Bayesian Machine Learning

A PyMCentric Introduction

Quan Nguyen

Modeling success rate

Question: What is the success rate of a binary event?

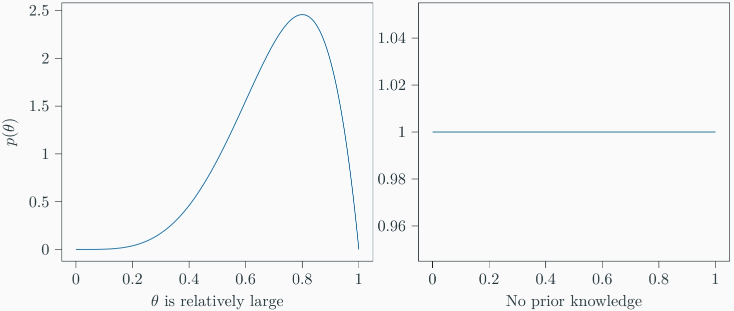
$$\theta = ?$$

Making inference: the prior distribution

$$\theta \in [0, 1]$$

Making inference: the prior distribution

$$\theta \in [0, 1]$$



Making inference: the likelihood

$\mathcal{D} = \{1, 0, 1, \ldots, 1\}$: k ones and $n - k$ zeros

$$p(\text{first one} \mid \theta) = \theta$$

$$p(\text{first zero} \mid \theta) = 1 - \theta$$

Making inference: the likelihood

$\mathcal{D} = \{1, 0, 1, \ldots, 1\}$: k ones and $n - k$ zeros

$$p(\text{first one} \mid \theta) = \theta$$

$$p(\text{first zero} \mid \theta) = 1 - \theta$$

$$\ldots$$

$$p(\mathcal{D} \mid \theta) = \theta^k (1 - \theta)^{n-k}$$

Making inference: Bayes' theorem

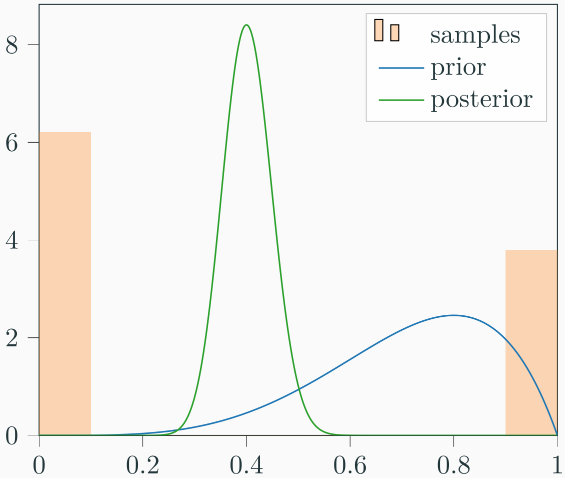
$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta)p(\theta)}{p(\mathcal{D})} = \frac{p(\mathcal{D} \mid \theta)p(\theta)}{\int p(\mathcal{D} \mid \theta)p(\theta)d\theta}$$

Making inference: Bayes' theorem

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta)p(\theta)}{p(\mathcal{D})} = \frac{p(\mathcal{D} \mid \theta)p(\theta)}{\int p(\mathcal{D} \mid \theta)p(\theta)d\theta}$$

Most of the time, the denominator cannot be computed exactly, but approximated using samples.

Making inference: the posterior distribution



Modeling a latent function

Gaussian Processes (GPs): normal distribution prior for every function value.

Modeling a latent function

Gaussian Processes (GPs): normal distribution prior for every function value.

- Mean function $\mu(x)$: central tendency of our function belief
- Covariance function $K(x, x')$: controls the smoothness of function belief

Modeling a latent function

Gaussian Processes (GPs): normal distribution prior for every function value.

- Mean function $\mu(x)$: central tendency of our function belief
- Covariance function $K(x, x')$: controls the smoothness of function belief
- Can be updated using Bayes' theorem

$$\mu_{\mathcal{D}}(x) = \mu(x) + K(x, \mathbf{x})(\mathbf{\Sigma} + \mathbf{N})^{-1}(\mathbf{y} - \boldsymbol{\mu})$$
$$K_{\mathcal{D}}(x, x') = K(x, x') - K(x, \mathbf{x})(\mathbf{\Sigma} + \mathbf{N})^{-1}K(\mathbf{x}, x')$$

GP inferences: the prior

```
ℓ = 1
η = 3

with pm.Model() as model:
    cov_func = η**2 * pm.gp.cov.Matern52(ℓ, 1)
    mean_func = pm.gp.mean.Zero()

    gp = pm.gp.Marginal(mean_func, cov_func)
```

GP inferences: the prior

```
ℓ = 1
η = 3

with pm.Model() as model:
    cov_func = η**2 * pm.gp.cov.Matern52(ℓ, 1)
    mean_func = pm.gp.mean.Zero()

    gp = pm.gp.Marginal(mean_func, cov_func)
```

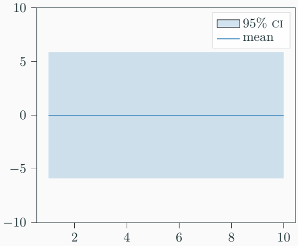


Figure 1: GP prior

GP inferences: the posterior

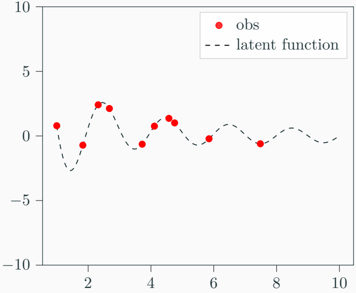


Figure 2: Observations from the latent function

GP inferences: the posterior

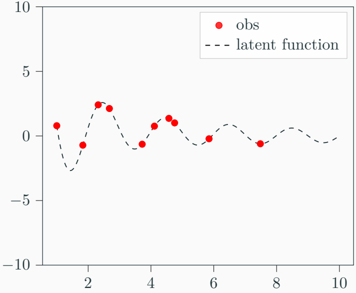


Figure 2: Observations from the latent function

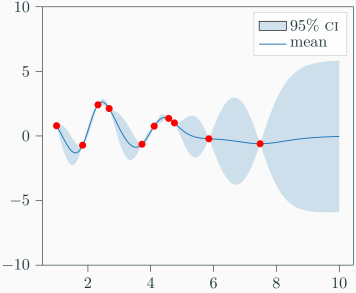


Figure 3: GP posterior conditioned on observations

GP hyper-parameters

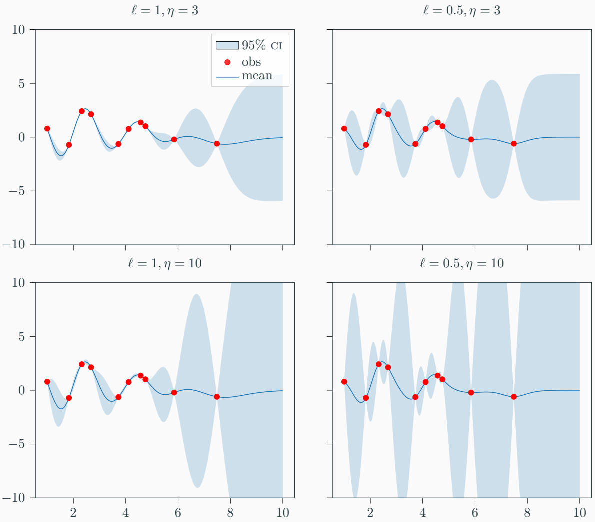


Figure 4: Effect of hyper-parameters on the posterior GP

GPs in PyMC3

```
with pm.Model() as model:
    ℓ = pm.Gamma('ℓ', alpha=2, beta=1)
    η = pm.HalfCauchy('η', beta=3)

    cov_func = η**2 * pm.gp.cov.Matern52(ℓ, 1)
    mean_func = pm.gp.mean.Zero()

    gp = pm.gp.Marginal(mean_func, cov_func)
    obs = gp.marginal_likelihood(
        'obs', X=X, y=y)

    map_post = pm.find_MAP()
```

GPs in PyMC3

```
with pm.Model() as model:
    ℓ = pm.Gamma('ℓ', alpha=2, beta=1)
    η = pm.HalfCauchy('η', beta=3)

    cov_func = η**2 * pm.gp.cov.Matern52(ℓ, 1)
    mean_func = pm.gp.mean.Zero()

    gp = pm.gp.Marginal(mean_func, cov_func)
    obs = gp.marginal_likelihood(
        'obs', X=X, y=y)

    map_post = pm.find_MAP()
```

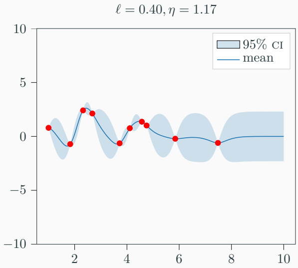


Figure 5: GP posterior via MAP

Linear regression

$$y = \mathbf{w}^T \mathbf{x} + \varepsilon$$

Linear regression

$$y = \mathbf{w}^T \mathbf{x} + \varepsilon$$

Optimal $\mathbf{w}$ via *least squares*:

$$\mathbf{w}^* = \arg\min \sum (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$
$$\overline{y} = \mathbf{w}^{*T} \mathbf{x}$$

Linear regression

$$y = \mathbf{w}^T \mathbf{x} + \varepsilon$$

Optimal $\mathbf{w}$ via *least squares*:

$$\mathbf{w}^* = \arg\min \sum (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$
$$\overline{y} = \mathbf{w}^{*T} \mathbf{x}$$

Bayesian linear regression:

$$p(\mathbf{w} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$$
$$p(\varepsilon \mid \sigma) = \mathcal{N}(\varepsilon; 0, \sigma)$$

Linear regression

$$y = \mathbf{w}^T \mathbf{x} + \varepsilon$$

Optimal $\mathbf{w}$ via *least squares*:

$$\mathbf{w}^* = \arg\min \sum (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$
$$\overline{y} = \mathbf{w}^{*T} \mathbf{x}$$

Bayesian linear regression:

$$p(\mathbf{w} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$$
$$p(\varepsilon \mid \sigma) = \mathcal{N}(\varepsilon; 0, \sigma)$$

$p(y \mid \mathbf{x}, \mathcal{D})$ can be computed.

Linear regression: an example

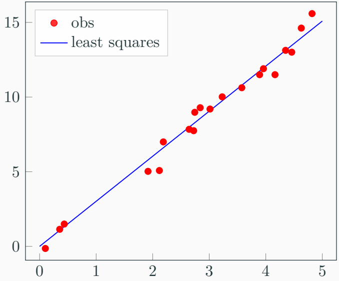


Figure 1: Least-squares solution

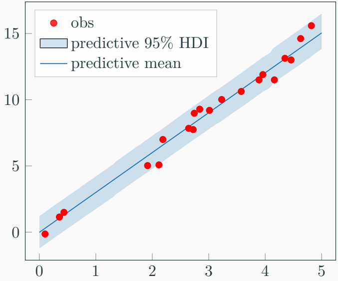


Figure 2: Bayesian solution

Linear regression: an example

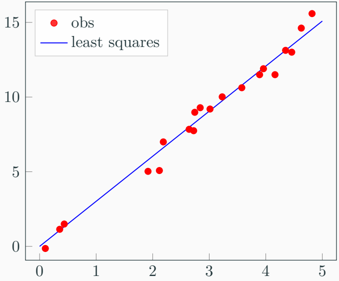


Figure 1: Least-squares solution

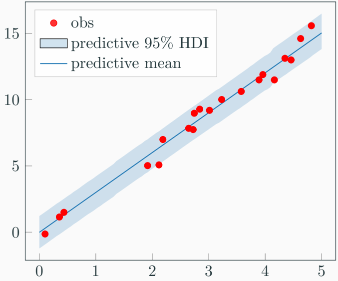


Figure 2: Bayesian solution

Bayesian neural networks: Thomas Wiecki - Probablistic Programming Data Science with PyMC3

Bayesian decision theory

Two main components:

- Bayesian beliefs about unknown quantities
- Utility function

Bayesian decision theory

Two main components:

- Bayesian beliefs about unknown quantities
- Utility function

$$u(\text{decision}, \text{outcome}) = v \in \mathbb{R},$$
$$d^* = \arg\max \mathbb{E}_{\text{outcome}} \left[u(\text{decision}) \right]$$

Bayesian-optimal The Price is Right

Setup:

- Compete against an opponent to guess product price p as $g \in \mathbb{R}$.
- Opponent's guess: $\bar{p}$.
- If a guess is above p then the player gets nothing.
- If a guess is not above p, then the player with the closer guess wins the product.
- Our belief about p: $\mathcal{N}(p; \mu, \sigma^2)$.
- What is the optimal decision d^*?

Deriving the optimal decision

General process: compute the utility of each action given the values of the unknown quantities and marginalize over those values according to our belief.

Deriving the optimal decision

General process: compute the utility of each action given the values of the unknown quantities and marginalize over those values according to our belief.

Utility function

- $u(g \mid p) = 0$ if $g > p$

Deriving the optimal decision

General process: compute the utility of each action given the values of the unknown quantities and marginalize over those values according to our belief.

Utility function

- $u(g \mid p) = 0$ if $g > p$
- $u(g \mid p) = 0$ if $g < \overline{p} <= p$

Deriving the optimal decision

General process: compute the utility of each action given the values of the unknown quantities and marginalize over those values according to our belief.

Utility function

- $u(g \mid p) = 0$ if $g > p$
- $u(g \mid p) = 0$ if $g < \overline{p} <= p$
- $u(g \mid p) = 0$ if $\overline{p} < g <= p$

Deriving the optimal decision

General process: compute the utility of each action given the values of the unknown quantities and marginalize over those values according to our belief.

Utility function

- $u(g \mid p) = 0$ if $g > p$
- $u(g \mid p) = 0$ if $g < \overline{p} <= p$
- $u(g \mid p) = 0$ if $\overline{p} < g <= p$

Expected utility of a decision g

$$\mathbb{E}\left[u(g)\right] = \int u(g \mid p)p(p)dp; \quad g^* = \arg\max_g \mathbb{E}\left[u(g))\right]$$

Visualizing the optimal decision

Example: $p \sim \mathcal{N}(100, 10^2)$; $\overline{p} = 75$.

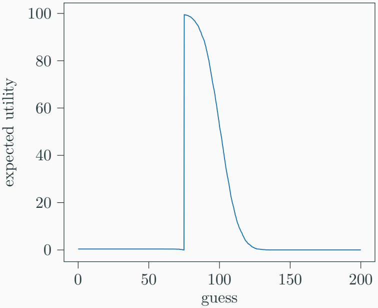


Figure 1: Expected utility as a function of our guess

The multi-armed bandit problem

Problem setup:

- k slot machines, each returns a coin with probability θ_i
- N pulls available
- Goal: maximize the expected number of coins (utility) received

The multi-armed bandit problem

Problem setup:

- k slot machines, each returns a coin with probability θ_i
- N pulls available
- Goal: maximize the expected number of coins (utility) received

Applications: designing clinical trials, personalized recommendations, etc.

The multi-armed bandit problem

Problem setup:

- k slot machines, each returns a coin with probability θ_i
- N pulls available
- Goal: maximize the expected number of coins (utility) received

Applications: designing clinical trials, personalized recommendations, etc.

Modeling the return rates: placing a prior on each θ_i and update accordingly.

The Bayesian optimal policy

$$i^* = \arg\max_i \mathbb{E}\left[\text{future reward} \mid \text{current outcome}\right]$$

The Bayesian optimal policy

$$i^* = \arg\max_i \mathbb{E}\left[\text{future reward} \mid \text{current outcome}\right]$$

- $\mathbb{E}\left[\text{future reward} \mid \text{current outcome}\right]$ is generally intractable.
- Need policies that *approximates* the optimal policy.
- Goal: have an $\mathcal{O}(\log t)$ upper-bound on the expected *regret*.

The Upper-Confidence Bound policy

At iteration $t = 1, 2, \dots, N$, for each arm i with the corresponding posterior belief $p(\theta_i)$, compute with constant c:

$$q_i(t) = Q_i \left(1 - \frac{1}{t \, (\log N)^c} \right),$$

where Q_i is the corresponding quantile function of $p(\theta_i)$.

Decision: pick $i^* = \arg\max_i q_i(t)$.

The Upper-Confidence Bound policy

At iteration $t = 1, 2, \ldots, N$, for each arm i with the corresponding posterior belief $p(\theta_i)$, compute with constant c:

$$q_i(t) = Q_i \left(1 - \frac{1}{t\ (\log N)^c}\right),$$

where Q_i is the corresponding quantile function of $p(\theta_i)$.

Decision: pick $i^* = \arg\max_i q_i(t)$.

Justification: $q_i(t)$ is high if either (1) $\mathbb{E}[\theta_i]$ is high or (2) there is significant uncertainty in $p(\theta_i)$.

$\rightarrow$ Exploration vs. exploitation

The Thompson Sampling policy

At iteration $t = 1, 2, \ldots, N$, for each arm i with the corresponding posterior belief $p(\theta_i)$, draw a sample as an approximation of the true rate:

$$\overline{\theta_i} \sim p(\theta_i).$$

Decision: pick $i^* = \arg\max_i \overline{\theta_i}$.

The Thompson Sampling policy

At iteration $t = 1, 2, \ldots, N$, for each arm i with the corresponding posterior belief $p(\theta_i)$, draw a sample as an approximation of the true rate:

$$\overline{\theta_i} \sim p(\theta_i).$$

Decision: pick $i^* = \arg\max_i \overline{\theta_i}$.

Justification: $\overline{\theta_i}$ is likely to be high if either (1) $\mathbb{E}[\theta_i]$ is high or (2) there is significant uncertainty in $p(\theta_i)$.

$\rightarrow$ Exploration vs. exploitation

The Thompson Sampling policy

At iteration $t = 1, 2, \ldots, N$, for each arm i with the corresponding posterior belief $p(\theta_i)$, draw a sample as an approximation of the true rate:

$$\overline{\theta_i} \sim p(\theta_i).$$

Decision: pick $i^* = \arg\max_i \overline{\theta_i}$.

Justification: $\overline{\theta_i}$ is likely to be high if either (1) $\mathbb{E}[\theta_i]$ is high or (2) there is significant uncertainty in $p(\theta_i)$.

$\rightarrow$ Exploration vs. exploitation

Thompson Sampling: Sid Ravinutala - Thompson Sampling and COVID testing

Bayesian optimization

Problem setup:

- Access to potentially noisy output of a black-box function $y = f(\cdot) + \varepsilon$ but not its gradients
- Expensive queries
- Goal: sequentially query the function to find the function maximizer $x^* = \arg\max_{x \in \mathcal{X}} f(x)$

Bayesian optimization

Problem setup:

- Access to potentially noisy output of a black-box function $y = f(\cdot) + \varepsilon$ but not its gradients
- Expensive queries
- Goal: sequentially query the function to find the function maximizer $x^* = \arg\max_{x \in \mathcal{X}} f(x)$

Modeling the objective function: Gaussian processes.

Using the posterior belief

Posterior predictive distribution of value y at point x:

$$p(y \mid x, \mathcal{D}).$$

Using the posterior belief

Posterior predictive distribution of value y at point x:

$$p(y \mid x, \mathcal{D}).$$

Probability of Improvement:

$$\Pr\left[y > \overline{y} \mid x, \mathcal{D}\right] = \int_{\overline{y}}^{\infty} p(y \mid x, \mathcal{D}) dy.$$

Using the posterior belief

Posterior predictive distribution of value y at point x:

$$p(y \mid x, \mathcal{D}).$$

Probability of Improvement:

$$\Pr\left[y > \overline{y} \mid x, \mathcal{D}\right] = \int_{\overline{y}}^{\infty} p(y \mid x, \mathcal{D}) dy.$$

Expected Improvement:

$$\mathbb{E}\left[y - \overline{y} \mid y > \overline{y}, x, \mathcal{D}\right] = \int_{\overline{y}}^{\infty} (y - \overline{y}) p(y \mid x, \mathcal{D}) dy.$$

Distribution of the true maximizer

From the posterior predictive $p(y \mid x, \mathcal{D})$, the probability of a point x being the maximizer can be considered: $p(x^* \mid \mathcal{D})$.

Distribution of the true maximizer

From the posterior predictive $p(y \mid x, \mathcal{D})$, the probability of a point x being the maximizer can be considered: $p(x^* \mid \mathcal{D})$.

Entropy Search: choosing the point that approximately causes the largest decrease (in differential entropy) of $p(x^* \mid \mathcal{D})$.

Predictive Entropy Search: having the same evaluation objective but approximating entropy reduction differently.

Distribution of the true maximizer

From the posterior predictive $p(y \mid x, \mathcal{D})$, the probability of a point x being the maximizer can be considered: $p(x^* \mid \mathcal{D})$.

Entropy Search: choosing the point that approximately causes the largest decrease (in differential entropy) of $p(x^* \mid \mathcal{D})$.

Predictive Entropy Search: having the same evaluation objective but approximating entropy reduction differently.

PyMC-powered Bayesian optimization: pyGPGO.

Other Bayesian decision-making problems

Active learning: interactively asking for new data points to minimize cost and maximize predictive performance.[1]

[1] Settles, Burr. Active learning literature survey. University of Wisconsin-Madison Department of Computer Sciences, 2009.

[2] Garnett, Roman, et al. "Bayesian optimal active search and surveying." arXiv preprint arXiv:1206.6406 (2012).

Other Bayesian decision-making problems

Active learning: interactively asking for new data points to minimize cost and maximize predictive performance.[1]

Active search: interactively asking for new data points to discover a rare class of data points while minimizing cost.[2]

[1] Settles, Burr. Active learning literature survey. University of Wisconsin-Madison Department of Computer Sciences, 2009.

[2] Garnett, Roman, et al. "Bayesian optimal active search and surveying." arXiv preprint arXiv:1206.6406 (2012).