower probability across the board for the recognition condition. This makes sense, given the participants could only respond “Yes” if they found the presented dots to be identical to one of the exemplars they previously viewed in the training phase. Furthermore, it is interesting to note that the Prototype stimulus and Old stimuli received nearly equal probabilities of endorsement in the recognition condition, whereas the probability was higher for both the prototype stimulus and Low stimuli in the categorical condition when compared with the Old stimuli. For the recognition condition, the similar probabilities for the Old and Prototype stimuli are likely due to the fact that while holding all of the exemplars in memory, the participants themselves may have subconsciously created a prototype of their own and falsely “recognized” the actual prototype as being one of the original exemplars. Additionally, it is rather surprising that probability of the Low stimuli in the categorical condition surpasses that of the prototype’s, given the prototype is what the “Old”, “Low”, and “High” stimuli are all based on.

3.2 Predictions for the exemplar model

The following cells set up the exemplar model using the equations described above.

[6]:
```python
################################
# unitdist:
# computes the euclidean distance between
# two dots
################################
def unitdist(x, y):
    x1 = np.array(x)
    y1 = np.array(y)
    return math.sqrt(sum(pow(x - y, 2.0)))


################################
# computeresponse
# computes the "activation" of each
# trace in memory
################################
def computeresponse(target, memory, c, k):
    res = []
    for mem in memory:
        res.append(
            math.log(
                1.0 + np.average(list(map(lambda x, y: unitdist(x, y), target,␣
 ↪mem)))
            )
        )
    resp = [math.exp(-c * x) for x in res]
    pofr = sum(resp) / (sum(resp) + k)
    return pofr
```

```
[7]: ################################
     # exemplar model
     # stores all 10 study items in memory
     # and computes the probability of endorsement
     # for each item type
     ################################


     def exemplarmodel(filename, c, k):
         data = readfile(filename)
         cond = data[-1][1]
         memory = []
         for line in data:
             if len(line) == 20 and line[1] == 2:
                 memory.append(np.resize(line[2:], (9, 2)))
         # print(memory)

         # prototype items
         proto = []
         for line in data:
             if len(line) == 20 and line[1] == 1:
                 item = np.resize(line[2:], (9, 2))
                 pofr = computeresponse(item, memory, c, k)
                 proto.append(pofr)
         # print(np.average(proto))

         # old items
         old = []
         for line in data:
             if len(line) == 20 and line[1] == 2:
                 item = np.resize(line[2:], (9, 2))
                 pofr = computeresponse(item, memory, c, k)
                 old.append(pofr)
         # print "p of r", old
         # print(np.average(old))

         # new high items
         newhigh = []
         for line in data:
             if len(line) == 20 and line[1] == 3:
                 item = np.resize(line[2:], (9, 2))
                 pofr = computeresponse(item, memory, c, k)
                 newhigh.append(pofr)
         # print(np.average(newhigh))

         # new low items
         newlow = []
```

```
    for line in data:
        if len(line) == 20 and line[1] == 4:
            item = np.resize(line[2:], (9, 2))
            pofr = computeresponse(item, memory, c, k)
            newlow.append(pofr)
    # print(np.average(newlow))

    # random items
    random = []
    for line in data:
        if len(line) == 20 and line[1] == 5:
            item = np.resize(line[2:], (9, 2))
            pofr = computeresponse(item, memory, c, k)
            random.append(pofr)
    # print(np.average(random))

    return [
        np.average(proto),
        np.average(newlow),
        np.average(newhigh),
        np.average(random),
        np.average(old),
        filename,
        cond,
    ]
```

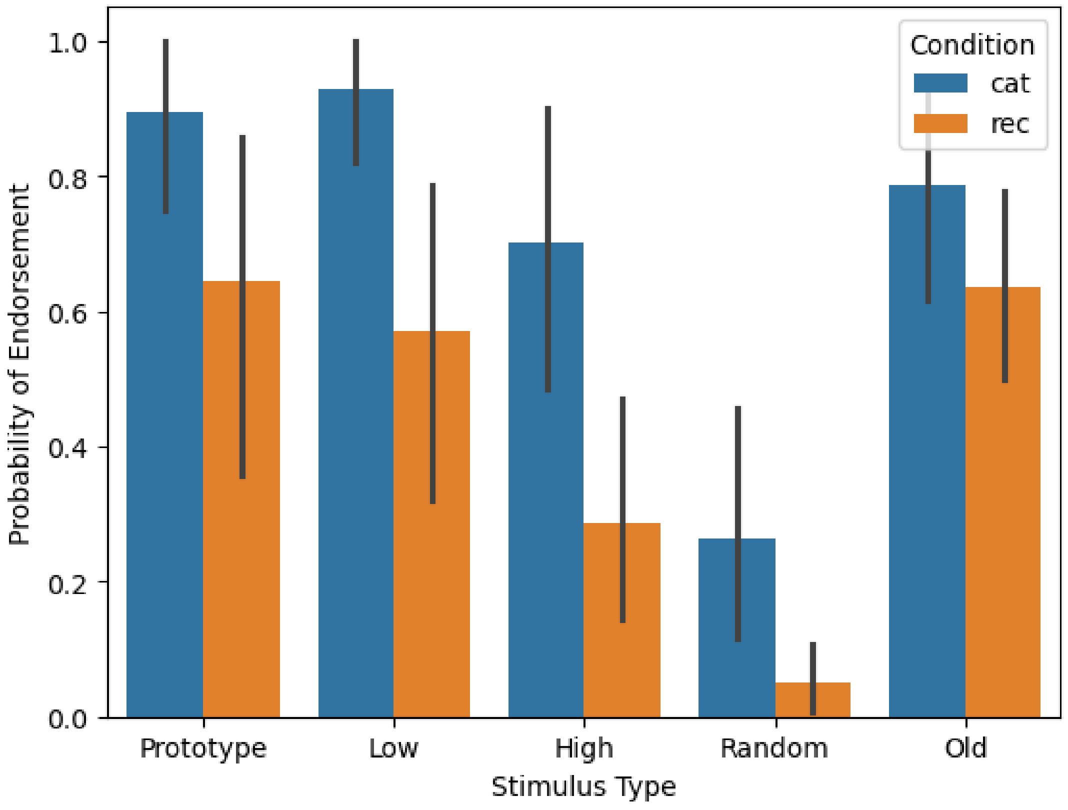
[8]:
```
def get_exemplar_results(c_cat, k_cat, c_rec, k_rec):
    allres = {fn: readfile(fn) for fn in get_all_filenames("data")}
    cat = []
    rec = []
    for filename in allres.keys():
        if allres[filename][-1][1] == 0:
            res = exemplarmodel(filename, c_cat, k_cat)
            cat.append(create_df(filename, "cat", res[:-2]))
        else:
            res = exemplarmodel(filename, c_rec, k_rec)
            rec.append(create_df(filename, "rec", res[:-2]))
    cat, rec = pd.concat(cat), pd.concat(rec)
    return pd.concat([cat, rec])
```

First let's replot the human results:

[9]:
```
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)
```

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



Problem 2 (10 points)

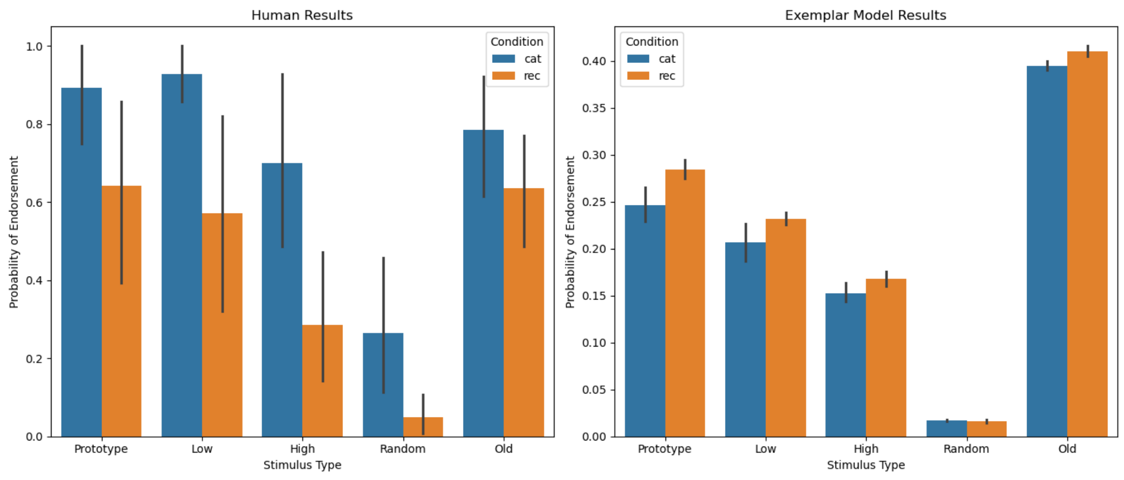
By hand adjust the setting of the model parameters to roughly fit the human data pattern shown above. How close can you get? What parameters did you find (report them) and you assessmnet of how well they fit. Was it a good fit or are there systematic problems with the fit? In addition, what are the parameter values and do they make sense in light of the equations described above? When the parameters are the same for recognition and categorization instructions why do the bars look a little different?

The following cell lets you plot the model predictions for the exemplar model fitted to the stimuli that participants in this experiment actually viewed. There is a k and a c parameter for both categorization and recogniton.

[25]:
```
# First, we maintain equal values for all parameters

# Set up subplots
plt.figure(figsize=(14, 6))

# First subplot for the human results
plt.subplot(1, 2, 1)
sns.barplot(
    x="Stimulus Type",
    y="Probability of Endorsement",
    hue="Condition",
    data=get_human_results(),
)
plt.title('Human Results')

# Second subplot for the exemplar results
plt.subplot(1, 2, 2)
c_cat, k_cat, c_rec, k_rec = 2.0, 2.0, 2.0, 2.0   # Maintaining equal values␣
 ↪for all parameters
sns.barplot(
    x="Stimulus Type",
    y="Probability of Endorsement",
    hue="Condition",
    data=get_exemplar_results(c_cat, k_cat, c_rec, k_rec),
)
plt.title('Exemplar Model Results')

# Show the plots
plt.tight_layout()
plt.show()
```

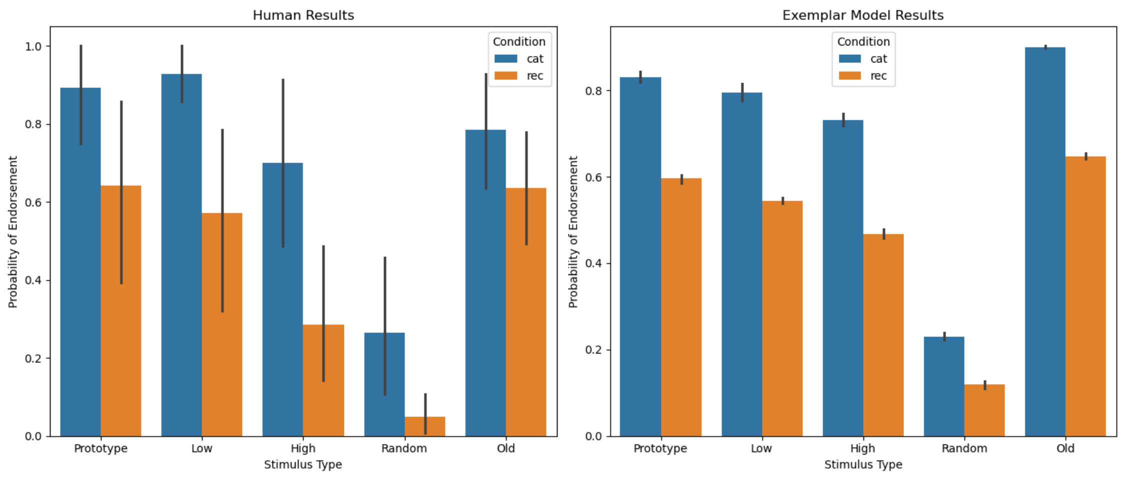
```
/opt/conda/envs/ccm/lib/python3.12/site-packages/seaborn/_base.py:949:
FutureWarning: When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass `(name,)`
instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/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)
/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]:
```python
# Set up subplots
plt.figure(figsize=(14, 6))

# First subplot for the human results
plt.subplot(1, 2, 1)
sns.barplot(
    x="Stimulus Type",
    y="Probability of Endorsement",
    hue="Condition",
    data=get_human_results(),
)
plt.title('Human Results')

# Second subplot for the exemplar results
plt.subplot(1, 2, 2)
c_cat, k_cat, c_rec, k_rec = 1.9, 0.15, 1.5, 1.0
sns.barplot(
    x="Stimulus Type",
    y="Probability of Endorsement",
    hue="Condition",
    data=get_exemplar_results(c_cat, k_cat, c_rec, k_rec),
)
plt.title('Exemplar Model Results')

# Show the plots
plt.tight_layout()
plt.show()
```

```
/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)
/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)
```



It looks like we can get fairly close to the human results when we use “c_cat, k_cat, c_rec, k_rec = 1.9, 0.15, 1.5, 1.0”, though the nuances of the empirical data are not exactly captured in the exemplar model. For example, in the human results, the Low distortion stimuli have a higher probability of endorsement than the prototype stimulus, and the prototype stimulus has a higher probability than the Old stimuli. However, in the exemplar model, the Old stimuli have the highest probability, which is likely due to the fact that the model directly compares the exemplars with the stimulus in question whereas humans are more likely to forget (which may explain why the prototype and old stimuli are nearly identical in recognition condition of the human results). Furthermore, the parameter values for categorization make sense in light of the equations described above because high values of c (c_cat = 1.9) indicate a need for very similar stimuli to be classified as the same category. However, the fact that the model is more aligned with the human data when c_rec = 1.5, which is lower than the c_cat, is rather surprising. If we change c_rec to 1.9, though, the model underestimates all of the recognition probabilities. Our k parameters on the other hand are drastically different, as the k value modulates the response bias or threshold for categorization. In the categorization condition, we have a lower k value to allow for more of the stimuli to be accepted as part of the category. For the recognition condition, we have a much higher k value to ensure that the model is relying more heavily on exact matches. Finally, when the parameters are the same for the recognition and categorization conditions, the output is most likely different due to the fact that each run of the model deals with a slightly different set of distorted stimuli (based on the prototype).

3.3 Predictions for the prototype model

The following cells set up the prototype model using the equations described above.

[18]:
```
################################
# prototype model
# stores an average of the study items in memory
# and computes the probability of endorsement
# for each item type
################################
def prototypemodel(filename, c, k):
    data = readfile(filename)
    cond = data[-1][1]
    # average all the old items in memory
    memory = []
    for line in data:
        if len(line) == 20 and line[1] == 2:
            memory.append(line[2:])
    memory = [np.resize(list(map(np.average, np.transpose(np.array(memory)))),␣
 ↪(9, 2))]

    # prototype items
    proto = []
    for line in data:
        if len(line) == 20 and line[1] == 1:
            item = np.resize(line[2:], (9, 2))
            pofr = computeresponse(item, memory, c, k)
            proto.append(pofr)
    # print(np.average(proto))
    # old items
    old = []
    for line in data:
        if len(line) == 20 and line[1] == 2:
            item = np.resize(line[2:], (9, 2))
            pofr = computeresponse(item, memory, c, k)
            old.append(pofr)
    # print "p of r", old
    # print(np.average(old))

    # new high items
    newhigh = []
    for line in data:
        if len(line) == 20 and line[1] == 3:
            item = np.resize(line[2:], (9, 2))
            pofr = computeresponse(item, memory, c, k)
            newhigh.append(pofr)
    # print(np.average(newhigh))

    # new low items
    newlow = []
    for line in data:
        if len(line) == 20 and line[1] == 4:
            item = np.resize(line[2:], (9, 2))
            pofr = computeresponse(item, memory, c, k)
            newlow.append(pofr)
    # print(np.average(newlow))

    # random items
    random = []
    for line in data:
        if len(line) == 20 and line[1] == 5:
            item = np.resize(line[2:], (9, 2))
            pofr = computeresponse(item, memory, c, k)
            random.append(pofr)
    # print(np.average(random))

    return [
        np.average(proto),
        np.average(newlow),
        np.average(newhigh),
        np.average(random),
        np.average(old),
        filename,
        cond,
    ]
```

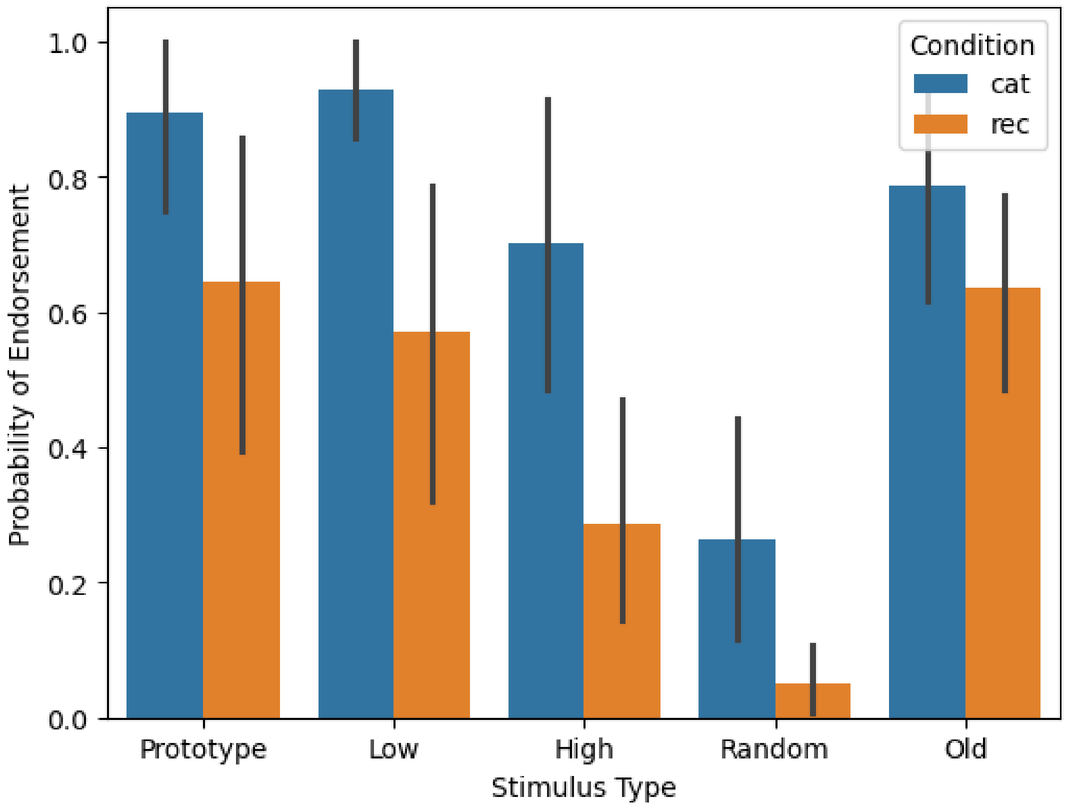
[19]:
```
def get_prototype_results(c_cat, k_cat, c_rec, k_rec):
    allres = {fn: readfile(fn) for fn in get_all_filenames("data")}
    cat = []
    rec = []
    for filename in allres.keys():
        if allres[filename][-1][1] == 0:
            res = prototypemodel(filename, c_cat, k_cat)
            cat.append(create_df(filename, "cat", res[:-2]))
        else:
            res = prototypemodel(filename, c_rec, k_rec)
            rec.append(create_df(filename, "rec", res[:-2]))
    cat, rec = pd.concat(cat), pd.concat(rec)
    return pd.concat([cat, rec])
```

Again lets replot the human results for easy reference.

[20]:
```
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)
```

[20]:
```
<Axes: xlabel='Stimulus Type', ylabel='Probability of Endorsement'>
```



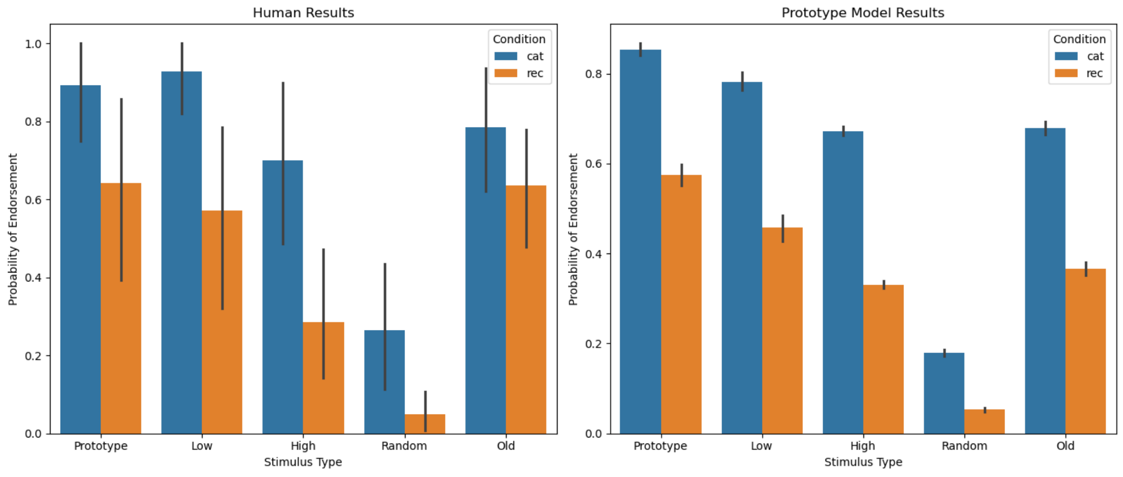
Problem 3 (10 points)

By hand adjust the setting of the model parameters in the next cell to roughly fit the human data pattern shown above. How close can you get? What parameters did you find (report them) and you assessment of how well they fit. Was it a good fit or are there systematic problems with the fit? In addition, what are the parameter values and do they make sense in light of the equations described above?

[53]:
```
# Set up a matplotlib figure with two subplots
plt.figure(figsize=(14, 6))

# First subplot for the human results
plt.subplot(1, 2, 1)
sns.barplot(
    x="Stimulus Type",
    y="Probability of Endorsement",
    hue="Condition",
    data=get_human_results(),
)
plt.title('Human Results')

# Second subplot for the prototype results
```

```python
plt.subplot(1, 2, 2)
c_cat, k_cat, c_rec, k_rec = 1.6, 0.05, 1.5, 0.25
sns.barplot(
    x="Stimulus Type",
    y="Probability of Endorsement",
    hue="Condition",
    data=get_prototype_results(c_cat, k_cat, c_rec, k_rec)
)
plt.title('Prototype Model Results')

# Show the plots
plt.tight_layout()
plt.show()
```

```
/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)
/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)
```



For the categorization condition in the protype model, we maintain a moderate sensitivity to similarity with c_cat = 1.6, and a much lower k value (k_cat = 0.05) in order to allow for a broader range of items to be accepted into the category, given we are now only comparing each stimulus with a single prototype (as opposed to comparing it with each possible exemplar in the exemplar model). We have similar parameters for the the recognition condition with c_rec = 1.5 and k_rec = 0.25, though the k_rec value is higher than that of the k_cat value in order to raise the bar for a pattern to be recognized as something the subject has seen before. In regards to systematic problems, the model underestimates the probability for Old stimuli in the recognition condition and again underestimates the probability for the Low stimuli in the categorization condition. Thus, again, while the prototype model captures the general trends of the human data, it doesn't fully account for the nuances of the empirical data.

3.4 Fitting the models using RMSE

Next we would like to come up with a more quantitative way to assess the quality of the model fits. The first technique we will use is the "goodness of fit" measures that were discussed in lecture. One of the most common measures of goodness of fit is the Root Mean Squared Error (RMSE). This measure compares the value of each data point x to each prediction y using the following equation:

$$RMSE = \sqrt{\frac{\sum_i (x_i - y_i)^2}{N}}$$

Often the RMSE is computed between the AVERAGE prediction of the model and the AVERAGE estimates of the behavior to all the subjects in an experiment. Using the code we developed above we can find the average endorsement curves for humans and both models like this:

[27]:
```
human_results = get_human_results()
exemplar_predictions = get_exemplar_results(c_cat, k_cat, c_rec, k_rec)
prototype_predictions = get_prototype_results(c_cat, k_cat, c_rec, k_rec)
```

[28]:
```
avghuman = human_results.groupby(['Condition', 'Stimulus Type'],as_index=False).
  ↪mean(numeric_only=True)
avghuman
```

```
[28]:   Condition Stimulus Type  Probability of Endorsement
      0       cat          High                    0.700000
      1       cat           Low                    0.928571
      2       cat           Old                    0.785714
      3       cat     Prototype                    0.892857
      4       cat        Random                    0.264286
      5       rec          High                    0.285714
      6       rec           Low                    0.571429
      7       rec           Old                    0.635714
      8       rec     Prototype                    0.642857
      9       rec        Random                    0.050000
```

```
[29]: avgexemplar = exemplar_predictions.groupby(
          ["Condition", "Stimulus Type"], as_index=False
      ).mean(numeric_only=True)
      avgexemplar
```

```
[29]:   Condition Stimulus Type  Probability of Endorsement
      0       cat          High                    0.929990
      1       cat           Low                    0.947209
      2       cat           Old                    0.969362
      3       cat     Prototype                    0.956653
      4       cat        Random                    0.672546
      5       rec          High                    0.747763
      6       rec           Low                    0.805181
      7       rec           Old                    0.872589
      8       rec     Prototype                    0.838669
      9       rec        Random                    0.287387
```

```
[30]: avgprototype = prototype_predictions.groupby(
          ["Condition", "Stimulus Type"], as_index=False
      ).mean(numeric_only=True)
      avgprototype
```

```
[30]:   Condition Stimulus Type  Probability of Endorsement
      0       cat          High                    0.671780
      1       cat           Low                    0.780866
      2       cat           Old                    0.678446
      3       cat     Prototype                    0.852571
      4       cat        Random                    0.178578
      5       rec          High                    0.300755
      6       rec           Low                    0.431930
      7       rec           Old                    0.338413
      8       rec     Prototype                    0.557275
      9       rec        Random                    0.040328
```

Problem 4 (20 points)

First, write a function below called `rmse` that computes the RMSE between two `numpy` vectors.

```
[31]: def rmse(human, model):
          return np.sqrt(np.mean(human-model)**2)
```

Write your code above. This code will then by used in the provided functions below to evaluate the fit of the prototype and exemplar models. The parameters to the model is provided as a list with `[c_cat, k_cat, c_rec, k_rec]` the implied order.

```
[32]: def fit_exemplar_model_rmse(params, human_results):
          [c_cat, k_cat, c_rec, k_rec] = params
          predictions = get_exemplar_results(c_cat, k_cat, c_rec, k_rec)
          avgpredict = predictions.groupby(
              ["Condition", "Stimulus Type"], as_index=False
          ).mean(numeric_only=True)
          model_results = avgpredict["Probability of Endorsement"].values
          return rmse(human_results, model_results)


      def fit_prototype_model_rmse(params, human_results):
          [c_cat, k_cat, c_rec, k_rec] = params
          predictions = get_prototype_results(c_cat, k_cat, c_rec, k_rec)
          avgpredict = predictions.groupby(
              ["Condition", "Stimulus Type"], as_index=False
          ).mean(numeric_only=True)
          model_results = avgpredict["Probability of Endorsement"].values
          return rmse(human_results, model_results)
```

Next adjust the parameters by hand for both the exemplar and prototype models to find values that appear to minimize the RMSE. Copy the code above for plotting the predictions of the models given your best fit parameters. Which model do you think fits better according to this fit statistic?

I already messed with the numbers quite extensively using the plots, and it looks like my parameter choices are pretty good. Thus, we will utilize the same ones I used earlier here:

```
[51]: human_results = get_human_results()
      avghuman = human_results.groupby(["Condition", "Stimulus Type"],␣
       ↪as_index=False).mean(numeric_only=True)
      human_results = avghuman["Probability of Endorsement"].values

      print(fit_exemplar_model_rmse([1.9, 0.15, 1.5, 1.0], human_results))
      print(fit_prototype_model_rmse([1.6, 0.05, 1.5, 0.25], human_results))
```

```
0.010356406114948546
0.08128634876549853
```

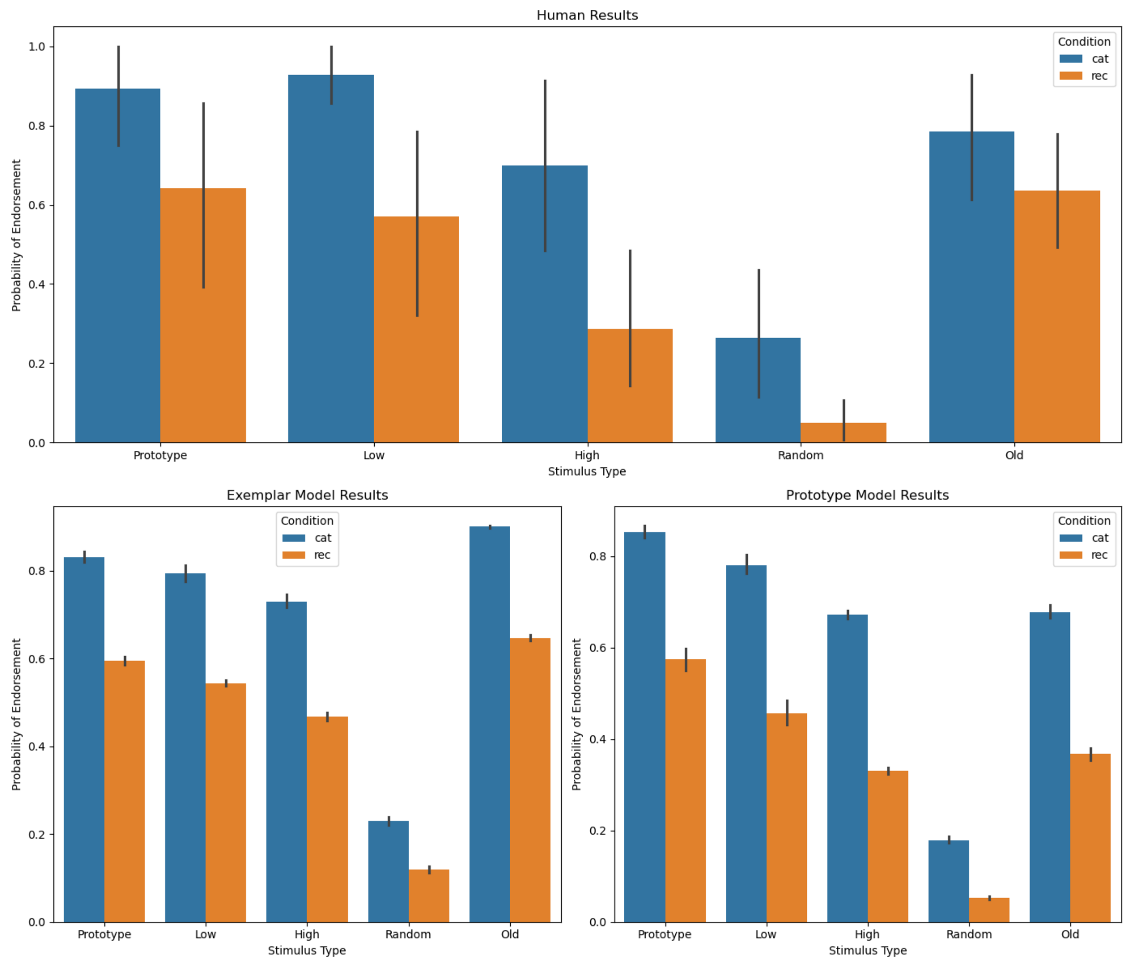
Given the rmse of the exemplar model is 0.0103 and the rmse of the prototype model is 0.0812, it can be argued that both models fit the human data quite well. When considering which is a better fit, it is hard to really say since they are performing so similarly and the rmse's are both < 0.1.

After running the fmin below, we will see.

```
[57]: # Set up a matplotlib figure with two subplots
plt.figure(figsize=(14, 12))  # Width, height in inches

# First subplot for the human results
plt.subplot(2, 2, (1,2))  # (number of rows, number of columns, subplot number)
sns.barplot(
    x="Stimulus Type",
    y="Probability of Endorsement",
    hue="Condition",
    data=get_human_results(),
)
plt.title('Human Results')

# Second subplot for the exemplar results
plt.subplot(2, 2, 3)
c_cat, k_cat, c_rec, k_rec = 1.9, 0.15, 1.5, 1.0
sns.barplot(
    x="Stimulus Type",
    y="Probability of Endorsement",
    hue="Condition",
    data=get_exemplar_results(c_cat, k_cat, c_rec, k_rec),
)
plt.title('Exemplar Model Results')

# Second subplot for the exemplar results
plt.subplot(2, 2, 4)
c_cat, k_cat, c_rec, k_rec = 1.6, 0.05, 1.5, 0.25
sns.barplot(
    x="Stimulus Type",
    y="Probability of Endorsement",
    hue="Condition",
    data=get_prototype_results(c_cat, k_cat, c_rec, k_rec)
)
plt.title('Prototype Model Results')

# Show the plots
plt.tight_layout()
plt.show()
```

```
/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)
/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)
/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)
```



Problem 5 (10 points)

Read about the scipy `fmin` function. Use fmin to algorithmically search for the best parameters for each model using the RMSE evaluation function described above.

[59]:
```
# Define the objective functions for fmin
def objective_function_exemplar(params):
    return fit_exemplar_model_rmse(params, human_results)

def objective_function_prototype(params):
    return fit_prototype_model_rmse(params, human_results)

# Define initial guesses for the parameters
initial_params_exemplar = [1.9, 0.15, 1.5, 1.0]
initial_params_prototype = [1.6, 0.05, 1.5, 0.25]

# Perform the optimization using fmin for the exemplar model
```

```
optimal_params_exemplar = fmin(objective_function_exemplar,␣
↪initial_params_exemplar)

# Do the same for the prototype model
optimal_params_prototype = fmin(objective_function_prototype,␣
↪initial_params_prototype)

# Print the optimal parameters found by fmin
print(f"Optimal Exemplar Model Parameters: {optimal_params_exemplar}")
print(f"Optimal Prototype Model Parameters: {optimal_params_prototype}")

# Finally, we use the optimal parameters to calculate the final RMSE and compare
final_rmse_exemplar = objective_function_exemplar(optimal_params_exemplar)
final_rmse_prototype = objective_function_prototype(optimal_params_prototype)

print(f"Final RMSE Exemplar Model: {final_rmse_exemplar}")
print(f"Final RMSE Prototype Model: {final_rmse_prototype}")
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 62
         Function evaluations: 115
Optimization terminated successfully.
         Current function value: 0.000001
         Iterations: 63
         Function evaluations: 118
Optimal Exemplar Model Parameters: [1.9329074  0.15040508 1.5354055  1.00636773]
Optimal Prototype Model Parameters: [0.95247697 0.05817547 1.30820838
0.29823728]
Final RMSE Exemplar Model: 7.668831003143639e-08
Final RMSE Prototype Model: 5.705257571084199e-07
```

Here, we find that the exemplar model slightly outperforms the prototype model (7.668831003143639e-08 vs. 5.705257571084199e-07), though, both models can perform amazingly well when we utilize fmin. The best performing parameters are:

Optimal Exemplar Model Parameters: [1.9329074 0.15040508 1.5354055 1.00636773]

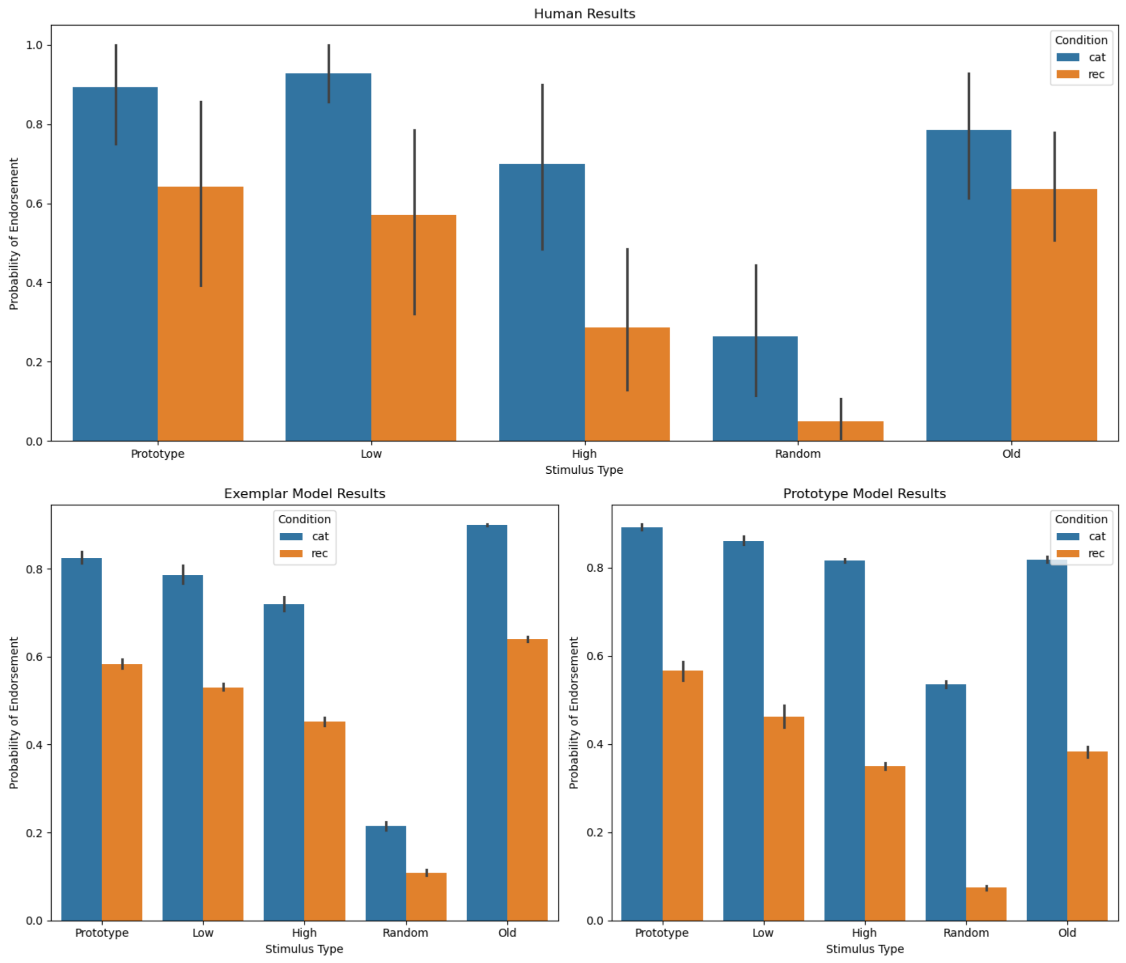
Optimal Prototype Model Parameters: [0.95247697 0.05817547 1.30820838 0.29823728]

And you can see the updated bar graphs below:

[62]:
```
# Print the bar graphs with the optimal solutions
plt.figure(figsize=(14, 12))  # Width, height in inches

# First subplot for the human results
plt.subplot(2, 2, (1,2))  # (number of rows, number of columns, subplot number)
sns.barplot(
    x="Stimulus Type",
    y="Probability of Endorsement",
    hue="Condition",
    data=get_human_results(),
)
plt.title('Human Results')

# Second subplot for the exemplar results
plt.subplot(2, 2, 3)
c_cat, k_cat, c_rec, k_rec = 1.9329074, 0.15040508, 1.5354055, 1.00636773
sns.barplot(
    x="Stimulus Type",
    y="Probability of Endorsement",
    hue="Condition",
    data=get_exemplar_results(c_cat, k_cat, c_rec, k_rec),
)
plt.title('Exemplar Model Results')

# Second subplot for the exemplar results
plt.subplot(2, 2, 4)
c_cat, k_cat, c_rec, k_rec = 0.95247697, 0.05817547, 1.30820838, 0.29823728
sns.barplot(
    x="Stimulus Type",
    y="Probability of Endorsement",
    hue="Condition",
    data=get_prototype_results(c_cat, k_cat, c_rec, k_rec)
)
plt.title('Prototype Model Results')

# Show the plots
plt.tight_layout()
plt.show()
```

```
/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)
/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)
/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)
```



[ ]:

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]:
```python
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]:
```python
#################################
# 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?

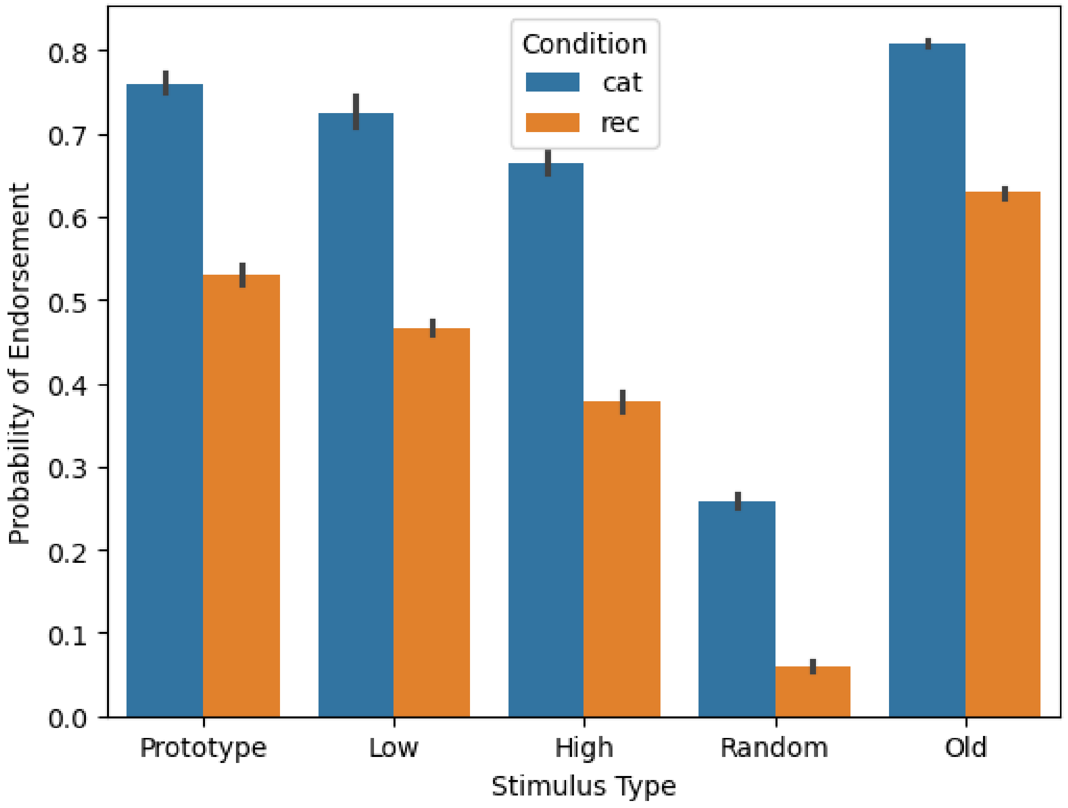
Exemplar model

[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'>
```



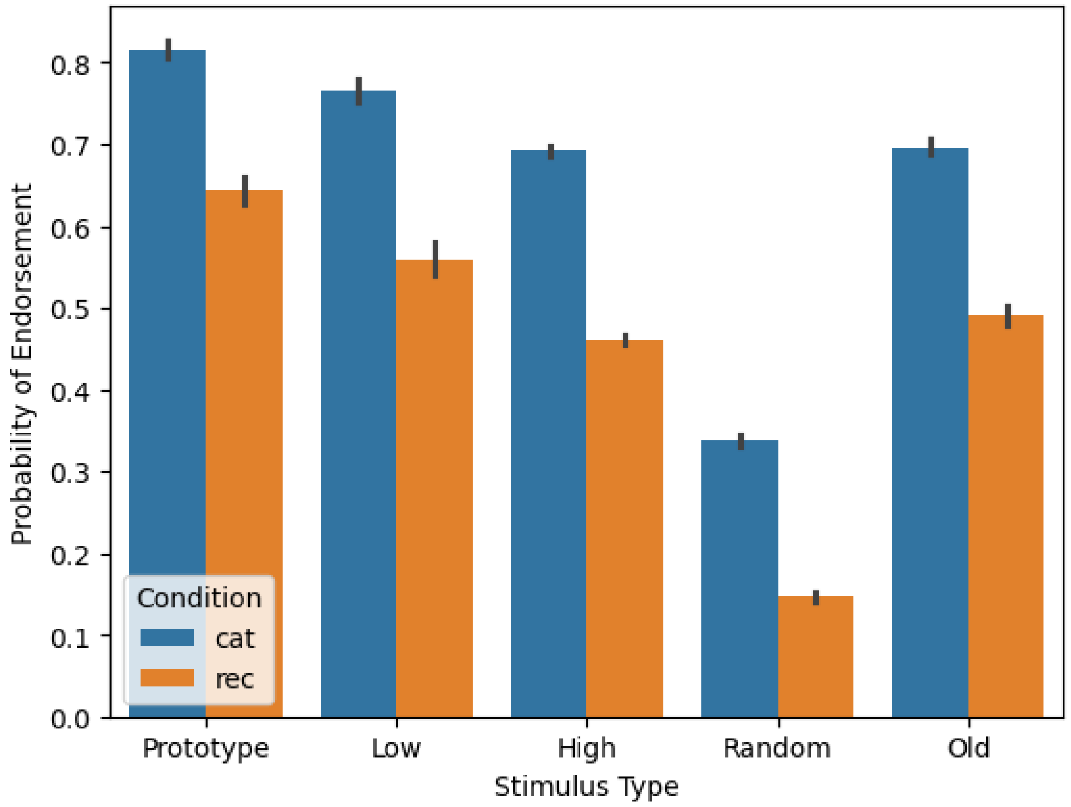
Prototype Model

```
[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'>
```

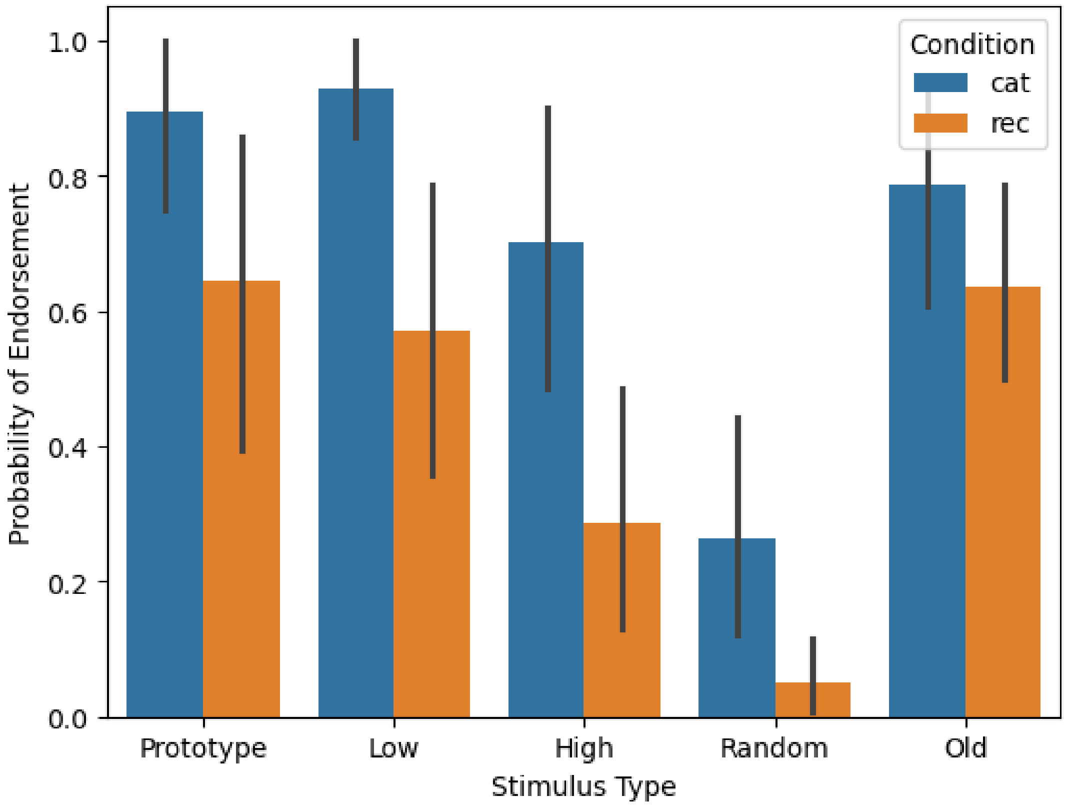


Human data again for reference

[32]:
```python
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)?

```
[ ]:
```