Forssen_lecture1

August 31, 2018

1 Nuclear theory and predictive power

1.1 Christian Forssén

Department of Phyics, Chalmers University of Technology, Sweden

2018-08-31, Euroschool on Exotic Beams

- This presentation is based on an ipython notebook. The presentation itself is a Reveal.js HTML slideshow created with nbconvert.
- All the material and accompanying source code is safely stored in a public git repository at github. Please feel free to download and try the examples yourself.

```
[~]$ git clone https://github.com/cforssen/Euroschool2018_Forssen.git
[~]$ cd Euroschool2018_Forssen
```

1.2 Preliminaries: Python installation

The installation of Python, together with the modules that allow scientific computations, is not very difficult.

I recommend *Anaconda*, with the package manager *conda*, the works on Linux, Mac OS X, and even Windows.

- Anaconda includes both Python and conda, plus a large number of preinstalled packages. However, this distribution requires quite some disk space. Miniconda is a good light-weight option . Read also the conda online documentation

Choose a Python-3 version and install the modules that are needed for these lectures:

```
[~]$ conda install numpy scipy pandas matplotlib seaborn jupyter
```

Even better, create a virtual environment (the modules are listed in the file 'environment.yml'):

```
[~]$ conda env create
[~]$ source activate euroschool-env
```

Let us start a jupyter (ipython) notebook:

```
[~]$ jupyter notebook Forssen_lecture1.ipynb
```

and import some important modules

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

        %matplotlib inline

In [2]: # Some care is needed when removing warnings. But for the final version of this notebo
        import warnings
        warnings.simplefilter("ignore", UserWarning)
        warnings.simplefilter("ignore", FutureWarning)

In [3]: # Not really needed, but nicer plots
        import seaborn as sns
        sns.set()
        sns.set_context("talk")
```

2 Learning from data

2.1 Inference

> the act of passing from one proposition, statement or judgment considered as true to another whose truth is believed to follow from that of the former

(Webster)
Do premises $A, B, \ldots \rightarrow$ hypothesis, H?

2.1.1 Deductive inference:

> Premises allow definite determination of truth/falsity of H (syllogisms, symbolic logic, Boolean algebra)

$B(H|A, B, ...) = 0$ or 1

2.1.2 Inductive inference

> Premises bear on truth/falsity of H, but don't allow its definite determination (weak syllogisms, analogies)

- A, B, C, D share properties x, y, z;
- E has properties x, y
- $\rightarrow E$ probably has property z.

2.2 Statistical Inference

- Quantify the strength of inductive inferences from facts, in the form of data (D), and other premises, e.g. models, to hypotheses about the phenomena producing the data.
- Quantify via probabilities, or averages calculated using probabilities. Frequentists ($\mathcal{F}$) and Bayesians ($\mathcal{B}$) use probabilities very differently for this.
- To the pioneers such as Bernoulli, Bayes and Laplace, a probability represented a *degree-of-belief* or plausability: how much they thought that something as true based on the evidence at hand. This is the Bayesian approach.
- To the 19th century scholars, this seemed too vague and subjective. They redefined probability as the *long run relative frequency* with which an event occurred, given (infinitely) many repeated (experimental) trials.

2.3 The Bayesian recipe

Assess hypotheses by calculating their probabilities $p(H_i|\ldots)$ conditional on known and/or presumed information using the rules of probability theory.

2.3.1 Probability Theory Axioms:

Product (AND) rule

$$p(A, B|I) = p(A|I)p(B|A, I) = p(B|I)p(A|B, I)$$

Should read $p(A, B|I)$ as the probability for propositions A AND B being true given that I is true.

Sum (OR) rule

$$p(A + B|I) = p(A|I) + p(B|I) - p(A, B|I)$$

$p(A + B|I)$ is the probability that proposition A OR B is true given that I is true.

Normalization

$$p(A|I) + p(\bar{A}|I) = 1$$

$\bar{A}$ denotes the proposition that A is false.

2.4 Bayes' theorem

Bayes' theorem follows directly from the product rule

$$p(A|B, I) = \frac{p(B|A, I)p(A|I)}{p(B|I)}.$$

The importance of this property to data analysis becomes apparent if we replace A and B by hypothesis(H) and data(D):

$$p(H|D, I) = \frac{p(D|H, I)p(H|I)}{p(D|I)}.$$

The power of Bayes' theorem lies in the fact that it relates the quantity of interest, the probability that the hypothesis is true given the data, to the term we have a better chance of being able to assign, the probability that we would have observed the measured data if the hypothesis was true.

The various terms in Bayes' theorem have formal names. * The quantity on the far right, $p(H|I)$, is called the **prior** probability; it represents our state of knowledge (or ignorance) about the truth of the hypothesis before we have analysed the current data. * This is modified by the experimental measurements through $p(D|H,I)$, the **likelihood** function, * The denominator $p(D|I)$ is called the **evidence**. It does not depend on the hypothesis and can be regarded as a normalization constant. * Together, these yield the **posterior** probability, $p(H|D,I)$, representing our state of knowledge about the truth of the hypothesis in the light of the data.

In a sense, Bayes' theorem encapsulates the process of learning.

2.5 The friends of Bayes' theorem

Normalization

$$\sum_i p(H_i|\ldots) = 1$$

In the above, H_i is an exclusive and exhaustive list of hypotheses. For example,let's imagine that there are five candidates in a presidential election; then H_1 could be the proposition that the first candidate will win, and so on.

Marginalization

$$\sum_i p(A, H_i|I) = \sum_i p(H_i|A,I)p(A|I) = p(A|I)$$

The probability that A is true, for example that unemployment will be lower in a year's time (given all relevant information I, but irrespective of whoever becomes president) is then given by $\sum_i p(A, H_i|I)$.

Marginalization (continuum limit)

$$\int dx p(A, H(x)|I) = p(A|I)$$

In the continuum limit of propositions we must understand $p(\ldots)$ as a pdf (probability density function).

Marginalization is a very powerful device in data analysis because it enables us to deal with nuisance parameters; that is, quantities which necessarily enter the analysis but are of no intrinsic interest. The unwanted background signal present in many experimental measurements are examples of nuisance parameters.

2.6 Inference With Parametric Models

Inductive inference with parametric models is a very important tool in the natural sciences.

- Consider N different models M_i ($i = 1,\ldots,N$), each with parameters $\boldsymbol{\alpha}_i$. Each of them implies a sampling distribution (conditional predictive distribution for possible data)

$$p(D|\boldsymbol{\alpha}_i, M_i)$$

- The $\boldsymbol{\alpha}_i$ dependence when we fix attention on the actual, observed data (D_{obs}) is the likelihood function
$$\mathcal{L}_i(\boldsymbol{\alpha}_i) \equiv p(D_{\text{obs}}|\boldsymbol{\alpha}_i, M_i)$$
- We may be uncertain about i (**model uncertainty**),
- or uncertain about $\boldsymbol{\alpha}_i$ (**parameter uncertainty**).

Parameter Estimation Premise = choice of model (pick specific i)
$\Rightarrow$ What can we say about $\boldsymbol{\alpha}_i$?

Model comparison: Premise = $\{M_i\}$
$\Rightarrow$ What can we say about model i compared to model j?

Model adequacy: Premise = M_1
$\Rightarrow$ Is M_1 adequate?

Hybrid Uncertainty: Models share some common params: $\boldsymbol{\alpha}_i = \{\boldsymbol{\varphi}, \boldsymbol{\eta}_i\}$
$\Rightarrow$ What can we say about $\boldsymbol{\varphi}$? (Systematic error is an example)

2.7 Illustrative example #1: A Bayesian billiard game

Adapted from the blog post Frequentism and Bayesianism II: When Results Differ

This example of nuisance parameters dates all the way back to the posthumous 1763 paper written by Thomas Bayes himself. The particular version of this problem used here is borrowed from Eddy 2004.

The setting is a rather contrived game in which Alice and Bob bet on the outcome of a process they can't directly observe:

Alice and Bob enter a room. Behind a curtain there is a billiard table, which they cannot see, but their friend Carol can. Carol rolls a ball down the table, and marks where it lands. Once this mark is in place, Carol begins rolling new balls down the table. If the ball lands to the left of the mark, Alice gets a point; if it lands to the right of the mark, Bob gets a point. We can assume for the sake of example that Carol's rolls are unbiased: that is, the balls have an equal chance of ending up anywhere on the table. The first person to reach **six points** wins the game.

Here the location of the mark (determined by the first roll) can be considered a nuisance parameter: it is unknown, and not of immediate interest, but it clearly must be accounted for when predicting the outcome of subsequent rolls. If the first roll settles far to the right, then subsequent rolls will favor Alice. If it settles far to the left, Bob will be favored instead.

Given this setup, here is the question we ask of ourselves:

> In a particular game, after eight rolls, Alice has five points and Bob has three points. What is the probability that Bob will go on to win the game?

Intuitively, you probably realize that because Alice received five of the eight points, the marker placement likely favors her. And given this, it's more likely that the next roll will go her way as well. And she has three opportunities to get a favorable roll before Bob can win; she seems to have clinched it. But, **quantitatively**, what is the probability that Bob will squeak-out a win?

2.7.1 A Naive Frequentist Approach

Someone following a classical frequentist approach might reason as follows:

To determine the result, we need an intermediate estimate of where the marker sits. We'll quantify this marker placement as a probability p that any given roll lands in Alice's favor. Because five balls out of eight fell on Alice's side of the marker, we can quickly show that the maximum likelihood estimate of p is given by:

$$\hat{p} = 5/8$$

(This result follows in a straightforward manner from the binomial likelihood). Assuming this maximum likelihood probability, we can compute the probability that Bob will win, which is given by:

$$P(B) = (1 - \hat{p})^3$$

That is, he needs to win three rolls in a row. Thus, we find that the following estimate of the probability:

```
In [4]: p_hat = 5. / 8.
        freq_prob = (1 - p_hat) ** 3
        print("Naive Frequentist Probability of Bob Winning: %.2f" %freq_prob)
```

```
Naive Frequentist Probability of Bob Winning: 0.05
```

In other words, we'd give Bob the following odds of winning:

```
In [5]: print("Odds against Bob winning: %i to 1" %((1. - freq_prob) / freq_prob))
```

```
Odds against Bob winning: 17 to 1
```

So we've estimated using frequentist ideas that Alice will win about 17 times for each time Bob wins. Let's try a Bayesian approach next.

2.7.2 Bayesian approach

We can also approach this problem from a Bayesian standpoint. This is slightly more involved, and requires us to first define some notation.

We'll consider the following random variables:

- B = Bob Wins
- D = observed data, i.e. $D = (n_A, n_B) = (5, 3)$
- p = unknown probability that a ball lands on Alice's side during the current game

We want to compute $P(B \mid D)$; that is, the probability that Bob wins given our observation that Alice currently has five points to Bob's three.

The general Bayesian method of treating nuisance parameters is *marginalization*, or integrating the joint probability over the entire range of the nuisance parameter. In this case, that means that we will first calculate the joint distribution

$$P(B, p \mid D)$$

and then marginalize over p using the following identity:

$$P(B \mid D) \equiv \int_{-\infty}^{\infty} P(B, p \mid D) \mathrm{d}p$$

This identity follows from the definition of conditional probability, and the law of total probability: that is, it is a fundamental consequence of probability axioms and will always be true. Even a frequentist would recognize this; they would simply disagree with our interpretation of $P(p)$ as being a measure of uncertainty of our own knowledge.

Building our Bayesian Expression To compute this result, we will manipulate the above expression for $P(B \mid D)$ until we can express it in terms of other quantities that we can compute.

We'll start by applying the following definition of conditional probability to expand the term $P(B, p \mid D)$:

$$P(B \mid D) = \int P(B \mid p, D) P(p \mid D) dp$$

Next we use Bayes' rule to rewrite $P(p \mid D)$:

$$P(B \mid D) = \int P(B \mid p, D) \frac{P(D \mid p) P(p)}{P(D)} dp$$

Finally, using the same probability identity we started with, we can expand $P(D)$ in the denominator to find:

$$P(B \mid D) = \frac{\int P(B \mid p, D) P(D \mid p) P(p) dp}{\int P(D \mid p) P(p) dp}$$

Now the desired probability is expressed in terms of three quantities that we can compute. Let's look at each of these in turn:

- $P(B \mid p, D)$: This term is exactly the frequentist likelihood we used above. In words: given a marker placement p and the fact that Alice has won 5 times and Bob 3 times, what is the probability that Bob will go on to six wins? Bob needs three wins in a row, i.e. $P(B \mid p, D) = (1-p)^3$.
- $P(D \mid p)$: this is another easy-to-compute term. In words: given a probability p, what is the likelihood of exactly 5 positive outcomes out of eight trials? The answer comes from the well-known Binomial distribution: in this case $P(D \mid p) \propto p^5(1-p)^3$
- $P(p)$: this is our prior on the probability p. By the problem definition, we can assume that p is evenly drawn between 0 and 1. That is, $P(p)$ is a uniform probability distribution in the range from 0 to 1.

Putting this all together, canceling some terms, and simplifying a bit, we find

$$P(B \mid D) = \frac{\int_0^1 (1-p)^6 p^5 dp}{\int_0^1 (1-p)^3 p^5 dp}$$

where both integrals are evaluated from 0 to 1.

These integrals might look a bit difficult, until we notice that they are special cases of the Beta Function:

$$\beta(n, m) = \int_0^1 (1-p)^{n-1} p^{m-1}$$

The Beta function can be further expressed in terms of gamma functions (i.e. factorials), but for simplicity we'll compute them directly using Scipy's beta function implementation:

```
In [6]: from scipy.special import beta
        bayes_prob = beta(6 + 1, 5 + 1) / beta(3 + 1, 5 + 1)

        print("P(B|D) = %.2f" %bayes_prob)
```

```
P(B|D) = 0.09
```

The associated odds are the following:

```
In [7]: print("Bayesian odds against Bob winning: %i to 1" %((1. - bayes_prob) / bayes_prob))
```

```
Bayesian odds against Bob winning: 10 to 1
```

So we see that the Bayesian result gives us 10 to 1 odds, which is quite different than the 17 to 1 odds found using the frequentist approach. So which one is correct?

2.7.3 Brute-force (Monte Carlo) approach

For this type of well-defined and simple setup, it is actually relatively easy to use a Monte Carlo simulation to determine the correct answer. This is essentially a brute-force tabulation of possible outcomes: we generate a large number of random games, and simply count the fraction of relevant games that Bob goes on to win. The current problem is especially simple because so many of the random variables involved are uniformly distributed. We can use the `numpy` package to do this as follows:

```
In [8]: np.random.seed(0)

        # play 100000 games with randomly-drawn p, between 0 and 1
        p = np.random.random(100000)

        # each game needs at most 11 rolls for one player to reach 6 wins
        rolls = np.random.random((11, len(p)))

        # count the cumulative wins for Alice and Bob at each roll
        Alice_count = np.cumsum(rolls < p, 0)
        Bob_count = np.cumsum(rolls >= p, 0)

        # sanity check: total number of wins should equal number of rolls
        total_wins = Alice_count + Bob_count
        assert np.all(total_wins.T == np.arange(1, 12))
        print("(Sanity check passed)")
```

```
(Sanity check passed)
```

```
In [9]: # determine number of games which meet our criterion of (A wins, B wins)=(5, 3)
        # this means Bob's win count at eight rolls must equal 3
        good_games = Bob_count[7] == 3
        print("Number of suitable games: {0}".format(good_games.sum()))

        # truncate our results to consider only these games
        Alice_count = Alice_count[:, good_games]
        Bob_count = Bob_count[:, good_games]

        # determine which of these games Bob won.
        # to win, he must reach six wins after 11 rolls.
        bob_won = np.sum(Bob_count[10] == 6)
        print("Number of these games Bob won: {0}".format(bob_won.sum()))

        # compute the probability
        mc_prob = bob_won.sum() * 1. / good_games.sum()
        print("Monte Carlo Probability of Bob winning: {0:.2f}".format(mc_prob))
        print("MC Odds against Bob winning: {0:.0f} to 1".format((1. - mc_prob) / mc_prob))
```

```
Number of suitable games: 11068
Number of these games Bob won: 979
Monte Carlo Probability of Bob winning: 0.09
MC Odds against Bob winning: 10 to 1
```

The Monte Carlo approach gives 10-to-1 odds on Bob, which agrees with the Bayesian approach. Apparently, our naive frequentist approach above was flawed.

2.7.4 Discussion

This example shows several different approaches to dealing with the presence of a nuisance parameter p. The Monte Carlo simulation gives us a close brute-force estimate of the true probability (assuming the validity of our assumptions), which the Bayesian approach matches. The naïve frequentist approach, by utilizing a single maximum likelihood estimate of the nuisance parameter p, arrives at the wrong result.

We should emphasize that **this does not imply frequentism itself is incorrect**. The incorrect result above is more a matter of the approach being "naive" than it being "frequentist". There certainly exist frequentist methods for handling this sort of nuisance parameter – for example, it is theoretically possible to apply a transformation and conditioning of the data to isolate the dependence on p – but it's hard to find any approach to this particular problem that does not somehow take advantage of Bayesian-like marginalization over p.

Another potential point of contention is that the question itself is posed in a way that is perhaps unfair to the classical, frequentist approach. A frequentist might instead hope to give the answer in terms of null tests or confidence intervals: that is, they might devise a procedure to construct limits which would provably bound the correct answer in $100 \times (1 - p)$ percent of similar trials, for some value of p – say, 0.05 (note this is a different p than the p we've been talking about

above). This might be classically accurate, but it doesn't quite answer the question at hand. I'll leave discussion of the meaning of such confidence intervals for my follow-up post on the subject.

There is one clear common point of these two potential frequentist responses: both require some degree of effort and/or special expertise; perhaps a suitable frequentist approach would be immediately obvious to someone with a PhD in statistics, but is most definitely *not* obvious to a statistical lay-person simply trying to answer the question at hand. In this sense, I think Bayesianism provides a better approach for this sort of problem: by simple algebraic manipulation of a few well-known axioms of probability within a Bayesian framework, we can straightforwardly arrive at the correct answer without need for other special expertise.

2.8 Example #2: Linear regression with data outliers

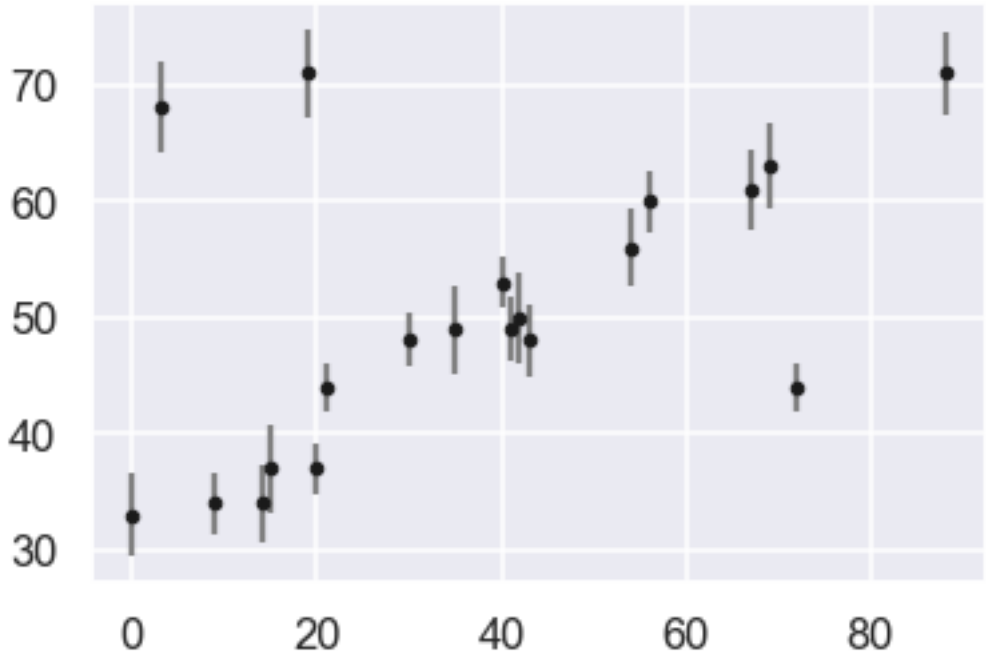
Adapted from the blog post Frequentism and Bayesianism II: When Results Differ

One situation where the concept of nuisance parameters can be helpful is accounting for outliers in data. Consider the following dataset, relating the observed variables x and y, and the error of y stored in e.

```
In [10]: x = np.array([ 0,  3,  9, 14, 15, 19, 20, 21, 30, 35,
                        40, 41, 42, 43, 54, 56, 67, 69, 72, 88])
         y = np.array([33, 68, 34, 34, 37, 71, 37, 44, 48, 49,
                        53, 49, 50, 48, 56, 60, 61, 63, 44, 71])
         e = np.array([ 3.6, 3.9, 2.6, 3.4, 3.8, 3.8, 2.2, 2.1, 2.3, 3.8,
                        2.2, 2.8, 3.9, 3.1, 3.4, 2.6, 3.4, 3.7, 2.0, 3.5])
```

We'll visualize this data below:

```
In [11]: plt.errorbar(x, y, e, fmt='.k', ecolor='gray');
```



Our task is to find a line of best-fit to the data. It's clear upon visual inspection that there are some outliers among these points, but let's start with a simple non-robust maximum likelihood approach.

Like we saw in the previous post, the following simple maximum likelihood result can be considered to be either frequentist or Bayesian (with uniform priors): in this sort of simple problem, the approaches are essentially equivalent.

We'll propose a simple linear model, which has a slope and an intercept encoded in a parameter vector θ. The model is defined as follows:

$$\hat{y}(x \mid \theta) = \theta_0 + \theta_1 x$$

Given this model, we can compute a Gaussian likelihood for each point:

$$p(x_i, y_i, e_i \mid \theta) \propto \exp\left[-\frac{1}{2e_i^2}\left(y_i - \hat{y}(x_i \mid \theta)\right)^2\right]$$

The total likelihood is the product of all the individual likelihoods. Computing this and taking the log, we have:

$$\log \mathcal{L}(D \mid \theta) = \text{const} - \sum_i \frac{1}{2e_i^2}\left(y_i - \hat{y}(x_i \mid \theta)\right)^2$$

This should all look pretty familiar if you read through the previous post. This final expression is the log-likelihood of the data given the model, which can be maximized to find the θ corresponding to the maximum-likelihood model. Equivalently, we can minimize the summation term, which is known as the *loss*:

$$\text{loss} = \sum_i \frac{1}{2e_i^2}\left(y_i - \hat{y}(x_i \mid \theta)\right)^2$$

This loss expression is known as a *squared loss*; here we've simply shown that the squared loss can be derived from the Gaussian log likelihood.

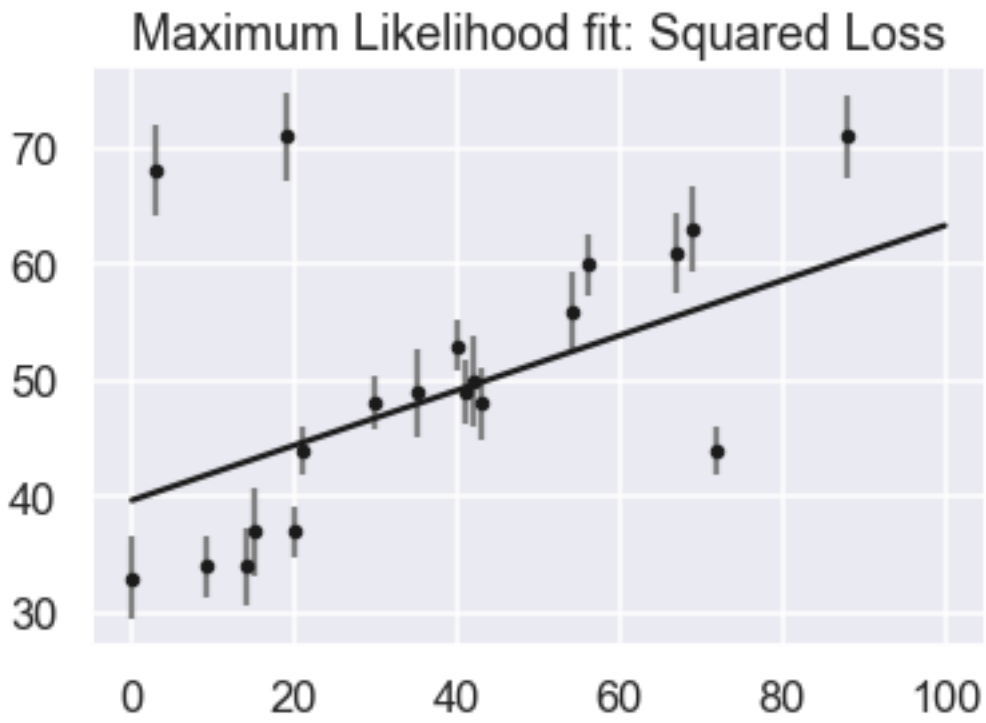
2.8.1 Standard Likelihood Approach

Following the logic of the previous post, we can maximize the likelihood (or, equivalently, minimize the loss) to find θ within a frequentist paradigm. For a flat prior in θ, the maximum of the Bayesian posterior will yield the same result. (note that there are good arguments based on the principle of maximum entropy that a flat prior is not the best choice here; we'll ignore that detail for now, as it's a very small effect for this problem).

For simplicity, we'll use scipy's `optimize` package to minimize the loss (in the case of squared loss, this computation can be done more efficiently using matrix methods, but we'll use numerical minimization for simplicity here)

```
In [12]: from scipy import optimize

         def squared_loss(theta, x=x, y=y, e=e):
             dy = y - theta[0] - theta[1] * x
             return np.sum(0.5 * (dy / e) ** 2)
```

```
theta1 = optimize.fmin(squared_loss, [0, 0], disp=False)

xfit = np.linspace(0, 100)
plt.errorbar(x, y, e, fmt='.k', ecolor='gray')
plt.plot(xfit, theta1[0] + theta1[1] * xfit, '-k')
plt.title('Maximum Likelihood fit: Squared Loss');
```



It's clear on examination that the outliers are exerting a disproportionate influence on the fit. This is due to the nature of the squared loss function. If you have a single outlier that is, say 10 standard deviations away from the fit, its contribution to the loss will out-weigh that of 25 points which are 2 standard deviations away!

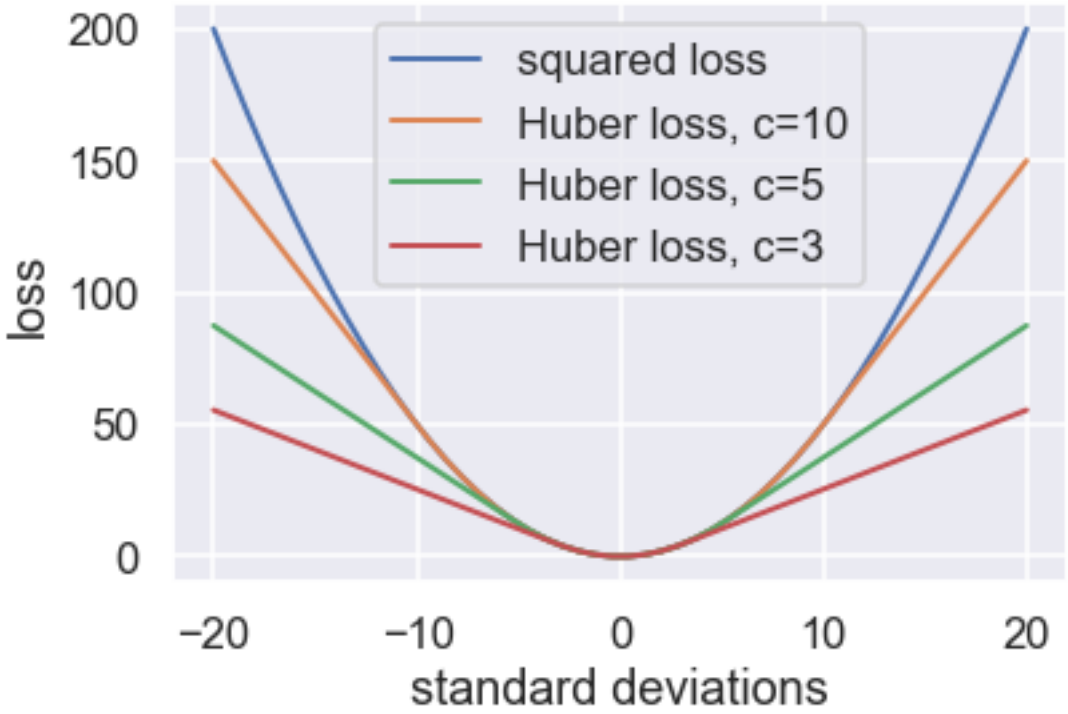
Clearly the squared loss is overly sensitive to outliers, and this is causing issues with our fit. One way to address this within the frequentist paradigm is to simply adjust the loss function to be more robust.

2.8.2 Frequentist Correction for Outliers: Huber Loss

The variety of possible loss functions is quite literally infinite, but one relatively well-motivated option is the Huber loss. The Huber loss defines a critical value at which the loss curve transitions from quadratic to linear. Let's create a plot which compares the Huber loss to the standard squared loss for several critical values c:

```
In [13]: t = np.linspace(-20, 20)

         def huber_loss(t, c=3):
```

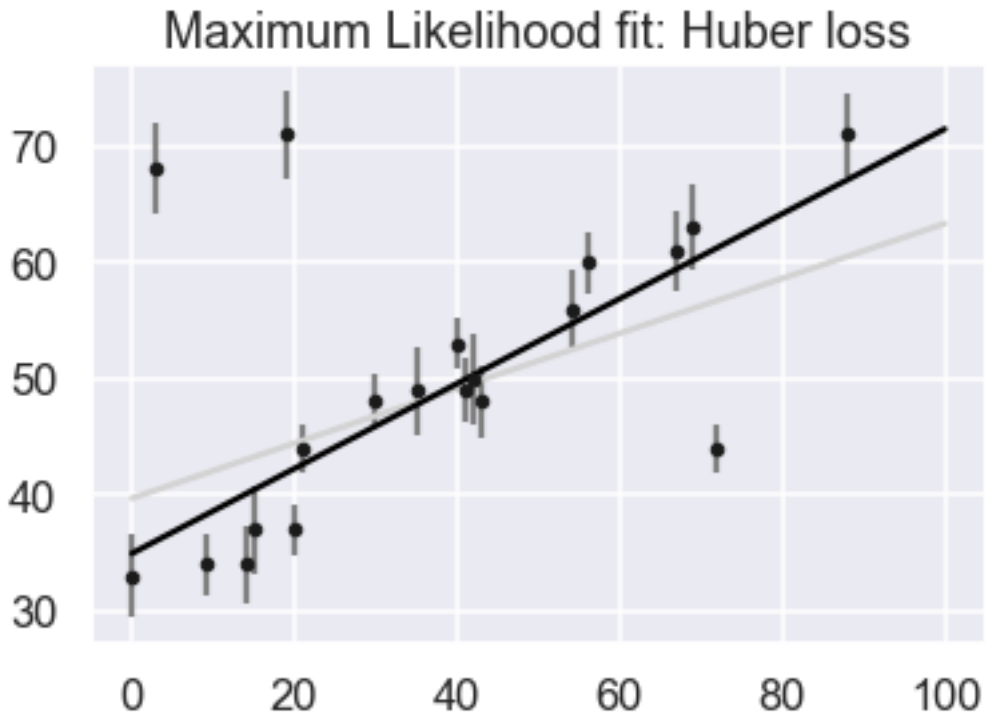
```
    return ((abs(t) < c) * 0.5 * t ** 2
            + (abs(t) >= c) * -c * (0.5 * c - abs(t)))

plt.plot(t, 0.5 * t ** 2, label="squared loss", lw=2)
for c in (10, 5, 3):
    plt.plot(t, huber_loss(t, c), label="Huber loss, c={0}".format(c), lw=2)
plt.ylabel('loss')
plt.xlabel('standard deviations')
plt.legend(loc='best');
```



The Huber loss is equivalent to the squared loss for points which are well-fit by the model, but reduces the loss contribution of outliers. For example, a point 20 standard deviations from the fit has a squared loss of 200, but a c=3 Huber loss of just over 55. Let's see the result of the best-fit line using the Huber loss rather than the squared loss. We'll plot the squared loss result in light gray for comparison:

```
In [14]: def total_huber_loss(theta, x=x, y=y, e=e, c=3):
             return huber_loss((y - theta[0] - theta[1] * x) / e, c).sum()

         theta2 = optimize.fmin(total_huber_loss, [0, 0], disp=False)

         plt.errorbar(x, y, e, fmt='.k', ecolor='gray')
         plt.plot(xfit, theta1[0] + theta1[1] * xfit, color='lightgray')
```

```
plt.plot(xfit, theta2[0] + theta2[1] * xfit, color='black')
plt.title('Maximum Likelihood fit: Huber loss');
```



By eye, this seems to have worked as desired: the fit is much closer to our intuition!

However a Bayesian might point out that the motivation for this new loss function is a bit suspect: as we showed, the squared-loss can be straightforwardly derived from a Gaussian likelihood. The Huber loss seems a bit *ad hoc*: where does it come from? How should we decide what value of c to use? Is there any good motivation for using a linear loss on outliers, or should we simply remove them instead? How might this choice affect our resulting model?

2.8.3 A Bayesian Approach to Outliers: Nuisance Parameters

The Bayesian approach to accounting for outliers generally involves *modifying the model* so that the outliers are accounted for. For this data, it is abundantly clear that a simple straight line is not a good fit to our data. So let's propose a more complicated model that has the flexibility to account for outliers. One option is to choose a mixture between a signal and a background:

$$p(\{x_i\},\{y_i\},\{e_i\} \mid \theta,\{g_i\},\sigma_b) = \frac{g_i}{\sqrt{2\pi e_i^2}}\exp\left[\frac{-(\hat{y}(x_i \mid \theta)-y_i)^2}{2e_i^2}\right] + \frac{1-g_i}{\sqrt{2\pi\sigma_B^2}}\exp\left[\frac{-(\hat{y}(x_i \mid \theta)-y_i)^2}{2\sigma_B^2}\right]$$

What we've done is expanded our model with some nuisance parameters: $\{g_i\}$ is a series of weights which range from 0 to 1 and encode for each point i the degree to which it fits the model.

$g_i = 0$ indicates an outlier, in which case a Gaussian of width σ_B is used in the computation of the likelihood. This σ_B can also be a nuisance parameter, or its value can be set at a sufficiently high number, say 50.

Our model is much more complicated now: it has 22 free parameters rather than 2, but the majority of these can be considered nuisance parameters, which can be marginalized-out in the end, just as we marginalized (integrated) over p in the Billiard example. Let's construct a function which implements this likelihood. We'll use the emcee package to explore the parameter space.

To actually compute this, we'll start by defining functions describing our prior, our likelihood function, and our posterior:

```
In [15]: # theta will be an array of length 2 + N, where N is the number of points
         # theta[0] is the intercept, theta[1] is the slope,
         # and theta[2 + i] is the weight g_i

         def log_prior(theta):
             #g_i needs to be between 0 and 1
             if (all(theta[2:] > 0) and all(theta[2:] < 1)):
                 return 0
             else:
                 return -np.inf  # recall log(0) = -inf

         def log_likelihood(theta, x, y, e, sigma_B):
             dy = y - theta[0] - theta[1] * x
             g = np.clip(theta[2:], 0, 1)  # g<0 or g>1 leads to NaNs in logarithm
             logL1 = np.log(g) - 0.5 * np.log(2 * np.pi * e ** 2) - 0.5 * (dy / e) ** 2
             logL2 = np.log(1 - g) - 0.5 * np.log(2 * np.pi * sigma_B ** 2) - 0.5 * (dy / sigma
             return np.sum(np.logaddexp(logL1, logL2))

         def log_posterior(theta, x, y, e, sigma_B):
             return log_prior(theta) + log_likelihood(theta, x, y, e, sigma_B)
```

Now we'll run the MCMC samples to explore the parameter space:

```
In [16]: # Note that this step will take a few minutes to run!

         ndim = 2 + len(x)  # number of parameters in the model
         nwalkers = 50  # number of MCMC walkers
         nburn = 10000  # "burn-in" period to let chains stabilize
         nsteps = 15000  # number of MCMC steps to take

         # set theta near the maximum likelihood, with
         np.random.seed(0)
         starting_guesses = np.zeros((nwalkers, ndim))
         starting_guesses[:, :2] = np.random.normal(theta1, 1, (nwalkers, 2))
         starting_guesses[:, 2:] = np.random.normal(0.5, 0.1, (nwalkers, ndim - 2))

         import emcee
         sampler = emcee.EnsembleSampler(nwalkers, ndim, log_posterior, args=[x, y, e, 50])
```

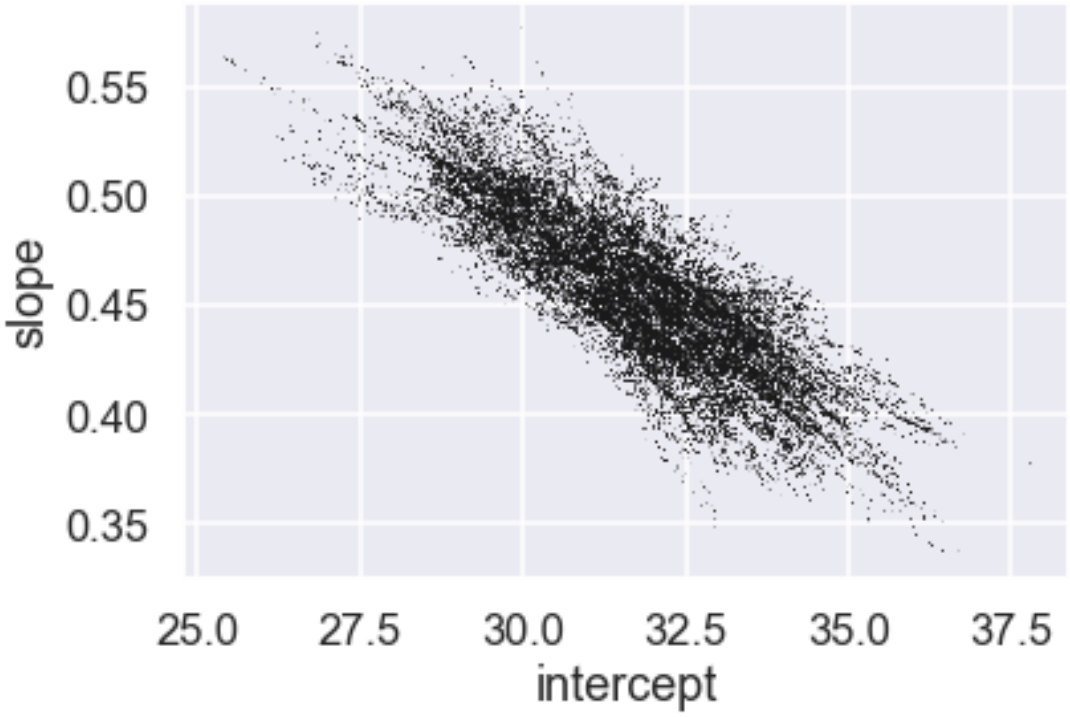
```
        sampler.run_mcmc(starting_guesses, nsteps)

        sample = sampler.chain  # shape = (nwalkers, nsteps, ndim)
        sample = sampler.chain[:, nburn:, :].reshape(-1, ndim)
```

```
/Applications/anaconda3/envs/euroschool-env/lib/python3.5/site-packages/ipykernel_launcher.py:
  from ipykernel import kernelapp as app
/Applications/anaconda3/envs/euroschool-env/lib/python3.5/site-packages/ipykernel_launcher.py:
  app.launch_new_instance()
```

Once we have these samples, we can exploit a very nice property of the Markov chains. Because their distribution models the posterior, we can integrate out (i.e. marginalize) over nuisance parameters simply by ignoring them!

We can look at the (marginalized) distribution of slopes and intercepts by examining the first two columns of the sample:

```
In [17]: plt.plot(sample[:, 0], sample[:, 1], ',k', alpha=0.1)
         plt.xlabel('intercept')
         plt.ylabel('slope');
```

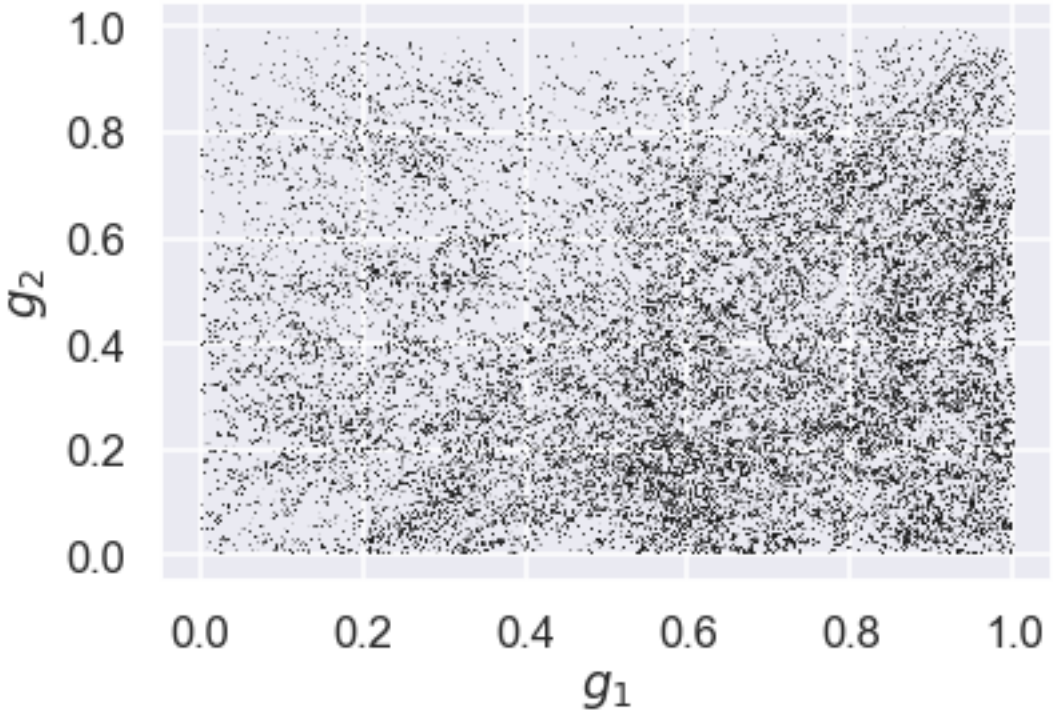


We see a distribution of points near a slope of $\sim 0.4 - 0.5$, and an intercept of $\sim 29 - 34$. We'll plot this model over the data below, but first let's see what other information we can extract from this trace.

One nice feature of analyzing MCMC samples is that the choice of nuisance parameters is completely symmetric: just as we can treat the $\{g_i\}$ as nuisance parameters, we can also treat the slope and intercept as nuisance parameters! Let's do this, and check the posterior for g_1 and g_2, the outlier flag for the first two points:

```
In [18]: plt.plot(sample[:, 2], sample[:, 3], ',k', alpha=0.1)
         plt.xlabel('$g_1$')
         plt.ylabel('$g_2$')

         print("g1 mean: {0:.2f}".format(sample[:, 2].mean()))
         print("g2 mean: {0:.2f}".format(sample[:, 3].mean()))
```

```
g1 mean: 0.61
g2 mean: 0.40
```

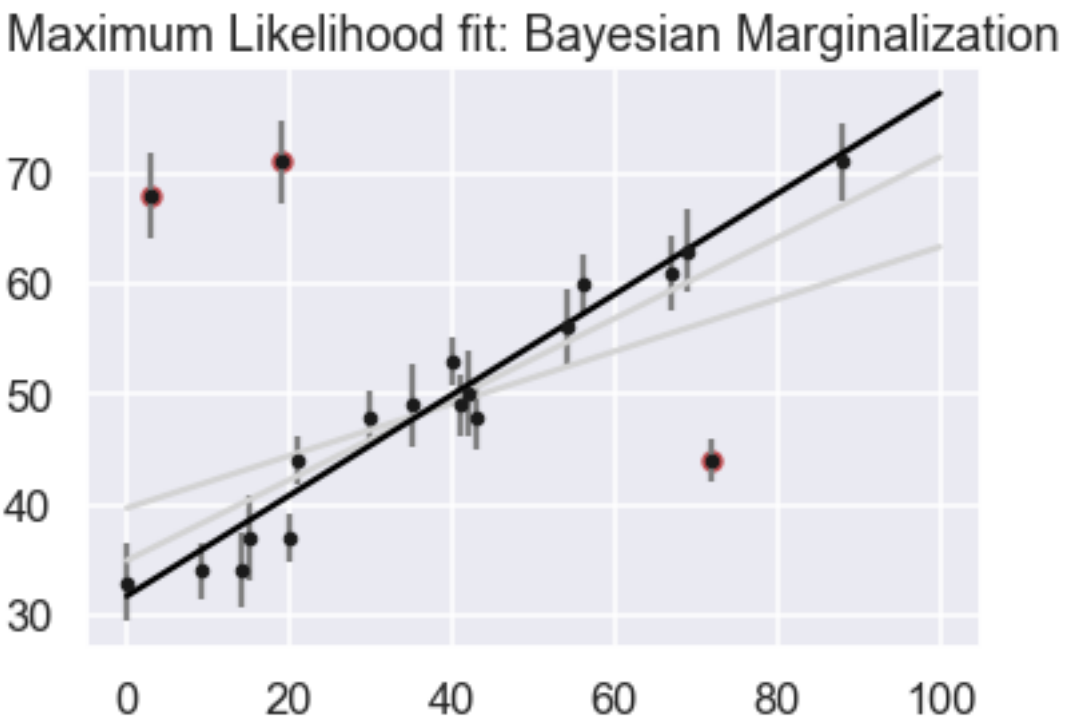


There is not an extremely strong constraint on either of these, but we do see that $(g_1, g_2) = (1, 0)$ is slightly favored: the means of g_1 and g_2 are greater than and less than 0.5, respecively. If we choose a cutoff at $g = 0.5$, our algorithm has identified g_2 as an outlier.

Let's make use of all this information, and plot the marginalized best model over the original data. As a bonus, we'll draw red circles to indicate which points the model detects as outliers:

```
In [19]: theta3 = np.mean(sample[:, :2], 0)
         g = np.mean(sample[:, 2:], 0)
         outliers = (g < 0.5)
```

```
In [20]: plt.errorbar(x, y, e, fmt='.k', ecolor='gray')
         plt.plot(xfit, theta1[0] + theta1[1] * xfit, color='lightgray')
         plt.plot(xfit, theta2[0] + theta2[1] * xfit, color='lightgray')
         plt.plot(xfit, theta3[0] + theta3[1] * xfit, color='black')
         plt.scatter(x[outliers], y[outliers],marker='o',s=40,edgecolors='r',linewidths=2,c='k
         plt.title('Maximum Likelihood fit: Bayesian Marginalization');
```



The result, shown by the dark line, matches our intuition! Furthermore, the points automatically identified as outliers are the ones we would identify by hand. For comparison, the gray lines show the two previous approaches: the simple maximum likelihood and the frequentist approach based on Huber loss.

2.8.4 Discussion

Here we've dived into linear regression in the presence of outliers. A typical Gaussian maximum likelihood approach fails to account for the outliers, but we were able to correct this in the frequentist paradigm by modifying the loss function, and in the Bayesian paradigm by adopting a mixture model with a large number of nuisance parameters.

Both approaches have their advantages and disadvantages: the frequentist approach here is relatively straightforward and computationally efficient, but is based on the use of a loss function which is not particularly well-motivated. The Bayesian approach is well-founded and produces very nice results, but requires a rather subjective specification of a prior. It is also much more intensive in both coding time and computational time.

For Bayes' billiard ball example, we showed that a naïve frequentist approach leads to the wrong answer, while a naïve Bayesian approach leads to the correct answer. This doesn't mean frequentism is wrong, but it does mean we must be very careful when applying it.

For the linear regression example, we showed one possible approach from both frequentism and Bayesianism for accounting for outliers in our data. Using a robust frequentist cost function is relatively fast and painless, but is dubiously motivated and leads to results which are difficult to interpret. Using a Bayesian mixture model takes more effort and requires more intensive computation, but leads to a very nice result in which multiple questions can be answered at once: in this case, marginalizing one way to find the best-fit model, and marginalizing another way to identify outliers in the data.