CHAPTER 1

REGRESSION

Machine learning is the study of systems that can learn tasks from data, without explicit instructions. Algorithms are typically deterministic- we specify a bunch of steps that need to be followed sequentially to produce a valid output. However, in Machine Learning, the programs we write are general, which can be applied to all sorts of data. The machine then learns from the data without any instruction as to how it should do so, other than by specifying the actual algorithm (which is very generic).

We will now illustrate the difference between a machine learning and a non-machine learning algorithm. Assume that we want to find out whether a picture has a red square. We can construct an algorithm for this easily- go through all the pixels and specify a range of rgb values to determine whether it is red, and ensure that they form a square. This is a very specific algorithm with full guaranteed accuracy. On the other hand, we can use Machine Learning for it. To do so, we would need to train the model, i.e. give it a bunch of pictures (with and without a red square) and say whether they have a red square. We can then use the model to determine whether the given picture contains a red square. This is a very generic algorithm, but does not have a performance guarantee. Note that the specific algorithm might not always be possible, e.g. to detect an animal, so the Machine Learning approach might be the best option.

Machine Learning relies on data. The data is a set of observations about some objects. We might want to make some predictions about these objects, e.g. predict an attribute given the others or a future using the past information. Alternatively, we might want to group the objects into different categories, labelled or unlabelled. Machine Learning can hence be thought of as a set of algorithms. So, it is important to understand how they work so that they can be correctly and effectively used.

Machine Learning has various applications, both in the real world and in academia. These include self-driving cars, recommender systems and autocorrect. ML is the better choice in many cases since there would be too many rules to write if we tried to create an algorithm for it, and it would be much easier to just see what would be the right thing to do in each scenario that a model could learn.

There are 2 types of learning- supervised and unsupervised learning. In both cases, we have a dataset. In supervised learning, there is a correct output associated to each data value, e.g. another value (a prediction). There are 2 types of supervised learning- regression and classification. In regression, the output we want to produce is a continuous value (e.g. a real number such as height, time taken, etc.). In classification, the output is a discrete set of values (e.g. yes or no). In unsupervised learning, we need to find a group of similar objects without knowing where the given values belong. There are 2 main types of unsupervised learning- clustering and projection. In clustering, where we place data into different clusters. Unlike with classification, we are not told

what values belong in the same group. In projection, we take a dataset and for each value, reduce the number of variables while preserving as much data as possible.

1.1 Linear and Polynomial Regression

Consider the following plot: The plot shows the winning times for the Men's Olympics 100m sprint from 1896 to 2008. We will use regression to predict the winning time for 2012. We do this by drawing a straight line through the data, and then reading the value at 2012. Formally, we first decide on the model to use to predict. In this case, we use a linear model. We then need to fit the model, and extended it to 2012 and evaluate the value. This gives the prediction.

When using linear regression, we ware making the assumption that there is a relationship between the two variables, and that the relationship is linear. It is quite reasonable to assume the existence of a relationship- we cannot do Machine Learning without it! On the other hand, a linear relationship is one that needs to be evaluated. We will later quantise how good it is. We also assume that the relationship holds in the future.

This is supervised learning- we are given the input and the output for the training points (i.e. the values before 2008 are known, both the year and the time). The attribute is the olympic year and the target is the winning time. We normally use the x-variable for the olympic year, and t-variable for the winning time. A model is a function that relates x to t, i.e. $t = f(x)$. We want to construct the model (i.e. the function f) and then predict the value at 2012 (i.e. compute $f(2012)$). These are N attributes (x_n, t_n), called the training data.

Assuming that the model is linear, it is of the form $f(x; w_0, w_1) = w_0 + w_1 x$. The values w_0 and w_1 are parameters of the model. Our goal is to choose the values w_0 and w_1 that 'fit' the training data. To do so, we need to quantise a good line- whether it passes through the line well or not. This can be done by computing the distance between the data and the line. In particular, if we have a model function f and the data points $(x_1, t_1), \ldots, (x_N, y_N)$, then the 'badness' of the model can be defined by the value

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} (t_n - f(x_n; w_0, w_1))^2.$$

We typically use the squared distance instead of absolute value. This is the called the mean squared loss of the model. The smaller the value $\mathcal{L}$, the better the model is. We can consider $\mathcal{L}$ as a function of w_0 and w_1, and then optimise it to compute the line of best fit. In particular, we want

$$\underset{w_0, w_1}{\operatorname{argmin}} \mathcal{L} = \underset{w_0, w_1}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^{N} (t_n - f(x_n; w_0, w_1)^2).$$

This can be computed using some optimisation algorithm, e.g. gradient descent. In this case, the loss function (with respect to w_0 and w_1) is convex, so we can find it instantly.

For a scalar-valued function $f(x_1, \ldots, x_n)$, its gradient at a point is given by

$$\Delta f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

is the vector of partial derivatives evaluated at some point. It points in the direction of steepest ascent, so taking its negative gives us the direction of steepest descent. We exploit this property to compute the (local) minimum. The algorithm chooses a starting point, and then uses the gradient to descend to the local minimum, until the value converges.

After doing gradient descent, for the Olympics data, we get $w_0 = -0.0133$ and $w_1 = 36.416$. Hence, the model is $t = 36.416 - 0.0133x$. So, we get the prediction 9.5947 seconds.

When making this prediction, we made the following assumptions:

- there is a relationship between Olympic year and winning time;
- the relationship is linear; and
- the relationship will continue into the future.

It is reasonable to assume that the relationship exists, e.g. people have gotten healthier over time and hence their speed has improved. This assumption must be made, or else there is nothing to be learned. The relationship however need not be linear. Moreover, the relationship cannot continue on in the future forever- the running time cannot be negative, for instance. However, we can reasonably expect the relation to still hold for the near future. It might not matter if the assumptions we make are wrong if they give us a reasonable prediction!

We will now generalise this model to linear models with multiple variables, and then non-linear models. Instead of the linear model $t = w_0 + w_1 x$, we can represent it as a vector:

$$t = \begin{bmatrix} w_0 & w_1 \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} = \mathbf{w}^T \mathbf{x}.$$

The mean loss is then given by

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} (t_n - \mathbf{w}^T \mathbf{x}_n)^2 = \frac{1}{N} \mathbf{q}^T \mathbf{q},$$

where entries in q are given by $q_n = t_n - \mathbf{w}^T \mathbf{x}_n$, i.e.

$$\mathbf{q} = \begin{bmatrix} t_1 - \mathbf{w}^T \mathbf{x}_1 \\ t_2 - \mathbf{w}^T \mathbf{x}_2 \\ \vdots \\ t_N - \mathbf{w}^T \mathbf{x}_N \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix} - \begin{bmatrix} \mathbf{w}^T \mathbf{x}_1 \\ \mathbf{w}^T \mathbf{x}_2 \\ \vdots \\ \mathbf{w}^T \mathbf{x}_N \end{bmatrix} = \mathbf{t} - X\mathbf{w},$$

with

$$X = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix}.$$

So, the mean loss is given by

$$\mathcal{L} = \frac{1}{N} (\mathbf{t} - X\mathbf{w})^T (\mathbf{t} - X\mathbf{w}).$$

In this format, it is much easier to generalise it to multiple variables and non-linear parameters. For instance, if we want more inputs, i.e. $t = w_0 + w_1 x + w_2 y$, then we can set

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}, \qquad \mathbf{x}_n = \begin{bmatrix} 1 \\ x_n \\ y_n \end{bmatrix}, \qquad X = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \vdots & \vdots & \vdots \\ 1 & x_N & y_N \end{bmatrix}.$$

The loss function is still

$$\mathcal{L} = \frac{1}{N}(\mathbf{t} - X\mathbf{w})^T(\mathbf{t} - X\mathbf{w}).$$

Alternatively, we can also increase the power, i.e. $t = w_0 + w_1 x + w_2 x^2 + \cdots + w_K x^K$. In that case,

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_K \end{bmatrix}, \qquad \mathbf{x}_n = \begin{bmatrix} 1 \\ x_n \\ x_n^2 \\ \vdots \\ x_n^K \end{bmatrix}, \qquad X = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^K \\ 1 & x_2 & x_2^2 & \dots & x_2^K \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \dots & x_N^K \end{bmatrix}.$$

We can minimise the loss using derivatives in the vector representation. We have the loss function:

$$\mathcal{L} = \frac{1}{N}(\mathbf{t} - X\mathbf{w})^T(\mathbf{t} - X\mathbf{w}).$$

So, we can take partial derivative with respect to $\mathbf{w}$, and set it to 0:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \nabla_w \mathcal{L} = 0.$$

The possible vectors we get are local extrema, which includes a local minimum if it exists. We can solve the vector equation by taking inverses. In particular,

$$\mathbf{w} = (X^T X)^{-1} X^T \mathbf{t}.$$

The vector $\mathbf{w}$ is the global minimum (since the loss equation is convex with respect to $\mathbf{w}$), and can be computed without any iterative optimisation method. For the Olympics data we saw, we have

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \qquad X = \begin{bmatrix} 1 & 1896 \\ 1 & 1900 \\ \vdots & \vdots \\ 1 & 2008 \end{bmatrix}, \qquad \mathbf{t} = \begin{bmatrix} 12.00 \\ 11.00 \\ \vdots \\ 9.85 \end{bmatrix}.$$

Hence,

$$\mathbf{w} = (X^T X)^{-1} X^T \mathbf{t} = \begin{bmatrix} 36.416 \\ -0.0133 \end{bmatrix}.$$

This is identical to the gradient descent solution, but was computed much more effectively. Using this approach, we can fit an 8th order model to the Olympic

data: Although it matches the given data quite well, it grows exponentially after 2008.

Our models could be further generalised from polynomials- they can be arbitrary functions of the data x, e.g. $t = w_0 + w_1 x + w_2 \sin x + w_3 x^{-1}$. There are some basis functions that can be used in general. These include:

- polynomial functions $h_k(x) = x^k$;
- radial basis functions

  $$h_k(x) = \exp\left(-\frac{(x-\mu_k)^2}{2s^2}\right),$$

  where μ_k is defined individually for each k, and s shared among all k- these look like bell curves with a peak at μ_k;
- signmoid functions

  $$h_k(x) = \sigma\left(\frac{(x-\mu_k)^2}{s}\right), \qquad \sigma(a) = \frac{1}{1+\exp -a}.$$

  This is a continuous step function with value between 0 and 1, with μ_k determining where the function is 1/2.

We can make predictions by computing the vector $\mathbf{t}_{new}$ using the value $\mathbf{w}$, i.e. $\mathbf{t}_{new} = \mathbf{w}^T \mathbf{x}_{new}$. If there are more basis functions than the dataset, then the matrix will not have an inverse, and so $\mathbf{w}$ cannot be predicted using this method. This is because it does not have a unique solution, e.g. if we have 1 observation and want to fit a line, then there are many lines that we could choose- it just has to go through the point.

If we have multi-dimensional data, we can flatten it to a single one-dimensional vector, e.g. an image of 2D rgb vectors into a 1D vector. It is likely that when we do so, the vector will have a lot of elements. Hence, it is possible that we do not have enough dataset to truly learn the data. Moreover, for images, not every pixel matters equally; it is typically quite complicated. A basis function is typically not powerful enough to capture any interesting property, e.g. to guess whether the image has a cat or not. To mitigate this, we can:

- normalise and standardise to highlight differences in images, e.g. center the faces in an image, subtract average colour, divide by the standard deviation;
- remove redundancy by subsampling, converting to greyscale or some other simpler representation;
- extract features, e.g. blob attributes, locations (which have much smaller dimension than an actual image).

If we have multiple attributes in data, they are expected to be quite different, e.g. temperature and year for Olympics data. This discrepancy can create problems in general, e.g. we cannot choose the same step value for both of the values. To mitigate this, we need to subtract the mean and divide by the standard deviation.

For the Olympics data, if we use a linear model, the prediction for 2012 was quite good. If we instead use an 8th order model, then the model is a good fit for the data, but the prediction is poor. Hence, it is better to use the linear model than the 8th order model. In general, we would want a model to not just fit the training data, but also the unseen data- this is called generalisation. In our case, the linear model generalises better than the 8th order.

We will now consider how to choose between the linear model and the 8th order model. We minimised the value $\mathcal{L}$ when constructing the model, so perhaps we can choose the model which has a smaller loss. The graph below illustrates how the loss varies for different polynomial-order models: As we can see, the higher the order of the model, the lower the loss. Hence, the 8th order model has a smaller loss than the linear one, so this won't work. In general, as the model becomes more complex, the loss gets smaller.

There is a trade-off between generalisation (i.e. how effectively we can predict) and over-fitting (i.e. how much we can decrease the loss value). The dataset will typically have some noise, and we do not want to fit the noise; just the actual data. Hence, if we minimise the loss $\mathcal{L}$, then the prediction won't necessarily be better.

To evaluate whether a model works well, we need to evaluate it on some data. We do this by subdividing the given data into training and validation data. In particular, if we have N data values, we can train it on $N - C$ data values (training data) and then check how well it can be used to predict the remaining C values (validation data). For instance, in the Olympics data, we can train it using data up to 1980, and then validate it with the remaining points. The following graph shows how the validation loss changes with the polynomial models. Clearly, the mean squared error on the validation data is the lowest for the linear model, and the model gets worse as the order of the polynomial increases.

We will now consider how to partition the dataset into training and validation data. For data related to time, we want to predict the future, so it makes sense to check whether the most recent data is accurate. If this pattern does not exist, then we can randomly choose the points. Also, we can do it many times and then take an average model.

We can do this more systematically using cross-validation. Here, we partition the data into N chunks. Then, we take out the first chunk and train on the remaining chunks; we validate using the first chunk. This is called the first fold. We repeat this for each of the chunk. We then average on each of the folds. This method allows us to choose every data point as a validation point (precisely once). In the extreme, each partition has precisely 1 data point, i.e. we validate on a single point. This is called the leave-one-out cross-validation. For the Olympics data, we get the following loss for this cross-validation technique: Here, the 3rd order is said to be better than linear. This is the best possible use of the available data, but it might be quite inefficient if there are too many data points (we have to train the model N times, or even in general, C times; for a small C, this might be manageable). Many approaches might give us different optimal models, and it is up to us to make the right choice using context, e.g. for the Olympics data, it makes more sense to use a linear model.

1.2 Noise in data

In this section, we will look at noise in data and how to model it. We will add to our model a term to represent noise. As the noise behaves randomly, we will treat noise as a random variable. We will use likelihood instead of loss in order to find the correct model. That is, we find the parameters that maximise the likelihood instead of minimising the loss. Moreover, we can compute the confidence in our predictions, and in our parameter estimates.

We will consider the Olympics data again. Assume now that the model is $t_n = \mathbf{w}^T\mathbf{x}_n + \epsilon_n$, where ϵ_n is the noise. Note that the value ϵ_n depends on the value n, and can be either positive or negative. Moreover, there is no relation between ϵ and n, which makes it hard to model them.

Since we cannot model the noise ϵ_n, we can treat it as a random variable. A random variable is a mathematical representation of a quantity that is uncertain. An example of a random variable is the result of a coin toss- there are 2 discrete values it could take. In general, we know the possible values a random variable might take, and how likely they are.

There are 2 types of random variables- discrete and continuous. A discrete random variable is one that takes up countably many outcomes, e.g. coin toss, rolling a die, word in a document, etc. A continuous random variable is one that takes up uncountably many outcomes, e.g. the outcome is a real number such as the winning time in the Olympic data.

For discrete random variables, we denote by $P(X = x)$ (or $P(x)$) the probability that the random variable X is equal to the outcome x. For example, $P(X = 1) = 0.5 = P(X = 0)$ for a fair coin (1 for heads, 0 for tails), and $P(Y = y) = \frac{1}{6}$ for $y \in \{1, \ldots, 6\}$ for a fair dice. We note that a probability value is between 0 and 1, and the sum of all outcomes is 1.

For continuous random variables, the value $P(x)$ denotes the distribution of the probability function around x- it is not the probability of x occurring. We can compute the probability for a range by integrating, e.g.

$$P(6 \leq X \leq 8) = \int_{x=6}^{x=8} p(x)\ dx,$$

where p is the probability density function. Like with the discrete random variables, we require $p(x) \geq 0$, but not $p(x) \leq 1$. Instead, we require

$$\int_{-\infty}^{\infty} p(x)\ dx = 1.$$

If we have two discrete random variables X and Y, then the probability $P(X = x, Y = y)$ is the probability that the random variable X has value x, and that the random variable Y has value y. This is the joint probability. We can similarly define joint density for continuous random variables.

Let X be the random variable for the toss of a coin (1 for heads, 0 for tails) and Y the random variable for rolling a dice. Then, $P(X = 1, Y = 3)$ is the probability that the coin is heads and the dice lands on a 3. These two events are independent, which means in general that

$$P(X = x, Y = y) = P(X = x) \cdot P(Y = y).$$

If two events are not independent, then the equation above doesn't hold, and they are called dependent.

Conditional probabilities are those that consider an event given another event is true. This is denoted $P(X = x|Y = y)$, which is the probability that $X = x$ given $Y = y$. We note that

$$P(X = x, Y = y) = P(X = x|Y = y) \cdot P(Y = y),$$

i.e. the probability of both x and y is the probability of y and the probability of x given y.

Our model is now $t_n = \mathbf{w}^T\mathbf{x} = \epsilon_n$. The value ϵ_n is a continuous value, so we need to choose some density function for it. This is typically a Gaussian distribution:

$$p(\epsilon|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}(\epsilon - \mu)^2\right),$$

and depends on the mean μ and the variance σ^2. Some graphs for different values of μ and σ are given below: We denote the Gaussian distribution with mean μ and variance σ^2 by $\mathcal{N}(\mu, \sigma^2)$.

Until now, we have used the data to produce a model. But, we can also get data using the model. In particular, our model is $t_n = \mathbf{w}^T\mathbf{x} + \epsilon_n$. We can fix the value $\mathbf{w}$ and a probability density for ϵ_n. Typically, we set $\mu = 0$ assuming that the error is equally likely to be positive or negative; any different value μ can be absorbed into the value $\mathbf{x}$. We can sample ϵ_n fo each $\mathbf{x}_n$ from the probability density function, and compute the value t_n. Instead of just sampling from $\mathcal{N}(0, \sigma^2)$ and adding $\mathbf{w}^T\mathbf{x}_n$, we could combine all the steps by sampling from $\mathcal{N}(\mathbf{w}^T\mathbf{x}_n, \sigma^2)$. Hence, t can be thought of as a random variable as well.

The value we get when we evaluate the density $\mathcal{N}(\mathbf{w}^T\mathbf{x}_n, \sigma^2)$ is called the likelihood. Note that it is not a probability (for continuous random variables). When we compute the model, we want to maximise likelihood, with respect to σ^2 and $\mathbf{w}$ (i.e. the t_n and $\mathbf{x}$ values are fixed).

We will now consider how to compute the likelihood for a model. We can use the generative process to compute the likelihood of a value, i.e. using $p(T_n = t_n|\mathbf{w}, \mathbf{x}_n, \sigma^2)$. To compute the value for the entire model, we use their joint likelihood, i.e.

$$p(T_1 = t_1, \ldots, T_N = t_N \mid \mathbf{w}, \sigma^2, \mathbf{x}_1, \ldots, \mathbf{x}_N).$$

We assume that all the noise values are independent, so we find that

$$p(T_1 = t_1, \ldots, T_N = t_N) = \prod_{n=1}^{N} p(T_n = t_n).$$

We want to optimise this value, i.e. compute

$$\underset{\mathbf{w},\sigma^2}{\operatorname{argmax}} \prod_{n=1}^{N} p(T_n = t_n \mid \mathbf{w}, \mathbf{x}_n, \sigma^2).$$

It is more computationally efficient to compute the log likelihood. So, we can optimise that value, i.e. compute

$$\underset{\mathbf{w},\sigma^2}{\operatorname{argmax}} \sum_{n=1}^{N} \log p(T_n = t_n \mid \mathbf{w}, \mathbf{x}_n, \sigma^2).$$

For Gaussian distribution, we have

$$\log L = \log \prod_{n=1}^{N} p(t_n \mid \mathbf{w}, \mathbf{x}_n, \sigma^2) = -N \log(\sigma\sqrt{2\pi}) - \frac{1}{2\sigma^2} \sum_{n=1}^{N} (t_n - \mathbf{w}^T \mathbf{x}_n)^2.$$

The values $-N \log(\sigma\sqrt{2\pi})$ and $\frac{1}{2\sigma^2}$ are constants, so the argmax just depends on

$$\underset{\mathbf{w},\sigma^2}{\operatorname{argmin}} (t_n - \mathbf{w}^T \mathbf{x}_n)^2.$$

We have solved this optimisation problem before, using the value $\hat{\mathbf{w}} = (X^T X)^{-1} X^T \mathbf{t}$. After having found the optimal $\mathbf{w}$ value, we can compute the optimal σ by

$$\hat{\sigma}^2 = \frac{1}{N} (\mathbf{t} - X\mathbf{w})^T (\mathbf{t} - X\mathbf{w}).$$

For the Olympics data, we find $\hat{\sigma}^2 = 0.0503$.

Because of the connection between the log likelihood function and the loss function, neither is better than the other to choose models. In particular, we have to use validation data to test the model. Nonetheless, it is possible to use the likelihood of the validation data to check the accuracy of the model; it is not possible to use the likelihood of the training data. Because of overfitting, the noise value σ^2 decreases as we make the model more complex.

In general, a loss curve might have more than one best value. The different values give rise to different predictions, and we might want to consider all of the possible solutions, along with their probabilities. We can find different predictions and take a weighted average of them. The higher the number of predictions, the better our average prediction.

We can generalise this by treating $\mathbf{w}$ itself as a random variable. We need to find its density to compute $p(\mathbf{w})$. Assuming that this exists, we can average over every possible $\mathbf{w}$. This is the expectation value:

$$\mathbf{E}_{p(\mathbf{w})} = \int f(\mathbf{x}_{new}, \mathbf{w}) p(\mathbf{w}) \, d\mathbf{w}.$$

The expectation value is the average of predictions from each possible $\mathbf{w}$ weighted by how likely the $\mathbf{w}$ value is.

The value $\mathbf{w}$ depends on some data, in particular the training data. So, we want to compute $p(\mathbf{w}|X, \mathbf{t})$. We know the likelihood $p(\mathbf{t}|\mathbf{w}, X, \sigma^2)$. So, to compute this value, we can use Bayes' Rule:

$$p(\mathbf{w}|X, \mathbf{t}) = \frac{p(\mathbf{t}|X, \mathbf{w}) \cdot p(\mathbf{w})}{p(\mathbf{t}|X)}.$$

The value $p(\mathbf{w}|X, \mathbf{t})$ is the posterior density and the value $p(\mathbf{t}|X, \mathbf{w})$ is the likelihood. Moreover, the value $p(\mathbf{w})$ is the prior and $p(\mathbf{t}|X)$ the marginal likelihood. The prior value must be chosen ourselves, based on how we believe the data will behave. The margin likelihood normalises and ensures that the probability is equal to 1, and it can be hard to compute. Typically, $\mathbf{w}$ is not part of the marginal likelihood and can be ignored. For now, all of our examples will feature a computable marginal likelihood. A prior $p(\mathbf{w})$ is conjugate to a likelihood $p(\mathbf{t}|X, \mathbf{w})$ if their product $p(\mathbf{w}) \cdot p(\approx|X, \mathbf{w})$ has the same type of density as the prior.

We will illustrate this for the Olympics data. The likelihood will be Gaussian, i.e. $p(\mathbf{t}|\mathbf{w}, X, \sigma^2) = \mathcal{N}(X\mathbf{w}, \sigma^2\mathbf{I})$. We choose a prior:

$$p(\mathbf{w}) = \mathcal{N}(0, S), \qquad S = \begin{bmatrix} 100 & 0 \\ 0 & 5 \end{bmatrix}.$$

The prior values allow for a lot of flexibility. The years will be normalised (i.e. they will start at 1 and increase by 1 every time). We get the following posterior: The plot on the left gives us the contour plot for the values w_0 and w_1. The closer it is to the ellipses, the more likely it is. The values we get by maximum likelihood/minimum loss corresponds to the center of all the ellipses. On the right, some of the plots are given.

There are 2 interpretations of probability- Frequentist and Bayesian. In Frequentist probability, we interpret the value $p(v)$ to be what we expect to happen when the event happens infinitely many times, e.g. the probability of a coin toss landing on heads is 50% if we throw it infinitely many times. On the other hand, in Bayesian approach, we update our confidence in the result, e.g. I have 50% confidence that the coin will land on heads when I flip it once. The probability represents a degree of belief in this scenario. We have taken the Bayesian approach when applying Bayes' Rule above. In particular, we update the prior belief we have using the likelihood (and margin likelihood) to get the posterior belief.

In general, Bayesian linear regression takes a prior belief about the random variable $\mathbf{w}$. We want the prior to be as general as possible, i.e. allowing all straight lines. If we had some knowledge about the lines in general (e.g. they have positive gradient), then we can make use of this information in the prior. We also need the likelihood- we have seen above how to compute the likelihood $p(\mathbf{t}|X, \mathbf{w})$ using the training data. We can then compute the posterior belief using Bayes' Rule.

If the prior is $\mathcal{N}(0, 1)$, then the posterior belief we get are likely to be shallow lines because of the prior. To avoid this bias, we can use uniform distribution over polar coordinates. The prior has little but significant effect, so we should choose it carefully to allow some possibilities to be allowed.

We looked at Bayesian linear regression to average over all the predictions over the possible values of $\mathbf{w}$. This is given by

$$\mathbf{E}_{p(\mathbf{w}|X,\mathbf{t},\sigma^2)} f(\mathbf{w}) = \int f(\mathbf{w}) p(\mathbf{w}|\mathbf{t}, X, \sigma^2)\, d\mathbf{w}.$$

In the model, $f(\mathbf{w}) \sim \mathcal{N}(\mathbf{w}^T\mathbf{x}_{new}, \sigma^2)$. We find that

$$p(\mathbf{t}_{new}|X, \mathbf{t}, \mathbf{x}_{new}, \sigma^2) = \mathcal{N}(\mathbf{x}_{new}^T, \sigma^2 + \mathbf{x}_{new}^T \Sigma x_{new}),$$

where

$$\mu = \frac{1}{\sigma^2}\Sigma X^T\mathbf{t}, \qquad \Sigma = \left(\frac{1}{\sigma^2}X^TX + S^{-1}\right).$$

We can plot the prediction for t_{new} that we get for the Olympics data- this is given below: This is the predictive density. We can plot the predictions over time. The pink strip gives the possible spread of the lines- note that as the year gets larger, the pink strip gets larger, i.e. we are less certain about the future.

We have mostly ignored the marginal likelihood $p(\mathbf{t}|X, \sigma^2)$ in Bayes' Rule. It is given by

$$p(\mathbf{t}|X, \sigma^2) = \int p(\mathbf{t}|X, w, \sigma^2)p(\mathbf{w})\ d\mathbf{w}.$$

We are averaging over all $\mathbf{w}$ to get a value for how good the model is, i.e. how likely is $\mathbf{t}$ given X and the model family. We can use this to compare model families (e.g. linear or polynomial). In general, if prior is $\mathcal{N}(\mu_0, \Sigma_0)$ and likelihood $\mathcal{N}(X\mathbf{w}, \sigma^2 I)$, the marginal likelihood is given by

$$p(\mathbf{t}|X, \sigma^2, \mu_0, \Sigma_0) = \mathcal{N}(X\mu_0, \sigma^2 I + X\Sigma_0 X^T).$$

This is an N-dimensional Gaussian evaluated at $\mathbf{t}$.

We can consider how the marginal likelihood favours the correct model. Consider the following graph: On the left, we have a cubic curve and a some samples created from it. On the right, we have the marginal likelihood for polynomial models of different order. Clearly, it can decode that the polynomial should be used on a 3rd order polynomial.

We know that the posterior $p(\mathbf{w}|X, \mathbf{t})$ incorporates observations and prior knowledge. In particular, the posterior is proportional to the product of likelihood and prior. We do not need to compute the entire posterior distribution; we can just compute the mode of the posterior (the value that is most likely to occur). This is called the maximum a posteriori (MAP) approach. We can use the MAP approach if the marginal likelihood is too hard to calculate, and it is also computationally more efficient.

For Olympics data, we had a Gaussian prior and likelihood, so the posterior is also Gaussian. Hence, the mode is equal to the mean (since the Gaussian model is symmetric). Hence, the MAP estimate of $\mathbf{w}$ is

$$\mathbf{w}^* = \frac{1}{\sigma^2}\Sigma X^T \mathbf{t}.$$

So, the predictive distribution is given by $\mathcal{N}((\mathbf{w}^*)^T \mathbf{x}_{new}, \sigma^2)$. We can use this to compute $\mathbf{t}_{new}$. This is quite closely related to the maximum likelihood- as the prior becomes more generic, the value $\mathbf{w}^*$ gets closer to the MLE.

If the likelihood is Guassian, then

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmax}}\, p(\mathbf{w}|X, \mathbf{t}) = \underset{\mathbf{w}}{\operatorname{argmax}} \log p(\mathbf{w}|X, \mathbf{t}).$$

So,

$$\log p(\mathbf{w}|X, \mathbf{t}) = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(\mathbf{t}_n - w^T \mathbf{x}_n)^2 - \frac{1}{2}\mathbf{w}^T S^{-1}\mathbf{w} + O(1).$$

This implies that

$$\begin{aligned}\mathbf{w}^* &= \underset{\mathbf{w}}{\operatorname{argmax}} \left(-\sum_{n=1}^{N}(t_n - \mathbf{w}^T \mathbf{x}_n)^2 - \frac{\lambda}{2}\sum_{k=1}^{K} w_k^2 \right) \\ &= \underset{\mathbf{w}}{\operatorname{argmin}} \left(\sum_{n=1}^{N}(t_n - \mathbf{w}^T \mathbf{x}_n)^2 + \frac{\lambda}{2}\sum_{k=1}^{K} w_k^2 \right).\end{aligned}$$

The second term is called the L2 regulariser. Instead of squaring, we can take the absolute value- this would be L1 regulariser, i.e.

$$\underset{\mathbf{w}}{\operatorname{argmin}} \left(\sum_{n=1}^{N} (t_n - \mathbf{w}^T \mathbf{x}_n)^2 + \frac{\lambda}{2} \sum_{k=1}^{K} w_k^2 \right).$$

The L1 regulariser encourages sparsity (many elements in $\mathbf{w}$ will be 0), while the L2 regulariser encourages small magnitude.