Exercises: Chapter 7

Exercise 7.1: Experimenting with Priors and Predictives

In our simple binomial model, we compared the parameter priors and posteriors to the corresponding **predictives** which tell us what data we should expect given our prior and posterior beliefs. For convenience, we've reproduced that model here.

Exercise 7.1.1

Notice that we used a uniform distribution over the interval [0, 1] as our prior, reflecting our assumption that a probability must lie between 0 and 1 but otherwise remaining agnostic to which values are most likely to be the case. While this is convenient, we may want to represent other assumptions.

The Beta distribution, expressed in WebPPL as `Beta({a:..., b:...})`' is a more general way of expressing beliefs over the interval $[0, 1]$.

The beta distribution is what's called the **conjugate prior** probability distribution for the binomial distribution due to its relationship between the prior and the posterior, and it also has a really neat interpretation that we will explore in this problem.

You may want to visualize the beta distribution a few times with different parameters to get a sense of its shape:

1. Beta(1, 1)
2. Beta(3, 3)
3. Beta(50, 50)
4. Beta(1, 10)
5. Beta(10, 1)
6. Beta(.2, .2)

```
viz(
  repeat(
    10000,
    function() {
      sample(
        Beta({a:1, b: 1})
      )
    }
  )
)
```

Here, we have the binomial distribution example from the chapter:

```
// Observed data
// Number of successes
const k = 1
const n = 20
var priorDist = Uniform({
  a: 0,
  b: 1
})
var model = function() {
  var p = sample(priorDist)
  // Observed k number of successes,
  // assuming a binomial
  observe(
    Binomial({p : p, n: n}),
    k
  )

  // Sample from binomial with updated p
  var posteriorPredictive = binomial(p, n)

  // Sample fresh p (for visualization)
  var prior_p = sample(priorDist)
  // Sample from binomial with fresh p (for visualization)
  var priorPredictive = binomial(prior_p, n)

  return {
    prior: prior_p,
    priorPredictive: priorPredictive,
    posterior: p,
```

```
    posteriorPredictive: posteriorPredictive
  }
}

var opts = {method: "MCMC", samples: 2500, lag: 50}
var posterior = Infer(opts, model)
viz.marginals(posterior)
```

Using the code above, answer the following questions.

1. Run the code as is. How does the posterior compare to Beta$(2, 20)$?
2. Set the prior to Beta$(1, 1)$. What do you notice about the posterior distribution?
3. Set $n = 10$ and the prior to Beta$(1, 11)$. What do you notice about the posterior distribution?
4. Set $k = 5$, $n = 15$, and the prior to Beta$(1, 1)$. Compare the posterior to Beta$(6, 11)$.
5. Set $k = 4$, $n = 10$, and the prior to Beta$(1, 1)$. What a and b values of Beta(a, b) does the posterior look like?
6. Set $k = 10$ and $n = 20$. What a and b values would make the posterior look like Beta$(12, 10)$?
7. Based on these observations (and any others you may have tried), what is the relationship between the beta distribution and the binomial distribution?

Exercise 7.1.2

Predictive distributions are not restricted to exactly the same experiment as the observed data, and can be used in the context of any experiment where the inferred model parameters make predictions. In the current simple binomial setting, for example, predictive distributions could be found by an experiment that is different because it has `n' != n` observations.

Change the model to implement an example of this:

```
// Observed data
// Number of successes
const k = 1
// Number of attempts
const n = 20
const priorDist = Uniform({
  a: 0,
  b: 1
})
var model = function() {
  var p = sample(priorDist)
```

```
  // Observed k number of successes,
  // assuming a binomial
  observe(
    Binomial({p : p, n: n}),
    k
  )

  // Sample from binomial with updated p
  var posteriorPredictive = binomial(p, n)

  // Sample fresh p (for visualization)
  var prior_p = sample(priorDist)
  // Sample from binomial with fresh p (for visualization)
  var priorPredictive = binomial(prior_p, n)

  return {
    prior: prior_p,
    priorPredictive: priorPredictive,
    posterior: p,
    posteriorPredictive: posteriorPredictive
  }
}

var opts = {method: "MCMC", samples: 2500, lag: 50}
var posterior = Infer(opts, model)
viz.marginals(posterior)
```

Exercise 7.2: Parameter Fitting vs. Parameter Integration

One of the strongest motivations for using Bayesian techniques for model-data evaluation is in how "nuisance" parameters are treated.

"Nuisance" parameters are parameters of no theoretical interest; their only purpose is to fill in a necessary slot in the model.

Classically, the most prominant technique (from the frequentist tradition) for dealing with these parameters is to fit them to the data, i.e., to set their value equal to whatever value maximizes the model-data fit (or, equivalently, minimizes some cost function).

The Bayesian approach is different. Since we have *a priori* uncertainty about the value of our parameter, we will also have *a posteriori* uncertainty about the value (though hopefully

the uncertainty will be reduced). What the Bayesian does is *integrate over* her posterior distribution of parameter values to make predictions.

Intuitively, rather than taking the value corresponding to the peak of the distribution (i.e., the maximum), she's considering all values with their respective probabilites.

Why might this be important for model assessment? Imagine the following situation. You are piloting a task and want to use Bayesian Data Analysis because you hear it is useful when you have few data points. You think that the task you've designed is a little too difficult for subjects. (Let's imagine that you're a psychophysicist, and your task pertains to contrast discriminiation in the peripheral visual field.) You think the current task design is too difficult, but you're not sure. It may well be that it's fine for subjects.

Here is your prior:

```
// Prior on task difficulty is uniform
// on [0, ..., 0.9], with a spike on 0.9
// i.e., you think it's likely that the task
// is too difficult
var sampleTaskDifficulty = function() {
  return flip()
    ? .9
    : randomInteger(10) / 10
}
var model = function() {
  return sampleTaskDifficulty();
}
viz.hist(
  Infer(
    {method: 'enumerate'},
    model
  ),
  {numBins: 9}
)
```

You now have a model of how subjects perform on your task. You could have a structured, probabilistic model here. For simplicity, let's assume you have the simplest model of task performance. It is a direct function of task-difficulty: subjects perform well if the task isn't too difficult.

```
var subjectPerformWell = !flip(taskDifficulty);
```

There's a lot of training involved in your task and it's very time consuming for you to collect data.

You run one subject through your training regime and have them do the task. The subject performs well! Soon after, your adviser drops by and wants you to make a decision to collect more data or tweak your experimental paradigm. You thought beforehand that your task was too difficult.

Since you wrote down your prior beliefs, we can examine how much the data update those beliefs about the `taskDifficulty` parameter. How does your degree of belief in task difficult change as a result of your one pilot subject performing well?

```
// Prior on task difficulty is uniform
// on [0, ..., 0.9], with a spike on 0.9
var sampleTaskDifficulty = function() {
  return flip()
    ? .9
    : randomInteger(10) / 10
}

// Compute posterior after seeing one subject perform well on the task
var taskDifficultyPosterior = Infer(
  {method: 'enumerate'},
  function() {
    var taskDifficulty = sampleTaskDifficulty()

    // Subject will perform well if the task is
    // not too difficult
    var subjectPerformsWell = !flip(taskDifficulty)

    // Observe that they perform well
    // (i.e. this value is true)
    condition(subjectPerformsWell)
    return taskDifficulty
  }
)

// Most likely task-difficulty is still .9
print('')
print("MAP: " + taskDifficultyPosterior.MAP().val)

// But a lot of probability mass is on lower values
viz.hist(taskDifficultyPosterior, {numBins: 9})

// Indeed, the expected subject ability is around .4
print("Expectation: " + expectation(taskDifficultyPosterior))
```

Exercise 7.2.1

Would you proceed with more data collection or would you change your experimental paradigm? In other words, do you still think your task is too hard?

Exercise 7.2.2

In Exercise 7.2.1, you probably used either one value of the task-difficulty or the full distribution of values to decide about whether to continue data collection or tweak the paradigm. We find ourselves in a similar situation when we have models of psychological phenomena and want to decide whether the model fits the data (or, equivalently, whether our psychological theory is capturing the phenomenon).

The traditional approach is the value (or "point-wise estimate") approach: take the value that corresponds to the best fit (e.g., by using least-squares or maximum-likelihood estimation; here, you would have taken the Maximum A Posteriori (or, MAP) estimate, which would be 0.9).

Why might this not be a good idea? Comment on the reliability of the MAP estimate and how MAP estimate compares to other values of the posterior distribution.

Exercise 7.3

Let's continue to explore the inferences you (as a scientist) can draw from the posterior over parameter values. This posterior can give you an idea of whether your model is well-behaved. In other words, do the predictions of your model depend heavily on the exact parameter value?

To help us understand how to examine posteriors over parameter settings, we're going to revisit the example of the blicket detector from the chapter on Conditional Dependence.

Here is the model, with slightly different names than the original example, and written in a parameter-friendly way. It is set up to display the "backwards blocking" phenomenon:

```
var blicketBaseRate = 0.4
var blicketPower = 0.9
var nonBlicketPower = 0.05
var machineSpontaneouslyGoesOff = 0.05

var blicketPosterior = function(evidence) {
  return Infer(
    {method: 'enumerate'},
    function() {
      var blicket = mem(
        function(block) {
```

```
          flip(blicketBaseRate)
        }
      )
      var power = function(block) {
        blicket(block)
        ? blicketPower
        : nonBlicketPower
      }
      var machine = function(blocks) {
        return blocks.length == 0
          ? flip(machineSpontaneouslyGoesOff)
          : (
            flip(power(first(blocks)))
            || machine(rest(blocks))
          )
      }
      // Condition on each of the pieces of evidence
      // making the machine go off
      map(
        function(blocks) {
          condition(machine(blocks))
        },
        evidence
      )
      return blicket('A')
    }
  )
}

// A & B make the blicket-detector go off
viz(blicketPosterior([['A', 'B']]))

// A & B make the blicket-detector go off,
// and then B makes the blicket detector go off
viz(blicketPosterior([['A', 'B'], ['B']]))
```

Exercise 7.3.1

What are the parameters of the above model? Explain what they represent in plain English.

Exercise 7.3.2

Let's analyze this model with respect to some data.

First, we'll put priors on these parameters, and then we'll do inference, conditioning on some data we might have collected in an experiment on 4 year olds, a la Sobel, Tenenbaum, and Gopnik (2004). [The data used in this exercise is schematic data].

Before running the program below, answer the following questions:

1. What does the `Infer` statement in `dataAnalysis` return?
2. What does the `Infer` statement in `detectingBlickets` return?

```
///fold:

// alternative proposal distribution for metropolis-hastings algorithm
var uniformKernel = function(prevVal) {
  return Uniform({
    a: prevVal - 0.2,
    b: prevVal + 0.2
  })
}

var toProbs = function(predictions) {
  var labels = map(
    function(i) {
      return "predictive: cond" + i + " P(true)"
    },
    _.range(1, predictions.length + 1)
  )
  var probs = map(
    function(model) {
      return Math.exp(model.score(true))
    },
    predictions
  )
  return _.zipObject(labels, probs)
}

var dataSummary = function(data) {
  _.map(data, _.mean)
}

var predictiveSummary = function(model) {
```

```
  var labels = map(
    function(i) {
      return "predictive: cond" + i + " P(true)"
    },
    _.range(1, 6)
  )
  return map(
    function(label) {
      return expectation(
        model,
        function(s) {
          return s[label]
        }
      )
    },
    labels
  )
}
///

// 5 experiment conditions / stimuli
var possibleEvidenceStream = [
  [['A']],
  [['A', 'B']],
  [['A', 'B'], ['B']],
  [['A', 'B'], ['A', 'B']],
  [[]]
];

// for each condition.
// note: the question is always "is A a blicket?"
var data = [
  repeat(
    10, function() { true }
  ).concat(false),
  repeat(
    6 , function() { true }
  ).concat(repeat(5, function() { false })),
  repeat(
    4, function() { true }
  ).concat(repeat(7, function() { false })),
  repeat(
```

```
    8, function() { true }
  ).concat(repeat(3, function() { false })),
  repeat(
    2, function() { true }
  ).concat(repeat(9, function() { false }))
]

// Same model as above, but parameterized
var detectingBlickets = mem(
  function(evidence, params) {
    return Infer(
      {method: 'enumerate'},
      function() {
        var blicket = mem(
          function(block) {
            flip(params.blicketBaseRate)
          }
        )
        var power = function(block) {
          blicket(block) ? params.blicketPower : params.nonBlicketPower
        }
        var machine = function(blocks) {
          return blocks.length == 0
            ? flip(params.machineSpontaneouslyGoesOff)
            : (
              flip(power(first(blocks)))
              || machine(rest(blocks))
            )
        }
        map(
          function(blocks){
            condition(machine(blocks))
          },
          evidence
        )
        return blicket('A')
      }
    )
  }
)

var dataAnalysis = Infer(
```

```
  {
    method: 'MCMC',
    samples: 5000,
    callbacks: [editor.MCMCProgress()]
  },
  function() {
    var params = {
      blicketBaseRate: sample(
        Uniform({a: 0, b: 1}), {driftKernel: uniformKernel}
      ),
      blicketPower: sample(
        Uniform({a: 0, b: 1}), {driftKernel: uniformKernel}
      ),
      nonBlicketPower: sample(
        Uniform({a: 0, b: 1}), {driftKernel: uniformKernel}
      ),
      machineSpontaneouslyGoesOff: sample(
        Uniform({a: 0, b: 1}), {driftKernel: uniformKernel}
      )
    }

    var cognitiveModelPredictions = map(
      function(evidence) {
        return detectingBlickets(evidence, params)
      },
      possibleEvidenceStream
    )

    // Observe each data point under the model's predictions
    map2(function(dataForStim, modelPosterior) {
      map(function(dataPoint) {
        observe(modelPosterior, dataPoint)
      }, dataForStim)
    }, data, cognitiveModelPredictions)

    var predictives = toProbs(cognitiveModelPredictions)
    return _.extend(params, predictives)
  }
)

viz.marginals(dataAnalysis)
viz.scatter(
```

```
  predictiveSummary(dataAnalysis),
  dataSummary(data),
  {xLabel: 'model', yLabel: 'data'}
)
```

Exercise 7.3.3

Now, run the program. [Note: This will take between 15-30 seconds to run.] Interpret each of the resulting plots.

Exercise 7.3.4

How do the posterior parameter values relate to the parameter values that were set in the original program?

Exercise 7.3.5

Look carefully at the priors (in the code) and the posteriors (in the plots) over `blicketPower` and `nonBlicketPower`.

Were there any a priori assumptions about the relationship between these parameters in the experimental setup? Do you think we would be justified in imposing any assumptions to the model?

Consider the posterior distributions. How was the data analysis model able to find the relationship between these parameters?

Exercise 7.3.6

Do you notice anything about the scatter plot? How would you interpret this?

Is there something we could add to the data analysis model to account for this?

Exercise 7.3.7

Now, we're going to examine the predictions of the model if we had done a more traditional analysis of point-estimates of parameters (i.e. fitting parameters).

Examine your histograms and determine the "maximum a posteriori" (MAP) value for each parameter. Plug those into the code below and run it:

```
///fold:

var toProbs = function(predictions) {
  var labels = map(
    function(i) {
      return "predictive: cond" + i + " P(true)"
    },
    _.range(1, predictions.length + 1)
  )
  var probs = map(
    function(model) {
      return Math.exp(model.score(true))
    },
    predictions
  )
  return _.zipObject(labels, probs)
}
var dataSummary = function(data) {
  _.map(data, _.mean)
}

// 5 experiment conditions / stimuli
var possibleEvidenceStream = [
  [['A']],
  [['A', 'B']],
  [['A', 'B'], ['B']],
  [['A', 'B'], ['A', 'B']],
  [[]]
];

// for each condition.
// note: the question is always "is A a blicket?"
var data = [
  repeat(
    10, function() { true }
```

```
  ).concat(false),
  repeat(
    6 , function() { true }
  ).concat(repeat(5, function() { false })),
  repeat(
    4, function() { true }
  ).concat(repeat(7, function() { false })),
  repeat(
    8, function() { true }
  ).concat(repeat(3, function() { false })),
  repeat(
    2, function() { true }
  ).concat(repeat(9, function() { false }))
];

var detectingBlickets = mem(
  function(evidence, params) {
    return Infer(
      {method: 'enumerate'},
      function() {
        var blicket = mem(
          function(block) {
            return flip(params.blicketBaseRate)
          }
        )
        var power = function(block) {
          return blicket(block)
            ? params.blicketPower
            : params.nonBlicketPower
        }
        var machine = function(blocks) {
          return blocks.length == 0
            ? flip(params.machineSpontaneouslyGoesOff)
            : (
              flip(power(first(blocks)))
              || machine(rest(blocks))
            )
        }
        map(
          function(blocks){
            return condition(machine(blocks))
          },
```

```
          evidence
        )
        return blicket('A')
      }
    )
  }
)
///

var params = {
  blicketBaseRate : ...,
  blicketPower: ...,
  nonBlicketPower: ...,
  machineSpontaneouslyGoesOff: ...
}

var bestFitModelPredictions = map(
  function(evidence) {
    return Math.exp(
      detectingBlickets(evidence, params).score(true)
    )
  },
  possibleEvidenceStream
)
viz.scatter(
  bestFitModelPredictions,
  dataSummary(data)
)
```

Exercise 7.3.8

What can you conclude about the two ways of looking at parameters in this model's case?

Sobel, David M., Joshua B. Tenenbaum, and Alison Gopnik. 2004. "Children's Causal Inferences from Indirect Evidence: Backwards Blocking and Bayesian Reasoning in Preschoolers." *Cognitive Science* 28 (3): 303–33.