Solution to COMP9318 Assignment 1

Q1

1. See the following table.

cuboid	*Location*	*Time*	*Item*	$SUM(Quantity)$
LTI	Sydney	2005	PS2	1400
LTI	Sydney	2006	PS2	1500
LTI	Sydney	2006	Wii	500
LTI	Melbourne	2005	XBox 360	1700
LT	Sydney	2005	ALL	1400
LT	Sydney	2006	ALL	2000
LT	Melbourne	2005	ALL	1700
LI	Sydney	ALL	PS2	2900
LI	Sydney	ALL	Wii	500
LI	Melbourne	ALL	XBox 360	1700
TI	ALL	2005	PS2	1400
TI	ALL	2006	PS2	1500
TI	ALL	2006	Wii	500
TI	ALL	2005	XBox 360	1700
L	Sydney	ALL	ALL	3400
L	Melbourne	ALL	ALL	1700
T	ALL	2005	ALL	3100
T	ALL	2006	ALL	2000
I	ALL	ALL	PS2	2900
I	ALL	ALL	Wii	500
I	ALL	ALL	XBox 360	1700
	ALL	ALL	ALL	5100

2. See below. Note that we

   (a) need to ensure all `SELECT` has the same schema (called "Union compatible"), and

   (b) should use `UNION ALL` as there is no duplicate across different `SELECT` queries.

```
SELECT   L, T, I, SUM(M)
FROM     R
GROUP BY L, T, I
UNION ALL
SELECT   L, T, ALL, SUM(M)
FROM     R
GROUP BY L, T
UNION ALL
SELECT   L, ALL, I, SUM(M)
FROM     R
GROUP BY L, I
UNION ALL
SELECT   ALL, T, I, SUM(M)
FROM     R
GROUP BY T, I
UNION ALL
SELECT   L, ALL, ALL, SUM(M)
FROM     R
GROUP BY L
UNION ALL
SELECT   ALL, T, ALL, SUM(M)
FROM     R
GROUP BY T
UNION ALL
SELECT   ALL, ALL, I, SUM(M)
FROM     R
GROUP BY I
UNION ALL
SELECT   ALL, ALL, ALL, SUM(M)
FROM     R
```

3. The iceberg cube is

cuboid	*Location*	*Time*	*Item*	$SUM(Quantity)$
LT	Sydney	2006	ALL	2000
LI	Sydney	ALL	PS2	2900
L	Sydney	ALL	ALL	3400
T	ALL	2005	ALL	3100
T	ALL	2006	ALL	2000
I	ALL	ALL	PS2	2900
	ALL	ALL	ALL	5100

4. The mapping function we choose should satisfy the property that it is a one-to-one function (such that we can always recover the original

value even after the mapping). The simplest form is $h(L,T,I) = 12L + 4T + I$. Hence,

Location	*Time*	*Item*	$SUM(Quantity)$	$h(L,T,I)$
1	1	1	1400	17
1	2	1	1500	21
1	2	3	500	23
2	1	2	1700	30
1	1	0	1400	16
1	2	0	2000	20
2	1	0	1700	28
1	0	1	2900	13
1	0	3	500	15
2	0	2	1700	26
0	1	1	1400	5
0	2	1	1500	9
0	2	3	500	11
0	1	2	1700	6
1	0	0	3400	12
2	0	0	1700	24
0	1	0	3100	4
0	2	0	2000	8
0	0	1	2900	1
0	0	3	500	3
0	0	2	1700	2
0	0	0	5100	0

So the final result is:

index	*value*
17	1400
21	1500
23	500
30	1700
16	1400
20	2000
28	1700
13	2900
15	500
26	1700
5	1400
9	1500
11	500
6	1700
12	3400
24	1700
4	3100
8	2000
1	2900
3	500
2	1700
0	5100

Q2

We define the *logOdds*, and if it is larger than 0, then the prediction is positive class; otherwise, the classification is the negative class.

$$
\begin{aligned}
\mathit{logOdds} &\stackrel{\text{def}}{=} \log\left(\frac{\mathbf{Pr}[C_+ \mid \mathbf{u}]}{\mathbf{Pr}[C_- \mid \mathbf{u}]}\right) \\
&= \log\left(\mathbf{Pr}[\mathbf{u} \mid C_+] \cdot \mathbf{Pr}[C_+]\right) - \log\left(\mathbf{Pr}[\mathbf{u} \mid C_-] \cdot \mathbf{Pr}[C_-]\right) \\
&= \log\left(\prod_{i=1}^{d} \mathbf{Pr}[x_i = u_i \mid C_+] + \log\left(\mathbf{Pr}[C_+]\right)\right) - \log\left(\prod_{i=1}^{d} \mathbf{Pr}[x_i = u_i \mid C_-] + \log\left(\mathbf{Pr}[C_-]\right)\right) \\
&= \left(\sum_{i=1}^{d} \log\left(\mathbf{Pr}[x_i = u_i \mid C_+]\right) + \log\left(\mathbf{Pr}[C_+]\right)\right) - \left(\sum_{i=1}^{d} \log\left(\mathbf{Pr}[x_i = u_i \mid C_-]\right) + \log\left(\mathbf{Pr}[C_-]\right)\right) \\
&= \left(\sum_{i=1}^{d} (\log\left(\mathbf{Pr}[x_i = u_i \mid C_+]\right) - \log\left(\mathbf{Pr}[x_i = u_i \mid C_-]\right))\right) + \left(\log\left(\mathbf{Pr}[C_+]\right) - \log\left(\mathbf{Pr}[C_-]\right)\right) \\
&= \sum_{i=1}^{d} \log\left(\frac{\mathbf{Pr}[x_i = u_i \mid C_+]}{\mathbf{Pr}[x_i = u_i \mid C_-]}\right) + \log\left(\frac{\mathbf{Pr}[C_+]}{\mathbf{Pr}[C_-]}\right)
\end{aligned}
$$

We define the following symbols to simply the above result:

$$
\begin{aligned}
\alpha(i, u_i) &\stackrel{\text{def}}{=} \log\left(\frac{\mathbf{Pr}[x_i = u_i \mid C_+]}{\mathbf{Pr}[x_i = u_i \mid C_-]}\right) \\
\beta &\stackrel{\text{def}}{=} \log\left(\frac{\mathbf{Pr}[C_+]}{\mathbf{Pr}[C_-]}\right)
\end{aligned}
$$

Then

$$
\begin{aligned}
\mathit{logOdds} &= \beta + \sum_{i=1}^{d} \alpha(i, u_i) \\
&= \beta + \sum_{i=1}^{d} (\alpha(i, 1) \cdot u_i + \alpha(i, 0) \cdot (1 - u_i)) \qquad (\because u_i \text{ can only take 0 or 1 value}) \\
&= \beta + \sum_{i=1}^{d} \alpha(i, 0) + (\alpha(i, 1) - \alpha(i, 0)) \cdot u_i \\
&= \gamma + \sum_{i=1}^{d} \delta_i \cdot u_i
\end{aligned}
$$

In the last step, we let $\gamma = \beta + \sum_{i=1}^{d} \delta(i, 0)$, and $\delta_i = \alpha(i, 1) - \alpha(i, 0)$

Therefore, it is a linear classifer for an **extended** feature vector $[1, \mathbf{x}]$, and the parameter

$$
\mathbf{w}^\top = [\gamma, \delta_1, \delta_2, \ldots, \delta_n]
$$

Remark 1

Note that the above maths proof does not necessarily convey the important intuitions and meanings of linear classifiers. I outline one intuitive thought that works better in this regard (at least to me). It is **important** that you shall arrive at your **own** deeper understanding of all major methods in this course, as this "generalizes" better in the future.

The outline:

1. First, we show that the score for either class, i.e., positive or negative, is a linear function of the (binary) vector $\mathbf{x}$. Geometrically, a linear function is just a hyperplane.

2. Seccondly, the decision boundary is the intersection of two hyperplanes, which will also be a hyperplane in the non-degenerated cases (albeit with one less degree of freedom), which is always a linear function.

You can start with 2D to gain some geometric intuition. Then perhaps 3D. You can also challenge yourself to visualize/understand what happens if there are more than two classes.

Remark 2

It is extremely helpful to contrast NB and LR (Logistic Regression) classifiers. For example, given the same set of training data $\mathbf{X}$, we can learn a NB classifier and a LR classifier, and both of them are a linear classifier in the same space. So how they derive the "best" $\mathbf{w}$ in their models and what are the key differences? What are the pros and cons of each?

Q3

(1) See Algorithm 1.

(2) It is obvious that we only need to prove the same conclusion for a cost function y that sums up square of the $dist$, i.e.,

$$\begin{aligned} f = cost'(g_1, \ldots, g_k) &= \sum_{i=1}^{k} cost'(g_i) \\ &= \sum_{i=1}^{k} \left(\sum_{p \in g_i} (dist(p, c_i))^2 \right) \end{aligned}$$

Within each iteration, we first assign each p to its nearest center, and then update the centers. It is easy to see first step always reduces f. For

Algorithm 1: k-means(D, k)

```
Data: D is a dataset of n d-dimensional points; k is the number of clusters.
Initialize k centers C = [c_1, c_2, ..., c_k];
canStop ← false;
while canStop = false do
    Initialize k empty clusters G = [g_1, g_2, ..., g_k];
    for each data point p ∈ D do
        c_x ← NearestCenter(p, C);
        g_{c_x}.append(p);
    canStop ← true;
    for each group g ∈ G do
        old_c_i ← c_i;
        c_i ← ComputeCenter(g);
        if old_c_i ≠ c_i then
            canStop ← false;
return G;
```

the second step, we study the extreme value of f. It is obvious that we can study $cost'(g_i)$ individually. Consider the d-dimensional point c_i to be represented as $(x_1, x_2, \ldots, x_d)$, then we take partial derivatives on every x_u:

$$\begin{aligned}\frac{\partial cost'(g_i)}{\partial x_u} &= \frac{\sum_{p \in g_i} \sum_{j=1}^{d} (p.x_j - x_j)^2}{\partial x_u} \\ &= -\sum_{p \in g_i} 2(p.x_u - x_u)\end{aligned}$$

Then x_u needs to be the mean values of $p.x_u$ for all points in the group, so that the partial derivatives are all 0. It is easy to check that this achieves the minimum value of the function. This shows that by choosing the new centers we also never increase the value of the objective function.

(3) Assume the conclusion in (2). This says the cost f at the end of each iteration is monotonically decreasing. Obviously f has a lower bound of 0. Therefore, according to the "monotone convergence theorem", the cost will converge.