MLPR Tutorial Sheet 3 (Answers)

Reminders: Attempt the tutorial questions, and ideally discuss them before your tutorial — for instance at ML-Base. You can seek clarifications and hints on Hypothesis. Full answers will be released.

1. A Gaussian classifier:

A training set consists of one-dimensional examples from two classes. The training examples from class 1 are \{0.5, 0.1, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.35, 0.25\} and the examples from class 2 are \{0.9, 0.8, 0.75, 1.0\}.

a) Fit a one-dimensional Gaussian to each class by matching the mean and variance. Also estimate the class probabilities \(\pi_1\) and \(\pi_2\) by matching the observed class fractions. (This procedure fits the model with maximum likelihood: it selects the parameters that give the training data the highest probability.) Sketch a plot of the scores \(p(x, y) = P(y) \cdot p(x \mid y)\) for each class \(y\), as functions of input location \(x\).

b) What is the probability that the test point \(x = 0.6\) belongs to class 1? Mark the decision boundary/ies on your sketch, the location(s) where \(P(\text{class 1} \mid x) = P(\text{class 2} \mid x) = 0.5\). You are not required to calculate the location(s) exactly.

c) Are the decisions that the model makes reasonable for very negative \(x\) and very positive \(x\)? Are there any changes we could consider making to the model if we wanted to change the model’s asymptotic behaviour?

Answer:

The mean of class one is at 0.26, and class two is at 0.86. The standard deviations of the classes are both about 0.1, with class one being slightly wider. The precise estimates you get will depend on whether you used the version containing a normalization of \(N\) or \(N-1\). The class 1 curve is higher, despite being slightly broader, because there are more points in class 1.

The classification probability is given by Bayes’ rule:

\[
P(y \mid x) = \frac{p(x \mid y) \cdot P(y)}{\sum_{k=1}^{2} p(x \mid y=k) \cdot P(y=k)}
\]

We set \(P(y=k) = \pi_k\), given by the empirical class fractions \(\pi_1 = 0.71\) and \(\pi_2 = 0.29\).
We set the class likelihoods to: 
\[ p(x \mid y=k) = \mathcal{N}(x; \mu_k, \sigma_k^2), \]
with means and variances set to the empirical values.

Substituting that all in we get
\[ P(y=1 \mid x=0.6) = 0.63 \]
if we use the “maximum likelihood” estimates of the variances (normalization by \(N\)). However we get
\[ P(y=1 \mid x=0.6) = 0.52 \]
if using the unbiased estimates for the variances, using the normalization by \(N-1\). Given this small dataset, we favour class 1, but not by much. The model fits are somewhat unreliable.

The diagram shows vertical lines at the decision boundaries where the posterior probabilities for the two classes are equal. To find these locations we equated the scores (prior times likelihood) for the two classes, took logs, and solved the resulting quadratic equation. (You were told you didn’t have to do that. Although you could recognize that there is a quadratic equation and that it might have two solutions, without writing out all the details or solving.)

If you want to check numbers or details of the class boundary solution, see the accompanying Matlab code.

Because class 2 has lower variance, the tail probability of class 1 eventually becomes bigger than class 2 to the far right of the plot, leading to the second decision boundary. Further to the right, beyond that boundary, we will become incredibly confident that points should be classified as class one. It is hard to justify making confident predictions far away from any of the observed data.

There is no correct answer here. If we changed the tails of the likelihood \(p(x \mid y)\), we could change the extreme behaviour. By using a mixture between a uniform and Gaussian, we could make it so that extreme points were given a nearly constant score by each class, leading to nearly uniform predictions. (Although Uniform distributions have to be bounded to be well-defined.)

We might simply give an alert when \(x\) appears far from any observed training input, and refuse to make a classification. How would we decide if \(x\) is “too far”? One idea is to create an extra “outliers” class with a large variance, and assign points to this class along with all of the others.

We might force the class variances to be equal. Then there would be no quadratic term when solving for the class boundary. There would be a single decision boundary, which may or may not be desirable.

**Intended lessons:** Be able to sketch Gaussians. Understand that a broad model can be more probable either side of a more focussed model. Begin to see some problems with fitting simple models to limited to data. Be able to discuss alternatives.

---

2. **More practice with Gaussians:**

\(N\) noisy independent observations are made of an unknown scalar quantity \(m\):

\[ x^{(n)} \sim \mathcal{N}(m, \sigma^2). \]

a) We don’t give you the raw data, \(\{x^{(n)}\}\), but tell you the mean of the observations:

\[ \bar{x} = \frac{1}{N} \sum_{n=1}^{N} x^{(n)}. \]

What is the likelihood\(^2\) of \(m\) given only this mean \(\bar{x}\)? That is, what is \(p(\bar{x} \mid m)^2\)?

---

2. We’re using the traditional statistics usage of the word “likelihood”: it’s a function of parameters given data, equal to the probability of the data given the parameters. You should avoid saying “likelihood of the data” (Cf p29 of MacKay’s textbook), although you’ll see that usage too.

3. The sum of Gaussian outcomes is Gaussian distributed; you only need to identify a mean and variance.
The mean of some Gaussian outcomes, like any linear combination, is Gaussian distributed. So we just have to identify the mean and variance of this derived quantity. The mean:

$$\mathbb{E}[\bar{x}] = \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}[x^{(n)}] = \frac{1}{N} \sum_{n=1}^{N} m = m. \quad (2)$$

The variance:

$$\text{var}[\bar{x}] = \frac{1}{N^2} \text{var} \left[ \sum_{n=1}^{N} x^{(n)} \right] = \frac{1}{N^2} \left[ N \sigma^2 \right] = \frac{\sigma^2}{N}. \quad (3)$$

where variances add because the noisy outcomes are independent. So:

$$p(\bar{x} | m) = \mathcal{N}(\bar{x}; m, \sigma^2 / N). \quad (4)$$

b) A sufficient statistic is a summary of some data that contains all of the information about a parameter.

i) Show that $\bar{x}$ is a sufficient statistic of the observations for $m$, assuming we know the noise variance $\sigma^2$. That is, show that $p(m | \bar{x}) = p(m | \{x^{(n)}\}_{n=1}^{N})$.

ii) Optional part: If we don’t know the noise variance $\sigma^2$ or the mean, is $\bar{x}$ still a sufficient statistic in the sense that $p(m | \bar{x}) = p(m | \{x^{(n)}\}_{n=1}^{N})$? Explain your reasoning.

Note: In i) we were implicitly conditioning on $\sigma^2$: $p(m | \bar{x}) = p(m | \bar{x}, \sigma^2)$. In this part, $\sigma^2$ is unknown, so $p(m | \bar{x}) = \int p(m, \sigma^2 | \bar{x}) \, d\sigma^2$. Although no detailed mathematical manipulation (or solving of integrals) is required.

Answer:

i) The posterior given the observations comes from Bayes’ rule:

$$p(m | \{x^{(n)}\}_{n=1}^{N}) \propto p(m) \, p(\{x^{(n)}\}_{n=1}^{N} | m) \quad (5)$$

$$\propto p(m) \prod_{n} \mathcal{N}(x^{(n)}; m, \sigma^2) \quad (6)$$

$$\propto p(m) \exp \left( -\frac{1}{2\sigma^2} \sum_{n}(x^{(n)} - m)^2 \right) \quad (7)$$

$$\propto p(m) \exp \left( -\frac{Nm^2}{2\sigma^2} + \frac{m \sum_{n} x^{(n)}}{\sigma^2} \right). \quad (8)$$

The proportionality sign, ‘$\propto$’, means that we can drop any multiplicative term that doesn’t depend on $m$. 

---

Given just the mean $\bar{x}$ the posterior would be:

$$p(m | \bar{x}) \propto p(m) p(\bar{x} | m)$$

$$\propto p(m) \mathcal{N}(\bar{x}; m, \sigma^2 / N)$$

$$\propto p(m) \exp \left( -\frac{N}{2\sigma^2} (\bar{x} - m)^2 \right)$$

$$\propto p(m) \exp \left( -\frac{Nm^2}{2\sigma^2} + \frac{mN\bar{x}}{\sigma^2} \right),$$

which, substituting the definition of $\bar{x}$, is equal to the posterior given all of the observations. Therefore $\bar{x}$ is a sufficient statistic. Our beliefs about the parameter $m$ given just $\bar{x}$ are identical to those given the whole dataset.

ii) The working above assumed that we knew the variance $\sigma^2$. If we don’t know $\sigma^2$ then $\bar{x}$ is not sufficient for computing the posterior over the mean. Counter example: if all the $x^{(n)}$ samples were identical, then we would infer that the variance is zero and have a delta-function posterior over the mean. If the $x^{(n)}$ samples had high variance (which they could have with the same mean $\bar{x}$), we’d realize the variance is larger, and not be certain about the mean. We can still use $\bar{x}$ as a point-estimate of the mean, it just doesn’t tell us the whole posterior distribution.

**Purpose of Q2:**
Computing the posterior in Bayesian linear regression is just an example of dealing with noisy observations of linear combinations of Gaussian variables. We will encounter similar examples later in the course (computing predictions in Bayesian linear regression and Gaussian processes). We can do a lot of useful statistical reasoning if we know how to deal with Gaussians, so it’s worth practicing manipulating them. This question was meant to contain some easier manipulations than some of the ones we’ve seen in lectures.

Much of machine learning is about compressing some useful knowledge from a large dataset into a learned model. The notion of a sufficient statistic is sometimes useful: it tells us some narrow situations in which we can replace a large dataset with some statistics without losing anything.

---

3. **Bayesian regression with multiple data chunks:**

This question involves simulations on a computer. You will generate datasets and calculate posterior distributions. If you have trouble, at least try to work out roughly what should happen for each part.

We will use the probabilistic model for regression models from the lectures:

$$p(y | x, w) = \mathcal{N}(y; f(x; w), \sigma_y^2).$$

In this question, we set $f(x; w) = w^T x$, where $w = [w_1 \ w_2]^T$ and $x = [x_1 \ x_2]^T$, and $\sigma_y^2 = 1^2 = 1$. We assume the following prior distribution:

$$p(w) = \mathcal{N}(w; [-5 \ 0]^T, \sigma_w^2 I),$$

where $\sigma_w^2 = 2^2 = 4$.

**Generating data:** Generate some synthetic dataset as below. Only generate the data once and do not change it when working on the parts of this question.

First, draw a $\tilde{w}$ from the prior distribution (i.e. draw a sample from the Gaussian). In the whole dataset generation process, draw this $\tilde{w}$ only once and keep it constant.
We will generate two chunks of data $D_1$ and $D_2$. For chunk $D_1 = \{x^{(n)}, y^{(n)}\}_{n=1}^{15}$, generate 15 input-output pairs where each $x^{(n)}$ is drawn from $\mathcal{N}(x^{(n)}; 0, 0.5^2 \mathbb{I})$. For chunk $D_2 = \{x^{(n)}, y^{(n)}\}_{n=1}^{45}$, generate 30 input-output pairs where each $x^{(n)}$ is drawn from $\mathcal{N}(x^{(n)}; [0.5, 0.5]^T, 0.1^2 \mathbb{I})$. For both chunks, draw the $y^{(n)}$ outputs using the $p(y|x,w)$ observation model above.

**Visualising distributions on the weights:** For each part that asks you to visualise a distribution, draw 400 weight vectors from the distribution and show the samples in a 2D scatter plot. In each plot, show $w$ as a bold cross.

a) Visualise the prior distribution and discuss the relationship between the parameters and the belief expressed by the prior.

**Answer:**
First, we implement the complete setting with all datasets and a general function to visualise a distribution. The code for the later questions will build on this earlier code, so that code will not work on its own without running the code for the earlier questions.

```matlab
% Matlab/Octave
D = 2; % Dimension of the weight space
N_Data_1 = 15; % Number of samples in dataset 1
N_Data_2 = 30; % Number of samples in dataset 2
sigma_w = 2.0;
prior_mean = [-5; 0];
prior_precision = eye(D) ./ sigma_w^2;
% We summarize distributions using their parameters
prior_par.mean = prior_mean;
prior_par.precision = prior_precision;
% Here we draw the true underlying w. We do this only once
w_tilde = sigma_w * randn(2, 1) + prior_mean;
% Draw the its for datasets 1 and 2
X_Data_1 = 0.5 * randn(N_Data_1, D);
X_Data_2 = 0.1 * randn(N_Data_2, D) + 0.5;
% Draw the outputs for the datasets
y_Data_1 = X_Data_1 * w_tilde + sigma_y * randn(N_Data_1, 1);
y_Data_2 = X_Data_2 * w_tilde + sigma_y * randn(N_Data_2, 1);
% The complete datasets
Data_1.X = X_Data_1;
Data_1.y = y_Data_1;
Data_2.X = X_Data_2;
Data_2.y = y_Data_2;

function visualise_par(par, colour)
% Visualises a multivariate normal distribution with given mean and
% covariance matrix in the par dictionary.
D = size(par.mean);
N = 400;
% The next two lines correspond to:
% samples = mvnrnd(par.mean, inv(par.precision), 400);
L = chol(par.precision, 'lower');
samples = (L' \ randn(D, N))' + repmat(par.mean, N, 1);
scatter(samples(:, 1), samples(:, 2), 1, colour);
xlim([-10, 0]);
ylim([-5, 5]);
```

xlabel('w_1');
ylabel('w_2');
end

figure;
visualise_par(prior_par, 'b');
legend('p(w)');
text(w_tilde(1), w_tilde(2), 'X');

Or

# Python
import numpy as np
from numpy.random import randn, uniform
from scipy.linalg import cholesky, solve, solve_triangular
import matplotlib.pyplot as plt

D = 2 # Dimension of the weight space
N_Data_1 = 15 # Number of samples in dataset 1
N_Data_2 = 30 # Number of samples in dataset 2
sigma_w = 2.0
prior_mean = [-5, 0]
prior_precision = np.eye(D) / sigma_w**2
# We summarize distributions using their parameters
prior_par = {'mean': prior_mean, 'precision': prior_precision}
# Here we draw the true underlying w. We do this only once
w_tilde = sigma_w * randn(2) + prior_mean
# Draw the inputs for datasets 1 and 2
X_Data_1 = 0.5 * randn(N_Data_1, D)
X_Data_2 = 0.1 * randn(N_Data_2, D) + 0.5
# Draw the outputs for the datasets
sigma_y = 1.0
y_Data_1 = np.dot(X_Data_1, w_tilde) + sigma_y * randn(N_Data_1)
y_Data_2 = np.dot(X_Data_2, w_tilde) + sigma_y * randn(N_Data_2)
# The complete datasets
Data_1 = {'X': X_Data_1, 'y': y_Data_1}
Data_2 = {'X': X_Data_2, 'y': y_Data_2}

def visualise_par(par, label, colour):
    # Visualises a multivariate normal distribution with given mean and
    # covariance matrix in the par dictionary.

    D = len(par['mean'])
    # The next three lines correspond to:
    # samples = multivariate_normal(par['mean'],
    #     inv(par['precision'])), 400)
    L = cholesky(par['precision'], lower=True)
samples = solve_triangular(L, randn(D, 400), trans=1, lower=True).T + par['mean']
plt.scatter(samples[:, 0], samples[:, 1], s=1, label=label, c=colour)
plt.plot(w_tilde[0], w_tilde[1], marker='X', c='k')
plt.xlim([-10, 0])
plt.ylim([-5, 5])
plt.xlabel('$w_1$')
plt.ylabel('$w_2$')
There will be variability in the plots you generate, in particular regarding the location of $\hat{w}$, but the plot should look similar to the following:

![Plot](image_url)

The prior mean can be interpreted as a belief that $w_1$ is probably negative (and around -5) while $w_2$ being negative is just as plausible as $w_2$ being positive. The standard deviation is the same for $w_1$ and $w_2$ meaning that we are just as certain about $w_1$ being close to -5 as we are about $w_2$ being close to 0. Moreover, the correlation is zero, meaning that we don’t believe in any relationship between $w_1$ and $w_2$.

b) Using equations from the lecture notes, calculate and visualise the posterior distributions that you get after observing:

i. dataset $D_1$, i.e. $p(w \mid D_1)$

ii. dataset $D_2$ but not $D_1$, i.e. $p(w \mid D_2)$

iii. datasets $D_1$ and $D_2$, i.e. $p(w \mid D_1, D_2)$

iv. no observations, i.e. $p(w \mid \emptyset)$

Answer:
Here see that the two data chunks constrain the broad prior distribution (no observations) in different ways. Conditioning on both datasets incorporates the constraints from both chunks.
From the lecture notes (w3b), we have $p(\mathbf{w} | \mathcal{D}) = \mathcal{N}(\mathbf{w}; \mathbf{w}_N, V_N)$ with:

$$V_N = \sigma_y^2(\sigma_y^2 V_0^{-1} + \Phi^\top \Phi)^{-1}, \quad (13)$$

$$\mathbf{w}_N = V_N V_0^{-1} \mathbf{w}_0 + \frac{1}{\sigma_y^2} V_N \Phi^\top \mathbf{y}. \quad (14)$$

With $\Phi = \mathbf{X}$ and $V_0 = \sigma_w^2 \mathbb{I}$, we get:

$$V_N = \sigma_y^2(\sigma_y^2 \frac{1}{\sigma_w^2} \mathbb{I} + \mathbf{X}^\top \mathbf{X})^{-1}, \quad (15)$$

$$\mathbf{w}_N = V_N \frac{1}{\sigma_w^2} \mathbf{w}_0 + \frac{1}{\sigma_y^2} V_N \mathbf{X}^\top \mathbf{y}. \quad (16)$$

Here, we implement the more general version with arbitrary $V_0$ to simplify question c). To avoid calculating inverse matrices, we keep track of the precision matrices $K_0 = V_0^{-1}$ and $K_N = V_N^{-1}$ and use the update equations:

$$K_N = K_0 + \frac{1}{\sigma_y^2} \mathbf{X}^\top \mathbf{X}, \quad (17)$$

$$\mathbf{w}_N = K_N^{-1} K_0 \mathbf{w}_0 + \frac{1}{\sigma_y^2} K_N^{-1} \mathbf{X}^\top \mathbf{y}. \quad (18)$$

% Matlab/Octave

function par = posterior_par(prior_par, Data, sigma_y)
% Calculates posterior mean and covariance for given prior mean and
% covariance in the par dictionary, given data and given noise
% standard deviation.
X = Data.X;
y = Data.y;
var_y = sigma_y^2;
w_0 = prior_par.mean;
V_0_inv = prior_par.precision;
if numel(X) > 0
% The next line corresponds to:
% V_N = var_y * inv(var_y * V_0_inv + X' * X);
V_N_inv = V_0_inv + X/var_y * X ./ var_y;
else
V_N_inv = V_0_inv;
end
% The next line corresponds to:
% w_N = V_N * V_0_inv * w_0;
w_N = V_N_inv \ (V_0_inv * w_0);
if numel(y) > 0
% The next line corresponds to:
% w_N = w_N + (V_N * X' * y) ./ var_y;
w_N = w_N + (V_N_inv \ (X' * y)) ./ var_y;
end
par.mean = w_N;
par.precision = V_N_inv;
end

% We combine datasets 1 and 2 by stacking the matrices / vectors
Data_1_2.X = [X_Data_1; X_Data_2];
Data_1_2.y = [y_Data_1; y_Data_2];
% Calculate posterior parameters
posterior_par_1 = posterior_par(prior_par, Data_1, sigma_y);
posterior_par_2 = posterior_par(prior_par, Data_2, sigma_y);
posterior_par_1_2 = posterior_par(prior_par, Data_1_2, sigma_y);
% Visualise prior and posterior distributions
figure;
hold on;
visualise_par(prior_par, 'b');
visualise_par(posterior_par_1, 'r');
visualise_par(posterior_par_2, 'g');
visualise_par(posterior_par_1_2, 'y');
legend('p(w)', 'p(w|D_1)', 'p(w|D_2)', 'p(w|D_1,D_2)');
text(w_tilde(1), w_tilde(2), 'X');
hold off;
% Here we construct an empty dataset
Data_nodat.X = [];
Data_nodat.y = [];
% The posterior for an empty dataset will be the prior
figure;
hold on;
posterior_par_nodat = posterior_par(prior_par, Data_nodat, sigma_y);
visualise_par(prior_par, 'b');
visualise_par(posterior_par_nodat, 'c');
legend('p(w)', 'p(w|\{\})');
text(w_tilde(1), w_tilde(2), 'X');
hold off;

Or

# Python
def posterior_par(prior_par, Data, sigma_y):
    ''' Calculates posterior mean and covariance for given prior mean and
    covariance in the par dictionary, given data and given noise
    standard deviation.
    '''
    X = Data['X']
y = Data['y']
var_y = sigma_y**2
w_0 = prior_par['mean']
V_0_inv = prior_par['precision']
# The next line corresponds to:
# V_N = var_y * inv(var_y * V_0_inv + np.matmul(X.T, X))
V_N_inv = V_0_inv + np.matmul(X.T, X) / var_y
# The next line corresponds to:
# w_N = np.dot(V_N, np.dot(V_0_inv, w_0))
w_N = solve(V_N_inv, np.dot(V_0_inv, w_0))
if y.size > 0:
    w_N += solve(V_N_inv, np.dot(X.T, y)) / var_y
return {'mean': w_N, 'precision': V_N_inv}

# We combine datasets 1 and 2 by stacking the matrices / vectors
Data_1_2 = {'X': np.vstack((X_Data_1, X_Data_2)),
            'y': np.concatenate((y_Data_1, y_Data_2))}
# Calculate posterior parameters
posterior_par_1 = posterior_par(prior_par, Data_1, sigma_y)
posterior_par_2 = posterior_par(prior_par, Data_2, sigma_y)
posterior_par_1_2 = posterior_par(prior_par, Data_1_2, sigma_y)
# Visualise prior and posterior distributions
plt.figure(2)
plt.clf()
visualise_par(prior_par,
label='$p(\mathbf{w})$'
visualise_par(posterior_par_1,
    label='$p(\mathbf{w}|\mathcal{D}_1)$',
    colour='r')
visualise_par(posterior_par_2,
    label='$p(\mathbf{w}|\mathcal{D}_2)$',
    colour='g')
visualise_par(posterior_par_1_2,
    label='$p(\mathbf{w}|\mathcal{D}_1,\mathcal{D}_2)$',
    colour='y')
plt.legend()
plt.show()

# Here we construct an empty dataset
Data_nodat = {'X': np.empty([0, D]),
        'y': np.empty([0, 1])}
# The posterior for an empty dataset will be the prior
plt.figure(3)
plt.clf()
posterior_par_nodat = posterior_par(prior_par, Data_nodat, sigma_y)
visualise_par(prior_par,
    label='$p(\mathbf{w})$',
    colour='b')
visualise_par(posterior_par_nodat,
    label='$p(\mathbf{w}|\emptyset)$',
    colour='c')
plt.legend()
plt.show()

With more data, the posterior should become sharper with a mean closer to $\mathbf{w}$:

With no observed data, the posterior corresponds to the prior:
c) Take the posterior that you got in b) for \( p(w \mid D_1) \) and now use it as a prior. Calculate the new posterior that you get after observing \( D_2 \). Numerically compare the mean and covariance with those of \( p(w \mid D_1, D_2) \).

**Answer:**

We should find that it doesn’t matter if we update our beliefs in two chunks, or all at once. Our beliefs should be the same given both datasets regardless of how we compute them.

We use the \( w_N \) and the \( K_N \) that we got for \( p(w \mid D_1) \) and use them as our new \( w_0 \) and \( K_0 \) in the equations from part b):

\[
K_N = K_0 + \frac{1}{\sigma_y^2} X^\top X, \quad (19)
\]

\[
w_N = K_N^{-1} K_0 w_0 + \frac{1}{\sigma_y^2} K_N^{-1} X^\top y. \quad (20)
\]

% Matlab/Octave
% Update the posterior that we got for Data_1 with Data_2
posterior_par_c = posterior_par(posterior_par_1, Data_2, sigma_y);
% The posterior is the same as the one we got in b)
figure;
hold on;
visualise_par(prior_par, 'b');
visualise_par(posterior_par_1_2, 'y');
visualise_par(posterior_par_c, 'm');
legend('p(w)', 'p(w|D_1,D_2)', 'Posterior update');
text(w_tilde(1), w_tilde(2), 'X');
hold off;
disp('p(w|D_1,D_2):');
disp(posterior_par_1_2);
disp('Posterior update:');
disp(posterior_par_c);

Or

# Python
# Update the posterior that we got for Data_1 with Data_2
posterior_par_c = posterior_par(posterior_par_1, Data_2, sigma_y)

# The posterior is the same as the one we got in b)
plt.figure(4)
plt.clf()
visualise_par(prior_par, label='$p(\mathbf{w})$', colour='b')
visualise_par(posterior_par_1_2, label='$p(\mathbf{w}|\mathcal{D}_1,\mathcal{D}_2)$', colour='y')
visualise_par(posterior_par_c, label='Posterior update', colour='m')
plt.legend()
plt.show()
print('\textit{p(w|D_1,D_2):}')
print(posterior_par_1_2)
print('Posterior update: ')
print(posterior_par_c)

The posteriors should be identical. While the samples will be different, the point clouds should be on top of each other:

![Posterior update](image)

The numerical means and covariance / precision matrices should match (almost) perfectly.

d) Is it important that the inputs were drawn from Gaussian distributions? What would happen if the $x^{(n)}$ in $D_2$ were drawn from a uniform distribution on the unit square?

**Answer:**
In our formalism, we don’t make use of any distribution of the input. So with the uniform distribution, we can still apply the same equations from the lecture notes.

% Matlab/Octave
% For dataset 2, draw uniform samples instead of Gaussian samples
X_Data_u = rand(N_Data_2, D);
% Draw the corresponding outputs
y_Data_u = X_Data_u * w_tilde + sigma_y * randn(N_Data_2, 1);
Data_u.X = X_Data_u;
Data_u.y = y_Data_u;
% Calculate the posterior for the new dataset
posterior_par_u = posterior_par(prior_par, Data_u, sigma_y);
% Compare to the posterior that we originally got for dataset 2
figure;
hold on;
visualise_par(prior_par, 'b');
visualise_par(posterior_par_2, 'g');
visualise_par(posterior_par_u, 'm');
legend('p(w)', 'p(w|\mathcal{D}_2)', 'p(w|\mathcal{D}_u)');
text(w_tilde(1), w_tilde(2), 'X');
hold off;

Or

# Python
# For dataset 2, draw uniform samples instead of Gaussian samples
X_Data_u = uniform(size=D*N_Data_2).reshape([N_Data_2, D])
# Draw the corresponding outputs
y_Data_u = np.dot(X_Data_u, w_tilde) + sigma_y * randn(N_Data_2)
Data_u = {'X': X_Data_u, 'y': y_Data_u}
# Calculate the posterior for the new dataset
posterior_par_u = posterior_par(prior_par, Data_u, sigma_y)
# Compare to the posterior that we originally got for dataset 2
plt.figure(5)
plt.clf()
visualise_par(prior_par, label='$p(\mathbf{w})$', colour='b')
visualise_par(posterior_par_2, label='$p(\mathbf{w}|\mathcal{D}_2)$', colour='g')
visualise_par(posterior_par_u, label='$p(\mathbf{w}|\mathcal{D}_u)$', colour='m')
plt.legend()
plt.show()

The data sampled from the uniform distribution will generally lead to a sharper posterior distribution. This is because the Gaussian that we used to generate $D_2$ forms a sharp blob around $[0.5, 0.5]^\top$, while the uniform distribution is more spread out with the same mean. Thus, the diagonal elements of $X^\top X$ will on average be greater when sampling from the uniform distribution rather than the Gaussian:

$$E[X_k X_k] = \sigma^2 + E[X_k]^2,$$

(21)

where $\sigma^2$ is the variance of the distribution sampled from (you get this equation from rearranging the equation for the variance). $E[X_k]$ is the same for both distributions, but the variance of the uniform distribution is $\frac{1}{12}$ while the variance of the Gaussian is 0.01. As can be seen in the equation for

$$V_N^{-1} = V_0^{-1} + \frac{1}{\sigma_0^2} X^\top X,$$

(22)

this results in higher precision and lower covariance:
e) What distribution is \( p(\mathbf{w} \mid D_1) \) approaching when we let \( \sigma_w^2 \) approach infinity?

**Answer:**
We continue with our equations for \( V_N \) and \( \mathbf{w}_N \) from above:

\[
V_N = \sigma_y^2 \left( \frac{1}{\sigma_w^2} \mathbb{I} + X^\top X \right)^{-1},
\]
\[
\mathbf{w}_N = V_N \left( \frac{1}{\sigma_w^2} \mathbf{I} \mathbf{w}_0 + \frac{1}{\sigma_y^2} V_N X^\top \mathbf{y} \right). \tag{24}
\]

In the limit \( \sigma_w^2 \to \infty \), we have \( V_N \to \sigma_y^2 \left( X^\top X \right)^{-1} \) and \( \mathbf{w}_N \to \left( X^\top X \right)^{-1} X^\top \mathbf{y} \), as long as \( \left( X^\top X \right)^{-1} \) exists.

*A connection you may want to return to later:* The posterior becomes peaked around \( \left( X^\top X \right)^{-1} X^\top \mathbf{y} \), which turns out to be the least squares weight vector (when there is a unique solution). We’ve also given you this expression in Q3 of assignment 1, and will derive it more directly later in the course. However, for now we can sketch why this should be the least squares solution: The posterior is Gaussian, so its mean is equal to the mode, or the mode of the log posterior, which up to a constant is the sum of the log likelihood and log prior. As the prior gets broader, and tends to a constant for reasonable weights, we just maximize the log likelihood, which we can do by minimizing the sum of square residuals.

f) Now assume that our observations \( y \) are corrupted by additional Gaussian noise:

\[
p(\mathbf{z} \mid \mathbf{x}, \mathbf{w}) = \mathcal{N}(\mathbf{z}; \mathbf{y}, \sigma_z^2).
\]

So we now observe datasets \( D_z = \{ \mathbf{x}^{(n)}, \mathbf{z}^{(n)} \}_{n=1}^N \). What is the new posterior distribution \( p(\mathbf{w} \mid D_z) \)? *Hint:* Recall how you did Question 2.a) of this sheet.

**Answer:**
We have

\[
p(y \mid \mathbf{x}, \mathbf{w}) = \mathcal{N}(y; f(\mathbf{x}; \mathbf{w}), \sigma_y^2), \tag{25}
\]
or put differently,

\[ y = f(x; w) + \nu_y, \]  

(26)

where \( \nu_y \sim \mathcal{N}(0, \sigma_y^2) \). With \( p(z \mid x, w) = \mathcal{N}(z; y, \sigma_z^2) \), we also have

\[ z = y + \nu_z = f(x; w) + \nu_y + \nu_z, \]  

(27)

where \( \nu_z \sim \mathcal{N}(0, \sigma_z^2) \). Recall from Question 2.a) that sums of independent Gaussian random variables are again Gaussian distributed, with its mean being the sum of the two means and its variance being the sum of the two variances. Thus, we can express \( z \) as \( z = f(x; w) + \nu_y + \nu_z \sim \mathcal{N}(0, \sigma_y^2 + \sigma_z^2) \). So, all we have to do is to replace \( \sigma_y^2 \) with \( \sigma_y^2 + \sigma_z^2 \) in the posterior distribution and get \( p(w \mid D_z) = \mathcal{N}(w; w_N, V_N) \), where

\[ V_N = (\sigma_y^2 + \sigma_z^2)((\sigma_y^2 + \sigma_z^2)\frac{1}{\sigma_w^2}I + X^T X)^{-1}, \]  

(28)

\[ w_N = V_N \frac{1}{\sigma_w^2}w_0 + \frac{1}{\sigma_y^2 + \sigma_z^2} V_N X^T z. \]  

(29)

**Notes:**

For generating the data, we have drawn a \( w \) from the same prior distribution that we used for calculating the posteriors. In real applications, the prior distribution expresses beliefs (which could be misguided) about an unknown constant, or is chosen for mathematical convenience. An informative prior could reduce the amount of data we need to observe, but an overly-confident prior could give us an overly-confident posterior.