03 Probabilistic Interpretation and Bayesian Methods

October 15, 2021

Information: *Basic concepts and simple examples of Probabilistic interpretation and Bayesian methods*

Written by: *Zihao Xu*

Last updated date: *Oct.15.2021*

1 Maximum Likelihood Estimation

1.1 Motivation

In the chapter talking about ***Generalization and Regularization***, the concepts of parameter estimation, bias and variance are used to formally characterize notions of generalization, underfitting and overfitting. Here are some important remarks.

- View the parameter estimator $\hat{\boldsymbol{\theta}}$ as a **function** of the sampled training dataset

$$\hat{\boldsymbol{\theta}} = g\left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(m)}\right)$$

- The datasets (training, testing and probably validation) are generated by a **i.i.d.** probability distribution over datasets called the **data-generating process** (i.i.d. assumptions can be applied to almost all the common tasks)
- Assume that the true parameter value $\boldsymbol{\theta}$ is fixed but unknown
- Since the **data** is drawn from a **random process**, any function of the data is random, which means the parameter estimator $\hat{\boldsymbol{\theta}}$ is a **random variable**

The concepts of **bias** and **variance** are used to measure the performance of a parameter estimator. However, **for obtaining a good estimator**, it's not a good idea to guess that some function might make a good estimator and then to analyze its bias and variance. This motivated some principles from which specific functions that are good estimators for different models can be derived.

1.2 Definition

- Consider a set of m examples $\mathcal{D} = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(m)}\right\}$ drawn independently from the true but unknown data-generating distribution $p_{\text{data}}(\mathbf{x})$. Let $p_{\text{model}}(\mathbf{x}|\boldsymbol{\theta})$ be a parametric family of probability distributions over the same space indexed by $\boldsymbol{\theta}$
  - That is to say, p_{model} maps any configuration $\mathbf{x}$ to a real number estimating the true probability $p_{\text{data}}(\mathbf{x})$

- Particularly, focus on the **likelihood** which is first introduced in the prerequisite chapter ***Probability Theory***

$$\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)} \mid \boldsymbol{\theta} \sim p_{\text{model}}\left(\mathbf{x}^{(1:m)} \mid \boldsymbol{\theta}\right)$$

  As a fast review, the likelihood tells us how *plausible* it is to observe $\mathbf{x}^{(1:m)}$ if we know the model parameters are θ

- Since the examples are assumed to be drawn **independently**, the likelihood can be factorized

$$p_{\text{model}}\left(\mathbf{x}^{(1:m)} \mid \boldsymbol{\theta}\right) = \prod_{i=1}^{m} p_{\text{model}}\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)$$

  Then **maximum likelihood** estimator for $\boldsymbol{\theta}$ is then defined as

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\text{model}}\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)$$

- While this simple production may lead to a lot of inconveniences such as **numerical underflow**, taking the **logarithm** of the likelihood does not change the location for maximum ($\arg\max_{\boldsymbol{\theta}}$) but does conveniently transform a product into a sum

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)$$

- Obviously, rescaling the likelihood does not change the location for maximum ($\arg\max_{\boldsymbol{\theta}}$), we can divide by m to obtain a version of the criterion that is expressed as an expectation with respect to the empirical distribution $\hat{p}_{\text{data}}$ defined by the training data

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}} \left[\log p_{\text{model}}\left(\mathbf{x} \mid \boldsymbol{\theta}\right)\right]$$

- The most **common** choice for the likelihood of a single measurement is to pick it to be **Gaussian**

1.3 KL divergence

- Maximum likelihood estimation can be viewed as minimizing the dissimilarity between the empirical distribution $\hat{p}_{\text{data}}$, defined by the training set and the model distribution, with the degree of dissimilarity between the two measure by the **KL divergence**

$$D_{\text{KL}}\left(\hat{p}_{\text{data}} \| p_{\text{model}}\right) = \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}} \left[\log \hat{p}_{\text{data}}\left(\mathbf{x}\right) - \log p_{\text{model}}\left(\mathbf{x} \mid \boldsymbol{\theta}\right)\right]$$

  The term on the left is a function only of the data-generating process, not the model. This means when we train the model to minimize the KL divergence, we need only minimize

$$-\mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}} \left[\log p_{\text{model}}\left(\mathbf{x} \mid \boldsymbol{\theta}\right)\right]$$

- Minimizing this KL divergence corresponds exactly to minimizing the cross-entropy between the distributions. By definition, any loss consisting a negative log-likelihood is a **cross-entropy** between the **empirical distribution** defined by the training set ($\hat{p}_{\text{data}}$), and the **probability distribution** defined by the model (p_{model})

1.4 Conditional Log-Likelihood

- To apply *MLE* to most **supervised learning** tasks of predicting $\mathbf{y}$ given $\mathbf{x}$, the maximum likelihood estimator is generalized to estimate a conditional probability $p_{\text{model}}(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta})$
- Consider a set of m examples $\mathcal{D} = \left\{ \left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \left(\mathbf{x}^{(2)}, \mathbf{y}^{(2)}\right), \cdots, \left(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}\right) \right\}$ drawn independently from the true but unknown data-generating distribution $p_{\text{data}}(\mathbf{x}, \mathbf{y})$. Factorize the data-generating process

$$p_{\text{data}}(\mathbf{x}, \mathbf{y}) = p_{\text{data}}(\mathbf{y} \mid \mathbf{x}) \, p_{\text{data}}(\mathbf{x})$$

Let $p_{\text{model}}(\mathbf{x}, \mathbf{y} \mid \boldsymbol{\theta})$ be a parametric family of probability distributions over the same space indexed by $\boldsymbol{\theta}$. It also can be factorized

$$p_{\text{model}}(\mathbf{x}, \mathbf{y} \mid \boldsymbol{\theta}) = p_{\text{model}}(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}) \, p_{\text{data}}(\mathbf{x})$$

Notice that the later part $p_{\text{data}}(\mathbf{x})$ is **fixed and shared**, the maximum likelihood estimation is going to focus on

$$p_{\text{model}}(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta})$$

Under the **i.i.d.** assumption, it can be decomposed into

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\text{model}}\left(\mathbf{y}^{(i)} \,\middle|\, \mathbf{x}^{(i)}, \boldsymbol{\theta}\right)$$

Similarly, this optimization problem is usually converted into a minimization problem by the **negative logarithm** operation considering computation issues

$$\boldsymbol{\theta}_{\text{ML}} = \arg\min_{\boldsymbol{\theta}} \left[-\sum_{i=1}^{m} \log \left[p_{\text{model}}\left(\mathbf{y}^{(i)} \,\middle|\, \mathbf{x}^{(i)}, \boldsymbol{\theta}\right) \right] \right]$$

1.5 Least Squares as Maximum Likelihood

- Least squares minimizing the mean square error is **equal** to maximum likelihood estimation when the likelihood is assigned to be **Gaussian**
- Assume the model is $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$ with the dataset

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \cdots & \mathbf{x}^{(m)} \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$$

The solution for $\boldsymbol{\theta}$ via least squares would be

$$\boldsymbol{\theta}_{\text{LS}} = \arg\min_{\boldsymbol{\theta}} \left\| \mathbf{y} - f(\mathbf{X}; \boldsymbol{\theta}) \right\|_2^2$$

- From the point of view of maximum likelihood estimation, think of the model as producing a conditional distribution $p_{\text{model}}(y \mid \mathbf{x}, \boldsymbol{\theta})$ instead of producing a single prediction $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$. Assign this likelihood of a single measurement to be Gaussian

$$p_{\text{model}}\left(y^{(i)} \,\middle|\, \mathbf{x}^{(i)}, \boldsymbol{\theta}, \sigma\right) = \mathcal{N}\left(y^{(i)} \,\middle|\, f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \sigma^2\right)$$

where σ models the **noise**. This correspond to the belief that the measurement is around the model prediction $f(\mathbf{x}; \boldsymbol{\theta})$ but it is contained with Gaussian noise of variance σ^2. For all the data, we have

$$\begin{aligned} p_{\text{model}}(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}, \sigma) &= \mathcal{N}\left(\mathbf{y} \,\middle|\, f(\mathbf{X}; \boldsymbol{\theta}), \sigma^2 \mathbf{I}_m\right) \\ &= (2\pi)^{-m/2} \sigma^{-m} \exp\left(-\frac{1}{2\sigma^2} \left\| \mathbf{y} - f(\mathbf{X}; \boldsymbol{\theta}) \right\|_2^2 \right) \end{aligned}$$

Then we have the maximum likelihood estimation to be

$$\begin{aligned}\boldsymbol{\theta}_{\mathrm{ML}} &= \arg\min_{\boldsymbol{\theta}} \left[-\log\left[p_{\mathrm{model}}\left(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}, \sigma\right)\right]\right] \\ &= \arg\min_{\boldsymbol{\theta}} \left[\frac{m}{2}\log\left(2\pi\right) + m\log(\sigma) + \frac{1}{2\sigma^2}\left\|\mathbf{y} - f\left(\mathbf{X}; \boldsymbol{\theta}\right)\right\|_2^2\right] \\ &= \arg\min_{\boldsymbol{\theta}} \left\|\mathbf{y} - f\left(\mathbf{X}; \boldsymbol{\theta}\right)\right\|_2^2 \\ &= \boldsymbol{\theta}_{\mathrm{LS}}\end{aligned}$$

Maximizing the likelihood with respect to $\boldsymbol{\theta}$ yields the same estimate as minimizing the squared error.

- The two criteria have **different values** but the **same location of the optimum**, which justifies the use of the LS as a maximum likelihood estimation procedure.
- Notice that σ is also a parameter to be optimized, maximize the likelihood with respect to σ

$$\begin{aligned}\sigma_{\mathrm{ML}} &= \arg\min_{\sigma} \left[-\log\left[p_{\mathrm{model}}\left(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}, \sigma\right)\right]\right] \\ &= \arg\min_{\sigma} \left[\frac{m}{2}\log\left(2\pi\right) + m\log(\sigma) + \frac{1}{2\sigma^2}\left\|\mathbf{y} - f\left(\mathbf{X}; \boldsymbol{\theta}\right)\right\|_2^2\right] \\ &= \arg\min_{\sigma} \left[m\log(\sigma) + \frac{1}{2\sigma^2}\left\|\mathbf{y} - f\left(\mathbf{X}; \boldsymbol{\theta}\right)\right\|_2^2\right]\end{aligned}$$

It can be easily solved by setting the derivative with respect to σ to zero

$$\begin{aligned} m\frac{1}{\sigma_{\mathrm{ML}}} - \frac{1}{\sigma_{\mathrm{ML}}^3}\left\|\mathbf{y} - f\left(\mathbf{X}; \boldsymbol{\theta}\right)\right\|_2^2 &= 0 \\ m\sigma_{\mathrm{ML}}^2 - \left\|\mathbf{y} - f\left(\mathbf{X}; \boldsymbol{\theta}\right)\right\|_2^2 &= 0 \\ \sigma_{\mathrm{ML}}^2 &= \frac{1}{m}\left\|\mathbf{y} - f\left(\mathbf{X}; \boldsymbol{\theta}\right)\right\|_2^2\end{aligned}$$

- With the maximum likelihood estimation $\boldsymbol{\theta}_{\mathrm{ML}}, \sigma_{\mathrm{ML}}$, we can make **predictions** about y at a new point $\mathbf{x}$

$$p\left(y \,|\mathbf{x}, \boldsymbol{\theta}_{\mathrm{ML}}, \sigma_{\mathrm{ML}}\right) = \mathcal{N}\left(y \,\middle|\, f(\mathbf{x}; \boldsymbol{\theta}_{\mathrm{ML}}), \sigma_{\mathrm{ML}}^2\right)$$

This means we're able to measure the **noise** occurred in sampling.

1.6 Properties of Maximum Likelihood Estimation

- Maximum likelihood estimator can be shown to be the **best** estimator asymptotically as the number of examples $m \to \infty$, in terms of its rate of convergence as m increases
- Whenever the **cross-entropy loss** or **least squares method** (e.g. MSE loss) is used, it can be viewed as computing the maximum likelihood estimate.
- Under appropriate conditions, the maximum likelihood estimator has the property of **consistency**
  - *Consistency*: **Unbiased**, converge to the true parameters as $m \to \infty$
  - The true distribution p_{data} must lie within the model family p_{model}
  - The true distribution p_{data} must correspond to exactly one value of $\boldsymbol{\theta}$

- The **statistical efficiency**, meaning that one consistent estimator may obtain lower generalization error for a fixed number of samples m, of the maximum likelihood estimator is very high among consistent estimators
  - It is asymptotically efficient, which means it achieves the Cramer Rao bound asymptotically
- It's mostly used when there is plenty of data
  - Tends to overfit when there is not enough data

2 Linear Regression via Maximum Likelihood Estimation

Here is a simple example showing how to apply **Maximum Likelihood Estimation** to linear regression and the correspondence between likelihood and least squares

2.1 Generate the dataset

The synthetic dataset is generated from

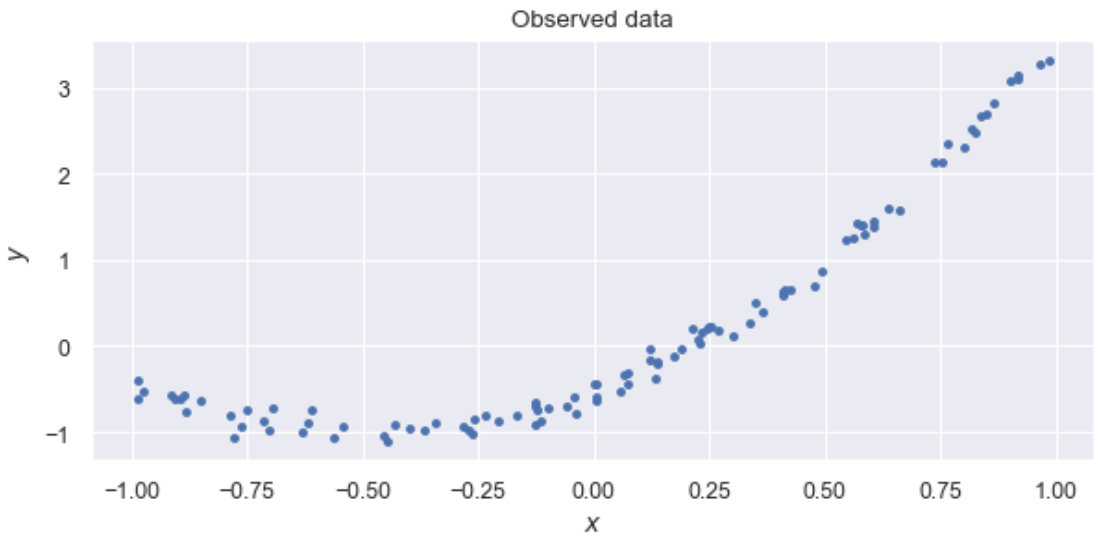
$$y_i = -0.5 + 2x_i + 2x_i^2 + 0.1\epsilon_i$$

where $\epsilon_i \sim \mathcal{N}(0, 1)$ and we sample $x_i \sim U([0, 1])$.

First generate this synthetic dataset and visualize the samples.

[1]:
```
# Necessary modules
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# Ensure reproducibility
np.random.seed(1234)
# Plot setting
sns.set()
sns.set_context('paper')
```

[2]:
```
# Number of observations
num_obs = 100
# Sample x
x = (-1.0 + 2 * np.random.rand(num_obs)).reshape(-1, 1)
# True parameters
theta = np.array([-0.5, 2.0, 2.0]).reshape(-1, 1)
sigma = 0.1
# Calculate the corresponding y
y = theta[0] + theta[1] * x + theta[2] * x**2 \
    + sigma * np.random.randn(num_obs).reshape(-1, 1)
# Visualize the dataset
fig, ax = plt.subplots(figsize=(7, 3), dpi=100)
ax.plot(x, y, '.', markersize=5)
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_title('Observed data')
plt.show()
```



2.2 Build the model and train

As mentioned above, with setting the likelihood to be Gaussian, linear regression via least squares is a part of the procedure of maximum likelihood estimation.

$$\boldsymbol{\theta}_{\mathrm{ML}} = \boldsymbol{\theta}_{\mathrm{LS}}$$

Therefore, first get the values of the parameters via least squares. For convenience, here the solutions are got with the help of **Scikit-Learn**.

In this case, assume we know the exact degree of the polynomial so that we're not going to worry about validation and generalization issues, which makes us focusing on the MLE part. In the meantime, the dataset is not divided into training and validation dataset.

[3]:
```
# Scikit-Learn Packages
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
# 'include_bias' and 'fir_intercept' cannot be both true
# Select one to be true according to personal preferrence
estimator = make_pipeline(PolynomialFeatures(2,include_bias=True),\
LinearRegression(fit_intercept=False))
# Train the model
estimator.fit(x, y)
# Compare the values
print("The true parameters are:\t", theta.T[0])
print("The estimated parameters are:\t", estimator[1].coef_[0])
```

```
The true parameters are:         [-0.5  2.   2. ]
The estimated parameters are:    [-0.52407683  1.99641392  2.08372088]
```

2.3 Estimate the noise variance

In maximum likelihood estimation, one important thing is to also estimate the variance by maximizing the likelihood. Mention that the likelihood is set to be Gaussian, the variance should be

$$\sigma_{\mathrm{ML}}^2 = \frac{1}{m} \left\| \mathbf{y} - f\left(\mathbf{X}; \boldsymbol{\theta}\right) \right\|_2^2$$

[4]:
```
# Get the predictions of the trained model
y_pred = estimator.predict(x).reshape(-1, 1)
# Calculate the estimated variance
sigma2_MLE = np.sum((y_pred - y)**2) / y.shape[0]
sigma_MLE = np.sqrt(sigma2_MLE)
# Compare the  values
print("True sigma = %1.4f" % sigma)
print("MLE  sigma = %1.4f" % sigma_MLE)
```

```
True sigma = 0.1000
MLE  sigma = 0.0928
```

2.4 Make predictions

Now we have the maximum likelihood estimation $\boldsymbol{\theta}_{\mathrm{ML}}, \sigma_{\mathrm{ML}}$, we can make **predictions** about y at a new point $\mathbf{x}$

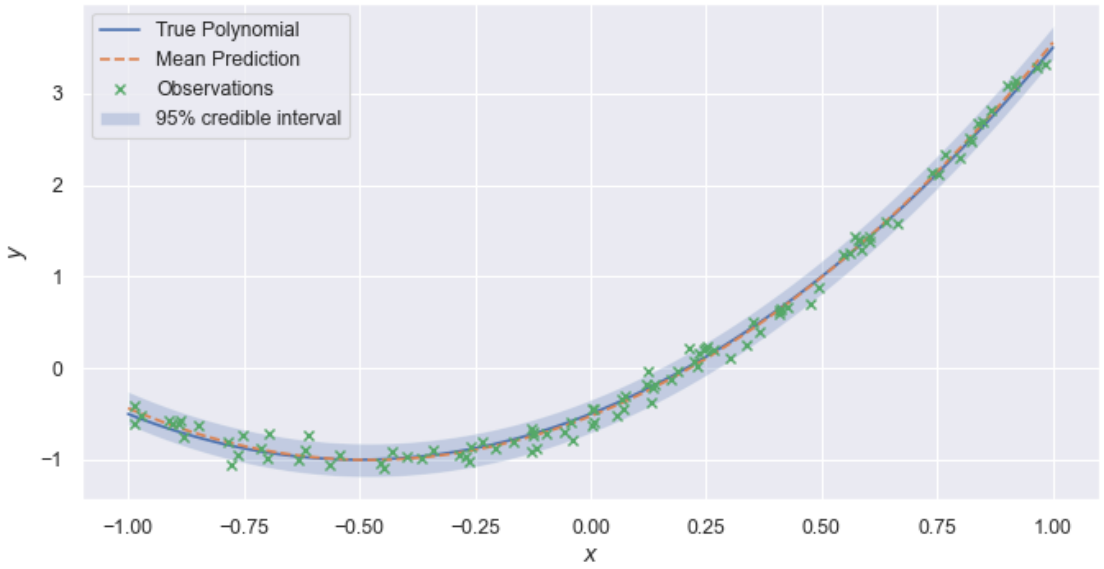
$$p\left(y \left| \mathbf{x}, \boldsymbol{\theta}_{\mathrm{ML}}, \sigma_{\mathrm{ML}} \right.\right) = \mathcal{N}\left(y \left| f(\mathbf{x}; \boldsymbol{\theta}_{\mathrm{ML}}), \sigma_{\mathrm{ML}}^2 \right.\right)$$

Let's visualize the 95% credible interval

$$(f(\mathbf{x}; \boldsymbol{\theta}_{\mathrm{ML}}) - 1.96\sigma_{\mathrm{ML}}, f(\mathbf{x}; \boldsymbol{\theta}_{\mathrm{ML}}) + 1.96\sigma_{\mathrm{ML}})$$

[5]:
```
fig, ax = plt.subplots(figsize=(8, 4), dpi=100)
# Points to be estimated
xe = np.linspace(-1, 1, 100)
# Predictions and true values
ye_pred = estimator.predict(xe.reshape(-1, 1)).reshape(-1)
ye_true = theta[0] + theta[1] * xe + theta[2] * xe**2
# Lower bound and upper bound
ye_pred_lb = ye_pred - 1.96 * sigma_MLE
ye_pred_ub = ye_pred + 1.96 * sigma_MLE
# True polynomial
ax.plot(xe, ye_true, label='True Polynomial')
# Mean prediction
ax.plot(xe, ye_pred, '--', label='Mean Prediction')
# 95% credible interval
ax.fill_between(xe, ye_pred_lb, ye_pred_ub,\
                alpha=0.25, label='95% credible interval')
# Observations
ax.plot(x, y, 'x', label='Observations')
# Labels
ax.set_xlabel('$x$')
```

```
ax.set_ylabel('$y$')
plt.legend(loc='upper left')
plt.show()
```



3 Bayesian methods

3.1 Frequentist approach

- Review: In frequentist view, when we say that an outcome has a probability p of occurring, it means that if we repeated the experiment infinite times, then a proportion p of the repetitions would result in that outcome
- The true parameter value $\boldsymbol{\theta}$ is supposed to be **fixed but unknown**
- The point estimate $\hat{\boldsymbol{\theta}}$ is a random variable on account of it being a function of the dataset, which is seen as random
- Predictions are made based on estimating a single value of $\boldsymbol{\theta}$
- Addresses the uncertainty in a given point estimate of $\boldsymbol{\theta}$ by evaluating its variance
  - The variance is an assessment of how the estimate might change with alternative samplings of the observed data
- **Maximum Likelihood Estimation** is a Frequentist approach

3.2 Bayesian Estimation

- Review: Bayesian uses the probability to reflect **degrees of certainty** in states of knowledge
- The true parameter $\boldsymbol{\theta}$ is unknown or uncertain and thus is represented as a **random variable**. Before observing the data, represent our knowledge of $\boldsymbol{\theta}$ using the **prior probability distribution** $p(\boldsymbol{\theta})$
  - Typically selects prior distribution that is quite broad (e.g. maximum entropy) to reflect a high degree of uncertainty in the value of $\boldsymbol{\theta}$ before observing any data
- The dataset is directly observed and so is not random. Consider a set of data examples $\left\{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\right\}$, we can recover the effect of data on our belief about $\boldsymbol{\theta}$ via Bayes' rule

$$p\left(\boldsymbol{\theta} \,\middle|\, \mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\right) = \frac{p\left(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)} \mid \boldsymbol{\theta}\right) p(\boldsymbol{\theta})}{p\left(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\right)}$$

  In the scenarios where Bayesian estimation is typically used, the prior begins as a relatively uniform or Gaussian distribution with high entropy, and the observation of the data usually causes the posterior to lose entropy and concentrate around a few highly likely values of the parameters

- Bayesian approach makes predictions using a full distance over $\boldsymbol{\theta}$. After observing m examples, the predicted distribution over the next data sample $\mathbf{x}^{(m+1)}$ is given by

$$p\left(\mathbf{x}^{(m+1)} \,\middle|\, \mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\right) = \int p\left(\mathbf{x}^{(m+1)} \mid \boldsymbol{\theta}\right) p\left(\boldsymbol{\theta} \,\middle|\, \mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\right)$$

  Each value of $\boldsymbol{\theta}$ with positive probability density contributes to the prediction of the next example, with the contribution weighted by the posterior density itself.
  - Enable us to quantify the **epistemic uncertainty** induced by the limited number of observations used to estimate the weights

- The Bayesian prior distribution has an influence by shifting probability mass density towards regions of the parameter space that are preferred a priori.
  - In practice, the prior often expresses a **preference for models that are simpler** or more smooth.
  - This naturally arose **regularization** effect tends to protect well against overfitting.
- Bayesian methods typically generalize much better when limited training data is available but typically suffer from high computational cost when the number of training examples is large.
- Similarly, for **supervised learning** tasks, use the **conditional likelihood** instead.

3.3 Bayesian vs Frequentist Estimation

3.3.1 Frequentist Estimation

- Usually used for parameter estimation
- Typically low-bias high-variance estimates
- Most appropriate when the
  - prior information is weak
  - amount of data is large
  - quality of data is high

3.3.2 Bayesian Estimation

- Usually used for Machine Learning inference
- Typically high-bias low-variance estimates
- Most appropriate when the
  - prior information is strong
  - amount of data is small
  - quality of data is poor

3.4 Maximum A Posterior Estimation

- While most operations involving the Bayesian posterior for most interesting models are **intractable**, a point estimate offers **a tractable approximation**
- **Maximum A Posterior** point estimate benefits from the Bayesian approach by **allowing the prior to influence the choice of the point estimate** rather than simply returning to the maximum likelihood estimate.
- The **MAP** estimate chooses the point of maximal posterior probability (or maximal probability density in the more common case of continuous $\boldsymbol{\theta}$)

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p\left(\boldsymbol{\theta} \,\middle|\, \mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\right) = \arg\max_{\boldsymbol{\theta}} \left\{ \log p\left(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)} \mid \boldsymbol{\theta}\right) + \log p(\boldsymbol{\theta}) \right\}$$

On the righthand side, $p\left(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)} \mid \boldsymbol{\theta}\right)$ is the standard likelihood term and $p(\boldsymbol{\theta})$ is the prior distribution

  - Reduce the bias at the price of increased bias (in comparison to the ML estimate)

- Many **regularized estimation strategies** can be interpreted as making the MAP approximation to Bayesian inference. This view applies when the regularization consists of **adding an extra term** to the objective function that corresponds to $p(\boldsymbol{\theta})$
  - Maximum likelihood learning regularized with weight decay

4 Linear Regression via Bayesian Estimation

4.1 Mathematical Representation

4.1.1 Linear Model

Assume the model is $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$ with the dataset

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \cdots & \mathbf{x}^{(m)} \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$$

In the case of linear regression, some **fixed** features are used to model the data

$$\hat{y} = \sum_{j=1}^{n} \theta_j \phi_j(\mathbf{x}) = \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})$$

4.1.2 Likelihood

Similar to what is did in MLE, assign the **likelihood** of a single measurement to be Gaussian, where σ models the **measurement noise**

$$p\left(y^{(i)} \,\middle|\, \mathbf{x}^{(i)}, \boldsymbol{\theta}, \sigma\right) = \mathcal{N}\left(y^{(i)} \,\middle|\, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}^{(i)}), \sigma^2\right)$$

Given that the measurements are made independently, for all the data

$$p\left(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}, \sigma\right) = \mathcal{N}\left(\mathbf{y} \,\middle|\, \boldsymbol{\phi}(\mathbf{X})\boldsymbol{\theta}, \sigma^2 \mathbf{I}_m\right) = (2\pi)^{-m/2} \sigma^{-m} \exp\left(-\frac{1}{2\sigma^2} \left\|\mathbf{y} - \boldsymbol{\phi}(\mathbf{X})\boldsymbol{\theta}\right\|_2^2\right)$$

4.1.3 Prior

In Bayesian methods, the **uncertainty in the model parameters** is modeled using a **prior**

$$\boldsymbol{\theta} \sim p(\boldsymbol{\theta})$$

The **simplest** possible choice for the parameters are the **zero mean Gaussian prior** with a **precision** of α

$$p(\boldsymbol{\theta}|\alpha) = \mathcal{N}\left(\boldsymbol{\theta} \,\middle|\, \mathbf{0}, \alpha^{-1}\mathbf{I}_n\right) = (2\pi)^{-n/2} \alpha^{n/2} \exp\left(-\frac{\alpha}{2} \|\boldsymbol{\theta}\|_2^2\right)$$

This shows the **belief** that $\boldsymbol{\theta}$ must be **around zero** with a precision of α before seeing any data. The bigger the precision parameter α is, the more the parameters $\boldsymbol{\theta}$ are pushed towards zero, which can be viewed as a kind of regularization.

4.1.4 Posterior

Combining the likelihood and the prior, get the **posterior** using Bayes' rule

$$p\left(\boldsymbol{\theta} \mid \mathbf{X}, \mathbf{y}, \sigma, \alpha\right) = \frac{p\left(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}, \sigma\right) p(\boldsymbol{\theta}|\alpha)}{\int p\left(\mathbf{y} \,\middle|\, \mathbf{X}, \boldsymbol{\theta}', \sigma\right) p(\boldsymbol{\theta}'|\alpha) d\boldsymbol{\theta}'}$$

This posterior stands for the **states of knowledge about $\boldsymbol{\theta}$ after seeing the data**, if α and σ are known. Given that the likelihood and prior are both Gaussian, the posterior is also Gaussian

$$p\left(\boldsymbol{\theta} \mid \mathbf{X}, \mathbf{y}, \sigma, \alpha\right) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{m}, \mathbf{S})$$

where

$$\mathbf{S} = \left(\sigma^{-2} \boldsymbol{\phi}\left(\mathbf{X}\right)^T \boldsymbol{\phi}\left(\mathbf{X}\right) + \alpha \mathbf{I}_n\right)^{-1}, \ \mathbf{m} = \sigma^{-2} \mathbf{S} \boldsymbol{\phi}\left(\mathbf{X}\right)^T \mathbf{y}$$

In the general case of non-Gaussian likelihood and non-linear models, the posterior will **not be analytically available**.

4.1.5 Posterior Predictive Distribution

With the posterior, the **predictive distribution** for y at a new $\mathbf{x}$ is available by using the **sum rule**

$$p\left(y \mid \mathbf{x}, \mathbf{X}, \mathbf{y}, \sigma, \alpha\right) = \int p\left(y \mid \mathbf{x}, \boldsymbol{\theta}, \sigma\right) p\left(\boldsymbol{\theta} \mid \mathbf{X}, \mathbf{y}, \sigma, \alpha\right) d\boldsymbol{\theta}$$

For the all-Gaussian case, the posterior predictive distribution is analytically available

$$p\left(y \mid \mathbf{x}, \mathbf{X}, \mathbf{y}, \sigma, \alpha\right) = \mathcal{N}\left(y \,\middle|\, m(\mathbf{x}), s^2(\mathbf{x})\right)$$

where

$$m(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})\mathbf{m} \text{ and } s^2(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})\mathbf{S}\boldsymbol{\phi}(\mathbf{x})^T + \sigma^2$$

The predictive uncertainty here is

$$s^2(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})\mathbf{S}\boldsymbol{\phi}(\mathbf{x})^T + \sigma^2$$

where σ^2 corresponds to the **measurement noise** and $\boldsymbol{\phi}(\mathbf{x})\mathbf{S}\boldsymbol{\phi}(\mathbf{x})^T$ is the **epistemic uncertainty** induced by limited data

4.2 Generate the dataset

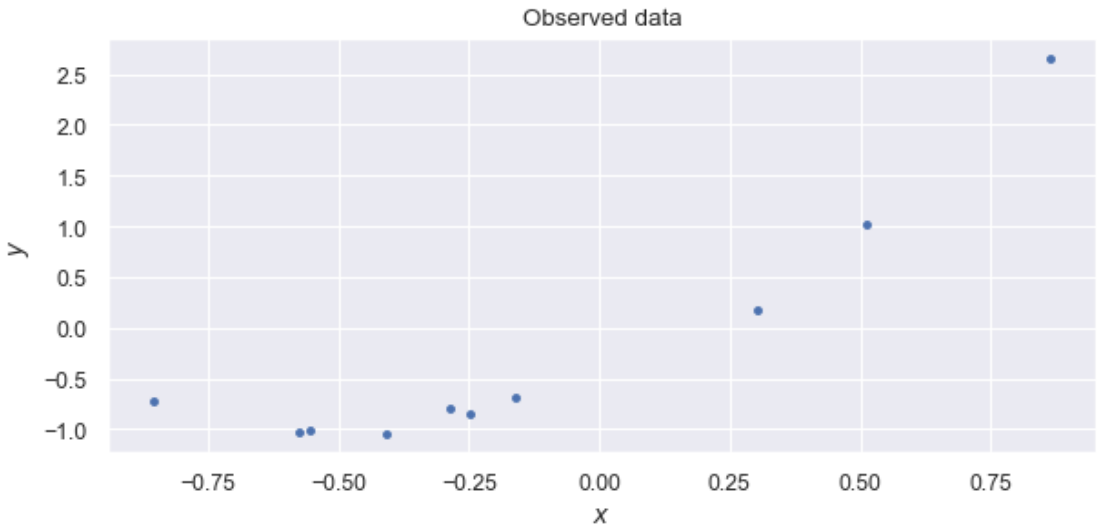
The synthetic dataset is still generated from

$$y_i = -0.5 + 2x_i + 2x_i^2 + 0.1\epsilon_i$$

where $\epsilon_i \sim \mathcal{N}(0, 1)$ and we sample $x_i \sim U([0, 1])$.

First generate this synthetic dataset and visualize the samples. For better observations of the **point-predictive distribution**, fewer observations are made.

[6]:
```
# Number of observations
num_obs = 10
# Sample x
x = (-1.0 + 2 * np.random.rand(num_obs)).reshape(-1, 1)
# True parameters
theta = np.array([-0.5, 2.0, 2.0]).reshape(-1, 1)
sigma = 0.1
# Calculate the corresponding y
y = theta[0] + theta[1] * x + theta[2] * x**2 \
    + sigma * np.random.randn(num_obs).reshape(-1, 1)
# Visualize the dataset
fig, ax = plt.subplots(figsize=(7, 3), dpi=100)
ax.plot(x, y, '.', markersize=5)
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_title('Observed data')
plt.show()
```



4.3 Get the posterior distribution

In the case of Gaussian likelihood and weight prior, the posterior of the weights is also Gaussian

$$p\left(\boldsymbol{\theta} \mid \mathbf{X}, \mathbf{y}, \sigma, \alpha\right) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{m}, \mathbf{S})$$

where

$$\mathbf{S} = \left(\sigma^{-2}\boldsymbol{\phi}\left(\mathbf{X}\right)^T \boldsymbol{\phi}\left(\mathbf{X}\right) + \alpha\mathbf{I}_n\right)^{-1}, \ \mathbf{m} = \sigma^{-2}\mathbf{S}\boldsymbol{\phi}\left(\mathbf{X}\right)^T \mathbf{y}$$

To get the posterior weight mean vector $\mathbf{m}$ and the posterior weight covariance matrix $\mathbf{S}$, the first step is to get the polynomial design matrices $\boldsymbol{\phi}(\mathbf{X})$. Assume we do not know the exact degree of the polynomial and we try to use **a 5 degree polynomial** to fit the data.

Here the attributes in **`Scikit-Learn`** are used for convenience.

[7]:
```
# Create the instance for calculating polynomial features
degree = 5
poly = PolynomialFeatures(degree, include_bias=True)
# Call the method of the instance
Phi = poly.fit_transform(x)
# Check the shape of the design matrix
print('The shape of the design matrix is', Phi.shape)
```

```
The shape of the design matrix is (10, 6)
```

Calculate the posterior distribution with the equations derived before. For stable solutions, the methods `scipy.linalg.cho_factor()` and `scipy.linalg.cho_solve()` are used.

- Construct the positive-definite matrix

$$\mathbf{A} = \sigma^{-2}\boldsymbol{\phi}\left(\mathbf{X}\right)^T \boldsymbol{\phi}\left(\mathbf{X}\right) + \alpha\mathbf{I}_n$$

- Use **`scipy.linalg.cho_factor()`** to compute the Cholesky decomposition of $\mathbf{A}$

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T$$

  where $\mathbf{L}$ is lower triangular
- Then use **`scipy.linalg.cho_solve()`** to solve the following equation to get $\mathbf{m}$

$$\mathbf{L}\mathbf{L}^T\mathbf{m} = \sigma^{-2}\boldsymbol{\phi}\left(\mathbf{X}\right)^T\mathbf{y}$$

- Similarly, use **`scipy.linalg.cho_solve()`** to solve the following equation to get $\mathbf{S}$

$$\mathbf{L}\mathbf{L}^T\mathbf{S} = \mathbf{I}$$

[8]:
```
import scipy
import scipy.stats as st

# Pick variance by hand. Here the true one is used
sigma = 0.1
# Pick the regularization paramter by hand
alpha = 5.0
# The prior for weights
theta_prior = st.multivariate_normal(mean=np.zeros(degree + 1),\
                                     cov=alpha * np.eye(degree + 1))
# Construct the positive-definite matrix
A = np.dot(Phi.T, Phi) / sigma**2 + alpha * np.eye(Phi.shape[1])
# Get the Cholesky decomposition of A
L = scipy.linalg.cho_factor(A)
# Solve for posterior weight mean vector
m = scipy.linalg.cho_solve(L, np.dot(Phi.T, y / sigma**2))
print("The posterior weight mean vector is:\n", m)
# Solve for posterior weight covariance matrix
S = scipy.linalg.cho_solve(L, np.eye(Phi.shape[1]))
print("The posterior weight covariance matrix is:\n", S)
# The posterior of the weights as a distribution
m = m.reshape(m.shape[0])
theta_post = st.multivariate_normal(mean=m, cov=S)
```

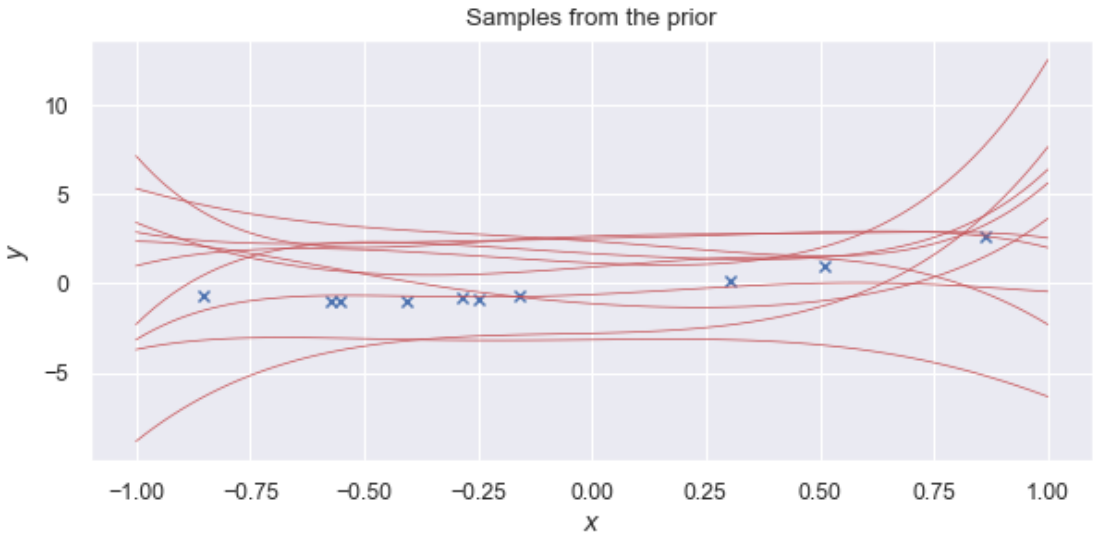
```
The posterior weight mean vector is:
 [[-0.41779846]
 [ 1.77955225]
 [ 1.15843691]
 [ 0.27039672]
 [ 0.88570988]
 [-0.0445176 ]]
The posterior weight covariance matrix is:
 [[ 0.00279796  0.00190825 -0.00881343 -0.00235031  0.00605631 -0.0003641 ]
 [ 0.00190825  0.01574982 -0.00038094 -0.0253839  -0.00213548  0.00370317]
 [-0.00881343 -0.00038094  0.0684496   0.00558418 -0.07259876 -0.00476975]
 [-0.00235031 -0.0253839   0.00558418  0.10218228 -0.00265889 -0.08216203]
 [ 0.00605631 -0.00213548 -0.07259876 -0.00265889  0.09859821  0.00437897]
 [-0.0003641   0.00370317 -0.00476975 -0.08216203  0.00437897  0.1158465 ]]
```

Thus we get the posterior distribution of parameters.

4.4 Some samples from the prior and the posterior

Let's visualize some samples from the prior:

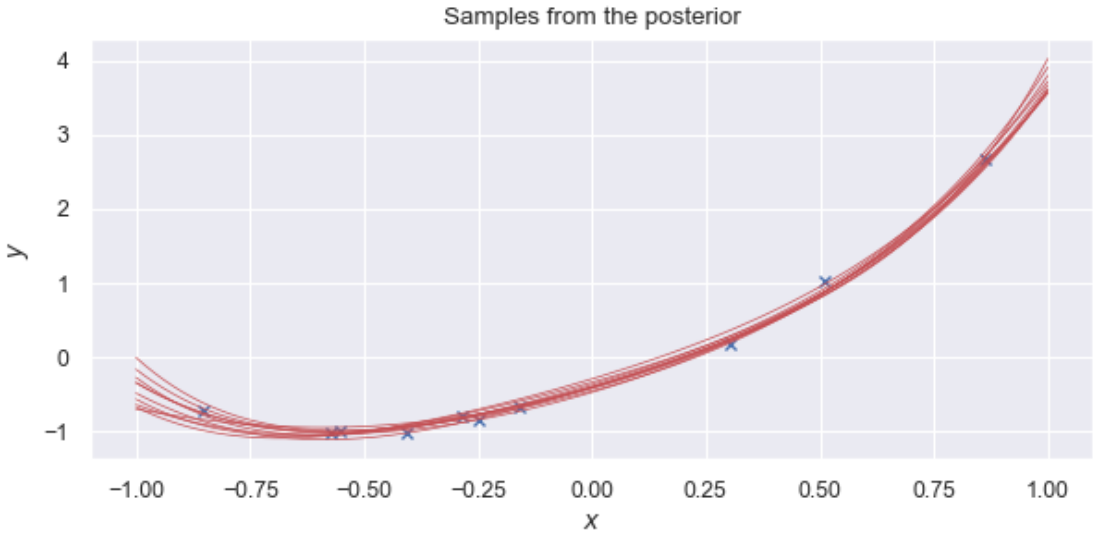
[9]:
```
fig, ax = plt.subplots(figsize=(7, 3), dpi=100)
# Some points on which to evaluate the regression function
xe = np.linspace(-1, 1, 100).reshape(-1, 1)
Phi_xe = poly.fit_transform(xe)
# Plot the observations
ax.plot(x, y, 'x')
# Sample from the prior
for ii in range(10):
    theta_sample = theta_prior.rvs()
    ye_sample = np.dot(Phi_xe, theta_sample)
    ax.plot(xe, ye_sample, 'r', lw=0.5)
# Set the axis labels and title
ax.set_title("Samples from the prior")
ax.set_xlabel("$x$")
ax.set_ylabel("$y$")
plt.show()
```



Then visualize the samples from the posterior:

[10]:
```
fig, ax = plt.subplots(figsize=(7, 3), dpi=100)
# Some points on which to evaluate the regression function
xe = np.linspace(-1, 1, 100).reshape(-1, 1)
Phi_xe = poly.fit_transform(xe)
```

```
# Plot the observations
ax.plot(x, y, 'x')
# Sample from the prior
for ii in range(10):
    theta_sample = theta_post.rvs()
    ye_sample = np.dot(Phi_xe, theta_sample)
    ax.plot(xe, ye_sample, 'r', lw=0.5)
# Set the axis labels and title
ax.set_title("Samples from the posterior")
ax.set_xlabel("$x$")
ax.set_ylabel("$y$")
plt.show()
```



4.5 Posterior predictive distribution

As mentioned in *section 4.1.1: Mathematical Representation*, for the all-Gaussian case, the posterior predictive distribution is analytically available

$$p\left(y \mid \mathbf{x}, \mathbf{X}, \mathbf{y}, \sigma, \alpha\right) = \mathcal{N}\left(y \mid m(\mathbf{x}), s^2(\mathbf{x})\right)$$

where

$$m(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})\mathbf{m} \text{ and } s^2(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})\mathbf{S}\boldsymbol{\phi}(\mathbf{x})^T + \sigma^2$$

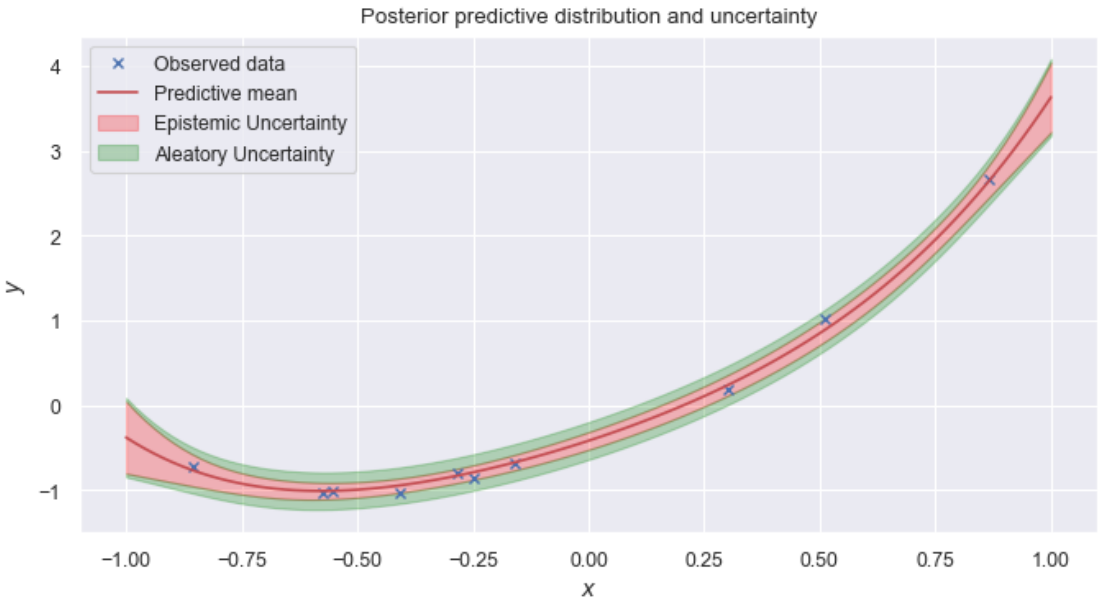
The predictive uncertainty here is

$$s^2(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})\mathbf{S}\boldsymbol{\phi}(\mathbf{x})^T + \sigma^2$$

where σ^2 corresponds to the **measurement noise** and $\boldsymbol{\phi}(\mathbf{x})\mathbf{S}\boldsymbol{\phi}(\mathbf{x})^T$ is the **epistemic uncertainty** induced by limited data.

Let's visualize the posterior predictive distribution and the predictive uncertainty. Notice that a diagonalization is used to the matrix $\boldsymbol{\phi}(\mathbf{x})\mathbf{S}\boldsymbol{\phi}(\mathbf{x})^T$.

[11]:
```
fig, ax = plt.subplots(figsize=(8, 4), dpi=100)
# Some points on which to evaluate the regression function
xe = np.linspace(-1, 1, 100)
Phi_xe = poly.fit_transform(xe.reshape(-1, 1))
# The posterior predictive mean
ye_mean = np.dot(Phi_xe, m)
# The epistemic uncertainty
ye_var_epi = np.diag(np.dot(np.dot(Phi_xe, S), Phi_xe.T))
# The total predictive uncertainty
ye_var_total = ye_var_epi + sigma**2
# 95% credible interval: Epistemic lower bound and upper bound
ye_lb_epi = ye_mean - 1.96 * np.sqrt(ye_var_epi)
ye_ub_epi = ye_mean + 1.96 * np.sqrt(ye_var_epi)
# 95% credible interval: Total uncertainty lower bound and upper bound
ye_lb_total = ye_mean - 1.96 * np.sqrt(ye_var_total)
ye_ub_total = ye_mean + 1.96 * np.sqrt(ye_var_total)
# Plot the observations
ax.plot(x, y, 'x', label="Observed data")
# Plot the prediction mean
ax.plot(xe, ye_mean, 'r', label="Predictive mean")
# Plot the epistemic uncertainty induced by limited data
ax.fill_between(xe,ye_lb_epi,ye_ub_epi,\
                color="red",alpha=0.25,\
                label="Epistemic Uncertainty")
# Plot the aleatory uncertainty induced by measurement noise
ax.fill_between(xe, ye_lb_total, ye_lb_epi,\
                color="green", alpha=0.25,\
                label="Aleatory Uncertainty")
ax.fill_between(xe, ye_ub_epi, ye_ub_total, color="green", alpha=0.25)
# Set the legend, axis labels and title
ax.set_title("Posterior predictive distribution and uncertainty")
ax.set_xlabel("$x$")
ax.set_ylabel("$y$")
plt.legend(loc="upper left")
plt.show()
```


Posterior predictive distribution and uncertainty
Observed data
Predictive mean
Epistemic Uncertainty
Aleatory Uncertainty
y
x
4
3
2
1
0
−1
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

5 Linear Regression via Maximum A Posteriori Estimation

5.1 Mathematical Representation

The steps towards obtaining the posterior is the same as those in Bayesian linear regression.

$$p\left(\boldsymbol{\theta} \mid \mathbf{X}, \mathbf{y}, \sigma, \alpha\right) = \frac{p\left(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}, \sigma\right) p(\boldsymbol{\theta}|\alpha)}{\int p\left(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}', \sigma\right) p(\boldsymbol{\theta}'|\alpha) d\boldsymbol{\theta}'}$$

Then we find a point estimate of $\boldsymbol{\theta}$ by solving

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\mathbf{y}|\ \mathbf{X}, \boldsymbol{\theta}, \sigma) p(\boldsymbol{\theta}|\ \alpha)$$

For Gaussian likelihood and zero-mean Gaussian weight prior, the logarithm of the posterior is:

$$\log p(\boldsymbol{\theta}|\ \mathbf{X}, \mathbf{y}, \sigma, \alpha) = -\frac{1}{2\sigma^2} \left\|\boldsymbol{\phi}\left(\mathbf{X}\right) \boldsymbol{\theta} - \mathbf{y}\right\|^2 - \frac{\alpha}{2} \|\boldsymbol{\theta}\|^2$$

Taking derivatives with respect to $\boldsymbol{\theta}$ and setting them equal to zero (necessary condition):

$$\boldsymbol{\theta}_{\text{MAP}} = \mathbf{m}$$

The point estimate is exactly the same as the weight mean vector of the posterior distribution of parameters:

$$\mathbf{S} = \left(\sigma^{-2} \boldsymbol{\phi}\left(\mathbf{X}\right)^T \boldsymbol{\phi}\left(\mathbf{X}\right) + \alpha \mathbf{I}_n\right)^{-1}, \ \mathbf{m} = \sigma^{-2} \mathbf{S} \boldsymbol{\phi}\left(\mathbf{X}\right)^T \mathbf{y}$$

5.2 Generate the dataset

The synthetic dataset is still generated from

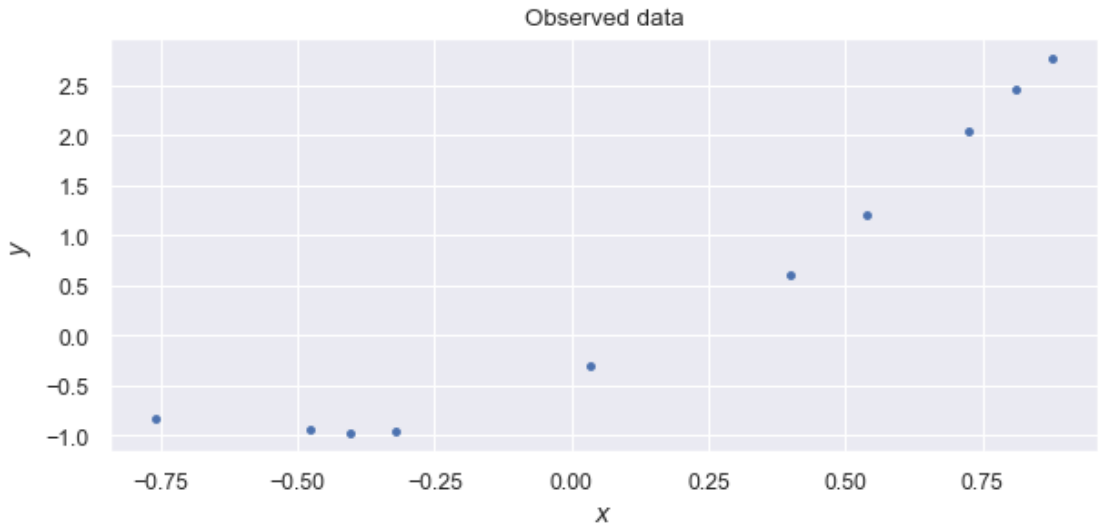
$$y_i = -0.5 + 2x_i + 2x_i^2 + 0.1\epsilon_i$$

where $\epsilon_i \sim \mathcal{N}(0, 1)$ and we sample $x_i \sim U([0, 1])$.

First generate this synthetic dataset and visualize the samples. For better observations of **regularization effect**, fewer observations are made.

[12]:
```
# Number of observations
num_obs = 10
# Sample x
x = (-1.0 + 2 * np.random.rand(num_obs)).reshape(-1, 1)
# True parameters
theta = np.array([-0.5, 2.0, 2.0]).reshape(-1, 1)
sigma = 0.1
# Calculate the corresponding y
y = theta[0] + theta[1] * x + theta[2] * x**2 \
    + sigma * np.random.randn(num_obs).reshape(-1, 1)
# Visualize the dataset
fig, ax = plt.subplots(figsize=(7, 3), dpi=100)
ax.plot(x, y, '.', markersize=5)
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

```
ax.set_title('Observed data')
plt.show()
```



5.3 Find the maximum a posterior estimation

Since the point estimate is exactly the same as the weight mean vector of the posterior distribution of parameters, the steps to get the maximum a posterior estimation is exactly the same as those steps in Bayesian Estimation. Also assume we do not know the exact degree of the polynomial and we try to use **a 5 degree polynomial** to fit the data.

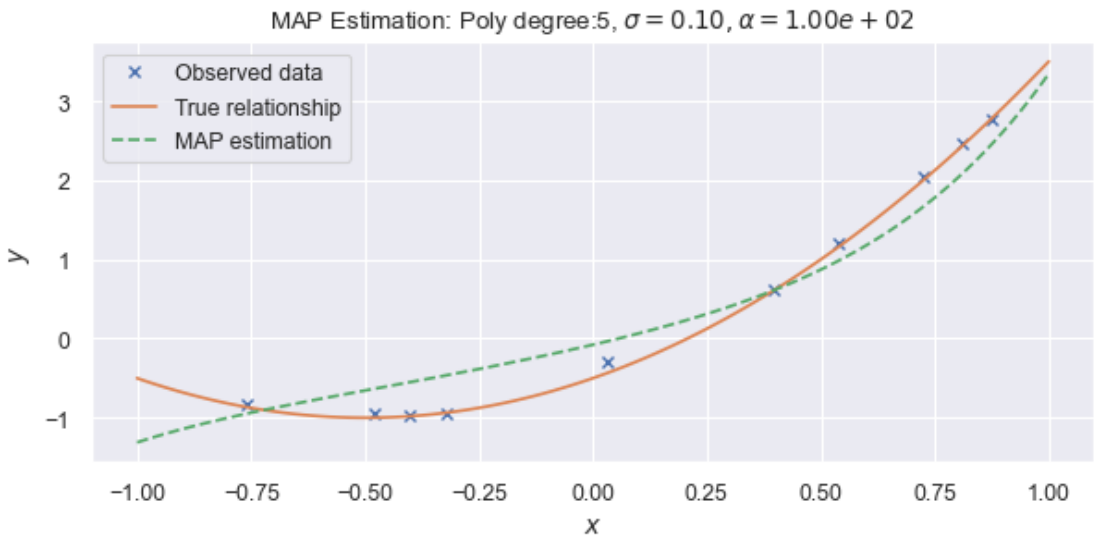
[13]:
```
# Define a function to find the maximum a posterior estimation
def find_MAP(Phi, y, sigma2, alpha):
    """
    Return the MAP estimation of a Bayesian linear regression problem
    with design matrix Phi, observations y, noise variance sigma2 and
    priors for the weights alpha.
    """
    # Construct the positive-definite matrix
    A = np.dot(Phi.T, Phi) / sigma2 + alpha * np.eye(Phi.shape[1])
    # Get the Cholesky decomposition of A
    L = scipy.linalg.cho_factor(A)
    # Solve for the maximum a posterior estimation
    MAP_estimation = scipy.linalg.cho_solve(L, np.dot(Phi.T, y / sigma2))
    return MAP_estimation


# Create the instance for calculating polynomial features
degree = 5
poly = PolynomialFeatures(degree, include_bias=True)
```

```
# Call the method of the instance
Phi = poly.fit_transform(x)
# Pick variance by hand. Here the true one is used
sigma = 0.1
# Pick the regularization paramter by hand
alpha = 100
# Get the MAP estimation
theta_map = find_MAP(Phi, y, sigma**2, alpha)
print("The maximum a posterior estimation when alpha = %1.2f is:\n"%alpha, m)
```

```
The maximum a posterior estimation when alpha = 100.00 is:
 [-0.41779846  1.77955225  1.15843691  0.27039672  0.88570988 -0.0445176 ]
```

Visualize the MAP estimation.

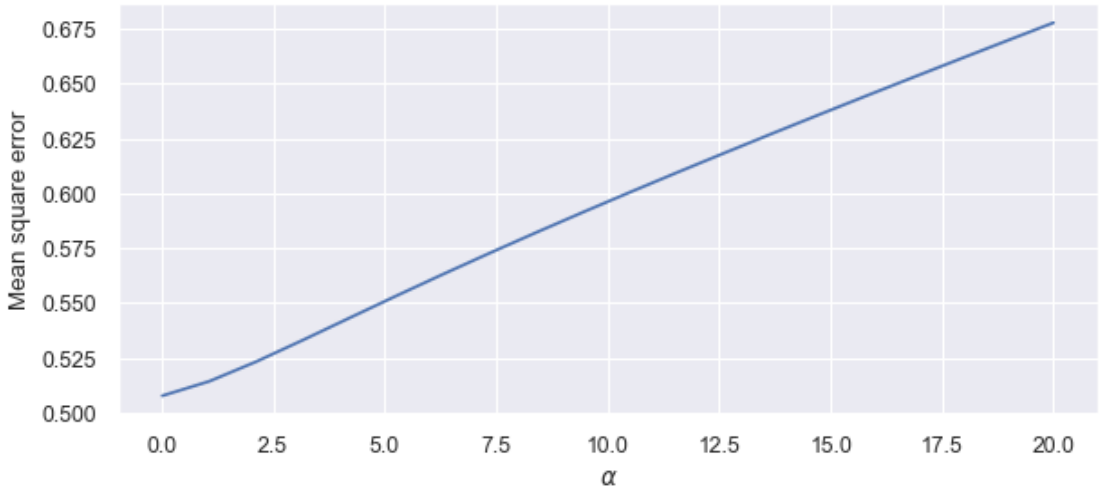
[14]:
```
fig, ax = plt.subplots(figsize=(7, 3), dpi=100)
# Some points on which to evaluate the regression function
xe = np.linspace(-1, 1, 100)
Phi_xe = poly.fit_transform(xe.reshape(-1, 1))
# The observations
ax.plot(x, y, 'x', label="Observed data")
# The true relationship between x and y
theta_true = np.array([-0.5, 2.0, 2.0, 0.0, 0.0, 0.0]).reshape(-1, 1)
ye_true = np.dot(Phi_xe, theta_true)
ax.plot(xe, ye_true, label="True relationship")
# The MAP estimation
ye_map = np.dot(Phi_xe, theta_map)
ax.plot(xe, ye_map, '--', label="MAP estimation")
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_title(\
    r'MAP Estimation: Poly degree:%d, $\sigma = %1.2f, \alpha=%1.2e$' %\
    (degree, sigma, alpha))
plt.legend(loc="upper left")
plt.show()
```



5.4 Selecting α through a validation set

Notice the α is actually the regularization parameter selected by hand, it can be optimized by plotting the mean square error of a validation set as a function of α.

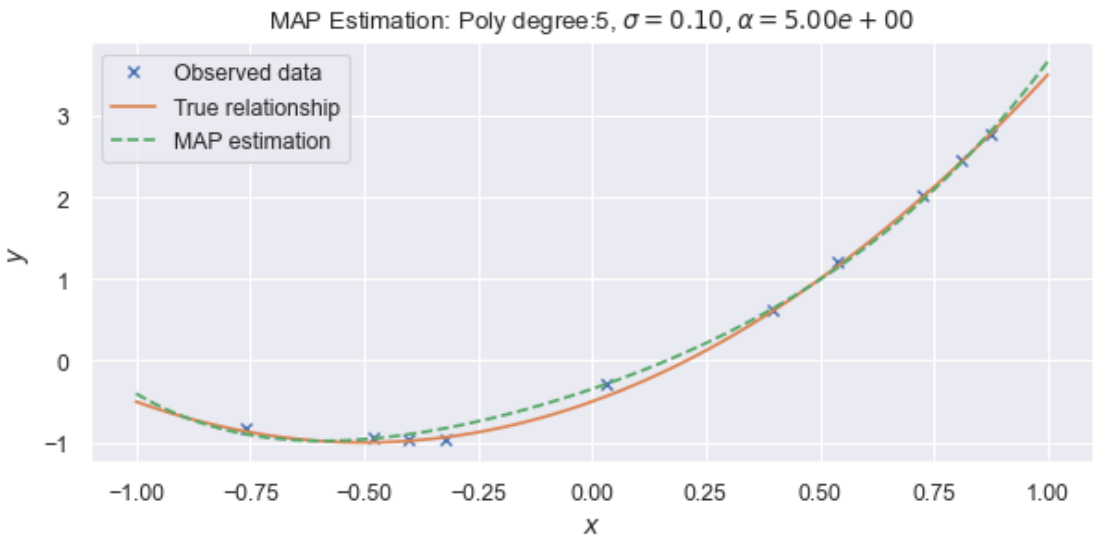
[15]:
```
# Create the validation dataset
num_valid = 20
x_valid = (- 1.0 + 2 * np.random.rand(num_valid)).reshape(-1,1)
Phi_valid = poly.fit_transform(x_valid)
y_valid = theta[0] + theta[1] * x_valid + theta[2] * x_valid**2 \
    + sigma * np.random.randn(num_valid).reshape(-1, 1)
# Get mean square errors corresponding to different alphas
MSE = []
alphas = np.linspace(0.01, 20.0, 20).reshape(-1,1)
for alpha in alphas:
    # Get the MAP estimation
    theta_map = find_MAP(Phi_valid, y_valid, sigma**2, alpha)
    # Compute the MAP prediction
    y_valid_pred = np.dot(Phi_valid, theta_map)
    # Calculate the meansquare error
    MSE_alpha = np.linalg.norm(y_valid_pred-y_valid)
    MSE.append(MSE_alpha)
# Plot the mean square errors out
fig, ax = plt.subplots(figsize=(7, 3), dpi=100)
ax.plot(alphas,MSE)
ax.set_xlabel(r'$\alpha$')
ax.set_ylabel('Mean square error');
```



However, in this case we should always expect lower MSE since we're using some polynomials of higher degree to fit the curve in a small region. Let's try some smaller α.

[16]:
```
alpha = 5
# Get the MAP estimation
theta_map = find_MAP(Phi, y, sigma**2, alpha)
print("The maximum a posterior estimation when alpha = %1.2f is:\n"%alpha, m)
fig, ax = plt.subplots(figsize=(7, 3), dpi=100)
# Some points on which to evaluate the regression function
xe = np.linspace(-1, 1, 100)
Phi_xe = poly.fit_transform(xe.reshape(-1, 1))
# The observations
ax.plot(x, y, 'x', label="Observed data")
# The true relationship between x and y
theta_true = np.array([-0.5, 2.0, 2.0, 0.0, 0.0, 0.0]).reshape(-1, 1)
ye_true = np.dot(Phi_xe, theta_true)
ax.plot(xe, ye_true, label="True relationship")
# The MAP estimation
ye_map = np.dot(Phi_xe, theta_map)
ax.plot(xe, ye_map, '--', label="MAP estimation")
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_title(\
    r'MAP Estimation: Poly degree:%d, $\sigma = %1.2f, \alpha=%1.2e$' %\
    (degree, sigma, alpha))
plt.legend(loc="upper left")
plt.show()
```

```
The maximum a posterior estimation when alpha = 5.00 is:
 [-0.41779846  1.77955225  1.15843691  0.27039672  0.88570988 -0.0445176 ]
```


MAP Estimation: Poly degree:5, $\sigma = 0.10$, $\alpha = 5.00e + 00$
Observed data
True relationship
MAP estimation
3
2
1
0
−1
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x