Chapter 9; Parameter Estiamtion and Likelihood

Steve Elston

10/4/2020

Introduction

The concept of **likelihood** and **maximum likelihood estimation (MLE)** have been at the core of much of statistical modeling for about 100 years. In 21st Century, these ideas continue to be foundational.

Understanding the concept of likelihood and the use of MLE methods is key to understanding many parametric statistical methods. Further, widely used machine learning models, including some deep learning models, use MLE. For example, in Chapter XXXXXXXXXXXX we will investigate the algorithms for generalized linear models which rely on MLE.

Stigler (2007) traces the long history of the concepts of likelihood and the MLE. The history starts with workers in the early 19th Century, including Gauss and Bernoulli. Major advances are marked by Fisher's seminal work in the 1920s and 1930s in establishing key properties of the MLE.

Likelihood

Likelihood is a measure of how well a **parametric model** fits a data sample. Consider a data sample, $\mathbf{X} = x_1, x_2, \ldots, x_n$. A model for these data, with a parameter vector $\vec{\theta}$, is the conditional density function given the observations $\mathbf{X}$. The relationship between the conditional likelihood and the conditional density function is then simply:

$$\mathcal{L}(\vec{\theta} \mid \mathbf{X}) = f(\mathbf{X} \mid \vec{\theta})$$

In the foregoing, $f(\mathbf{X} \mid \vec{\theta}$ can be either a probability density function (PDF), for continuous distributions, or a probability mass function (PMF), for discrete distributions.

In most practical cases, we work with the **log likelibood**. For a set of observations, $\mathbf{X} = x_1, x_2, \ldots, x_n$, the log likelihood is expressed:

$$l(\vec{\theta} \mid \mathbf{X}) = log\big(\mathcal{L}(\vec{\theta} \mid \mathbf{X})\big) = \sum_j log\Big(f(x_j \mid \vec{\theta})\Big)$$

Example: The Normal likelihood

Consider the example of the univariate Normal distribution. Recall that the probability density function with parameter vector (μ, σ) for a single observation, x, is expressed:

$$f(x, \mu, \sigma^2) = -\frac{1}{(2\pi\sigma^2)^{1/2}} exp\Big[-\frac{1}{2\sigma^2}(x-\mu)^2\Big]$$

For a set of n observations, $\mathbf{X} = x_1, x_2, \ldots, x_n$, the log-likelihood can then be written:

$$l(\mu, \sigma \mid \mathbf{X}) = -\frac{n}{2} log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{j=1}^{n} (x_j - \mu)^2$$

It is clear that the log-likelihood is a function of the parameters, (μ, σ). Notice also that as the number of observations increases so does the likelihood.

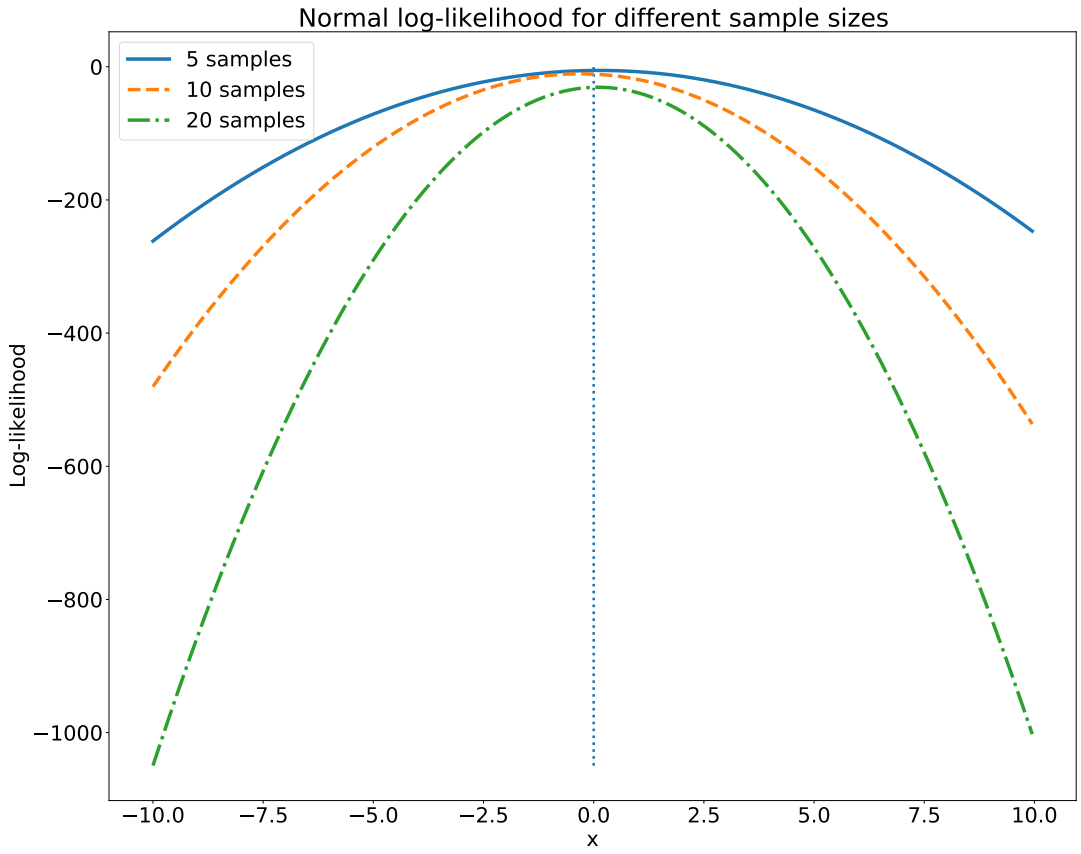
An example will help illustrate the foregoing concepts. The code below plots the likelihood for 5, 10 and 20 samples from a standard Normal distribution. In this example, we vary the parameter μ, and assume the parameter σ is fixed and known. The steps are:
- A random sample is drawn from a standard Normal distribution.
- For the random sample the log-likelihood is computed at each location parameter value. In code we do this by summing the logarithm of the PDFs for each observation at a value of the parameter μ. The lambda (anonymous function) specified the log of the PDF using the scipy.stats.norm.logpdf function.
- The log-likelihood is plotted for each of the location parameter values.

```
def plot_likelihood_1(sample_dist, pdf, num_samples, start, stop, linestyle):
    ## Setup for plot
    plt.rc('font', size=24)
    fig, ax = plt.subplots(figsize=(20, 16), )
    X = np.arange(start, stop, step=0.05)

    ## Loop over number of samples
    for i,samps in enumerate(num_samples):
        ## Compute a sample from standard Normal
        sample = sample_dist(samps)
        ## Loop over the x values and compute the likelihood
        y=[]
        for mu in X:
            y.append(pdf(sample, mu).sum())
        ## Plot the likelihood
        _=ax.plot(X, y, linewidth=4, label= str(samps) + ' samples', linestyle=linestyle[i] )

    ## Add annotions to plot
    ax.vlines(0.0, ymin=min(y), ymax=0.0, linewidth=3, linestyles='dotted')
    ax.set_ylabel('Log-likelihood')
    ax.set_xlabel('x')
    ax.set_title('Normal log-likelihood for different sample sizes')
    ax.legend()
    plt.show()


sample_dist = lambda x: nr.normal(size=x)
pdf = lambda x, y: norm.logpdf(x, loc=y)
num_samples = [5, 10, 20]
start = -10.0
stop = 10.0
linestyle = ['solid','dashed','dashdot']
plot_likelihood_1(sample_dist, pdf, num_samples, start, stop, linestyle)
```



Notice the following expected properties:
- The maximum of the likelihood is near the actual location parmater value the samples were drawen from.
- As the number of samples increases the curvature of the log-likelihood increases and the width decreases.

Exercise: With the foregoing theory and example in mind, its time to work with a real-world example. One might consider modeling the price per square foot of housing using the parametric log-Normal denisty function. The univariate log-Normal density function has two parameters, (μ, σ). Create a plot of the likelihood for random samples of size XXXXXXXXX by the following steps. You should write functions to create this plot, as you will use the same plot in subsequent exercises.
- Use the pandas.notnull method to remove rows where the medListPriceSqft column values are missing.
- Use numpy.log to compute the log of the housing prices.
- Use numpy.random.choice to Bernoulli sample 100 values.
- Compute the likelihood on a grid of values of the two parameters, (μ, σ). Use the numpy.meshgrid function to create the grid with the following values; $0.5 \leq location \leq 6.0$ with steps of 0.05, and $0.15 \leq scale \leq 2.0$ with steps of 0.05.
- Use the matplotlib.pyplot.contour function to display your grid as the Z argument. *Hint:* to improve the interpretablity of the plot use set the following arguments, `levels=500, cmap='RdGy'`.
Next, answer the following questions:
- Which parameter values are approimately the maximum of the log-likelihood?

- For which parameter is the log-likelihood not symmetric about the maximum. Does this asymmetry make sense in terms of the limits on the parameter values?
- Is there any evidence of correlation between the likelihoods of the two parameters?

Example: Binomial Likelihood

In the foregoing example, we investigated the log-likelihood of the Normal distribution. Now, we will explore a different example of log-likelihood for the Binomial distribution. Some differences include:
- The Binomial distribution models **discrete events**. The Normal distribution is for continous valued observations.
- The range of the single parameter, π, of the Binomial distribution is restricted to the range $0 \leq \pi \leq 1$. At least in principle, the range of the location parameter, μ, of the Normal distribution is in the range $-\infty \leq \mu \leq \infty$, with the scale parameter limited to the range $0 < \mu \leq \infty$.

Recall that the Binomial distribution has the following probability mass function (PMF) for k successes in n trials:

$$f(k, n, \pi) = \binom{n}{y} \pi^k (1-\pi)^{n-k}$$
$$0 \leq \pi \leq 1$$
$$0 \leq k \leq n$$

Given this PDF, the log-likelihood is easily found:

$$l(k, n|\pi) = log\binom{n}{k} + k\ log(\pi) + (n-k)\ log(1-\pi)$$

You can see that the Binomial log-likelihood has a strong dependence on both the sample size, n and the number of successes, k.

Exercise: Consider how one might use the concept of likelihood to determine if a fair coin is being used in a toss. Let 'heads' be a success. The question is, how likely are the observed results to be from a fair coin? Perform the following steps:
- Simulate tosses of a fair coin ($\pi = 0.5$) with random draws from a Binomial distribution for sample sizes, 5, 20 and 100, using numpy.random.binomial with $n = 1$.
- Create plots of the likehihood for $0.05 \leq \pi \leq 0.95$ with a different line type for each set of simulated data. Use scipy.stats.binom.logpmfto compute the log-likelihood for these plots. You will need to compute n and k from the randowm draws of the samples.
- Repeat this process four times so that you can compare results from different random draws of the samples.

Answer the following questions:
- How does the shape of the likelihood curve change with sample size? - How does the shape of the likelihood curve change with the random draw for each sample size?
- Are the maximums of the likelihood curves near the expected value of 0.5?
- Based on your above answers, what can you say about the confidence you should have in determining if a coin is fair for these different sample sizes?

The Maximum Likehihood Estimator

As has already been stated, the maximum likelihood estimator (MLE) is a foundational tool for much of statistical inference and machine learning. The concept is attractive and intuitive. Given a log-likelihood

function, find the model parameters which maximize it. Fruther, knowing the distribution allows us to quantify the uncertainty of the MLE parameter estiamtes. As we will see shortly, the distribution of the model parameter estiamtes found by MLE is Normal for large samples, a remarkable property.

We will only give an overview of the essential elements of the theory. Many standard statistics and machine learning texts contain much greater detail. For example, Chapter 7 of Freedman (2009) provides an overview of MLE theory along with many specific examples. Section 4.4 of Davidson (2008) provides a rigorous derviation of significant properties of the MLE.

The MLE is a **point estimator**. The solution is an estimate of a single parameter value, or point value, with the highest likelihood. We will explore measures of uncertainty for point estimates in the next section of this book.

The maximum likelihood for the model parameters is achieved when two conditions are met:

$$\frac{\partial\ l(\theta)|\mathbf{X}}{\partial\theta} = 0 \frac{\partial^2\ l(\theta)|\mathbf{X}}{\partial\theta^2} < 0$$

You can interpret these two conditions as follows:
- The first derivative of the log-likelihood function, or slope, will be 0 at either maximum or minimum points. In general, $\vec{\theta}$ is a vector of model parameters, and the partial derivatives of log-likelihood are a vector; the gradient of the log likelihood with respect to the model parameters. These first derivatives or gradient of the log-likelihood are known as the **score function**.
- The second derivatives of the log-likelihood indicates the curvature. A maximum has negative curvature, whereas, a minimum has positive curvature.

Fisher information and properties of MLE

The maximum likelihood estimator has some useful and desireable properties. These properties form a basis for understanding MLE methods. To understand these concepts we will outline the derivation of some key properties. Additional details can be found in many reference, including Effron and Hastie (2016) Section 4.2 or Davidson (2008) Section 4.4.

We start with a matrix of second partial derivatives of the log-likelihood function. For a multi-parameter model, the second derivatives of the log-likelihood function form a matrix. The elements of the matrix include all second partial derivatives with respect to each model parameter. This matrix is known as the **observed information matrix** of the model, $\mathcal{J}(\vec{\theta})$.

$$\mathcal{J}(\vec{\theta}) = \begin{pmatrix} \frac{\partial^2 l(\vec{\theta}\ |\ \mathbf{X})}{\partial\theta_1^2} & \frac{\partial^2 l(\vec{\theta}\ |\ \mathbf{X})}{\partial\theta_2\ \partial\theta_1} & \cdots & \frac{\partial^2 l(\vec{\theta}\ |\ \mathbf{X})}{\partial\theta_n\ \partial\theta_1} \\ \frac{\partial^2 l(\vec{\theta}\ |\ \mathbf{X})}{\partial\theta_1\ \partial\theta_2} & \frac{\partial^2 l(\vec{\theta}\ |\ \mathbf{X})}{\partial\theta_2^2} & \cdots & \frac{\partial^2 l(\vec{\theta}\ |\ \mathbf{X})}{\partial\theta_2\ \partial\theta_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 l(\vec{\theta}\ |\ \mathbf{X})}{\partial\theta_n\ \partial\theta_1} & \frac{\partial^2 l(\vec{\theta}\ |\ \mathbf{X})}{\partial\theta_n\ \partial\theta_2} & \cdots & \frac{\partial^2 l(\vec{\theta}\ |\ \mathbf{X})}{\partial\theta_n^2} \end{pmatrix}$$

The observed information matrix has some useful properties:
- The more negative the values of the second partial derivatives, the greater the curvature of the log-likehihood. This means that the log-likelihood likelihood function has a narrower range. The narow peak implies that the information on the parameter value for the model is high.
- The matrix is symmetric, so the information is symmetric around the maximum likelihood point.

The computing the observed information matrix is useful if you have obervations. But, how can one consider the information of the MLE before sampling data or performing an experiment? The answer is to use the **expected information** or **Fisher information**. Fisher information is computed by taking the expectation over the second derivative of the log-likelihood, or the observed information:

$$\mathcal{J}(\vec{\theta}) = -\mathbf{E}\big\{\mathcal{I}(\vec{\theta})\big\} = -\mathbf{E}\big\{\frac{\partial^2\ l(\mathbf{X}|\theta)}{\partial\theta^2}\big\}$$

The Fisher information leads to an important relationship. The MLE parameter estimate $\hat{\theta}$ is a Normally distributed random variable. To see this we start with a Taylor expansion of the maximum likelihood estimator:

$$0 = \frac{\partial\ l(\hat{\theta})_{\mathbf{X}}}{\partial\theta} = \frac{\partial\ l(\theta)_{\mathbf{X}}}{\partial\theta} + \frac{\partial^2\ l(\hat{\theta})_{\mathbf{X}}}{\partial\theta^2}(\hat{\theta} - \theta)$$

or

$$0 = l'(\theta)_{\mathbf{X}} + l''(\hat{\theta})_{\mathbf{X}}$$

Where, $l'(\theta)_{\mathbf{X}}$ symbolizes the first partial derivative and $l''(\hat{\theta})_{\mathbf{X}}$ symbolizes the second partial derivative. Contiuing with the simplified notation, and solving for $\hat{\theta}$;

$$\hat{\theta} = \theta + \frac{l'(\theta)_{\mathbf{X}}/n}{-l''(\hat{\theta})_{\mathbf{X}}/n}$$

Now the Fisher information is related to the score function as its variance:

$$\frac{\partial\ l(\theta)}{\partial\theta} \dot{\sim} \mathcal{N}\big(0, 1/\mathcal{I}_\theta\big)$$

For a **large sample**, $n \to \infty$, we can take the expectation of the above relation over $\mathbf{X}$. Assuming the first and second derivatives of the log-likelihood function exist and are continious, and applying the Central Limit Theorem we arrive at a remarkable result:

$$\hat{\theta}\dot{\sim}\mathcal{N}\Big(\theta, \frac{1}{n\mathcal{I}(\theta)}\Big)$$

This relationship shows several important properties of the MLE:
- The maximum likelihood estimate of the model parameters, $\hat{\theta}$, is Normally distributed.
- The larger the Fisher information, the lower the variance of the parameter estimate. This observation is consistent with the idea that greater curvature of the log likelihood function the more certain the parameter estimate.
- The variance of the parameter estimate is inversly proportional to the number of samples, n.

These concepts will play a key role in our analysis of confidence intervals in the next section of this book.

Example of MLE for Normal distribution

To help make the foregoing more concrete, we will work out the maximum likelihood estimator for the Normal distribution. As a first step we work out the derivatives of the log-likelihood function with respect to the model parameters, μ and $\sigma 2$. The first derivatives are:

$$\begin{pmatrix} \frac{\partial l}{\partial \mu} \\ \frac{\partial l}{\partial \sigma^2} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sigma^2}\sum_j (x_j - \mu) \\ -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4}\sum_j (x_j - \mu)^2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Starting with the above, it is easy to find the well known formula to compute the maximum likelihood estimate of μ. This estimate is the mean, $\bar{x}$. The steps to verify this relationship are:

$$\sum_{j=1} n(x_j - \mu) = 0$$

$$\bar{x} = \frac{1}{n}\sum_{j=1} x_j$$

Finding the maximum likelihood estimate of σ^2 is straight forward as well. We use the notation s^2 to represent the MLE of σ^2. The familar relationship to estiamte the variance is found as follows:

$$\frac{1}{s^2}\sum_{j=1}(x_j - \mu)^2 = n$$

$$s^2 = \frac{1}{n}\sum_{j=1}^{n}(x_j - \bar{y})^2$$

$$\mathcal{J}(\vec{\theta}) = \begin{pmatrix} \frac{\partial^2 l}{\partial \mu^2} & \frac{\partial^2 l}{\partial \mu\ \partial \sigma^2} \\ \frac{\partial^2 l}{\partial \mu\ \partial \sigma^2} & \frac{\partial^2 l}{\partial (\sigma^2)^2} \end{pmatrix} = \begin{pmatrix} -\frac{n}{\sigma^2} & -\frac{n}{\sigma^4}(\bar{x} - \mu) \\ -\frac{n}{\sigma^4}(\bar{x} - \mu) & -\frac{n}{2\sigma^4} + \frac{1}{\sigma^6}\sum_j (x_j - \mu)^2 \end{pmatrix} = \begin{pmatrix} -\frac{n}{\sigma^2} & 0 \\ 0 & -\frac{n}{2\sigma^4} \end{pmatrix}$$

The simplification results from the fact that $\bar{x} \to \mathbf{E}(\mathbf{x_j}) = \mu$ and $s^2 \to \mathbf{E}\{(x_j - \mu)^2\} = \sigma^2$ in the limit of a large sample from the law of large numbers.

There are some aspects of these relationships which make the MLE method attractive:
- The curvature of the MLE for both parameters increases with the number of samples n. In other words, the peak of the log-likelihood function becomes better defined as n increases.
- The maximum likelihood estimates of the parameters, μ and σ^2 are independent. The off-diagonal terms are 0.

Exercise: The observed information matrix provides a measure of the log-likelihood curveature at the estimated parameter values. In a previous exercise you plotted the log-likelihood for the approximately log-Normally distributed housing price data. Create a plot the Fisher information for the two parameters of the univariate Normal disribution. To create this plot, do the following:
- Use ramdom Bernoulli samples of size, 10, 20, 50 and 100, from the log housing price values. To do so, you will need an estimate of $\sigma 2$ for the population.
- To create consistent values of Fisher Informatiion for each of the parameters, you must use an extimated value for the other parameter. To esimate the Fisher information for the μ parameter use the emperical estimate of the variance, s^2. For the Fisher information estimate of the $\sigma 2$ parameter, use the estimated mean, $\bar{x}$. Compute the estimate of $s2$ and $\bar{x}$ from the entire set of log price samples.
- Use numpy.apply_along_axis with a Python lambda (anonymous function) to compute the Fisher information for each parameter and sample Numpy functions.
- Plot the Fisher information for both parameters on the vertical axis against sample size on the horrizontal axis. Be sure to follow proper chart paractice to ensure these curves are distinguishable.
Examine these plots and nswer the following questions:
- Do the values of Fisher informatuion for both parameters change in a similar way? If not, what does this tell you about your ability to estimate these paramters?
- Is the change in Fisher inforamtion consistent with the curvature of the likelihood functions plotted in the previous eercise?
- How might consideration of Fisher information help you construct a sampling plan?

Example; Binomial distribution

As another example of likelihood estimation we turn out attention to the Binomial distribution. This distribution is discrete and has a single parameter. We start with the derivative of the log-likelihood function:

$$\frac{\partial\ l(n,k|\pi)}{\partial\theta} = \frac{k}{n} - \frac{n-k}{1-\pi} = 0$$

The solution to the above has a simple closed form:

$$\pi = \frac{k}{n}$$

The information for the maximum likelihood estimator is found by taking the second derivative of the log-likelihood function:

$$\frac{\partial^2 l(n,k|\pi)}{\partial\theta^2} = -\frac{k}{\pi^2} - \frac{n-k}{1-\pi^2} = -\frac{n}{\pi(1-\pi)}$$

You can see that the curvature increases with the number of samples n. Further, there is a dependency on π. As $\pi \to 0$ or $\pi \to 1$ the log-likelihoood becomes negative extremely rapidly.

> **Exercise:** For a fixed sample size of 100, plot the information of the binomial maximum likelihood estimator. Make the range of the parameter, $0.1 \leq \pi \leq 0.99$ in steps of 0.01. Examine your plot and answer these questions:
> - Given the curvature around the actual parameter value, how much confidence can you have in the maximum likelihood parameter estimate?
> - How would you describe the change in information at the limts of the range of π ploted? How do you think this behavor would impeed estimates of a parameter close to these limits?

Finding Solutions Without a Closed Form

So far in this discussion, we have only explored cases where the maximum likelihood solution is a closed form. In the foregoing examples, simple algebra produced solutions. In general, this will not be the case. A common example, is logistic regression.

When no closed form solution is available an approximate solution can be found by numerical **optimization methods** or **root finding methods**. Here, we will briefly examine two methods, gradient descent and Newton's method. These methods are similar, but differ in important detail. We will only present an outline here. Many texts give full details of a number of different numerical methods for finding a maximum of a function.

For the example of logistic regression, you can find many details on MLE methods in some machine learning and statitics texts. Section 4.4 of Hastie, Tibshirani and Friendmen (2009), or Section 8.3 of Murphy (2012) provide overviews.

Gradient descent methods

The gradient descent method has an intuitive explanation. The maximum of the log-likelihood function can be found by following the gradient 'uphill' until the gradient is approximately 0. Notice that if we used the negative of the gradient we would find a minimum.

To formulate this algorithm, we start with the gradient of the log-likelihood function with respect to the parameters, θ. The gradient is the vector of partial derivatives with respect to each of the parameters.

$$grad(l(\vec{\theta})) = \nabla_\theta \; l(\vec{\theta}) = \begin{pmatrix} \frac{\partial l(\vec{\theta} \mid \mathbf{X})}{\partial \theta_1} \\ \frac{\partial l(\vec{\theta} \mid \mathbf{X})}{\partial \theta_2} \\ \vdots \\ \frac{\partial l(\vec{\theta} \mid \mathbf{X})}{\partial \theta_n} \end{pmatrix}$$

We will use the notation, $\hat{\theta}$, to indicate an estimate of the true parameter vector, $\vec{\theta}$.Given a current parameter estimate vector at step n , $\hat{\theta}_n$, the improved parameter estimate vector, $\hat{\theta}_{n+1}$, is found:

$$\hat{\theta}_{n+1} = \hat{\theta}_n + \gamma \; \nabla_\theta \; l(\hat{\theta})$$

The hyperparameter γ is the **learning rate** or **step size**. Determining a learning rate can have a significant effect on the performance of the gradient. This hyperparameter can be chosen manually, often with by a search of the hyperparameter space.

Using a fixed γ is far from optimal. At the magnitude of the gradient changes toward the maximum point the optimal step size changes. More sophisticated algorithms use an adaptive method to determine an optimal step at each step. Finding this step size can be found dynamically using a **line search** procedure. The line search typically uses a quadratic approximation to find a maximum (or minimum).

Exercise: As you have already seen, maximum likelihood estimation of Normal distribution parameters can be done using simple colsed form solutions. Now, ou will use the gradient descent method to estimate these parameters. Using a Bernoulli sample of 100 log housing price values, follow these steps.
- Compute and print the mean, *mu*, and variance, σ^2, using numpy functions. - Create python functions to compute the gradients for both parameters, given the parameter values and the data sample. - Create and execute python code to perform the gradient descent update, by calling the gradient function. Use starting parameter values, $\mu = 1.0$ and $\sigma^2 = 1.0$. To simplify the algorithm use a fixed learning rate, $\gamma = 0.0001$. Use a simple stopping criteria of the norm of the gradient being less than 0.001. Save the norm of the gradient and both parameter values at each iteration of the algorithm. Hints: 1) Keep in mind that the paramters and the graident are both vectors, so vector operations from the Numpy package should be used. 2) Depending on the Bernoulli sample of log price values your algorithm may not converge, in which case you should decrease the learning rate parameter, *gamma*. - Plot the norm of the gradient along with the values of both parameters vs. the iteration number.
- Plot the values of both parameters on a contour plot of the likelihood function. See pervious exercise for instructions to make the contour plot background.
Examine your plot and answer these questions:
- Compare the parameter values estimated with the gradient descent algorithm with the values computed using closed form solutons with Numpy functions. Are you estimated parameters close to the closed form solutions?
- Examine the plots of the parameter values and norm of the gradient vs. iteration. Can you see evidence that one parameter is easier to fit than the other?
- Examine the countour plot with descent path. Does the trajectory of the descent path always follow the maxium gradient direction? In other words, is the trajectory of the descent path always perpendicular to the countour lines of the log likelihood function?
-Can you see evidence that the descent path is affected by the possible range of the parameter values?
Tip: you can reuse the plot functions you create for this exercise for the next exercise.

Stochastic gadient descent (SGD)

The simple gradient descent algorithm and Newton's method have limited scalability. Both methods require sums over the entire gradient vector. This calculation must be done as a single batch in memory. Computing the full gradient at each step limits scalability. As a result the simple version of gradient descent is referred to as **batch gradient descent**.

To achieve scalability, the **stochastic gradient decent (SGD)** algorithm computes the expected gradient using a **mini-batch**. The mini-batch is a limited-size Bernoulli sample from the full set of cases. As opposed to batch gradient decent, SGD uses a series of gradient approximations computed from the mini-batches. These gradient approximations are inherently noisy or stochastic, giving rise to the method's name.

Using mini-batches greatly increases scalability. While the gradient estimates are less accurate, these estiamtes can be computed very quickly. As a result of scalability, SGD is the workhorse of many large-scale statistical methods. Mini-batch optimization is often referred to as **online optimization** since the optimizer algorithm can update the solution as cases arrive.

The basic idea of stochastic optimization is using a Bernoulli random sample of the data to estimate the **expected update** of the model weights. The weight update for SGD then becomes:

$$W_{t+1} = W_t + \alpha \; E_{\hat{p}data}\Big[\nabla_W J(W_t)\Big]$$

where,
$E_{\hat{p}data}[]$ is the expected value of the gradient given the Bernoulli sample of the data $\hat{p}data$.

Choosing batch size can require some tuning. If the batch is too small, the gradient estimate will be poor. Further, hardware resources will not be fully utilized. Large batches require significant memory and slow the calculation.

Empirically, SGD has good convergence properties. This behavior seems to arise since stochastic gradient samples provide a better exploration of the loss function space. The variations in the gradient from one mini-batch sample to another help the algorithm escape from saddle points or other areas of the loss function with poor convergence properties. In fact, for very large data sets, the SGD algorithm often converges before the first pass through the data is completed.

The pseudo code for the SGD algorithm is: `Random_sort(cases) while(grad > stopping_criteria): mini-batch = sample_next_n(cases) grad = compute_expected_grad(mini_batch) weights = update_weights(weights, grad)`

Notice that if the sampling continues for more than one cycle through the cases, the samples are biased. In practice, this small bias does not seem to mater much.

> **Exercise:** In the previous exercise, you used batch gradient descent to estimate the parameters of the distributio of the log housing prices. Now, you will use SGD to estimate these parameters. Using a Bernoulli mini=batch sample size of 100 log housing price values, follow these steps:
> - Create python functions to compute the Hessian of the log-likelihood, given the parameter values and the data sample. For the univariate Normal distribution, you can - Create and execute python code to perform the gradient descent update, by calling the gradient function. Use starting parameter values, $\mu = 1.0$ and $\sigma^2 = 1.0$. To simplify the algorithm use a fixed learning rate, $\gamma = 0.0001$. Use a simple stopping criteria of the norm of the gradient being less than 0.001. Save the norm of the gradient and both parameter values at each iteration of the algorithm. Hints: 1) Keep in mind that the paramters and the graident are both vectors, so vector operations from the Numpy package should be used. 2) Depending on the Bernoulli sample of log price values your algorithm may not converge, in which case you should decrease the learning rate parameter, *gamma*. - Plot the norm of the gradient along with the values of both parameters vs. the iteration number.
> - Plot the values of both parameters on a contour plot of the likelihood function. See pervious

exercise for instructions to make the contour plot background.
Examine your plot and answer these questions:
- Compare the parameter values estimated with the gradient descent algorithm with the values computed using closed form solutons with Numpy functions. Are you estimated parameters close to the closed form solutions?
- Examine the plots of the parameter values and norm of the gradient vs. iteration. Can you see evidence that one parameter is easier to fit than the other?
- Examine the countour plot with descent path. Does the trajectory of the descent path always follow the maxium gradient direction? In other words, is the trajectory of the descent path always perpendicular to the countour lines of the log likelihood function?
-Can you see evidence that the descent path is affected by the possible range of the parameter values?

Newton's method

Newton's method, and related methods, employ a **quadratic approximation** to optimization. For MLE, Newton's method uses both the first and second derivatives of the log-likelihood function.

Consider a nonlinear log-likelihood function, $l(\theta \,|\mathbf{X})$. We use a Taylor expansion to find the tangent point of the log-likelihood, $\theta_n = \theta_k + \delta\theta$. The Taylor expansion is:

$$l(\theta_k + \delta\theta) = l(\theta_k) + l'(\theta_k)\delta\theta + \frac{1}{2}l''(\theta_k)\delta\theta$$

Setting this expansion to 0, we have:

$$0 = \frac{d}{d\ \delta\theta}\Big(l(\theta_k) + l'(\theta_k)\delta\theta + \frac{1}{2}l''(\theta_k)\delta\theta^2\Big) \tag{1}$$
$$0 = l'(\theta_k) + l''(\theta_k)\delta\theta \tag{2}$$

It is simple to solve for $\delta\theta$:

$$\delta\theta = \frac{l'(\theta_k)}{l''(\theta_k)}$$

The foregoing suggests an iterative method to find a maximum. At each step in the iteration, we find the update x_{n+1} by the following relationship:

$$\theta_{n+1} = \theta_n + \gamma\frac{l'(\theta_k)}{l''(\theta_k)} \tag{3}$$
$$= \theta_n + \gamma\ \delta\theta \tag{4}$$

Here, γ is a learning rate of step size.

It can now be seen that Newton's method has a quadratic form. The quadratic form is not just a mathematical curiosity. Newton's method exhibits convergence quadratic in the number of iterations. This rate of convergence can be compared to the approximate linear convergence for gradient descent methods.

In higher dimensions we use the gradient, $\nabla l(\vec{\theta})$ for the first derivatives of the likelihood. The second derivative is represented as a matrix, $\nabla^2 l(\vec{\theta})$, known as the **Hessian**. In higher dimensions, the quadratic update is:

$$x_{n+1} = x_n + \gamma |\nabla^2_\theta l(\vec{\theta})|^{-1} \nabla_\theta l(\vec{\theta})$$

This update requires computing the **inverse Hessian** matrix. There are several practical difficulties with this proceedure:
- The inverse of the Hessian may not exist as this matrix may be singular. In Section XXXXXXXXXXXX of this book we will explore methods for dealing with this problem.
- If there are a large number of model parameters, the Hessian will have high dimensionality. Computing the inverse of of a high dimensional matrix is computationally intensive.
- Computing the full gradient and Hessian requires summing over all observations. For large data sets, this process can become prohibitive.

For large scale problems **quasi-Newton** methods are used. These methods use an approximation to avoid the need to compute the full inverse Hessian. The **limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS)** (Fletcher 1987; Nocedal and Wright 2006) algorithm the most widely used quasi-Newton method.

As with the gradient descent method, the learning rate or step size must be selected. This can be done with a hyperparameter search. Or more commonly, using a line search proceedure. Line search is used in the BFGS algorithm, for example.

> **Computational Note:** You can find implementations the L-BFGS (limited-memory-BFGS) algorithm in many packages with Python APIs. These tyupically implement algorithms including line search options. An example, is the scipy.optimize.minimize with method='L-BFGS-B' function.

Limitations of MLE

The maximum likelihood estimator has a number of important limitations which should be kept in mind whenever you use it. In this section we will explore some, but far from all, of these limitations.

Incorrect model and complex distributions

In many real-world problems the distribution and, therefore the likelihood function, are not simple. These situations are problematic for maximum likelihood methods. The MLE will generally not find a useful solution for a maximum of the distribution.

For example, consider what happens when we assume a likelihood function that is only approximately correct. This situation could arise from either the population being modeled having a different distribution or simply from outliers in the form of erroneous samples. Regardless, parameter estimates using an incorrect likelihood function are usually biased.

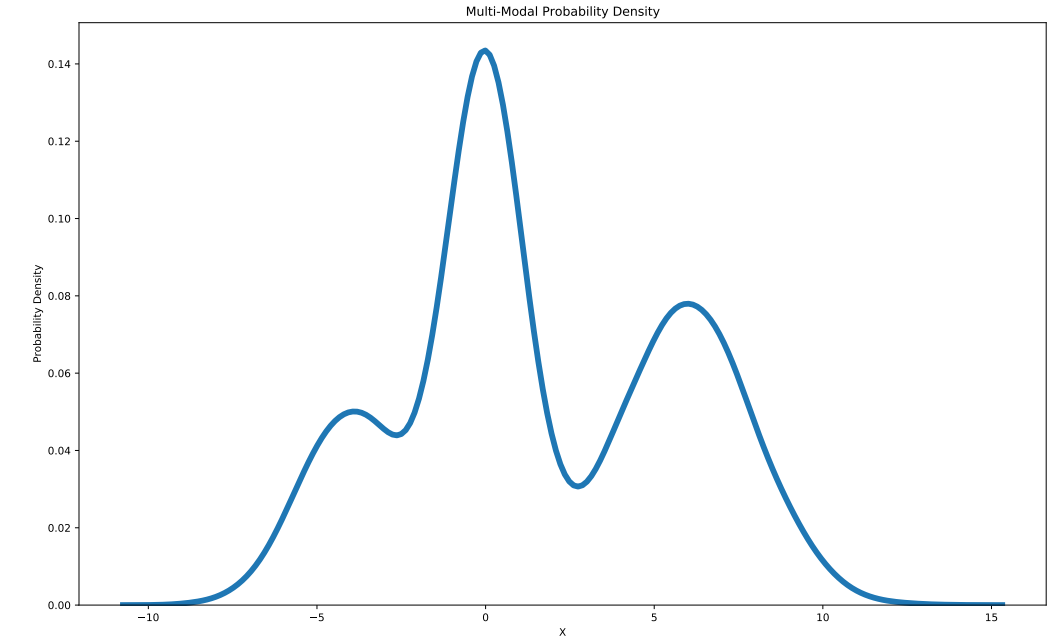
Consider what happens to the maximum likelihood estimator if the population distribution has much more complex behavior than the model. For example, the likelihood might a distribution might be mixture of simple distributions. In this case, the mixture will have multiple modes. One mode of the distribution will be the maximum. However, the mode found by any gradient-based algorithm is dependent on the starting point. An MLE algorithm will find the nearest **local maximum**. But, there is no guarantee of fiding the actual maximum likelihood.

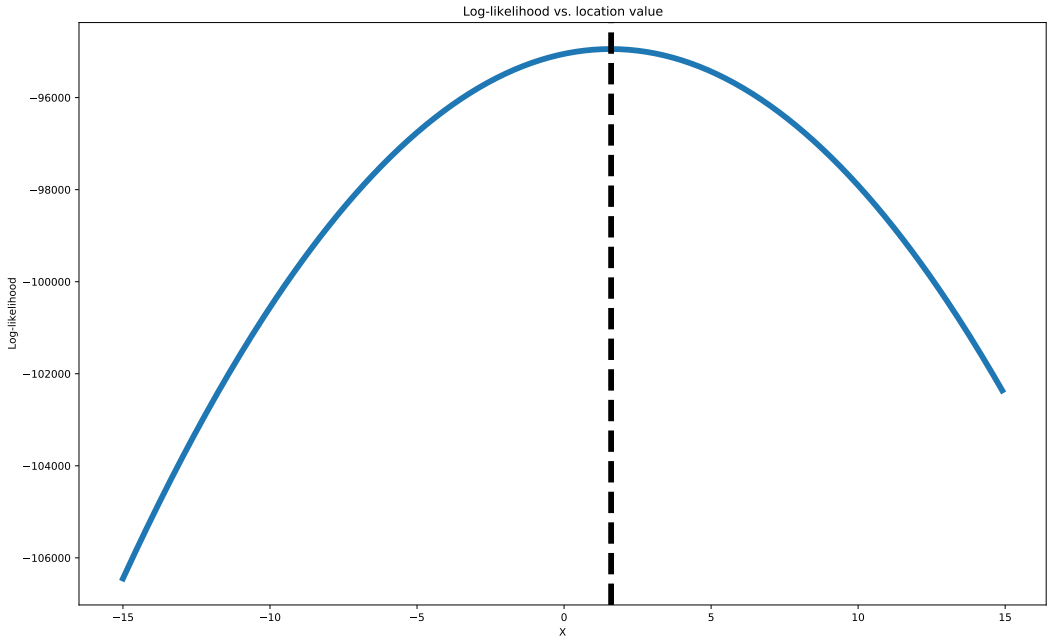
Another

```
samples = np.concatenate((nr.normal(loc=0.0, scale=1.0, size=10000),
                          nr.normal(loc=6.0, scale=2.0, size=10000),
                          nr.normal(loc=-4.0, scale=1.5, size=5000)))
samples = pd.Series(samples) #, name='Multi_Modal_Samples')
```

```
plt.rc('font', size=9)
fig, ax = plt.subplots(2, 1, figsize=(15, 20))
#_=plt.hist(samples, bins=100)
sns.kdeplot(data=samples, ax=ax[0], linewidth=5)
ax[0].set_xlabel('X')
ax[0].set_ylabel('Probability Density')
ax[0].set_title('Multi-Modal Probability Density')

scale = np.var(samples)
logpdf = lambda x, y: norm.logpdf(x, loc=y, scale=scale)
mu_values = np.arange(-15.0, 15.0, 0.1)
log_likelihood = []
for mu in mu_values:
    log_likelihood.append(logpdf(samples,mu).sum())
max_LL = mu_values[np.argmax(log_likelihood)]
ax[1].plot(mu_values,log_likelihood, linewidth=5)
ax[1].axvline(max_LL, linewidth=5, linestyle='--', color='black')
ax[1].set_title('Log-likelihood vs. location value')
ax[1].set_xlabel('X')
ax[1].set_ylabel('Log-likelihood')
```


Multi-Modal Probability Density
Probability Density
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−10
−5
0
5
10
15
X


Log-likelihood vs. location value
Log-likelihood
−96000
−98000
−100000
−102000
−104000
−106000
−15
−10
−5
0
5
10
15
X

Parameter near limits

For many distributions, parameter values have limits. In these cases, log-likelihood function may have an extremely high gradient near the limit. The result can be poorly determined parameter estimates and slow convergence.

You have already seen the example of estimating the variance near 0 for a Normal distribution model. In another example, you have seen that finding a maximum likelihood estimate for the parameter of the Binomial distribution can be difficult near the limits of the range; 0 and 1.

High dimensional problems

The MLE method often finds poor solutions to problems in high dimensions. By high dimensions we mean a large numbers of parameters. For these problems, the likelihood function has corresponding high dimensionality; one dimension for each parameter.

For high dimensional problems, it is often the case that the gradient and Hessian are not well determined. Sources of uncertainty in the variables can lead to considerable uncertainty in determining the gradient in high dimensions. MLE algorithms may not converge of will converge to results with a large uncertainty.

Even it cases with only a few parameters MLE methods can have convergence problems. In this chapter you have seen the affect of difficulties fitting the variance parameter have on fitting the location parameter for a univariate Normal distribution.

We will address methods to deal with some of these problems in Section XXXXX of this book.

Correlated features

The forgoing discussion assumes that the variables used for the MLE process are independent. In reality, this is never truly the case. If some variables have a high correlations, the MLE algorithm can breakdown since gradients will not be well determined. We will have considerably more to say about this problem in Section XXXXXXX of this book.

Bibliography

Davidson, A. C. 2008. *Statistical Models*. First Paperback Edition. Cambridge University Press.

Efron, B, and T Hastie. 2016. *Computer Age Statistical Inference: Algorithms, Evidence, and Data Science*. Cambridge University Press.

Fletcher, Roger. 1987. *Practical Methods of Optimization*. 2nd ed. New York: John Wiley & Sons.

Freedman, D. A. 2009. *Statistical Models: Theory and Paractice*. Cambridge University Press.

Hastie, T, R Tibshirani, and J Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Second. Springer.

Murphy, K. P. 2012. *Machine Learning: A Probabilistic Perspective*. MIT Press.

Nocedal, Jorge, and Stephen J. Wright. 2006. *Numerical Optimization*. Second. New York: Springer.

Stigler, Stephen M. 2007. "The Epic Story of Maximum Likelihood." *Statistical Science* 22 (4): 598–620.