Binned Maximum Likelihood Fit

A review of the method of maximum likelihood applied to binned data

Toki Nishikawa

September 2021

School of Engineering
Supervisor: Vince Croft

Contents

1 Theory

Suppose we have N measurements $x_i, ..., x_N$ in the range $x_0 \leq x_i < x_h$. Let $f(x; \boldsymbol{\theta})$ be proportional to the *assumed* probability density function for our N measurements where $\boldsymbol{\theta}$ represents our fit parameters and $\int_{x_0}^{x_h} f(x; \boldsymbol{\theta})\, dx \equiv A(\boldsymbol{\theta})$. Let's assign our data to M bins, each with the same bin-width $h \equiv \frac{x_h - x_0}{M}$. The measured contents of each bin b $(b = 1...M)$ are described by **Equation 1**.

$$n_b = \int_{x_0+h(b-1)}^{x_0+hb} \sum_{i=1}^{N} \delta(x - x_i) dx \tag{1}$$

*Want to see an example of **Equation 1**?*

Equation 1 can be expanded to the following form.

$$n_b = \textstyle\int_{x_0+h(b-1)}^{x_0+hb} [\delta(x - x_1) + \delta(x - x_2) + \delta(x - x_3) + \delta(x - x_4)] dx$$

Suppose we have $N = 4$ measurements – let's say $\boldsymbol{x} = [x_1, x_2, x_3, x_4] = [1, 2, 3, 4]$ - and we assign our data to $M = 2$ bins, each with bin width $h \equiv \frac{x_h - x_0}{M} = \frac{4-1}{2} = \frac{3}{2}$. For the first bin $b = 1$, we get retrieve the following.

$$\begin{gathered} n_1 = \textstyle\int_{1+\frac{3}{2}(1-1)=1}^{1+\frac{3}{2}(1)=\frac{5}{2}} [\delta(x-1) + \delta(x-2) + \delta(x-3) + \delta(x-4)] dx \\ = \textstyle\int_1^{\frac{5}{2}} \delta(x-1) dx + \int_1^{\frac{5}{2}} \delta(x-2) dx + \int_1^{\frac{5}{2}} \delta(x-3) dx + \int_1^{\frac{5}{2}} \delta(x-4) dx \\ = 1 + 1 + 0 + 0 \\ = 2 \end{gathered}$$

Thus, the measured contents for the first bin would be $n_1 = 2$. This makes sense because the first bin should include all the measurements which fall between 1 and $\frac{5}{2}$, and two of our four measurements do this, namely $x_1 = 1$ and $x_2 = 2$.

The fit contents, ie the contents we will eventually fit to our data, are described by **Equation 2**.

$$f_b(\boldsymbol{\theta}) = \int_{x_0+h(b-1)}^{x_0+hb} f(x; \boldsymbol{\theta}) dx \tag{2}$$

In this case, $f(x; \boldsymbol{\theta})$ represents a histogram that is proportional to the assumed probability density function for our data. We see that $f_1(\boldsymbol{\theta}) = \int_{x_0}^{x_0+h} f(x; \boldsymbol{\theta}) dx$ is just the area of the first bar in the histogram which is proportional to the number of events in that bin. The actual number of events in the bin would be $f_1(\boldsymbol{\theta})/h$ (e.g. if the bin width was $h = 0.1$ and there were three events in the bin, $f_1(\boldsymbol{\theta}) = 0.3$ and $f_1(\boldsymbol{\theta})/h = 3$). It is only important that $f_b(\boldsymbol{\theta})$ is proportional to the number of events in bin b - after all, we are *fitting* this histogram to our data.

Let's assume that the measured contents in each bin n_b are Poisson-distributed, i.e. the probability density function for n_b is defined as $f(n_b; f_b) = \frac{f_b^{n_b} e^{-f_b}}{n_b!}$ where $f_b \equiv f_b(\boldsymbol{\theta})$. If we just consider the first bin, $f(n_1, f_1)$ is the probability of n_1 given some configuration of f_1. It becomes evident that, by multiplying these probabilities for each bin, we will retrieve the likelihood function. Thus, we take the product of the individual p.d.f.s for each bin.

$$L(\boldsymbol{\theta}) = \prod_{b=1}^{M} f(n_b; f_b) = \prod_{b=1}^{M} \frac{f_b^{n_b} e^{-f_b}}{n_b!} \tag{3}$$

It is more convenient to work with the negative log of the likelihood function.

$$F(\boldsymbol{\theta}) \equiv -\ln L(\boldsymbol{\theta}) = -\sum_{b=1}^{M}(n_b \ln f_b - f_b - \ln n_b!) \tag{4}$$

Recognizing that $\sum_{b=1}^{M} f_b = \int_{x_0}^{x_h} f(x;\boldsymbol{\theta})\,dx = A(\boldsymbol{\theta})$ and $\sum_{b=1}^{M} \ln n_b!$ is just an additive constant, we can simplify our expression to the following.

$$F(\boldsymbol{\theta}) = -\sum_{b=1}^{M} n_b \ln f_b + A(\boldsymbol{\theta}) \tag{5}$$

2 Examples in PyTorch

2.1 Single histogram fit (with arbitrary data)

Let's start with the simplest example. Suppose we have 15 measurements assigned to 5 bins of equal width between 0 and 1 – the associated histogram is displayed in **Figure 1**. We choose to fit the histogram shown in **Figure 2** to our data. Notice that the fit histogram has exactly double the measured bin contents. It is evident that multiplying our fit histogram by a factor of 0.5 will return the data histogram. Let's see if we can retrieve this result using PyTorch autograd and the mathematics described in the first section.

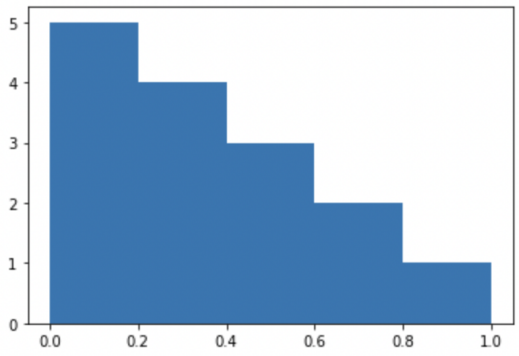

Figure 1: Data histogram

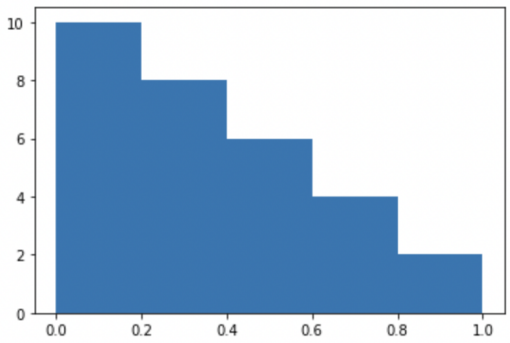

Figure 2: Fit histogram

Code:

```
# generates the measured bin contents. these values were chosen arbitrarily.
dataHist = torch.tensor([[5.],[4.],[3],[2],[1.]])
# generates the fit bin contents. these values are double the measured bin contents.
hist1 = torch.tensor([[10.],[8.],[6],[4],[2.]])

# initializes our paramter p.
p = Variable(torch.rand(1), requires_grad=True)

# model: our model takes the measured bin contents and fit bin contents as inputs
# and outputs the negative log likelihood using the function we derived earlier.
class MLE(nn.Module):
  def __init__(self, dataHist, hist1):
    super(MLE, self).__init__()
    self.nll = lambda p: -((dataHist*torch.log(hist1*p)).sum() - torch.sum(hist1*p))
  def forward(self, p):
    out = self.nll(p)
    return out

model = MLE(dataHist, hist1)

# initializes our optimizer. in this example, we implement adam algorithm.
optimizer = torch.optim.Adam([p], lr=0.1)

# training loop: every iteration will update our fit parameter to a value that
# lowers the negative log likelihood function.
for t in range(1000):
    optimizer.zero_grad()
    NLL = model(p)
    NLL.backward()
    optimizer.step()
    if t % 200 == 0:
      print(f'epoch: {t}, loss: {NLL.item():.4f}, p: {p.detach().numpy()}')
    if t == 999:
      print(f'epoch: {t}, loss: {NLL.item():.4f}, p: {p.detach().numpy()}')

# outputs our results.
bins = np.array([0,1,2,3,4,5])
bin_centres = bins[:-1]+np.diff(bins)/2
plt.scatter(bin_centres, dataHist.detach().numpy(), label='data')
plt.step(bin_centres, hist1.detach().numpy()*p.detach().numpy(), where='mid', label='fit')
plt.legend()
print ('\n')
plt.show()
```

The output of this code is shown in **Figure 3**.

```
epoch: 0, loss: 0.7472, p: [0.8604592]
epoch: 200, loss: -3.2745, p: [0.50000864]
epoch: 400, loss: -3.2745, p: [0.5]
epoch: 600, loss: -3.2745, p: [0.5]
epoch: 800, loss: -3.2745, p: [0.5]
epoch: 999, loss: -3.2745, p: [0.5]
```

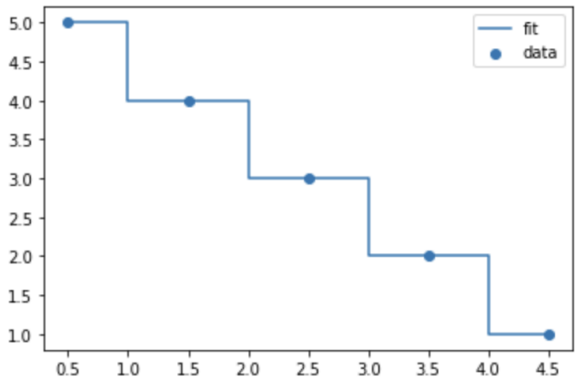

Figure 3: Fitted histogram

Our neural network has navigated us to the minimum of our negative log loss function and it appears that the product of the fit bin contents and 0.5 yields the best fit for our data. This is the result we expected.

2.2 Single histogram fit (with data sampled from an exponential distribution)

Let's move on to a more complicated example. Suppose we have 50 measurements assigned to 8 bins of equal width between 0 and 1 – the associated histogram is displayed in **Figure 4**. At this point, we hypothesize the probability density function that models our N measurements. In this case, the data appears to have been sampled from an exponential distribution (that's because it was!). As a result, we want to create a fit histogram which models an exponential distribution. It is important to note that *it is the task of the researcher to identify the p.d.f. from which the fit histogram will sample its data.* In this example, I have generated a somewhat arbitrary histogram (shown in **Figure 5**) to fit to our data instead. The bin contents are as follows: [13],[11],[9],[7],[5],[4],[3],[1]. Although these numbers reflect the downward trend in the data, they were not sampled from any particular probability density function.

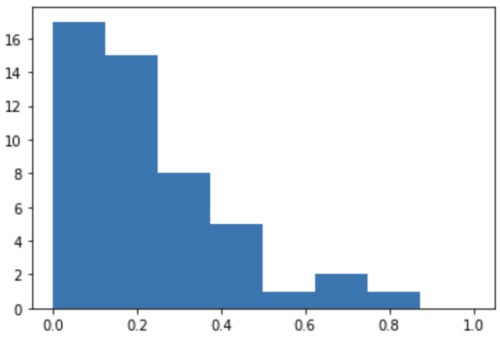

Figure 4: Data histogram

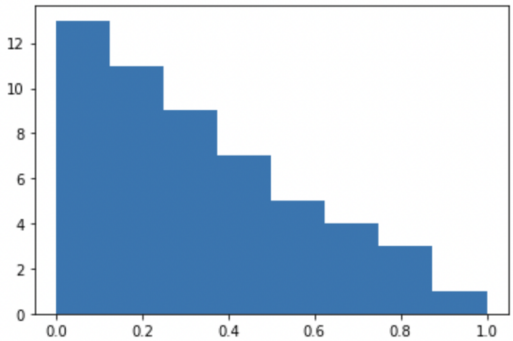

Figure 5: Fit histogram

Code:

```
# generates 50 samples from an exponential distribution with rate  = 5. in
# reality, we wouldn't know the p.d.f. from which our data is sampled from.
data = Exponential(torch.tensor([5.0])).sample().view(1,1)
for s in range (49):
  value = Exponential(torch.tensor([5.0])).sample().view(1,1)
  data = torch.cat((data, value), 0)
# retrieves the measured bin contents.
dataHist = torch.histc(data, bins=8, min=0, max=1).view(8,1)
# generates the fit bin contents. these values were chosen arbitrarily.
hist1 = torch.tensor([[13.],[11.],[9],[7],[5.],[4.],[3.],[1.]])

# initialize our paramter p.
p = Variable(torch.rand(1), requires_grad=True)

# model: our model takes the measured bin contents and fit bin contents as inputs
# and outputs the negative log likelihood using the function derived earlier.
class MLE(nn.Module):
  def __init__(self, dataHist, hist1):
    super(MLE, self).__init__()
    self.nll = lambda p: -((dataHist*torch.log(hist1*p)).sum() - torch.sum(hist1*p))
  def forward(self, p):
    out = self.nll(p)
    return out

model = MLE(dataHist, hist1)

# initializes our optimizer. in this example, we implement adam algorithm.
optimizer = torch.optim.Adam([p], lr=0.1)

# training loop: every iteration will update our fit parameter to a value that
# lowers the negative log likelihood function.
for t in range(1000):
    optimizer.zero_grad()
    NLL = model(p)
    NLL.backward()
    optimizer.step()
    if t % 200 == 0:
      print(f'epoch: {t}, loss: {NLL.item():.4f}, p: {p.detach().numpy()}')
```

```
# outputs our results.
bins = np.array([0,1,2,3,4,5,6,7,8])
bin_centres = bins[:-1]+np.diff(bins)/2
plt.scatter(bin_centres, dataHist.detach().numpy(), label='data')
plt.step(bin_centres, hist1.detach().numpy()*p.detach().numpy(), where='mid', label='fit')
plt.legend()
print ('\n')
plt.show()
```

The output of this code is shown in **Figure 6**.

```
epoch: 0, loss: 19.0883, p: [0.17408863]
epoch: 200, loss: -59.5154, p: [0.92455983]
epoch: 400, loss: -59.5154, p: [0.9245284]
epoch: 600, loss: -59.5154, p: [0.9245284]
epoch: 800, loss: -59.5154, p: [0.9245284]
epoch: 999, loss: -59.5154, p: [0.9245284]
```

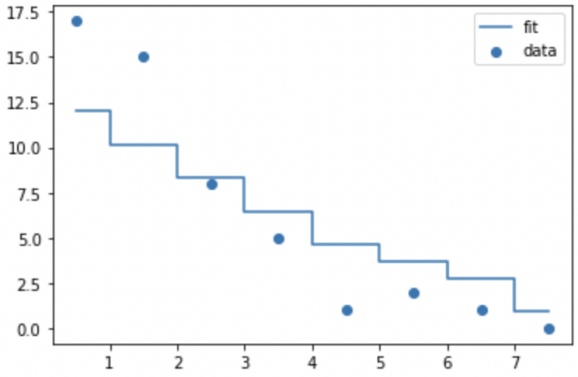

Figure 6: Fitted histogram

Our neural network has navigated us to the minimum of our negative log loss function and it appears that the product of the fit bin contents and 0.9245284 yields the best fit for our data.