

# Homework - Bayesian modeling - Part A (100 points)

## Bayesian concept learning with the number game

by *Brenden Lake* and *Todd Gureckis*

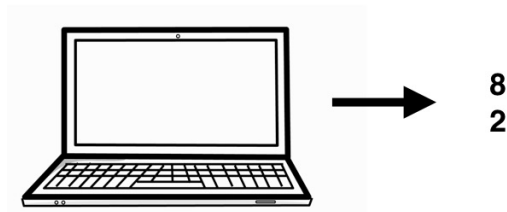
Computational Cognitive Modeling

NYU class webpage: <https://brendenlake.github.io/CCM-site/>

In this notebook, you will implement (mostly from scratch!) a Bayesian concept learning model and the "number game," as covered in lecture. As with so many of our everyday inferences, the data we receive is far too sparse and noisy to be conclusive. Nevertheless, people must make generalizations and take actions based on imperfect and insufficient data. In data science and machine learning, the situation is often the same: the data is not enough to produce an answer with certainty, yet we can make meaningful generalizations anyway. What computational mechanisms can support these types of inferences?

The number game is a quintessential inductive problem. In the number game, there is an unknown computer program that generates numbers in the range 1 to 100. You are provided with a small set of random examples from this program. For instance, in the figure below, you get two random examples from the program: the numbers '8' and '2'.

**random "yes" examples:**



**Which numbers will be accepted by the same computer program?**

**9? 10? 16?**

Which numbers will also be accepted by the same program? Of course, it depends what the program is, and you don't have enough information to be sure. Should '9' be accepted? Perhaps, if the concept is "all numbers up to 10." What about '10'? A better candidate, since the program could again be "numbers up to 10", or "all even numbers." What about '16'? This is another good candidate, and the program "powers of 2" is also consistent with the examples so far. How should one generalize based on the evidence so far? This homework explores how the Bayesian framework provides an answer to this question.

**You should read the following paper carefully.**

Josh Tenenbaum's paper introduced the number game. You can download the paper on EdStem:

- Tenenbaum, J. B. (2000). Rules and similarity in concept learning. In Advances in Neural Information Processing Systems (NIPS).

## The Bayesian model

In the number game, we receive a set of  $n$  positive examples  $X = \{x^{(1)}, \dots, x^{(n)}\}$  of an unknown concept  $C$ . In a Bayesian analysis of the task, the goal is predict  $P(y \in C \mid X)$ , which is the probability that a new number  $y$  is also a member of the concept  $C$  after receiving the set of examples  $X$ .

### Updating beliefs with Bayes' rule

Let's proceed with the Bayesian model of the task. There is a hypothesis space  $H$  of concepts, where a particular member of the hypothesis space (i.e., a particular concept) is denoted  $h \in H$ . The Bayesian model includes a prior distribution  $P(h)$  over the hypotheses and a likelihood  $P(X|h)$ . Bayes' rule specifies how to compute the posterior distribution over hypotheses given these two pieces:

$$P(h|X) = \frac{P(X|h)P(h)}{\sum_{h' \in H} P(X|h')P(h')} \quad (1)$$

The likelihood and prior are specified below.

### Likelihood

We assume that each number in  $X$  is an independent sample from the set of all valid numbers. Thus, the likelihood decomposes as a product of individual probabilities,

$$P(X|h) = \prod_{i=1}^n P(x^{(i)}|h). \quad (2)$$

We assume that the numbers are sampled uniformly at random from the set of valid numbers, such that  $P(x^{(i)}|h) = \frac{1}{|h|}$  if  $x^{(i)} \in h$  and  $P(x^{(i)}|h) = 0$  otherwise. The term  $|h|$  is the cardinality or set size of the hypothesis  $h$ .

### Prior

The hypothesis space  $H$  includes two main kinds of hypotheses. You can think of each hypothesis as a list of the numbers that fit that hypothesis.

- The first kind consists of mathematical hypotheses such as odd numbers, even numbers, square numbers, cube numbers, primes, multiples of  $n$ , powers of  $n$ , and numbers ending with a particular digit. Each mathematical hypothesis is given equal weight in the prior.

- The second kind consists of interval hypotheses, which are solid intervals of numbers, such as 12, 13, 14, 15, 16, 17. Intervals of intermediate size are favored (rather than very small or large hypotheses) by reweighting according to an Erlang distribution,

$$P(h) \propto \frac{|h|}{\sigma^2} \exp -|h|/\sigma \text{ such that } \sigma = 10.$$

There is a free parameters `mylambda` controls how much of the prior is specified by each type of hypothesis, with `mylambda` weight going to the mathematical hypotheses and `1 - mylambda` weights going to the interval hypotheses.

We provide starter code below that generates the mathematical hypotheses and their prior probabilities (in natural log space).

## Making Bayesian predictions

Once we have the posterior beliefs over hypotheses, we want to be able to make predictions about the membership of a new number  $y$  in the concept  $C$ , or as mentioned  $P(y \in C \mid X)$ . To compute this, we average over all possible hypotheses weighted by the posterior probability,

$$P(y \in C \mid X) = \sum_{h \in H} P(y \in C \mid h) P(h \mid X), \quad (3)$$

where the first term is simply 1 or 0 based on the membership of  $y$  in  $h$ , and the second term is the posterior weight.

```
In [ ]: # Here are some packages that may be useful
from __future__ import print_function
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from scipy.special import logsumexp
x_max = 100 # (numbers between 1 and 100 are allowed)
```

```
In [ ]: # Generate a list of all mathematical hypotheses
def make_h_odd():
    return list(range(1,x_max+1,2))

def make_h_even():
    return list(range(2,x_max+1,2))

def make_h_square():
    h = []
    for x in range(1,x_max+1):
        if x**2 <= x_max:
            h.append(x**2)
    return h

def make_h_cube():
    h = []
    for x in range(1,x_max+1):
        if x**3 <= x_max:
            h.append(x**3)
    return h
```

```

def make_h_primes():
    return [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71]

def make_h_mult_of_y(y):
    h = []
    for x in range(1, x_max+1):
        if x*y <= x_max:
            h.append(x*y)
    return h

def make_h_powers_of_y(y):
    h = []
    for x in range(1, x_max+1):
        if y**x <= x_max:
            h.append(y**x)
    return h

def make_h_numbers_ending_in_y(y):
    h = []
    for x in range(1, x_max+1):
        if str(x)[-1] == str(y):
            h.append(x)
    return h

def generate_math_hypotheses(mylambda):
    h_set = [make_h_odd(), make_h_even(), make_h_square(), make_h_cube(), make_h_prime()]
    h_set += [make_h_mult_of_y(y) for y in range(3,13)]
    h_set += [make_h_powers_of_y(y) for y in range(2,11)]
    h_set += [make_h_numbers_ending_in_y(y) for y in range(0,10)]
    n_hyp = len(h_set)
    log_prior = np.log(mylambda * np.ones(n_hyp) / float(n_hyp))
    return h_set, log_prior

h_set_math, log_prior_math = generate_math_hypotheses(2./3)
print("Four examples of math hypotheses:")
for i in range(4):
    print(h_set_math[i])
    print("")
print("Their prior log-probabilities:")
print(log_prior_math[0:4])

```

Four examples of math hypotheses:

[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99]

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100]

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

[1, 8, 27, 64]

Their prior log-probabilities:

[-3.93182563 -3.93182563 -3.93182563 -3.93182563]

```

In [ ]: ## Generate a list of all interval hypotheses
def make_h_between_y_and_z(y,z):

```

```

assert(y >= 1 and z <= x_max)
return list(range(y,z+1))

def pdf_erlang(x,sigma=10.):
    return (x / sigma**2) * np.exp(-x/sigma)

def generate_interval_hypotheses(mylambda):
    h_set = []
    for y in range(1,x_max+1):
        for z in range(y,x_max+1):
            h_set.append(make_h_between_y_and_z(y,z))
    nh = len(h_set)
    pv = np.ones(nh)
    for idx,h in enumerate(h_set): # prior based on length
        pv[idx] = pdf_erlang(len(h))
    pv = pv / np.sum(pv)
    pv = (1-mylambda) * pv
    log_prior = np.log(pv)
    return h_set, log_prior

h_set_int, log_prior_int = generate_interval_hypotheses(2./3)
print("Four examples of interval hypotheses")
for i in range(4):
    print(h_set_int[i])
    print("")
print("Their prior log-probabilities:")
print(log_prior_int[0:4])

```

Four examples of interval hypotheses

[1]

[1, 2]

[1, 2, 3]

[1, 2, 3, 4]

Their prior log-probabilities:

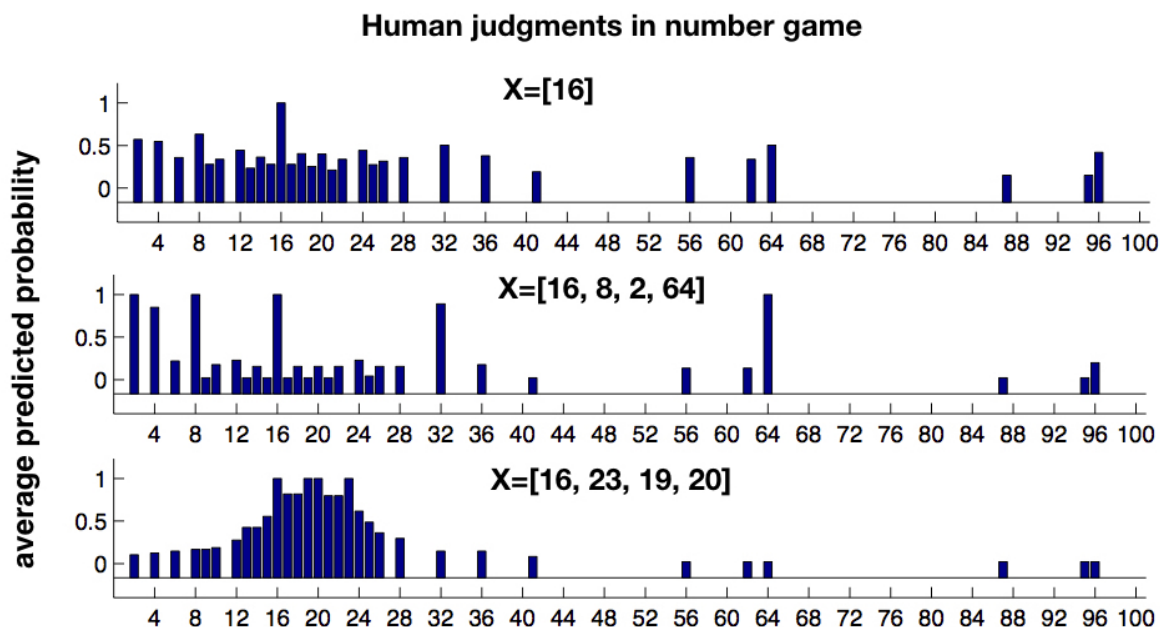
[-10.197254    -9.60410682   -9.29864171   -9.11095964]

## Human behavioral judgments

Tenenbaum ran eight participants in an experiment where they were provided with various sets  $X$  of random positive examples from a concept. They were asked to rate the probability that each of 30 test numbers would belong to the same concept of the observed examples.

The following plot shows the mean rating across the human participants for three different sets. Note that since only 30 test numbers were evaluated, and thus a value of 0 in the plot indicates

missing data (rather than zero probability).



Your goal is to implement the Bayesian concept learning model in order to produce the same plots, although with the model judgements rather than the human judgements.

```
In [ ]: # The 30 test numbers that Tenenbaum used are here
x_eval = [2,4,6,8,9,10]+list(range(12,23))+[24,25,26,28,32,36,41,56,62,64,87,95,96]
```

## Implementation

### Problem 1 (90 points)

Your main task is to produce the same three plots as shown above, although showing model predictions rather than human behavioral judgements. To do so, you'll need to implement the Bayesian concept learning model. A successful implementation will include the following components:

- A function for computing the log-probability of a hypothesis  $h$  according to the prior (largely provided in starter code).
- A function for computing the log-likelihood of a set of numbers  $X$  given a particular hypothesis  $h$ .
- A function for computing the log-posterior over all hypotheses  $h$  given a set of numbers  $X$  that were sampled from  $h$ .
- According to the "Making Bayesian predictions" section above, a function for computing the probability that a new number  $y$  belongs to the same concept as a set of sampled numbers  $X$
- Code for making the plots

Tip: For probabilistic modeling in general, we like to compute probabilities in log-space to help avoid numerical issues such as underflow. For instance, rather than multiplying the prior and likelihood (resulting in potentially very small numbers), we sum the log-prior and the log-likelihood. Also, check out the nifty `logsumexp` function ([see scipy doc](#)) which is used to normalize log-probability distributions in a numerically safer way. This function is already loaded.

```
In [ ]: # Prior and hypotheses
h_set = h_set_math + h_set_int
log_prior = np.concatenate((log_prior_math, log_prior_int))
```

```
In [ ]: # LogLikelihood
def compute_log_likelihood(X, h):
    log_likelihood = 0
    for x in X:
        if x in h:
            log_likelihood += np.log(1/len(h))
        else:
            return -np.inf
    return log_likelihood
```

```
In [ ]: # Log-posterior
def compute_log_posterior(X, h_set, log_prior):
    log_posterior = np.zeros(len(h_set))
    for i, h in enumerate(h_set):
        log_posterior[i] = compute_log_likelihood(X, h) + log_prior[i]
    return log_posterior - logsumexp(log_posterior)
```

```
In [ ]: # Prob y belongs to same concept as X
def bayes_Prediction(y, X, h_set, log_prior):
    posterior = np.exp(compute_log_posterior(X, h_set, log_prior), )
    prob = 0
    for h in h_set:
        if y in h:
            prob += posterior[h_set.index(h)]
    return prob
```

```
In [ ]: # function for plotting
def plot_prediction(y_set, X, h_set, log_prior):
    p = []
    for y in y_set:
        p.append(bayes_Prediction(y, X, h_set, log_prior))

    plt.figure(figsize=(10,2))
    plt.bar(y_set, p, bottom=0)
    plt.xlim(0,100)
    plt.xlabel("y")
    plt.ylabel("P(y|X)")
    plt.title(f"Bayes prediction for X = {X}")
    plt.yticks([0,0.5,1])
    plt.xticks(np.arange(0, x_max+1, step=4))
    plt.show()
```

```
In [ ]: X = [16]
```

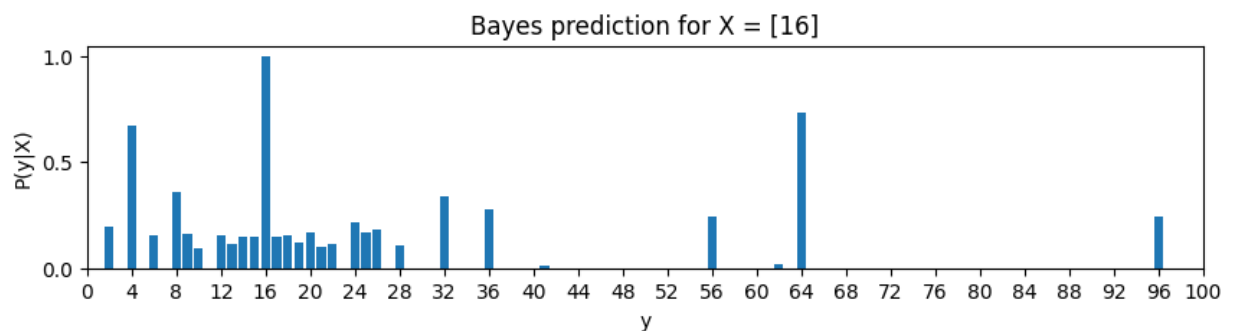
```
p = []
for y in x_eval:
    p.append(bayes_Prediction(y, X, h_set, log_prior))
p
```

```
Out[ ]: [0.19394788511466782,
0.6735467829235368,
0.1531754882030396,
0.35805071361931945,
0.16011114289506695,
0.09181103688167794,
0.1566097425656057,
0.11188895636422394,
0.1478506585029739,
0.14594202330170472,
0.9999999999999972,
0.14992774594054578,
0.15544281212335131,
0.12274434368225971,
0.1704178258370784,
0.10048845628011119,
0.11070829291747189,
0.21623307679437342,
0.1662772424253282,
0.17965078101037257,
0.10924147456380928,
0.34010689730319726,
0.27961103415045546,
0.013570475958588152,
0.243728855770476,
0.0214179866013078,
0.7367019554597903,
0.00010302572372842273,
2.7723024784020907e-05,
0.24074894627294663]
```

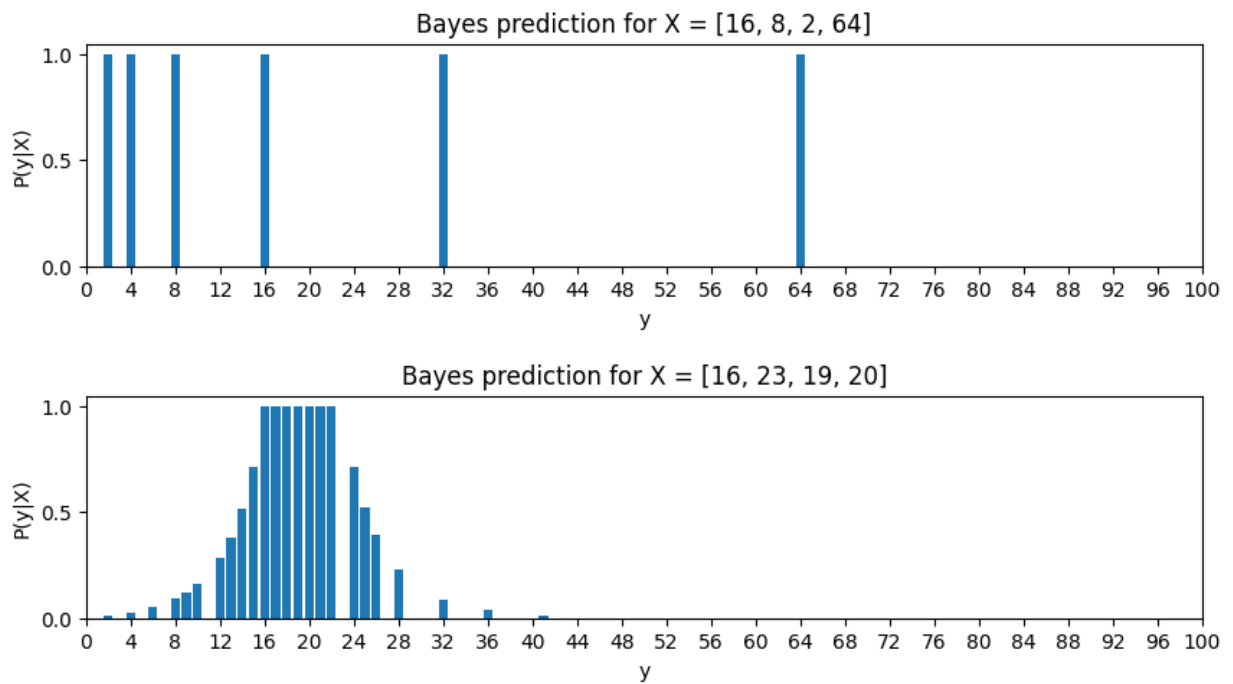
```
In [ ]: # Plots
X = [16]
plot_prediction(x_eval, X, h_set, log_prior)

X = [16, 8, 2, 64]
plot_prediction(x_eval, X, h_set, log_prior)

X = [16, 23, 19, 20]
plot_prediction(x_eval, X, h_set, log_prior)
```







## Problem 2 (10 points)

Discuss your general thoughts on this Bayesian model to understand human judgments in the number game. Discussion questions could include the following (as well as others):

- Is the model convincing? Why or why not?
- Is the number game and Bayesian model relevant to more naturalistic settings for concept learning in childhood or everyday life?
- Where could the hypothesis space come from?
- What algorithms could people be using to approximate Bayesian inference, rather than enumerating all the hypotheses, as in the current implementation?

Please write a short response in the cell below. Your response should be about two paragraphs.

YOUR RESPONSE GOES HERE

The model did a great job at mimicking human judgement in this game, though noticeably not perfect. To start, the model is way more decisive than the averages of a bunch of people, noticed particularly in the second graph where the scattered small probabilities in the human graph are non-existent in the model's. Perhaps this is due to human error, but it shows the model is too perfect to perfectly mimic the noisy decisions of humans, though the logic it develops is sound. This kind of model does a great job of reflecting pattern recognition in humans though, a trait we especially learn as we grow older and learn languages and object recognition.

The hypothesis space most likely comes from common guesses humans look for in patterns, more specifically in numeric patterns. We pick up on odd/even, intervals, and simple multiplicities rather well as they are simple rules nearly everyone learns in elementary school, so it is rather ingrained in us. However, when humans see this problem, they are most likely not checking every single rule they know and doing all the likelihood math in their head. Instead, I believe we cheat a bit and try and identify a few rules that could be in place, then generate numbers that agree with that rule and mark those as more likely, which is similar to this Bayes but the rules are filtered, and the numbers considered are prediction instead of testing a large amount.