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

Q2.1 We define the logOdds, and if it is larger than 0, then the prediction is positive class; otherwise, the classification is the negative class.

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

We define the following symbols to simply the above result:

$$
\alpha(i, u_i) \stackrel{\text{def}}{=} \log \frac{\Pr\{x_i = u_i \mid C_+\}}{\Pr\{x_i = u_i \mid C_-\}}
$$

$$
\beta \stackrel{\text{def}}{=} \log \frac{\Pr\{C_+\}}{\Pr\{C_-\}}
$$

Then

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

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

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

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

[1] δ_i has some intuitive interpretations after some rewriting. You can show that $\delta_i = \log(\text{odds}(\Pr\{x_i = 1 \mid C_+\})) - \log(\text{odds}(\Pr\{x_i = 1 \mid C_-\}))$, where $odds(p) \stackrel{\text{def}}{=} \frac{p}{1-p}$. One can interpret δ_i as the net contribution/score towards the positive class by having feature i.

Remark 1

Note that the above maths proof does not necessarily convey the important intuitions and meanings of linear classifiers *intuitively*. 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.

I outline one intuitive thought that works better in this regard (at least to me — you should seek your own):

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.[2] 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?

Q2.2 (There is no fixed answer to this question, but I hope everyone has similar kind of understanding)

Note that both models uses Maximum Likelihood Estimation (MLE) to find the parameters. The main difference is that NB model uses the conditional independence assumption so that there is no correlation between features (given a class label). The MLE of the parameters, i.e., $\Pr\{x_i = v_j \mid C_k\}$ becomes very easy (essentially some kind of counting). LR does not make such an assumption, and hence the MLE of the parameters involves all the parameters simultaneously, and has no closed form solution.

Q3

1 Steps:

[2] If you want to challenge yourself, you can ask the same questions for all other linear classifiers (e.g., Perceptron, SVM, and Adaboost).

- The *only* parameters of our model is $\mathbf{Q} = \begin{bmatrix} q_1 & q_2 \end{bmatrix}^\top$. As clarified in the Piazza post, we can assume that we sampled n units, and each unit has to be one of the O_js, and n is large enough (so that sampling without replacement can be well approximated by sampling with replacement).
- Then our observation is observing $u_j \cdot n$ units of O_j $(1 \leq j \leq 3)$, respectively
- Given fixed q_is, the probability that a randomly sampled unit is O_js is:
$$p_e(O_j) = \sum_i q_i \cdot p_{i,j}$$
- Hence, the likelihood function is
$$\begin{aligned} L(\mathbf{Q}) &= P(\#O_1 = u_1 \cdot n, \#O_2 = u_2 \cdot n, \#O_3 = u_3 \cdot n \mid \mathbf{Q}) \\ &= \prod_j (p_e(O_j))^{u_j \cdot n} \\ &= \prod_j \left(\left(\sum_i q_i \cdot p_{i,j} \right)^{u_j \cdot n} \right) \end{aligned}$$
The log likelihood function is:
$$l(\mathbf{Q}) = \log L(\mathbf{Q}) = n \cdot \sum_j u_j \cdot \log \left(\sum_i q_i \cdot p_{i,j} \right)$$
Note:
  - This is very similar to the 2nd example in the MLE slides. However, notice that we have three outcomes (O_j) instead of two outcomes (remember or forget).
  - If we do not do MLE, but instead uses the loss function based model. One can think of our model predicts a discrete distribution of a sampled unit as $\begin{bmatrix} p_e(O_1), p_e(O_2), p_e(O_3) \end{bmatrix}^\top$, and our observed probability distribution is $\begin{bmatrix} u_1, u_2, u_3 \end{bmatrix}^\top$ (i.e., ground truth). A natural loss function between two distributions is the KL-divergence. This will result in the same optimization problem (and its optimal solution) as we derived using MLE. This is *not* an coincidence, and interestest readers can look at the proof[3] (the blogger is the *Director of Engineering, Data Science and Machine Learning at Etsy Inc.*)
  - Some of the alternative models:

[3] `https://www.hongliangjie.com/2012/07/12/maximum-likelihood-as-minimize-kl-divergence/`

* Model the probability as an additive Gaussian error, i.e.,

$$P(\big[u_1, u_2, u_3\big]^\top \mid \big[p_e(O_1), p_e(O_2), p_e(O_3)\big]^\top) = \prod_j f(u_j - p_e(O_j) \mid 0, \sigma^2),$$

where $f(x \mid 0, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{x^2}{\sigma^2}\Big)$.

2 We only need to maximimize the log likelihood, subject to the following constraints:

$$q_i \geq 0$$
$$\sum_i q_i = 1$$

By running a solver, e.g., `scipy.optimize`, we can find the MLE estimate as:

$$\hat{q}_1 = 0.6344$$
$$\hat{q}_2 = 0.3656$$

The expected percentage of each component is:

$$\hat{O}_1 = 0.2097$$
$$\hat{O}_2 = 0.3097$$
$$\hat{O}_3 = 0.4807$$

Note that these numbers do not agree with the observed u_js (and that's exactly the reason why we need to use MLE to perform the estimation).