MaximumLikelihood

August 11, 2018

I would argue that maximum likelihood represents one of the most important concepts in machine learning. When we perform machine learning we typically have three main components:

- A dataset
- A model
- A method for fitting model parameters

In general we make assumptions about all three of these components in order to complete the task at hand. Maximum likelihood is part of the third component because it allows us to quantify how well our model parameters fit a given dataset.

The main goal of many machine learning techniques is to find the set of parameters for a given model that best fits the data. In probability terms, this means finding the set of parameters that is most probable given the data:

$$\arg\max_{\theta} P(\theta \mid D)$$

i.e. we want to maximise the **posterior** distribution. Under Bayes theorem we know that:

$$P(\theta \mid D) \propto P(D \mid \theta)P(\theta)$$

$$posterior \propto likelihood * prior$$

When people refer to maximimum likelihood however, they are ignoring the prior over parameters ($P(\theta)$) and just maximising over the **likelihood**:

$$\arg\max_{\theta} P(D \mid \theta)$$

As a side note if one maximises over the posterior, taking into account both the likelihood and the prior, then this is known as the $Maximum\ A\ Posteriori\ (MAP)$ estimate. The inclusion of the prior is often important because it allows for regularization of the parameters θ. One problem with performing maximum likelihood estimation alone is that it is prone to overfitting the data. Unfortunately the discussion of MAP estimates, regularization and fully Bayesian approaches

are for another tutorial. For now we shall just focus on maximum likelihood estimation as it is fundamental to these other extensions.

So we have defined maximum likelihood as maximising the probability of the data given a set of parameters θ. The exact form of this probability is dependent on the assumptions we make about our dataset. As a motivating example we shall just consider simple linear regression. When conducting linear regression we make the following assumptions:

1. i.i.d

The first assumption we make about the data is that it is independent and identically distributed (i.i.d). This means that the probability of one datapoint is independent from other datapoints and that all data points are generated from the same probability distribution. This is an extremely common assumption in classic machine learning apporaches (see Markov Models for exmaples of when we don't assume independence between datapoints). Importantly, the independence assumption allows us to re-write the likelihood as a product of each datapoint:

$$P(D \mid \theta) = \prod_{i=1}^{N} P(d_i \mid \theta)$$

where i indexes an individual data point from our dataset. We shall see later that writing the likelihood as a product will make our life much easier when it comes to calculating it.

2. Zero-Mean Gaussian Noise

The other assumption we make about the data is more specific to linear regression. In the most common form of linear regression we assume that the target variable t_i has been generated by adding some zero-mean gaussian noise to the actual value y_i, which is a function of the input variables x_i and our parameters θ:

$$t_i = y(x_i, \theta) + \epsilon_i$$
$$e_i \sim \mathcal{N}(0,\, \sigma^2)$$

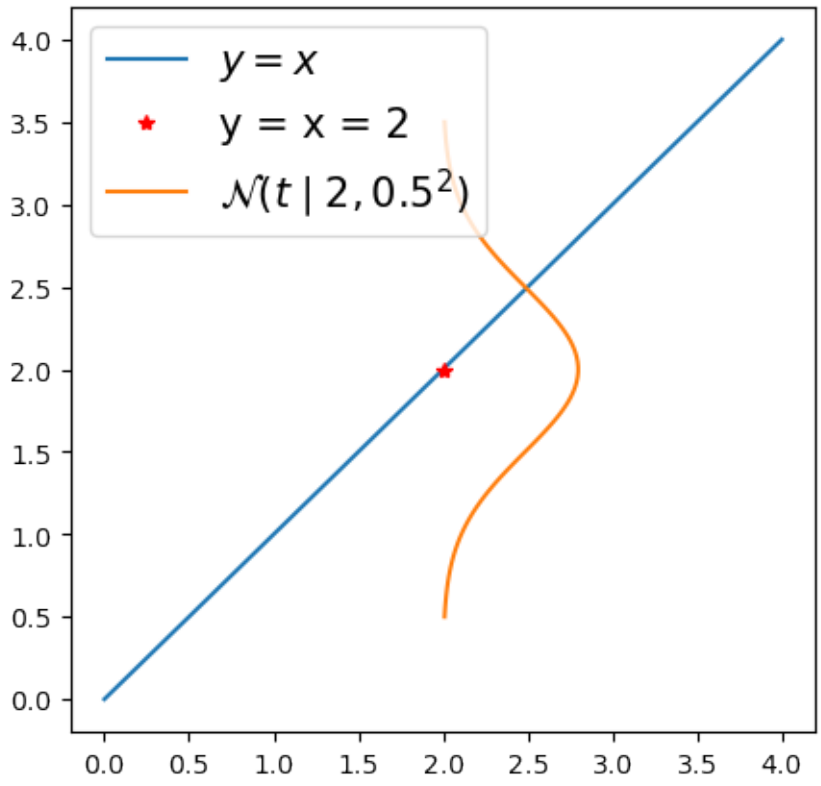
The probability of a single target variable value therefore becomes:

$$P(t_i \mid x_i, \theta, \sigma^2) = \mathcal{N}(t_i \mid y(x_i, \theta), \sigma^2)$$

The noise is zero-mean so we can just take the mean of our gaussian to be the determinstic function we are trying to fit ($y(x_i, \theta)$ is the 'most probable value for t_i').

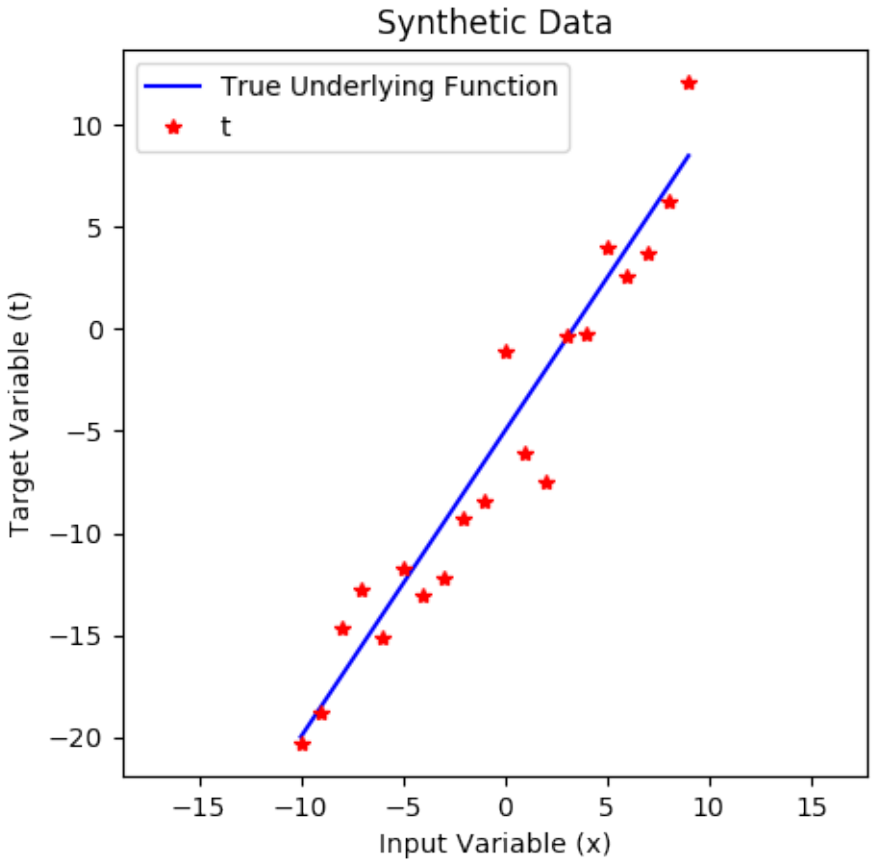
The code below plots a graph that shows the zero-mean gaussian noise assumption for the function $y = x$. The zero-mean gaussian noise means that the t_i has a gaussian probability distribution centred on the true value y_i.

```
In [2]: %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.mlab as mlab
        import matplotlib as mpl
        mpl.rcParams['figure.dpi']= 100

        fig = plt.figure(figsize=(5,5))
        plt.plot(np.arange(5), np.arange(5), label='$y = x$')
        plt.plot(2,2,'r*', label='y = x = 2')
        x_loc = 2
        mu = 0
        sigma = .5
        y = np.linspace(mu - 3*sigma, mu + 3*sigma, 100)
        plt.plot(mlab.normpdf(y, mu, sigma) + x_loc, y + x_loc,
                 label='$\mathcal{N}(t \mid 2, 0.5^2)$')
        plt.legend(prop={'size': 15}, loc='upper left')
        plt.show()
```



Now lets generate some synthetic data that uses these assumptions so that we can calculate the likelihood later on in this notebook.

```
In [4]: num_points = 20
        sigma = 2.5
        theta = np.array([1.5, -5])

        X = np.arange(num_points) - 10
        y = (theta[0] * X) + theta[1]
        t = y + np.random.normal(0, sigma, num_points)

        fig = plt.figure(figsize=(5,5))
        plt.plot(X, y, 'b', label='True Underlying Function')
        plt.plot(X, t, 'r*', label='t')
        plt.axis('equal')
        plt.legend(loc='upper left')
        plt.xlabel('Input Variable (x)')
        plt.ylabel('Target Variable (t)')
        plt.title('Synthetic Data')
        plt.show()
```



If we combine the two aforementioned assumptions together we get the following expression for the likelihood:

$$P(\mathbf{t} \mid \mathbf{X}, \theta, \sigma^2) = \prod_{i=1}^{N} \mathcal{N}(t_i \mid y(x_i, \theta), \sigma^2)$$
$$\mathbf{X} = \{x_1, ..., x_N\}$$
$$\mathbf{t} = \{t_1, ..., t_N\}$$

where **X** is our entire set of input values and **t** is our entire set of target values. Fortunately, as you will see in much of machine learning, gaussian distributions are fairly easy to work with and so we can simplify this expression down to something more manageable:

$$\begin{aligned}
\arg\max_{\theta} P(D \mid \theta) &= \arg\max_{\theta} P(\mathbf{t} \mid \mathbf{X}, \theta, \sigma^2) \\
&= \arg\max_{\theta} \prod_{i=1}^{N} \mathcal{N}(t_i \mid y(x_i, \theta), \sigma^2) \\
&= \arg\max_{\theta} \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(t_i - y(x_i, \theta))^2}
\end{aligned}$$

We can take the logarithm of the likelihood so that we have a sum of values rather than a product. Often this is called maximising the log-likelihood. Note that taking the logarithm of the likelihood doesn't affect which values of θ maximise it.

$$\begin{aligned}
\arg\max_{\theta} P(D \mid \theta) &= \arg\max_{\theta} \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(t_i - y(x_i, \theta))^2} \\
&= \arg\max_{\theta} \sum_{i=1}^{N} \ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{1}{2\sigma^2}(t_i - y(x_i, \theta))^2
\end{aligned}$$

Notice how the first term is indepedent of theta and so doesn't affect which values of θ maximise the likelihood. We can therefore just drop this first term because it is a constant.

$$\begin{aligned}
\arg\max_{\theta} P(D \mid \theta) &= \arg\max_{\theta} \sum_{i=1}^{N} \ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{1}{2\sigma^2}(t_i - y(x_i, \theta))^2 \\
&= \arg\max_{\theta} \sum_{i=1}^{N} -\frac{1}{2\sigma^2}(t_i - y(x_i, \theta))^2
\end{aligned}$$

Finally by minimising the negative of the above and dropping the last few constants we get the following final expression for the maximum likelihood:

$$\arg\max_{\theta} P(D \mid \theta) = \arg\max_{\theta} \sum_{i=1}^{N} -\frac{1}{2\sigma^2}(t_i - y(x_i, \theta))^2$$
$$= \arg\min_{\theta} \sum_{i=1}^{N} (t_i - y(x_i, \theta))^2$$

If you are already familiar with objective / cost functions in machine learning then you will notice that the maximum likelihood solution is just the same as minimizing the sum of squared errors (SSE) between the target variables (t_i) and our predictions ($y(x_i, \theta)$)!

Indeed the majority of the objective functions you learn about in machine learning will involve a term that is equivalent to the likelihood. This is a very powerful result and it gives you a really good understanding of why we chose the objective functions we do and what assumptions they are actually making about the data! It also gives you the necessary tools to devise your own objective functions in a mathematically sound way using probability theory, taking into account the specific assumptions you wish to make about your data.

This is just one example of how to obtain an expression for the maximum likelihood solution based on simple linear regression. Many other machine learning algorithms make different assumptions about the nature of the data and so the maximum likelihood solution will take on a different form. I would strongly urge you to see if you can find one of these other exmaples and follow the reasoning behind it!

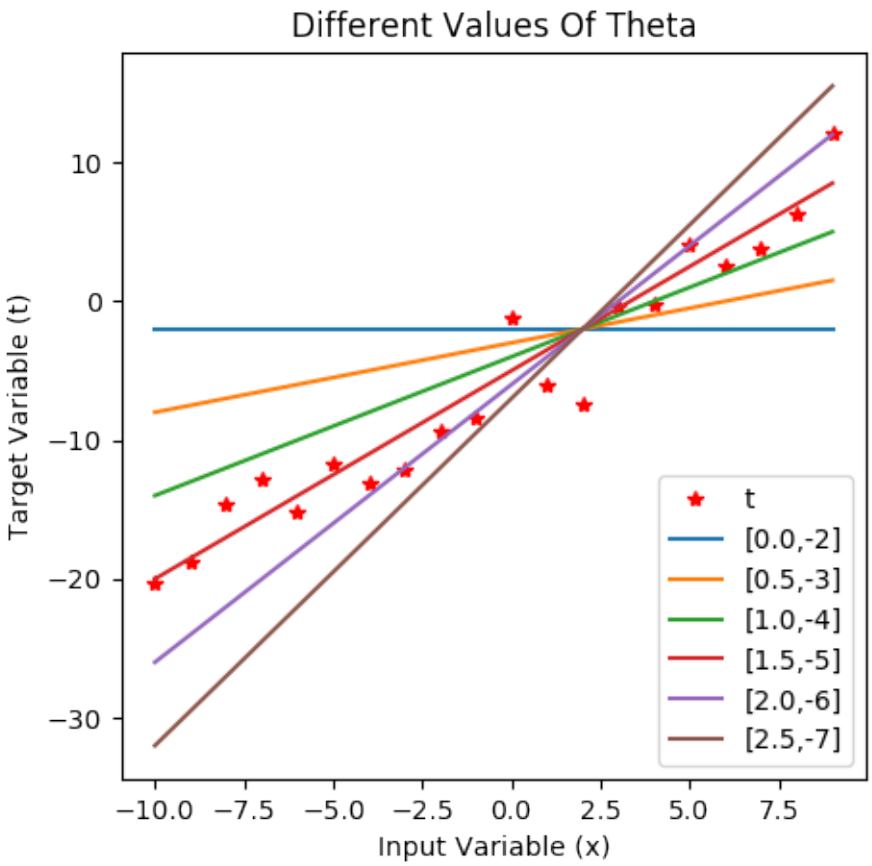
Unfortunately, how to actually find the maximum likelihood solution of the linear regression problem is beyond the scope of this notebook i.e. how to find the minimum of $\sum_{i=1}^{N}(t_i - y(x_i, \theta))^2$ with respect to θ. There are many optimization tehcniques out there to solve this problem, from closed loop solutions (normal equations) to iterative solutions (gradient descent). The key point in this tutorial is that the likelihood gives us a mathematically sound way of evaluating how good our parameters are at fitting the data. Other techniques can then use this to find the best fitting parameters.

Just to reinforce this idea, there is some code below that uses the synthetic data we created above to calculate the SSE for different values of θ. Notice how as θ approaches the true values of the linear function used to generate the data, the SSE decreases. The SSE doesn't reach 0 because a linear model of this form cannot account for the residual noise in our target variables.

```
In [7]: print('True Theta Values = Theta_1: ' + str(theta[0]) +
              ' Theta_2: ' + str(theta[1]) + '\n')

        theta_1s = np.arange(0,3,.5)
        theta_2s = np.arange(-2,-8,-1)
```

```
        fig = plt.figure(figsize=(5,5))
        plt.plot(X, t, 'r*', label='t')

        for theta_1, theta_2 in zip(theta_1s, theta_2s):
            predictions = theta_1 * X + theta_2
            SSE = np.sum((t - predictions)**2)
            print('Theta_1: ' + str(theta_1) + ' Theta_2: ' + str(theta_2) +
                  ' SSE: ' + str(SSE))

            plt.plot(X, predictions, label= '[' + str(theta_1) + ',' +
                     str(theta_2) + ']')

        plt.legend(loc='lower right')
        plt.xlabel('Input Variable (x)')
        plt.ylabel('Target Variable (t)')
        plt.title('Different Values Of Theta')
        plt.show()
```

```
True Theta Values = Theta_1: 1.5 Theta_2: -5.0

Theta_1: 0.0 Theta_2: -2 SSE: 1891.30289655
Theta_1: 0.5 Theta_2: -3 SSE: 901.173303124
Theta_1: 1.0 Theta_2: -4 SSE: 306.043709694
Theta_1: 1.5 Theta_2: -5 SSE: 105.914116265
Theta_1: 2.0 Theta_2: -6 SSE: 300.784522835
Theta_1: 2.5 Theta_2: -7 SSE: 890.654929405
```


Different Values Of Theta
Target Variable (t)
Input Variable (x)
10
0
−10
−20
−30
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
t
[0.0,-2]
[0.5,-3]
[1.0,-4]
[1.5,-5]
[2.0,-6]
[2.5,-7]