COMP9417 - Machine Learning
Tutorial: Regression I

In this tutorial we will explore the theoretical foundations of linear regression. We first work through linear regression in 1 dimension (univariate linear regression), and then extend the analysis to multivariate linear regression. If you find yourself unsure about any of the math covered in this tutorial, we strongly recommend reading the corresponding material in the text (freely available online):

Mathematics for Machine Learning by Marc Peter Deisenroth, A. Aldo Faisal and Cheng Soon Ong.

We will refer to this as the MML book throughout the course.

Question 1. (Univariate Least Squares)

A *univariate linear regression model* is a linear equation $y = w_0 + w_1 x$. Learning such a model requires fitting it to a sample of training data, $(x_1, y_1), \ldots, (x_n, y_n)$, so as to minimize the loss (usually Mean Squared Error (MSE)), $\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2$. To find the best parameters w_0 and w_1 that minimize this error function we need to find the error *gradients* $\frac{\partial \mathcal{L}}{\partial w_0}$ and $\frac{\partial \mathcal{L}}{\partial w_1}$. So we need to derive these expressions by taking partial derivatives, set them to zero, and solve for w_0 and w_1.

(a) Derive the least-squares estimates (minimizers of the MSE loss function) for the univariate linear regression model.

Solution:

First we write the loss function for the univariate linear regression $y = w_0 + w_1 x$ as

$$\mathcal{L} = \mathcal{L}(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i)^2$$

At a minimum of $\mathcal{L}$, the partial derivatives with respect to w_0, w_1 should be zero. We will start with taking the partial derivative of $\mathcal{L}$ with respect to w_0:

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_0} &= \frac{\partial}{\partial w_0} \frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)^2 \\
&= \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial w_0} (y_i - w_0 - w_1 x_i)^2 \\
&= \frac{1}{n} \sum_{i=1}^{n} -2(y_i - w_0 - w_1 x_i) \\
&= -2 \left[\frac{1}{n} \sum_{i=1}^{n} y_i - w_0 \frac{1}{n} \sum_{i=1}^{n} 1 - w_1 \frac{1}{n} \sum_{i=1}^{n} x_i \right] \\
&= -2 \left[\overline{y} - w_0 - w_1 \overline{x} \right],
\end{aligned}
$$

where we have introduced the notation $\overline{f}$ to mean the sample average of f, i.e. $\overline{f} = \frac{1}{m} \sum_{j=1}^{m} f_j$, where m is the length of f. Now, we equate this to zero and solve for w_0 to get

$$
-2 \left[\overline{y} - w_0 - w_1 \overline{x} \right] = 0 \implies w_0 = \overline{y} - w_1 \overline{x}
$$

Note, we have not actually solved for w_0 yet, since our expression depends on w_1, which we must also optimise over. Taking the partial derivative of $\mathcal{L}$ with respect to w_1:

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_1} &= \frac{\partial}{\partial w_1} \frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)^2 \\
&= \frac{1}{n} \sum_{i=1}^{n} -2x_i(y_i - w_0 - w_1 x_i) \\
&= -2 \left[\frac{1}{n} \sum_{i=1}^{n} x_i y_i - w_0 \frac{1}{n} \sum_{i=1}^{n} x_i - w_1 \frac{1}{n} \sum_{i=1}^{n} x_i^2 \right] \\
&= -2 \left[\overline{xy} - w_0 \overline{x} - w_1 \overline{x^2} \right],
\end{aligned}
$$

Now, we equate this to zero and solve for w_1 to get

$$
-2 \left[\overline{xy} - w_0 \overline{x} - w_1 \overline{x^2} \right] = 0 \implies w_1 = \frac{\overline{xy} - w_0 \overline{x}}{\overline{x^2}}
$$

We now have an expression for w_0 in terms of w_1, and an expression for w_1 in terms of w_0. These are known as the Normal Equations. In order to get an explicit solution for w_0, w_1, we can plug w_0 into w_1 and solve:

$$
\begin{aligned}
w_1 &= \frac{\overline{xy} - w_0 \overline{x}}{\overline{x^2}} \\
&= \frac{\overline{xy} - (\overline{y} - w_1 \overline{x}) \overline{x}}{\overline{x^2}} \\
&= \frac{\overline{xy} - \overline{x}\,\overline{y} + w_1 \overline{x}^2}{\overline{x^2}}
\end{aligned}
$$

Rearranging and solving for w_1 gives us

$$w_1 = \frac{\overline{xy} - \overline{x}\,\overline{y}}{\overline{x^2} - \overline{x}^2}$$

So now we have an explicit solution for the regression parameters w_0 and w_1, and so we are done.

(b) Show that the centroid of the data, i.e. the point $(\overline{x}, \overline{y})$ is always on the least squares regression line.

Solution:
The least-squares regression line is:

$$\begin{aligned}\hat{y}(x) &= w_0 + w_1 x \\ &= \left(\overline{y} - \frac{\overline{xy} - \overline{x}\,\overline{y}}{\overline{x^2} - \overline{x}^2}\overline{x}\right) + \left(\frac{\overline{xy} - \overline{x}\,\overline{y}}{\overline{x^2} - \overline{x}^2}\right) x,\end{aligned}$$

where the 'hat' notation is usually used to mean the estimated value or predicted value, in this case, $\hat{y}(x)$ is our estimate of y at the evaluation point x. Subsituting $x = \overline{x}$ gives:

$$\hat{y}(\overline{x}) = \overline{y},$$

and so the least squares regression line must pass through the data centroid: $(\overline{x}, \overline{y})$.

(c) To make sure you understand the process, try to solve the following loss function for linear regression with a version of "$L2$" regularization, in which we add a penalty that penalizes the size of w_1. Let $\lambda > 0$ and consider the regularised loss

$$\mathcal{L}(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2 + \lambda w_1^2$$

Solution:
Repeating the steps above for the new problem yields

$$\begin{aligned}w_0 &= \overline{y} - w_1\overline{x}, \\ w_1 &= \frac{\overline{xy} - \overline{x}\,\overline{y}}{\overline{x^2} - \overline{x}^2 + \lambda}.\end{aligned}$$

Question 2. (Multivariate Least Squares)
In the previous question, we found the least squares solution for the univariate (single feature) problem. We now generalise this for the case when we have p features. Let $x_1, x_2, \ldots, x_n$ be n feature vectors (e.g.

corresponding to n instances) in $\mathbb{R}^p$, that is:

$$x_i = \begin{bmatrix} x_{i0} \\ x_{i1} \\ \vdots \\ x_{ip-1} \end{bmatrix}$$

We stack these feature vectors into a single matrix, $X \in \mathbb{R}^{n \times p}$, called the *design* matrix. The convention is to stack the feature vectors so that each row of X corresponds to a particular instance, that is:

$$X = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{bmatrix} = \begin{bmatrix} x_{10} & x_{11} & \cdots & x_{1,p-1} \\ x_{20} & x_{21} & \cdots & x_{2,p-1} \\ \vdots & \vdots & \cdots & \vdots \\ x_{n0} & x_{n1} & \cdots & x_{n,p-1} \end{bmatrix}$$

where the superscript T denotes the transpose operation. Note that it is standard to take the first element of the feature vectors to be 1 to account for the bias term, so we will assume $x_{i0} = 1$ for all $i = 1, \ldots, n$. Analagously to the previous question, the goal is to learn a weight vector $w \in \mathbb{R}^p$, and make predictions:

$$\hat{y}_i = w^T x_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \cdots + w_{p-1} x_{i,p-1},$$

where $\hat{y}_i$ denotes the i-th predicted value. To solve for the optimal weights in w, we can use the same procedure as before and use the MSE:

$$\begin{aligned} \mathcal{L}(w_0, w_1, \ldots, w_{p-1}) &= \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \\ &= \frac{1}{n} \sum_{i=1}^{n} (y_i - (w_0 + w_1 x_{i1} + w_2 x_{i2} + \cdots + w_{p-1} x_{i,p-1}))^2. \end{aligned}$$

One approach to solve this would be to take derivatives with respect to each of the p weights and solve the resulting equations, but this would be **extremely** tedious. The more efficient way to solve this problem is to appeal to matrix notation. We can write the above loss as:

$$\mathcal{L}(w) = \frac{1}{n} \|y - Xw\|_2^2,$$

where $\| \cdot \|_2$ is the Euclidean norm. For the remainder of this question, we will assume that X is a full-rank matrix, which means that we are able to compute the inverse of $X^T X$.

(a) Show that $\mathcal{L}(w)$ has a critical point:

$$\hat{w} = (X^T X)^{-1} X^T y,$$

note: critical point here means that $(\frac{d\mathcal{L}(w)}{w})(\hat{w}) = 0$, i.e. the gradient evaluated at the critical point $\hat{w}$ is zero).

Hint 1: if u is a vector in $\mathbb{R}^n$, and v is a fixed vector in $\mathbb{R}^n$, then $\frac{\partial v^T u}{\partial u} = v$.

Hint 2: if A is a fixed $n \times n$ matrix, and if $f = z^T A z$, then $\frac{\partial u^T A u}{\partial u} = Au + A^T u$.

Solution:
We have

$$\begin{aligned}\|y - Xw\|_2^2 &= (y - Xw)^T(y - Xw) \\ &= y^T y - 2y^T Xw + w^T X^T Xw.\end{aligned}$$

To minimise the above, we take derivatives with respect to w and set equal to zero as follows:

$$\frac{d}{dw}(y^T y - 2y^T Xw + w^T X^T Xw) = -2X^T y + 2X^T Xw \overset{\text{(set)}}{=} 0.$$

Solving for w in the above yields:

$$2X^T Xw = 2X^T y \implies \hat{w} = (X^T X)^{-1} X^T y$$

(b) The condition $(\frac{d\mathcal{L}(w)}{w})(\hat{w}) = 0$ is necessary but not sufficient to show that $\hat{w}$ is a (global) minimizer of $\mathcal{L}$, since this point could be a local minimum or a saddle point. Show that the critical point in part (a) is indeed a global minimizer of $\mathcal{L}$.
Hint 1: $\mathcal{L}(w)$ is a function of $w \in \mathbb{R}^p$, and so its Hessian, H, is the $p \times p$ matrix of second order partial derivatives, that is, the (k, l)-th element of H is

$$H_{kl} = \frac{\partial \mathcal{L}(w)}{\partial w_k \partial w_l}.$$

We will often write $H = \nabla_w^2 \mathcal{L}(w)$, where ∇ is the gradient operator, and ∇^2 means taking the gradient twice. Note that the Hessian plays the role of second derivative for multivariate functions.
Hint 2: a function is convex if its Hessian is positive semi-definite, which means that for any vector u,

$$u^T Hu \geq 0.$$

Note also that this condition means that for any choice of u, the product term will always be non-negative.
Hint 3: Any critical point of a convex function is a global minimum.

Solution:
Using the first hint, we first compute the Hessian of $\mathcal{L}$:

$$\begin{aligned}\nabla_w^2 \mathcal{L}(w) &= \nabla_w(\nabla_w \mathcal{L}(w)) \\ &= \nabla_w(-2X^T y + 2X^T Xw) \\ &= 2X^T X.\end{aligned}$$

Then, for any vector $u \in \mathbb{R}^p$, we have

$$u^T(2X^T X)u = 2(Xu)^T(Xu) = 2\|Xu\|_2^2 \geq 0,$$

since norms are always non-negative. Therefore, $\mathcal{L}$ is indeed convex, and so the critical point found earlier is indeed a global minimizer of $\mathcal{L}$.

(c) In the next parts, we will use the formula derived above to verify our solution in the univariate case. We assume that $p = 2$, so that we have a two dimensional feature vector (one term for the intercept, and another for our feature). Write down the following values:

$$x_i, \quad y, \quad w, \quad X, \quad X^T X, \quad (X^T X)^{-1}, \quad X^T y.$$

Solution:
We have the following results:

$$x_i = \begin{bmatrix} 1 \\ x_{i1} \end{bmatrix} \in \mathbb{R}^2, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \in \mathbb{R}^n, \quad w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \in \mathbb{R}^2$$

$$X = \begin{bmatrix} 1 & x_{11} \\ 1 & x_{21} \\ \vdots & \vdots \\ 1 & x_{n1} \end{bmatrix} \in \mathbb{R}^{n \times 2} \qquad X^T X = \begin{bmatrix} n & n\overline{x} \\ n\overline{x} & n\overline{x^2} \end{bmatrix} \in \mathbb{R}^{2 \times 2}$$

Finally, we have

$$(X^T X)^{-1} = \frac{1}{n(\overline{x^2} - \overline{x}^2)} \begin{bmatrix} \overline{x^2} & -\overline{x} \\ -\overline{x} & 1 \end{bmatrix}, \qquad X^T y = \begin{bmatrix} n\overline{y} \\ n\overline{xy} \end{bmatrix}$$

(d) Compute the least-squares estimate for the $p = 2$ case using the results from the previous part.

Solution:
Putting the results from parts (a) and (b) together:

$$\hat{w} = (X^T X)^{-1} X^T y = \begin{bmatrix} \overline{y} - \hat{w}_1 \overline{x} \\ \frac{\overline{xy} - \overline{x}\,\overline{y}}{\overline{x^2} - \overline{x}^2} \end{bmatrix},$$

which is exactly the same doing the brute-force approach in the previous question.

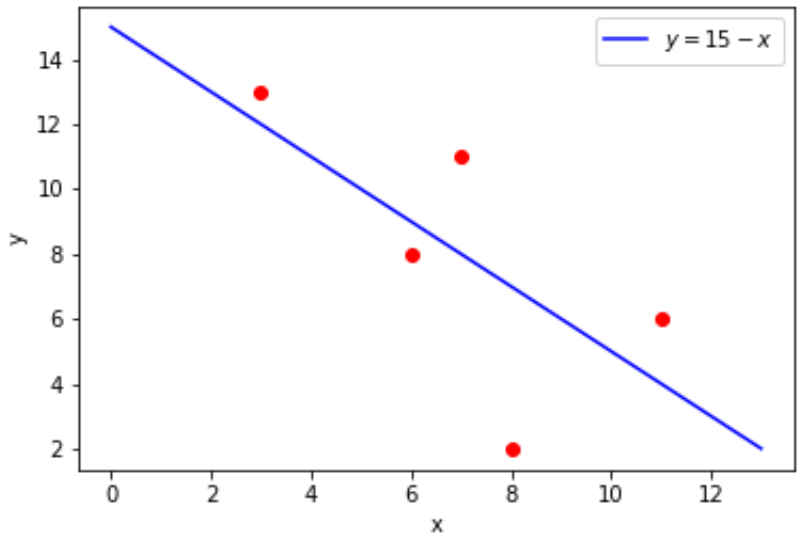
(e) Consider the following problem: we have inputs $x_1, \ldots, x_5 = 3, 6, 7, 8, 11$ and outputs $y_1, \ldots, y_5 = 13, 8, 11, 2, 6$. Compute the least-squares solution and plot the results both by hand and using python. Finally, use the sklearn implementation to check your results.

Solution:
This should be straight-forward, we can use the following python code:

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

```
            from sklearn import linear_model

            y = np.array([13, 8, 11, 2, 6])
            x = np.array([3, 6, 7, 8, 11])
            n = x.shape[0]
            X = np.stack((np.ones(n), x), axis=1)

            # compute the least-squares solution
            XTX = X.T @ X
            XTXinv = np.linalg.inv(XTX)
            XTy = X.T @ y
            LeastSquaresEstimate = XTXinv @ XTy

            # sklearn comparison
            model = linear_model.LinearRegression()
            model.fit(x.reshape(-1,1), y)

            # plot
            xx = np.linspace(0,13,1000)
            plt.scatter(x, y, color='red')
            plt.plot(xx, 15. - xx, color='blue', label='$y = 15 - x$')
            plt.legend()
            plt.xlabel('x'); plt.ylabel('y')
            plt.savefig("LSLine.png")
            plt.show()
```

We should end up with the following



(f) **Advanced:** Some of you may have been concerned by our assumption that X is a full-rank matrix, which is an assumption made to ensure that $X^T X$ is an invertible matrix. If $X^T X$ was not invertible, then we would not have been able to compute the least squares solution. So what happens

if X is not full-rank? Does least squares fail? The answer is no, we can use something called the pseudo-inverse, also referred to as the Moore-Penrose Inverse. This is outside the scope of this course, and the interested reader can refer to the following notes, or chapter 2 in MML. At a very high level, the pseudo-inverse is a matrix that acts like the inverse for non-invertible matrices. In NumPy, we can easily compute the pseudo inverse using the command 'np.linalg.pinv'.

(g) Discuss the idea of a *feature map*. How would you use a feature map in the context of least squares regression?

Solution:

If we are dealing with a regression problem where we have data (x_i, y_i), $i = 1, \ldots, n$ where $x_i \in \mathbb{R}^p$, a feature map is any function $\phi : \mathbb{R}^p \to \mathbb{R}^K$ that maps the data from the original p-dimensional space to some new K dimensional space. K can be either larger or smaller than p. For example, if $p = 1$, then a feature map might look look like:

$$\phi : \mathbb{R} \to \mathbb{R}^4, \qquad \phi(x) = [x, x^2, x^3, \log x]^T,$$

i.e. we map each point a four dimensional space where we take polynomial and log transforms. Doing this might yield a better model (more on this next week). Consider another example where $p = 3$, so that $x = [x_1, x_2, x_3]^T$, then we might choose to consider the feature map:

$$\phi : \mathbb{R}^3 \to \mathbb{R}^2, \qquad \phi(x) = \left[\frac{1}{2}(x_1 + x_2), x_3^2\right]^T.$$

We saw that in the original setting, our least-squares regression model is

$$\hat{y}_i = w^T x_i,$$

and now our model is simply

$$\hat{y}_i = w^T \phi(x_i).$$

Therefore, nothing really changes except that we are dealing with a K dimensional problem instead of a p dimensional one. We simply construct the new design matrix

$$\Phi = \begin{bmatrix} \phi(x_1)^T \\ \phi(x_2)^T \\ \vdots \\ \phi(x_n)^T \end{bmatrix} \in \mathbb{R}^{n \times K},$$

and solve the same minimisation:

$$\hat{w} = \arg\min_{w \in \mathbb{R}^K} \frac{1}{n} \|y - \Phi w\|_2^2,$$

which yields

$$\hat{w} = (\Phi^T \Phi)^{-1} \Phi^T y.$$

(h) The mean-squared error (MSE) is

$$\text{MSE}(w) = \frac{1}{n}\|y - Xw\|_2^2,$$

whereas the sum of squared errors (SSE) (also referred to as the residual sum of squares (RSS)) is

$$\text{SSE}(w) = \|y - Xw\|_2^2.$$

Are the following statements True or False. Explain why.

(i) $\arg\min_{w\in\mathbb{R}^p} \text{MSE}(w) = \arg\min_{w\in\mathbb{R}^p} \text{SSE}(w)$

(ii) $\min_{w\in\mathbb{R}^p} \text{MSE}(w) = \min_{w\in\mathbb{R}^p} \text{SSE}(w)$

Notation: recall that $\min_x g(x)$ is the smallest value of $g(x)$, whereas $\arg\min_x g(x)$ is the value of x that minimizes $g(x)$. So $\min_x (x-2)^2 = 0$ but $\arg\min_x (x-2)^2 = 2$.

Solution:

The first statement is correct, but the second statement is not correct. To understand this, let's consider a simpler example of minimising the function $g(x) = (x-2)^2+4$. A simple calculation shows that

$$\arg\min_{x\in\mathbb{R}} g(x) = 2,$$

and

$$\arg\min_{x\in\mathbb{R}} \frac{1}{n} g(x) = 2.$$

In other words, it doesn't matter if you multiply the function by a (positive) constant, the minimizer is always going to be the same. However,

$$\min_{x\in\mathbb{R}} g(x) = g(2) = 4,$$

and

$$\min_{x\in\mathbb{R}} \frac{1}{n} g(x) = \frac{1}{n} g(2) = \frac{4}{n}.$$

So, in summary, when finding the minimizer $\hat{w}$ of the least squares problem, it doesn't matter if we consider the MSE or SSE, we will get the same solution in the end. What does matter though is interpretation of the minimum value of an objective. The MSE gives an idea of the average squared error made by a model, where as the SSE gives the total squared error. The MSE is therefore somewhat more meaningful, since you might have a higher SSE just because you have a larger sample size n.

Question 3. (Population Versus Sample Parameters)

(a) What is the difference between a population and a sample?

Solution:
A population is the quantity we are interested in learning something about, and a sample is the subset of the population we have access to. For example, in a COVID-19 vaccine trial, the population is the entire population of humans on earth. We would ideally like to understand how a dose of the vaccine reduces the risk of death from COVID-19 in the entire population, but realistically we cannot conduct this study. What we do instead is run clinical trials on much smaller but hopefully representative samples of the population, and try to estimate/infer things about the population.

(b) What is a population parameter? How can we estimate it?

Solution:
A population parameter is something that is unknown and that we wish to estimate using data. For example, assume we are interested in the mean (average) weight of all kangaroos in Australia. It would be impossible to catch and weigh all kangaroos so this quantity cannot ever be known exactly. When dealing with data, it is up to the modeler to make certain assumptions about the true underlying population that are reasonable and that make the statistical problem become more tractable.

One reasonable assumption is that the weights of kangaroos are normally distributed with mean μ, and standard deviation 10 (of course we can also treat the standard deviation as unknown, but for now we focus solely on the mean). The assumption here lets us think of weights of kangaroos as a random variable, which we denote by X, and we write $X \sim N(\mu, 10^2)$. Now, say we only have access to 100 kangaroos (at the local zoo), and we weigh each of them and record the numbers - we now have access to a sample, which we denote $X_1, \ldots, X_{100}$. Since we know that $X_i \sim N(\mu, 10^2)$, we can try to use the data to estimate μ. There are many ways to construct an estimator for the population mean, and a common one is the sample mean: $\overline{X} = \frac{1}{100}\sum_{i=1}^{n} X_i$. Another example is the sample median. Note that in general, we use greek letters, such as μ and σ to refer to the the population parameters, you should generally not use them to refer to sample statisitcs such as the sample mean or sample standard deviation.

It is important not to get too hung up on populations representing some actual population in the real world, it is best to think of population parameters as things we would like to know but cannot ever measure. A concrete example to keep in mind is the example of a coin toss. We know a coin has two sides, heads/tails. Assume that the coin might be bent, so we do not know the true probability of getting heads. Let's call this unkown probability p. In this case, we can think of p as our population parameter. We can gather data by tossing the coin multiple times and recording the outcomes. A good statistical assumption is then that $X \sim$ Bernoulli(p).

From now on, whenever you see the statement $X_1, \ldots, X_n \sim F(\theta)$, where F is some distribution and θ is some parameter, you can simply assume that we are interested in knowing θ, but only have access to samples (data) $X_1, \ldots, X_n$, that we want to use in a smart way to estimate θ.