Statistical Machine Learning: Assignment 2

Joris van Vugt, s4279859

October 19, 2016

Exercise 1 – Sequential learning

Part 1 – Obtaining the prior

1. First we compute the precision matrix using `numpy.linalg.inv`:

$$\tilde{\mathbf{\Lambda}} = \tilde{\mathbf{\Sigma}}^{-1} = \left(\begin{array}{cc|cc} 60 & 50 & -48 & 38 \\ 50 & 50 & -50 & 40 \\ \hline -48 & -50 & 52.4 & -41.4 \\ 38 & 40 & -41.4 & 33.4 \end{array}\right) = \left(\begin{array}{c|c} \tilde{\mathbf{\Lambda}}_{aa} & \tilde{\mathbf{\Lambda}}_{ab} \\ \hline \tilde{\mathbf{\Lambda}}_{ba} & \tilde{\mathbf{\Lambda}}_{bb} \end{array}\right)$$

Now, using equations 2.73 and 2.75 from Bishop, we can compute the conditional covariance:

$$\mathbf{\Sigma}_p = \tilde{\mathbf{\Lambda}}_{aa}^{-1} = \begin{pmatrix} 0.1 & -0.1 \\ -0.1 & 0.12 \end{pmatrix}$$

and the conditional mean:

$$\begin{aligned}
\boldsymbol{\mu}_p &= \boldsymbol{\mu}_a - \tilde{\mathbf{\Lambda}}_{aa}^{-1}\tilde{\mathbf{\Lambda}}_{ab}(\boldsymbol{x}_b - \boldsymbol{\mu}_b) \\
&= \begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 0.1 & -0.1 \\ -0.1 & 0.12 \end{pmatrix} \begin{pmatrix} -48 & 38 \\ -50 & 40 \end{pmatrix} \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 1 \\ 2 \end{pmatrix} \right) \\
&= \begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 0.1 & -0.1 \\ -0.1 & 0.12 \end{pmatrix} \begin{pmatrix} -48 & 38 \\ -50 & 40 \end{pmatrix} \begin{pmatrix} -1 \\ -2 \end{pmatrix} \\
&= \begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 0.2 & -0.2 \\ -1.2 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ -2 \end{pmatrix} \\
&= \begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 0.2 \\ -0.8 \end{pmatrix} \\
&= \begin{pmatrix} 0.8 \\ 0.8 \end{pmatrix}
\end{aligned}$$

2.
```python
def generate_pair():
    return np.random.multivariate_normal([0.8, 0.8],
                                         [[0.1, -0.1],
                                          [-0.1, 0.12]])
```

$$\boldsymbol{\mu}_t = \begin{pmatrix} 0.28 \\ 1.18 \end{pmatrix}$$

3. To calculate the probability density of our multivariate Gaussian random variable, we use `scipy.stats.multivariate_normal` and its `pdf` method.

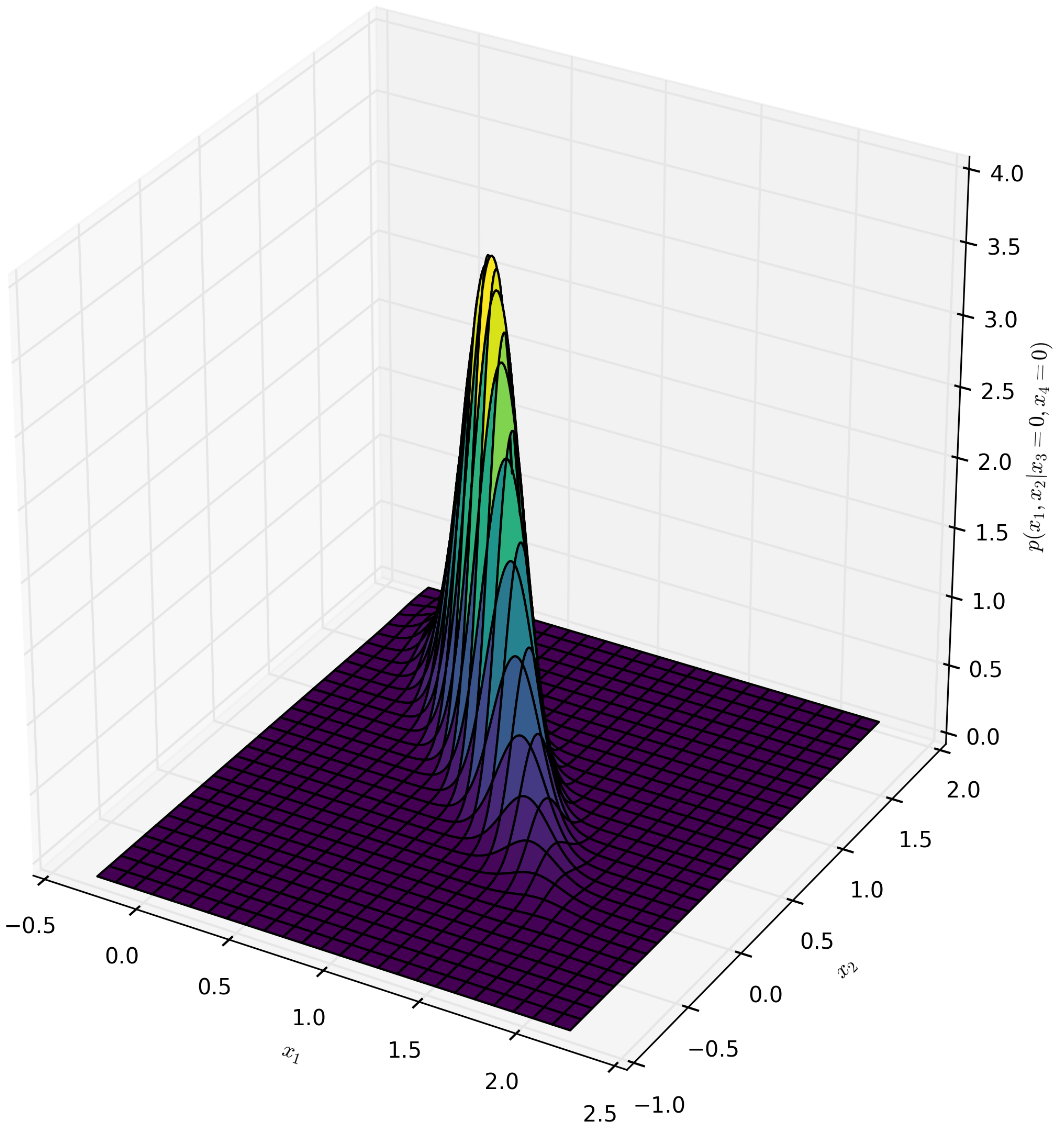

Figure 1: Probability density plot of the multivariate Gaussian

Part 2 – Generating the data

1.
```
N = 1000
data = np.random.multivariate_normal([0.28, 1.18],
                                     [[2.0, 0.8],
                                      [0.8, 4.0]],
                                     N)
np.savetxt('data.txt', data)
```

2. The maximum likelihood estimate of the mean is simply the mean of the observed data:

$$\boldsymbol{\mu}_{\mathrm{ML}} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n = \begin{pmatrix}0.25\\1.21\end{pmatrix}$$

Computing the maximum likelihood estimate of the covariance is slightly more involved:

$$\boldsymbol{\Sigma}_{\mathrm{ML}} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n - \boldsymbol{\mu}_{\mathrm{ML}})(\boldsymbol{x}_n - \boldsymbol{\mu}_{\mathrm{ML}})^T = \begin{pmatrix}2.023 & 0.828\\0.828 & 3.626\end{pmatrix}$$

To compute the unbiased maximum likelihood covariance estimate, we normalize by $N-1$ instead of N:

$$\boldsymbol{\Sigma}_{\mathrm{ML}} = \frac{1}{N-1}\sum_{n=1}^{N}(\boldsymbol{x}_n - \boldsymbol{\mu}_{\mathrm{ML}})(\boldsymbol{x}_n - \boldsymbol{\mu}_{\mathrm{ML}})^T = \begin{pmatrix}2.025 & 0.829\\0.829 & 3.629\end{pmatrix}$$

These results were obtained with the following code:

```
mu_ml = data.mean(axis=0)
x = data - mu_ml
cov_ml = np.dot(x.T, x) / N
cov_ml_unbiased = np.dot(x.T, x) / (N - 1)
```

Note that the left factor is transposed instead of the right factor in the covariance estimates. This is because our points are row vectors instead of column vectors (i.e., `data` has shape $N \times 2$).

We can compare our estimates to the true statistics:

$$\boldsymbol{\mu}_t = \begin{pmatrix}0.28\\1.18\end{pmatrix} \qquad \boldsymbol{\Sigma}_t = \begin{pmatrix}2.0 & 0.8\\0.8 & 4.0\end{pmatrix}$$

Our estimates are pretty close to their true values. The unbiased estimate of the covariance is not much closer than the biased estimate. Because N is relatively high, the slight change in normalization does not have much effect.

Part 3 – Sequential learning algorithms

1. Using equation 2.126 from Bishop

$$\boldsymbol{\mu}_{\mathrm{ML}}^{(N)} = \boldsymbol{\mu}_{\mathrm{ML}}^{(N-1)} + \frac{1}{N}(\boldsymbol{x}_N - \boldsymbol{\mu}_{\mathrm{ML}}^{(N-1)})$$

   we can come up with a Python procedure for sequential learning:

```python
mu_ml = 0
for i in range(N):
    mu_ml += (data[i]-mu_ml) / (i+1)
```

   The starting value $\boldsymbol{\mu}_{\mathrm{ML}}^{(0)}$ does not matter, but is arbitrarily set to 0. We divide by $i+1$ instead of just i, because Python uses 0-indexing.

2.