COMS30007 - Machine Learning

Linear Regression

Carl Henrik Ek

Week 3

Abstract

In this lab we will extend the example from last week to more complicated data. In this case we will look at a model that relates two continuous variables to each other also known as a *regression task*. To simplify things we will limit the hypothesis space and only look at relationships that are linear. Importantly the concepts of the lab is *exactly* the same as in the binary case and its important that you realise this. However differently from last weeks lab I will be less verbose when deriving the model and inference scheme so some parts are left for you to fill in.

We are interested in the task of observing a data-set $\mathcal{D} = \{\tilde{x}_i, y_i\}_{i=1}^{N}$ where we assume the following relationship between the variates,

$$y_i = f(\tilde{x}_i). \tag{1}$$

Our task is to infer the function $f(\cdot)$ from $\mathcal{D}$. To simplify things we are going to limit the hypothesis space to be only of linear functions. This means that we can write Eq 1 as,

$$y_i = w_1\tilde{x}_i + w_0 = \mathbf{w}^{\mathrm{T}}\mathbf{x}_i = \begin{bmatrix} w_1 \\ w_0 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \tilde{x}_i \\ 1 \end{bmatrix} \tag{2}$$

where we have rewritten the input variate by appending a one so that we can write everything on matrix form. The task that we will perform in this lab is to infer the function parametrised by $\mathbf{w}$ by observing $\mathcal{D}$.

1 Model

Now we will make our first assumption, we will assume that the data we observed is not instantiations of the "true" underlying function but rather have been corrupted by *addative* noise. This means that we get the following model,

$$y_i = \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + \epsilon. \tag{3}$$

Now can we make an argument what form this noise will take? One assumption would be to say that the noise is independent of where in the input space we evaluate the function, this is called *homogenous* noise. Furthermore we could argue that we know the form of the noise, one idea would be to say that the noise is Gaussian,

$$\epsilon \sim \mathcal{N}(0, \beta^{-1}),$$

with precision β. Now what this would mean is that if we could directly observe the noise we could formulate a likelihood as,

$$p(\epsilon) = \mathcal{N}(\epsilon|0, \beta^{-1}). \tag{4}$$

Now we can use Eq.3 and rewrite,

$$y_i = \mathbf{w}^{\mathrm{T}}x_i + \epsilon \tag{5}$$

$$y_i - \mathbf{w}^{\mathrm{T}}x_i = \epsilon \tag{6}$$

If we now combine this new expression of the noise with the assumption of the stochastic form in Eq.4 we get,

$$
\begin{aligned}
p(\epsilon) &= \mathcal{N}(\epsilon|0, \beta^{-1}) && (7)\\
&= \mathcal{N}(y_i - \mathbf{w}^{\mathrm{T}} x_i|0, \beta^{-1}) && (8)\\
&= \mathcal{N}(y_i|\mathbf{w}^{\mathrm{T}} x_i, \beta^{-1}). && (9)
\end{aligned}
$$

The last step can be done because we can "translate" a Gaussian distribution.

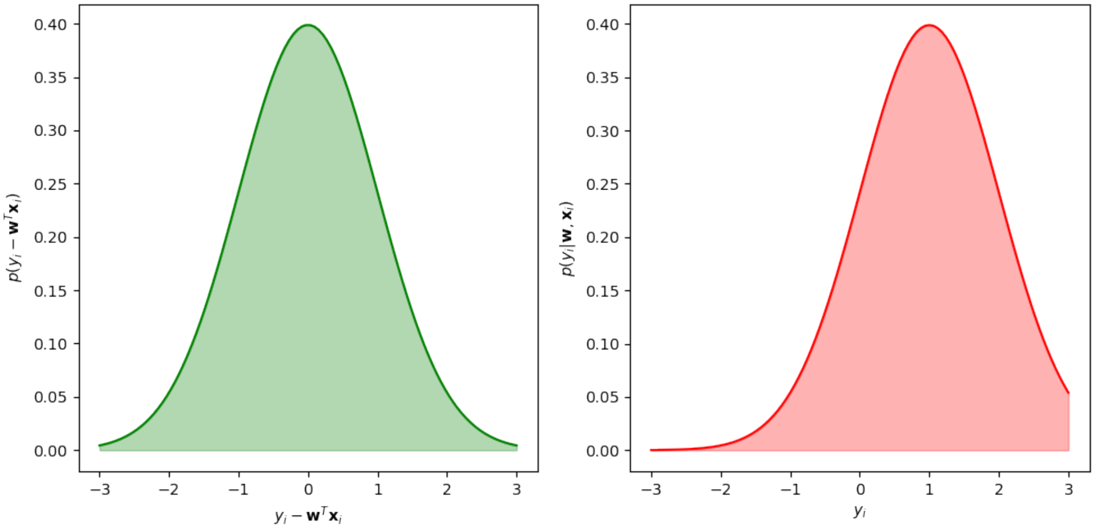

Figure 1: Figure showing how we can re-parametrise a Gaussian by translation. In the first case we have a Gaussian with mean zero over the difference between the observed output y_i and the output of the function $\mathbf{w}^{\mathrm{T}}\mathbf{x}_i$ which is the same as a Gaussian over the output y_i with mean $\mathbf{w}^{\mathrm{T}}\mathbf{x}_i$ where in this case the latter equates to 1.

What we have just done is to formulate a likelihood function, how likely is an observed output location given that we know the parametrisation of the function. Now the formulation is for a single data-point but also assuming that the noise is independent we can easily formulate the likelihood for a sequence of data,

$$
p(\mathbf{y}|\mathbf{w}, \mathbf{X}) = \prod_{i=1}^{N} p(y_i|\mathbf{w}, \mathbf{x}_i), \tag{10}
$$

where $\mathbf{X} = [x_1, \ldots, x_N]^{\mathrm{T}}$ and $\mathbf{y} = [y_1, \ldots, y_N]^{\mathrm{T}}$. Think back to the Bernoulli trial in last weeks lab, its exactly the same thing. One important difference is how we motivated the construction of the likelihood, it came through making an assumption of the structure of the noise. Concretely we made three assumptions,

1. the noise is addative
2. the noise is Gaussian
3. the noise is independent of the input location.

These assumptions needs to be justified on a problem to problem basis. Say for example that we are measuring the vibrations in a car as a function of speed. It might be the case that the noise in the sensor measuring the vibrations also depends on the speed of the car then we cannot use a homoscedastic noise model. In this case we use these assumptions because we want to show how to build models rather than looking at specific data.

So now we have our likelihood function and if we knew the parameters of the function $\mathbf{w}$ we would be able to generate data. Now our task is to recover the function by observing data. In order to do so we first need to specify our prior belief in the parameters.

2 Prior

In order to specify our prior distribution we will again use the concept of conjugacy. If we look at the likelihood in Eq.10 it is itself a Gaussian distribution[1]. We will also assume that the parameters of the noise β is known. Now we can again look-up what the conjugate prior is for a Gaussian with known variance[2]] as it turns out this in itself is another Gaussian distribution. So in order to exploit conjugacy we will use a Gaussian prior for the parameters of the function such that,

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}_0, \mathbf{S}_0). \tag{11}$$

The structure of the prior covariance tells us that we assume that the two parameters w_1 and w_0 are independent with equal variance. Think about this assumption, does this make sense for a line? Well the grate thing about distributions is that you can sample from it and generate the results.

[1] check this for yourself by writing up the product of the individual terms

[2] `https://en.wikipedia.org/wiki/Conjugate_prior`

Code

```
import numpy as np
import matplotlib.pyplot as plt

def plot_line(ax, w):
    # input data
    X = np.zeros((2,2))
    X[0,0] = -5.0
    X[1,0] = 5.0
    X[:,1] = 1.0

    # because of the concatenation we have to flip the transpose
    y = w.dot(X.T)
    ax.plot(X[:,0], y)

# create prior distribution
tau = 1.0*np.eye(2)
w_0 = np.zeros((2,1))

# sample from prior
n_samples = 100

w_samp = np.random.multivariate_normal(w_0.flatten(), tau, size=n_samples)

# create plot
fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(111)

for i in range(0, w_samp.shape[0]):
    plot_line(ax, w_samp[i,:])

# save fig
plt.tight_layout()
plt.savefig(path, transparent=True)
return path
```

1. what is the most likely line according to your prior belief?
2. what is the least likely line according to your prior belief?
3. is there any lines that have zero probability in this belief?

3 Posterior

Now we have encoded our prior belief and we have formulated our likelihood function and its time to formulate the posterior distribution. We will do exactly the same thing as we did in the Bernoulli trial but the math will be a bit more complicated. Go back and look at lecture 4 and make sure that you understand the derivation that we did as we will now use these results.

To derive the posterior distribution we will use two concepts,

1. that the posterior is proportional to the likelihood times the prior

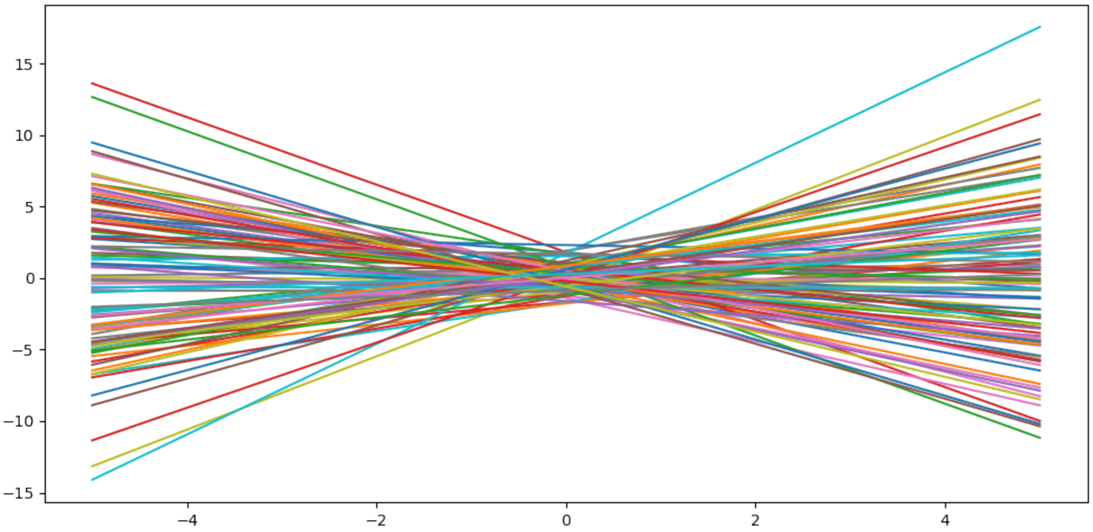

Figure 2: Samples from the prior with the prior covariance set to identitiy and prior mean of 0.

2. due to conjugacy we know that the posterior is a Gaussian

Proportionality means that,

$$p(\mathbf{w}|\mathbf{y},\mathbf{X}) \propto p(\mathbf{y}|\mathbf{X},\mathbf{w})p(\mathbf{w}). \tag{12}$$

Conjugacy means that we know the form of the right-hand side,

$$p(\mathbf{w}|\mathbf{y},\mathbf{X}) = \mathcal{N}(\mathbf{w}|\mu(\mathbf{y},\mathbf{X}),\sigma(\mathbf{y},\mathbf{X})). \tag{13}$$

Now what remains to be done is to first multiply the left-hand side of Eq.12 and identify the unknown terms $\mu(\cdot)$ and $\sigma(\cdot)$ in Eq.13. [3]. Doing so will lead to the posterior distrubution,

$$p(\mathbf{w}|\mathbf{y},\mathbf{X}) = \mathcal{N}\left(\mathbf{w}|\left(\mathbf{S}_0^{-1}+\beta\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\left(\mathbf{S}_0^{-1}\mathbf{w}_0+\beta\mathbf{X}^{\mathrm{T}}\mathbf{y}\right),\left(\mathbf{S}_0^{-1}+\beta\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\right). \tag{14}$$

The expression above looks rather daunting at first but actually does make a lot of sense when you start looking at it. One way of making sense of the posterior is to look at some extreme scenarios, think about the following

1. what would happen if you assume a noise-free situation i.e. $\beta \to \infty$
2. what would happen if we assume a zero mean prior?
3. what happens if we do not observe any data?
4. when you observe more and more data which terms are going to dominate posterior?

Make sure that the expression makes sense and that you build an intuition. We will now proceed to experiment with the expression

[3] look through the lecture hand-out for the full derivation

4 Implementation

Once you have done the mathematical interogation of the posterior it is time to evaluate the model by generating some data and then aim to recover the parameters that generated this specific data. Again, just the same proceedure as we did in the Bernoulli trial. The first thing we need to do is to decide on some paramters, let us assume that the data have been generated as,

$$y_i = \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + \epsilon \tag{15}$$

$$\epsilon \sim \mathcal{N}(0, 0.3) \tag{16}$$

$$\mathbf{X} = \begin{bmatrix} -1 & 1 \\ -0.99 & 1 \\ \vdots & \vdots \\ 1 & 1 \end{bmatrix} \tag{17}$$

$$\mathbf{w} = \begin{bmatrix} -1.3 \\ 0.5 \end{bmatrix} \tag{18}$$

First visualise the prior distribution over $\mathbf{w}$. As this is a two dimensional distribution we have to show this as an image. One simple way is to create a function that samples evaluates the distribution on a grid and then creates a contour plot,

Code

```
"""
Create a contour plot of a two-dimensional normal distribution

Parameters
----------
ax : axis handle to plot
mu : mean vector 2x1
Sigma : covariance matrix 2x2

"""

def plotdistribution(ax,mu,Sigma):
    x = np.linspace(-1.5,1.5,100)
    x1p, x2p = np.meshgrid(x,x)
    pos = np.vstack((x1p.flatten(), x2p.flatten())).T

    pdf = multivariate_normal(mu.flatten(), Sigma)
    Z = pdf.pdf(pos)
    Z = Z.reshape(100,100)

    ax.contour(x1p,x2p,Z, 5, colors='r', lw=5, alpha=0.7)
    ax.set_xlabel('w_0')
    ax.set_ylabel('w_1')

    return
```

Now we will do an iterative proceedure where we pick a random point from the data-set, compute and visualise the posterior and visualise the sample functions from the same distribution. So first combine the code to plot sample with the visualisation of the distribution and then generate a loop similar to this,

Code

```python
index = np.random.permutation(X.shape[0])
for i in range(0, index.shape[0]):
    X_i = X[index,:]
    y_i = y[index]

    # compute posterior
    # visualise posterior
    # visualise samples from posterior with the data
    # print out the mean of the posterior
```

You can iterate through this and add a pause statement in your loop or you can skip the loop completely and just run the code above by setting `i` as a variable and testing for interesting values. If this works as it should you should be able to regenerate the plots from the lecture. Observe what the mean of the posterior is, in an ideal setting we should eventually recover the parameters that generated the data. Play with the parameters of the model, the noise variance, the prior and get an intuitive feeling for how everything fits together.

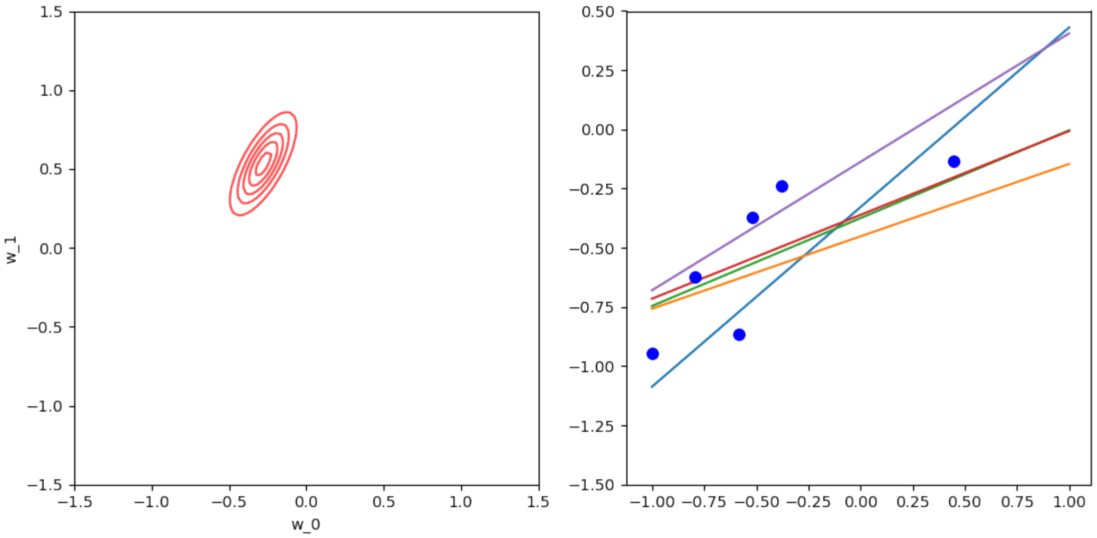

Figure 3: The right pane shows the posterior distribution after seeing the 6 data-points on the left. The left pane shows the samples from this posterior.

Now think about the following questions,

1. our prior is spherical, what assumption does this encode? Does this make sense for a line?
2. with a few data-points the posterior starts quickly to look non-spherical, what does this mean? Does this make sense?
3. with many data-points the posterior becomes spherical again? Why is this? Look at Eq.14 can you see why this is the case for this data?

5 Predictive Posterior

So far we have a way of learning the parameters of the function, but the parameters is just a means to an end, what we really want is to perform predictions. This means that given a new input location $\mathbf{x}_*$ we want to have a distribution over what we believe the output location to be. Of course this distribution should take into account the training data we have used to learn the weights of the function. The way to get to this point is to *marginalise out* the parameters of the function $\mathbf{w}$. In other words generate all possible lines and weigh them with how much we believe in each of them based on what we have learned. This can be done as follows,

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \int p(y_*|\mathbf{x}_*, \mathbf{w})p(\mathbf{w}|\mathbf{X}, \mathbf{y})\mathrm{d}\mathbf{w}, \tag{19}$$

where we integrate the likelihood for a new point with the posterior distribution over the parameters. Being that both these are Gaussian we can compute this integral again in closed form which leads to the following distribution,

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}\left(\mathbf{y}|\mathbf{m}_N^{\mathrm{T}}\mathbf{X}, \beta^{-1} + \mathbf{X}^{\mathrm{T}}\mathbf{S}_N\mathbf{X}\right), \tag{20}$$

where $\mathbf{m}_N$ and $\mathbf{S}_N$ is the posterior mean and variance for $\mathbf{w}$ having seen N points from the training data. As they have this nice dependency just write a new function that wraps the posterior over the weights something like,

Code

```
def predictiveposterior(m0, S0, beta, x_star, X, y):
    mN, SN = posterior(m0, S0, beta, X, y)

    m_star = mN.T.dot(x_star)
    S_star = 1.0/beta + x_star.T.dot(SN).dot(x_star)

    return m_star, S_star
```

6 EXTRA Non-linear basis functions

If you are interested you can extend the linear case and work with a non-linear basis function instead. The code should be exactly the same you just first need to map the input data to a different space Z and then perform the linear regression there,

$$\Phi : \tilde{\mathcal{X}} \rightarrow \mathcal{Z} \tag{21}$$

$$y_i = \mathbf{w}^{\mathrm{T}}\Phi(\mathbf{x}_i) + \epsilon \tag{22}$$

A simple example of a mapping is to use an exponential function as,

$$\phi_d(\tilde{x}_i) = e^{-(\tilde{x}_i - b_d)^{\mathrm{T}}(\tilde{x}_i - b_d)},$$

where b_d is the "center" of the basis function. You can distribute these linearly along the input space, say that if you pick 10 basis functions you will do linear regression in a 10 dimensional space but map the result back to the original problem space where the 10 dimensional line will look like a non-linear function. It is a bit mind boggling to get your head around this one so experiment till you understand the concept. We will look at this more in future labs and already next lab we will take this to the extreme and let the number of basis-functions go towards infinity and then some really cool stuff starts to happen. But don't worry to much about the interpretation for now just justify for yourself that you do get non-linear functions when doing this approach.

7 Conclusion

Now you have reached the end of this lab and hopefully you manged to generate the plots and have understood the connection between the assumptions, the mathematical inference proceedure and the results. Even though it might feel like a very simple example it really is not, you have taken prior knowledge, combined this with data and generated new knowledge. Now we will move onto more complicated models but it is important that you see that it is all the same proceedure.