Question 1: $y(x,w) = w_0 + w_1 x + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$

$L(w) = \frac{1}{2}\sum_{i=1}^{n}[y_i - \sum_{j=0}^{M} w_j x^j]^2$ 分别对 $w_1, w_2 \cdots w_M$ 求偏导

有 $\frac{\partial L}{\partial (w_k)} = \sum_{i=1}^{n}[y - \sum_{j=0}^{M} w_j x^j] x^k$ 分别令 $\frac{\partial L}{\partial (w_k)} = 0$ 得

$$\begin{bmatrix} n & \sum x_i & \cdots & \sum x_i^M \\ \sum x_i & \sum x_i^2 & \cdots & \sum x_i^{M+1} \\ \vdots & \vdots & & \vdots \\ \sum x_i^M & \sum x_i^{M+1} & \cdots & \sum x_i^{2M} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \vdots \\ \sum x_i^k y_i \end{bmatrix}$$

有 $\begin{bmatrix} 1 & x_1 & \cdots & x_1^m \\ 1 & x_2 & \cdots & x_2^m \\ \vdots & & & \\ 1 & x_n & \cdots & x_n^m \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$ $w = (X^T X)^{-1} X^T y$

Question 2:

$P(\text{apple}) = P(r)\cdot\frac{3}{3+4+3} + P(b)\cdot\frac{1}{1+1} + P(g)\cdot\frac{3}{3+4+3}$

$= 0.34$

$P(g|\text{orange}) = \frac{P(g)\cdot\frac{3}{3+4+3}}{P(\text{orange})} = \frac{0.6\cdot\frac{3}{10}}{P(r)\cdot\frac{4}{3+4+3} + P(b)\cdot\frac{1}{1+1} + P(g)\cdot\frac{3}{3+4+3}}$

$= \frac{1.8}{10} = \frac{1.8}{0.36} = 0.5$

Question 3:

$E(x+z) = \frac{1}{n}\sum_{i=1}^{n}(x_i + z_i) = \frac{1}{n}\sum_{i=1}^{n} x_i + \frac{1}{n}\sum_{i=1}^{n} z_i$

$= E(x) + E(z)$

∵ x and y are independent variables

$\text{var}(x+z) = E(x+z)^2 - [E(x+z)]^2$

$= E(x^2 + 2xz + z^2) - (Ex + Ez)^2$

$= E(x^2) + E(z^2) + 2E(xz) - (Ex)^2 - 2ExEz - E(z)^2$

$= \text{var}(x) + \text{var}(z) + 2E(xz) - 2ExEz$

$= \text{var}(x) + \text{var}(z)$

Question 4:

$P(X|\lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$

$L(x_1, x_2 \cdots, x_n|\lambda) = \prod_{i=1}^{n}\frac{\lambda^{x_i} e^{-\lambda}}{x_i!} = e^{-n\lambda}\prod_{i=1}^{n}\frac{\lambda^{x_i}}{x_i!}$

$\ln L = -n\lambda + \sum_{i=1}^{n}(x_i \ln\lambda - \ln x_i)$

$\frac{d\ln L}{d\lambda} = -n + \sum_{i=1}^{n}\frac{x_i}{\lambda}$

令 $\frac{d\ln L}{d\lambda} = 0$ 得 $\hat{\lambda} = \frac{1}{n}\sum_{i=1}^{n} x_i$

exponential distribution:

$f(x) = \frac{1}{\lambda} e^{-\frac{x}{\lambda}}$

$L(x_1, x_2, \cdots, x_n|\lambda) = \prod_{i=1}^{n}\frac{1}{\lambda} e^{-\frac{x_i}{\lambda}} = \frac{1}{\lambda^n} \cdot e^{-\frac{\sum x_i}{\lambda}}$

$\ln L = -n\ln\lambda + (-\frac{\sum x_i}{\lambda})$

$\frac{d\ln L}{d\lambda} = -\frac{n}{\lambda} + \frac{\sum x_i}{\lambda^2}$ 令 $\frac{d\ln L}{d\lambda} = 0$

得 $\hat{\lambda} = \frac{1}{n}\sum_{i=1}^{n} x_i$

Question 5

a) $P(\text{mistake}) = \int_{R_1} P(x, c_2)\,dx + \int_{R_2} P(x, c_1)\,dx$

$P(\text{correct}) = \int_{R_1} P(x, c_1)\,dx + \int_{R_2} P(x, c_2)\,dx$

b)

$E[L(\vec{t}, \vec{y}(x))] = \int\int \|\vec{y}(x) - \vec{t}\|^2 p(\vec{x}, \vec{t})\,dx\,dt$

$\|\vec{y}(x) - \vec{t}\|^2 = \|\vec{y}(x) - \vec{E}_t[t|x] + \vec{E}_t[t|x] - \vec{t}\|^2$

$= \|\vec{y}(x) - \vec{E}_t[t|x]\|^2 +$

Question 6:

a) $p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

$H[x] = -\int p(x)\ln p(x)\,dx$

$-H[x] = -\int \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \ln\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\,dx$

$= -\frac{1}{\sqrt{2\pi}\sigma}\int e^{-\frac{(x-\mu)^2}{2\sigma^2}}\left(-\ln\sqrt{2\pi}\sigma - \frac{(x-\mu)^2}{2\sigma^2}\right)dx$

$= \frac{\ln\sqrt{2\pi}\sigma}{\sqrt{2\pi}\sigma}\int e^{-\frac{(x-\mu)^2}{2\sigma^2}}\,dx - \frac{1}{\sqrt{2\pi}\sigma}\int e^{-\frac{(x-\mu)^2}{2\sigma^2}}\cdot\frac{(x-\mu)^2}{2\sigma^2}\,dx$

$= \frac{\ln\sqrt{2\pi}\sigma}{\sqrt{2\pi}\sigma}\sqrt{2}\sigma\int e^{-\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)^2} d\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)$

$+ \frac{1}{\sqrt{2\pi}\sigma}\sqrt{2}\sigma\int e^{-\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)^2}\cdot\frac{(x-\mu)^2}{2\sigma^2} d\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)$

记 $y = \frac{x-\mu}{\sqrt{2}\sigma}$

有 $H[x] = \frac{\ln(\sqrt{2\pi}\sigma)}{\sqrt{\pi}} \int_{-\infty}^{+\infty} e^{-y^2} dy + \frac{1}{\sqrt{\pi}} \int_{-\infty}^{+\infty} e^{-y^2} y^2 dy$

$= \ln(\sqrt{2\pi}\sigma) + \frac{1}{\sqrt{\pi}} \int_{-\infty}^{+\infty} e^{-y^2} y^2 dy$

$= \ln(\sqrt{2\pi}\sigma) + \frac{1}{\sqrt{\pi}} \cdot (-\frac{1}{2}) \cdot (-\int_{-\infty}^{+\infty} e^{-y^2} dy)$

$= \ln(\sqrt{2\pi}\sigma) + \frac{1}{2}$

$= \frac{1}{2}\{1 + \ln(2\pi\sigma^2)\}$

b)

~~$KL(p\|q) = -\int p(x) \ln\{\frac{q(x)}{p(x)}\} dx$~~

~~$I[x,y] = KL(p(x,y)\|p(x)p(y))$~~

~~$= -\iint p(x,y) \cdot \ln\left(\frac{p(x)p(y)}{p(x,y)}\right) dxdy$~~

~~$I[x,y] = \sum_{x,y} p(x,y) \ln \frac{p(x,y)}{p(x)p(y)}$~~

~~$= \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$~~

b)

$I[x,y] = \sum_{x,y} p(x,y) \ln \frac{p(x,y)}{p(x)p(y)}$

$= \sum_{x,y} p(x,y) \ln \frac{p(x,y)}{p(x)} - \sum_{x,y} p(x,y) \ln p(y)$

$= \sum_{x,y} p(x)p(y|x) \ln p(y|x) - \sum_{x,y} p(x,y) \ln p(y)$

$= \sum_x p(x)\left(\sum_y p(y|x) \ln p(y|x)\right) - \sum_y \ln p(y)\left(\sum_x p(x,y)\right)$

$= -\sum_x p(x) H(Y|X=x) - \sum_y \ln p(y) \cdot p(y)$

$= H(Y) - H(Y|X)$

$I[x,y] = \sum_{x,y} p(x,y) \ln \frac{p(x,y)}{p(x)p(y)}$

$= \sum_{x,y} p(x,y) \ln \frac{p(x,y)}{p(y)} - \sum_{x,y} p(x,y) \ln p(x)$

$= \sum_{x,y} p(x,y) \ln p(x|y) - \sum_{x,y} p(x,y) \ln p(x)$

$= \sum_{x,y} p(y) \cdot p(x|y) \cdot \ln(x|y) - \sum_x \ln p(x)\left(\sum_y p(x,y)\right)$

$= -\sum_y p(y) H(x|Y=y) - \sum_x p(x) \cdot \ln p(x)$

$= H[x] - H[x|y]$

~~$I[x,y] = H[x] - H[Y|X] = H[x]$~~

$I[x,y] = H[Y] - H[Y|X] = H[X] - H[X|y]$

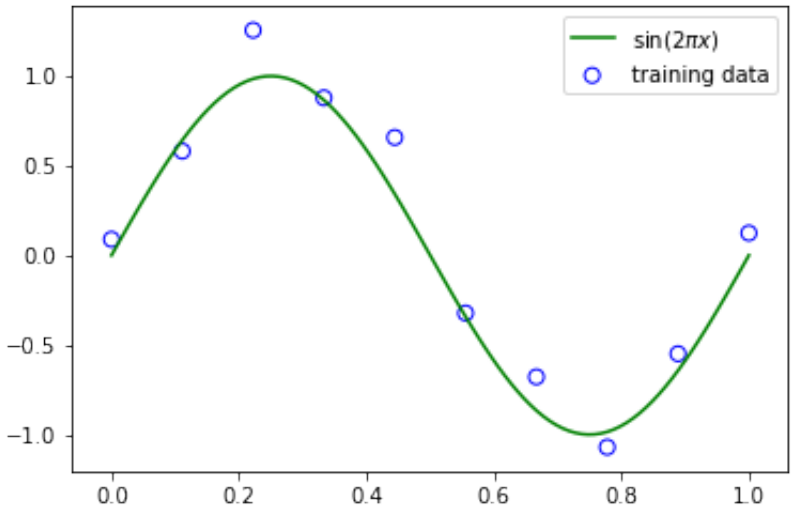
HW1_programQuestion

September 9, 2019

```
In [ ]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
```

```
In [ ]: def create_toy_data(func, sample_size, std):
            x = np.linspace(0, 1, sample_size)
            t = func(x) + np.random.normal(scale=std, size=x.shape)
            return x, t

        def func(x):
            return np.sin(2 * np.pi * x)

        x_train, y_train = create_toy_data(func, 10, 0.25)
        x_test = np.linspace(0, 1, 100)
        y_test = func(x_test)
```

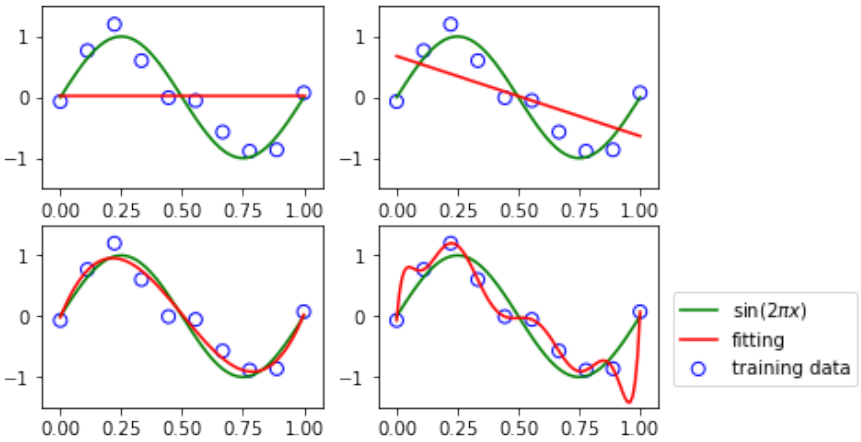
(a) Plot the graph with given code, the result should like below.



`x_train` and `y_train` are the datas you need to create, `sample_size` is 10 and `std` is 0.25.

```
In [ ]: # Write you codes here.
```

(b) On the basis of the results, you should try 0^{th} order polynomial, 1^{st} order polynomial, 3^{rd} order polynomial and some other order polynomial, show the results include fitting and



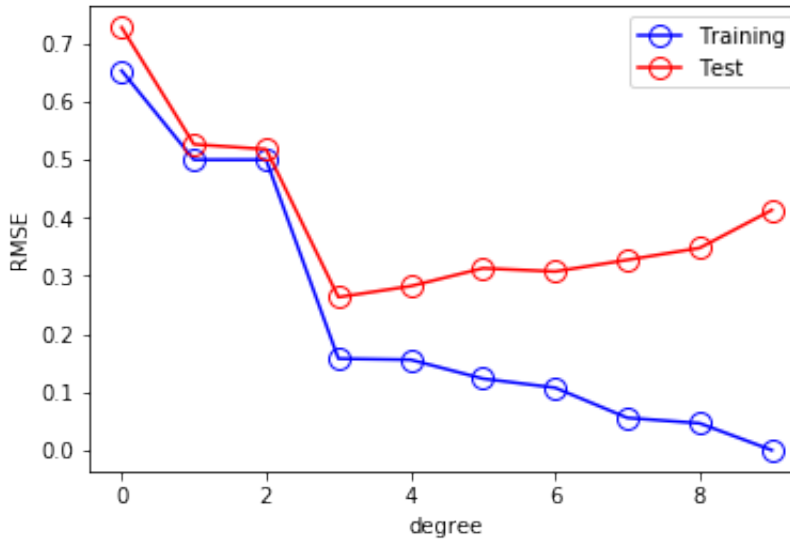
over-fitting.

```
In [ ]: class PolynomialFeature(object):
            """
            polynomial features

            transforms input array with polynomial features

            Example
            =======
            x =
            [[a, b],
            [c, d]]

            y = PolynomialFeatures(degree=2).transform(x)
            y =
            [[1, a, b, a^2, a * b, b^2],
            [1, c, d, c^2, c * d, d^2]]
            """

            def __init__(self, degree=2):
                """
                construct polynomial features

                Parameters
                ----------
                degree : int
                    degree of polynomial
                """
                assert isinstance(degree, int)
                self.degree = degree

            def transform(self, x):
                """
                transforms input array with polynomial features
```

```
        Parameters
        ----------
        x : (sample_size, n) ndarray
            input array

        Returns
        -------
        output : (sample_size, 1 + nC1 + ... + nCd) ndarray
            polynomial features
        """
        if x.ndim == 1:
            x = x[:, None]
        x_t = x.transpose()
        features = [np.ones(len(x))]
        for degree in range(1, self.degree + 1):
            for items in itertools.combinations_with_replacement(x_t, degree):
                features.append(functools.reduce(lambda x, y: x * y, items))
        return np.asarray(features).transpose()

class Regression(object):
    """
    Base class for regressors
    """
    pass

class LinearRegression(Regression):
    """
    Linear regression model
    y = X @ w
    t ~ N(t|X @ w, var)
    """

    def fit(self, X:np.ndarray, t:np.ndarray):
        """
        perform least squares fitting

        Parameters
        ----------
        X : (N, D) np.ndarray
            training independent variable
        t : (N,) np.ndarray
            training dependent variable
        """
        self.w = np.linalg.pinv(X) @ t
        self.var = np.mean(np.square(X @ self.w - t))

    def predict(self, X:np.ndarray, return_std:bool=False):
```

```
        """
        make prediction given input

        Parameters
        ----------
        X : (N, D) np.ndarray
            samples to predict their output
        return_std : bool, optional
            returns standard deviation of each predition if True

        Returns
        -------
        y : (N,) np.ndarray
            prediction of each sample
        y_std : (N,) np.ndarray
            standard deviation of each predition
        """
        y = X @ self.w
        if return_std:
            y_std = np.sqrt(self.var) + np.zeros_like(y)
            return y, y_std
        return y
```

In [ ]:
```
# Write your codes here.
```

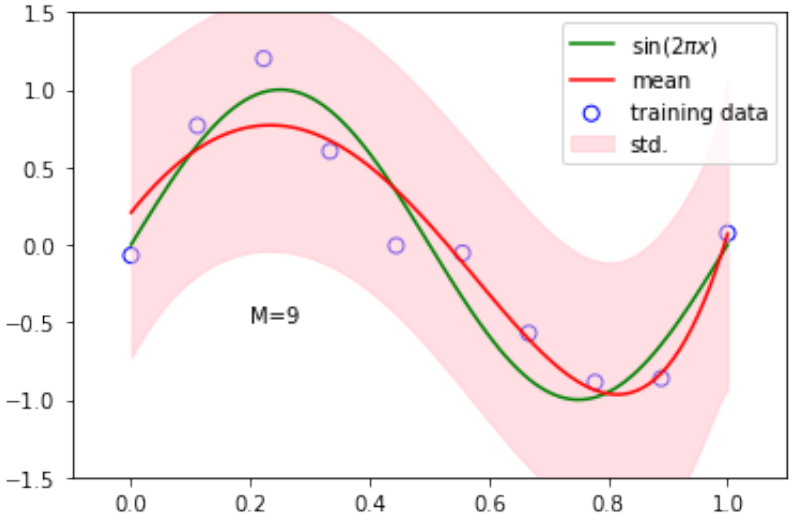


(c) Plot the graph of the root-mean-square error.

In [ ]:
```
def rmse(a, b):
    # Complete this function
```

In [ ]:
```
# Write your codes here.
training_erroes = []
test_errors = []
```

(d) Plot the graph of the predictive distribution resulting from a Bayesian treatment of polynomial curve fitting using an M=9 polynomial, with the fixed parameters $\alpha = 5 \times 10^{-3}$ and $\beta = 11.1$(corresponding to the known noise variance).




```
In [ ]: class BayesianRegression(Regression):
            """
            Bayesian regression model

            w ~ N(w|0, alpha^(-1)I)
            y = X @ w
            t ~ N(t|X @ w, beta^(-1))
            """

            def __init__(self, alpha:float=1., beta:float=1.):
                self.alpha = alpha
                self.beta = beta
                self.w_mean = None
                self.w_precision = None

            def _is_prior_defined(self) -> bool:
                return self.w_mean is not None and self.w_precision is not None

            def _get_prior(self, ndim:int) -> tuple:
                if self._is_prior_defined():
                    return self.w_mean, self.w_precision
                else:
                    return np.zeros(ndim), self.alpha * np.eye(ndim)

            def fit(self, X:np.ndarray, t:np.ndarray):
                """
                bayesian update of parameters given training dataset

                Parameters
```

```
        ----------
        X : (N, n_features) np.ndarray
            training data independent variable
        t : (N,) np.ndarray
            training data dependent variable
        """

        mean_prev, precision_prev = self._get_prior(np.size(X, 1))

        w_precision = precision_prev + self.beta * X.T @ X
        w_mean = np.linalg.solve(
            w_precision,
            precision_prev @ mean_prev + self.beta * X.T @ t
        )
        self.w_mean = w_mean
        self.w_precision = w_precision
        self.w_cov = np.linalg.inv(self.w_precision)

    def predict(self, X:np.ndarray, return_std:bool=False, sample_size:int=None):
        """
        return mean (and standard deviation) of predictive distribution

        Parameters
        ----------
        X : (N, n_features) np.ndarray
            independent variable
        return_std : bool, optional
            flag to return standard deviation (the default is False)
        sample_size : int, optional
            number of samples to draw from the predictive distribution
            (the default is None, no sampling from the distribution)

        Returns
        -------
        y : (N,) np.ndarray
            mean of the predictive distribution
        y_std : (N,) np.ndarray
            standard deviation of the predictive distribution
        y_sample : (N, sample_size) np.ndarray
            samples from the predictive distribution
        """

        if sample_size is not None:
            w_sample = np.random.multivariate_normal(
                self.w_mean, self.w_cov, size=sample_size
            )
            y_sample = X @ w_sample.T
            return y_sample
```

```
        y = X @ self.w_mean
        if return_std:
            y_var = 1 / self.beta + np.sum(X @ self.w_cov * X, axis=1)
            y_std = np.sqrt(y_var)
            return y, y_std
        return y
```

```
In [ ]: # Write your codes here.
```

(e) Change the *sample_size* to 2, 3 or 10 times than before, explain the change of M.

```
In [ ]: # Write your codes here.
```

随着测试训练集的增多，rmse 呈现下降趋势，对每个样例施加的干扰是固定的，随着样本案例的增多，多个样例的均值的期望的方差降低，使得最终 rmse 下降，但是在不同样例的情况下也可能出现 rmse 没有下降的情况，但是频率较低。