COMP9417 - Machine Learning
Tutorial: Linear Regression

Question 1. (Least Squares Regression)

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), which is also referred to as the mean sum of squares loss), $\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2$. 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) 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

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

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^TX.

(a) Show that $\mathcal{L}(w)$ is minimised by the least-squares estimator:

$$\hat{w} = (X^TX)^{-1}X^Ty.$$

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

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

Solution:
We have

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

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

$$\frac{\partial}{\partial w}(y^Ty - 2y^TXw + w^TX^TXw) = -2X^Ty + 2X^TXw \overset{\text{(set)}}{=} 0.$$

Solving for w in the above yields:

$$2X^TXw = 2X^Ty \implies \hat{w} = (X^TX)^{-1}X^Ty$$

(b) 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 bias, and another for our feature). Write down the following values:

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

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

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

(c) 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.

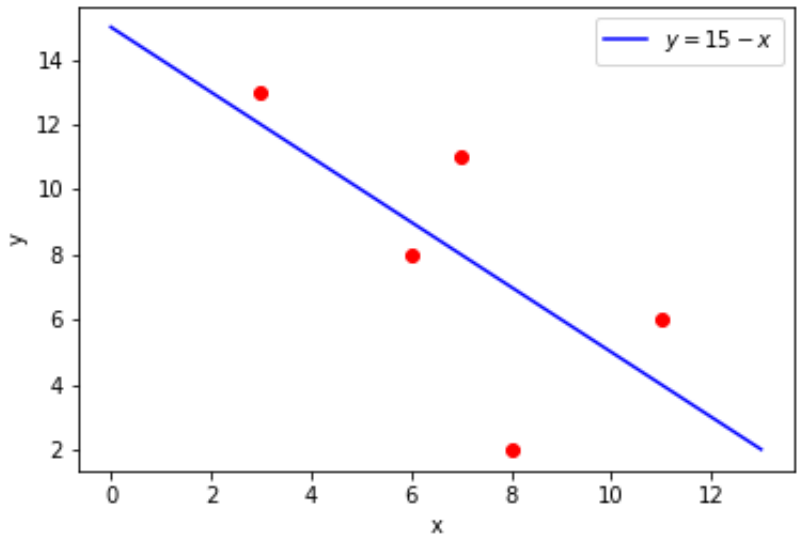
(d) Consider the following problem: we have inputs $x_1, \ldots, x_5 = 3, 6, 7, 8, 11$ and outputs $y_1, \ldots, y_5 = 13, 8, 11, 2, 6$. Compute the least-squares solution and plot the results both by hand and using python. Finally, use the sklearn implementation to check your results.

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

```
import matplotlib.pyplot as plt
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
LeastSqaresEstimate = 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



(e) **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. 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'.