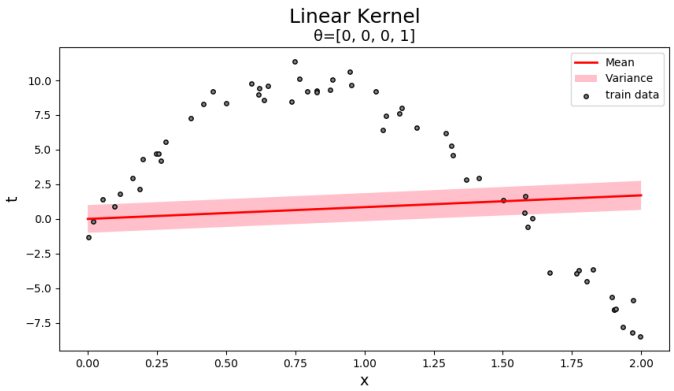
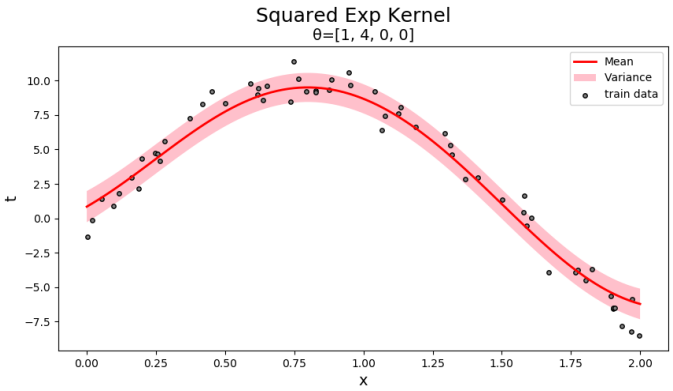
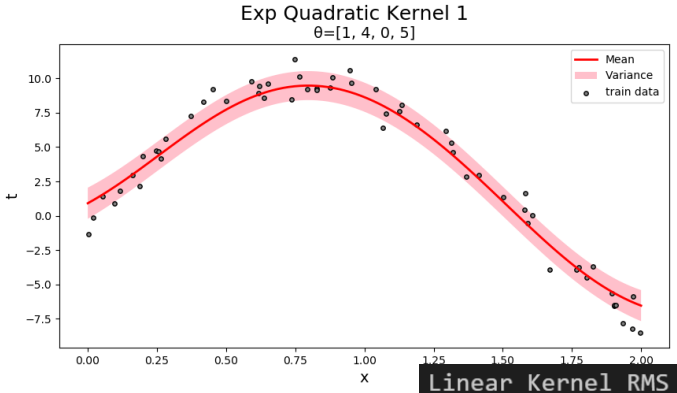
ML Hw3 Report

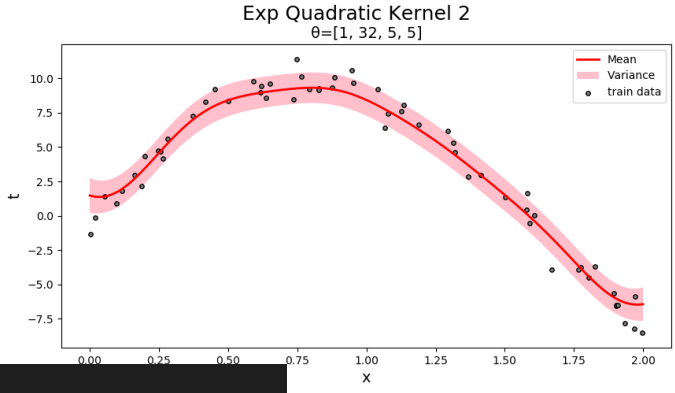
電機乙 0850736 楊登宇

1. Gaussian Process for Regression










```
Linear Kernel RMS
        Train data:6.658 , Test data:6.749

Squared Exp Kernel RMS
        Train data:1.052 , Test data:1.299

Exp Quadratic Kernel-1 RMS
        Train data:1.029 , Test data:1.286

Exp Quadratic Kernel-2 RMS
        Train data:0.964 , Test data:1.258
```

這一題在練習使用 kernel 的方法，一開始在寫的時候一直不太懂所謂的 kernel 到底有甚麼好處，但實作後才懂原來使用 kernel 可以減少一些對於超參數的假設與計算（雖然最後還是需要去調整θ），讓整體計算平均值與變異數的過程更加直覺。

其中，關於平均值與變異數的計算方法如下：

$$\mathrm{P}(t_{N+1}|t) = N(m(X_{N+1}), \sigma^2(X_{N+1}))$$

$$m(X_{N+1}) = K^T C_N^{-1} t$$

$$\sigma^2(X_{N+1}) = c - K^T C_N^{-1} K$$

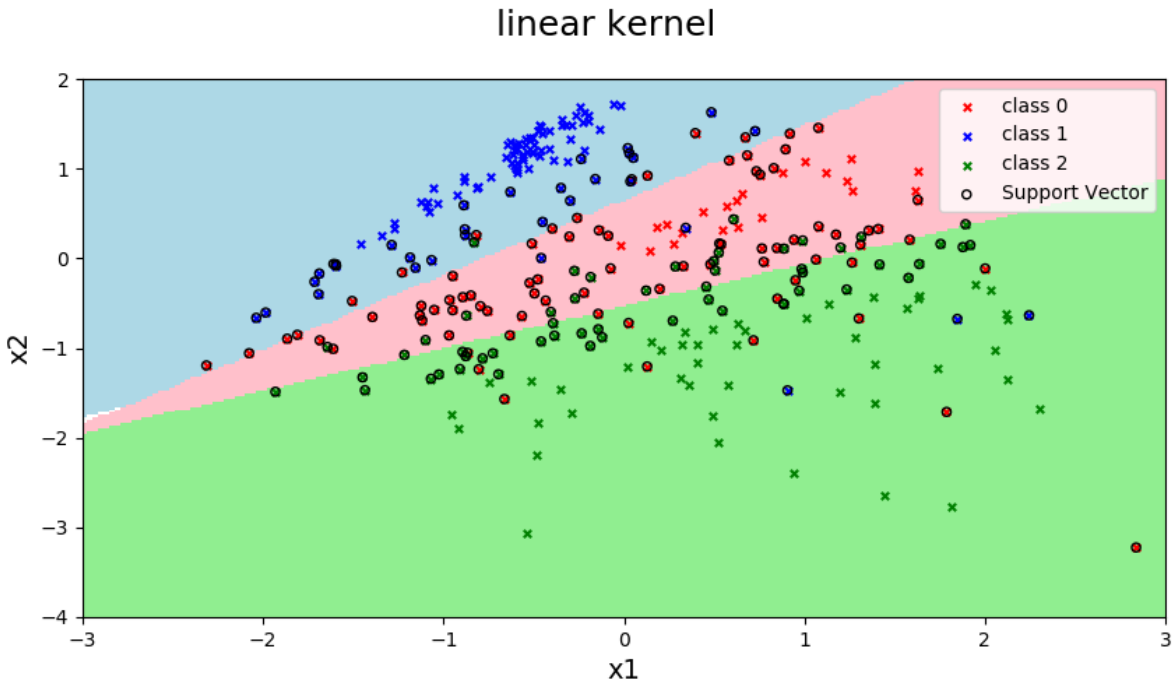
$$K_{Nx1} = \begin{bmatrix} k(x_1, x_{N+1}) \\ \vdots \\ k(x_N, x_{N+1}) \end{bmatrix}$$

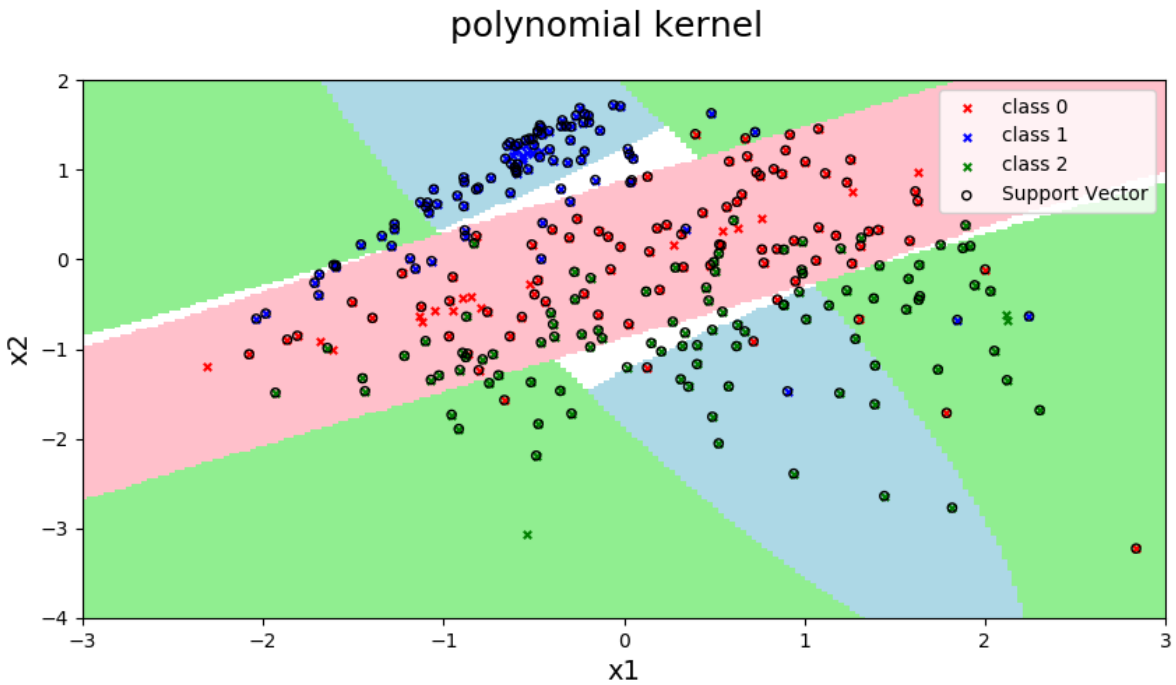
$$\mathrm{c} = k(x_{N+1}, x_{N+1}) + \beta^{-1}$$

$$C_N(x_n, x_m) = k(x_n, x_m) + \beta^{-1}\delta_{nm} \quad , \begin{cases} \delta_{nm} = 1, & n = m \\ \delta_{nm} = 0, & n \neq m \end{cases}$$

而結果可以清楚看見，透過去調整θ，可以讓 kernel 所代表的意義有所變化，而此作業所使用的訓練資料來看，使用 Exponential Quadratic Kernel 的參數最可以貼近該組資料，其中第二組 $\theta = [1,32,5,5]$在訓練資料的 RMS 可以到達最低值 0.964。但是在使用測試資料來計算誤差時會發現跟第一組$\theta = [1,4,0,5]$差別不大，我懷疑可能是因為第二組的參數有 overfitting 的趨勢，使得測試資料 RMS 下降程度不像使用訓練資料來做 RMS 一樣的大。也有可能是這已經是 Exponential Quadratic Kernel 參數對該題測試資料所能影響的最大範圍了。

2. Support Vector Machine





第二題我使用 python 的 scikit-learn 套件來求出該題 SVM 所要計算的 Lagrange Multipliers 與支援向量（已跟助教確認過可以直接求得）。在這一題支援向量機的求法，我使用的是 one versus one 的方法來實作。其中關於我選擇方法的原因如下；

One vs. one：可一次分類多類別的資料（multiclass），假設有 n 個類別，可以一次找出其中一個類別與其他(n-1)類別的邊界 y(x)，這樣子可以一次就訓練好模型，在預測的時候比較方便，所以我是選擇使用此方式。

One vs. the rest：一次只會判斷是不是屬於該類別，一次只會計算出一條邊界 y(x)。假設有 n 個類別，就需要訓練 n 次才能進行後續的預測，因為需要耗費的計算量我覺得比較大，因此我並沒有選擇此方法。

在得到 multipliers 後，根據以下的公式：

$$y(x) = \sum_{n=1}^{N} \alpha_n t_n k(x, x_n) + b$$

$$b = \frac{1}{N_s} \sum_{n \in S} (t_n - \sum_{m \in S} \alpha_m t_m k(x_m, x_m))$$

可以求出三條決策邊界，我這邊計算出來的邊界定義如下：

$$\begin{cases} y_0(x) : class\ 0\ vs.\ class\ 1 \\ y_1(x) : class\ 0\ vs.\ class\ 2 \\ y_2(x) : class\ 1\ vs.\ class\ 2 \end{cases}$$

接著，可以帶入 x 來計算出在這三條邊界上的正負來判斷是屬於哪一類別：

$y_0(x)$	+	−	+	+	−	−	+	−
$y_1(x)$	+	+	−	+	−	+	−	−
$y_2(x)$	+	+	+	−	+	−	−	−
class	0	1	X	0	1	X	2	2

其中會發現，會出現三種皆為無法分類的情形，在我的實作中我會將分不出哪一類的情形畫成白色底，用以跟其他類別作出區分。

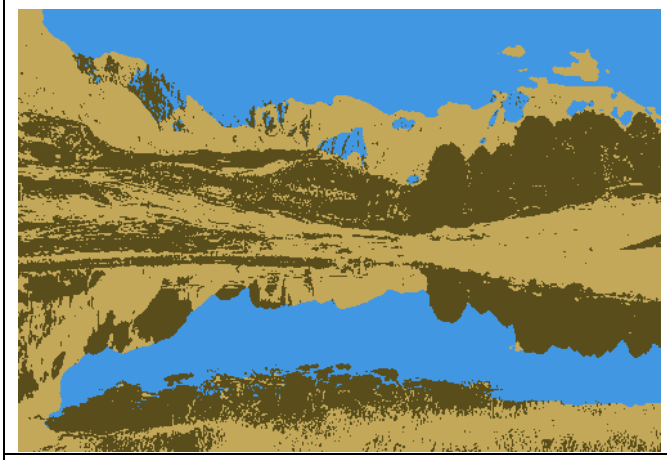
實作部分我是先將原本的資料進行 PCA 降成 2 維後，再進行標準化。因為若沒有進行標準化的話，每個樣本點之間的離散程度會滿大的，在畫圖時的邊界會差距過大，所以為了繪圖與分析資料方便，我線進行標準化讓資料漂亮一點。
接著，將資料透過 scikit-learn 來求出參數後以求得$y_0(x)$、$y_1(x)$、$y_2(x)$，接著繪圖。

在線性部分的結果，分起來的效果看起來還算不錯，且幾乎沒有分不出來的部分（白色底）；而 2 次多項式的結果看起來就有明顯差異了，其中更多出了許多白色底的無法分類的區域，而且分類的情況也沒有線性的理想。我推測是因為此題應該使用到線性 kernel 即可，使用到多項式反而有點過頭，才導致分類反而會分不好。

3. Gaussian Mixture Model

a. kmeans only

K=3



K=5

K=7

K=10

我將每個 pixel 的 rgb 顏色取出來，然後進行 kmeans 來做初步分類。程式在運作時，會發現在 K=5, 7 時，計算次數會比較久一點，反而 k=10 還比較快，這讓我感到蠻驚訝的。我猜測可能與 kmeans 一開始取初始中心點有關係。我本身的取法是把所有資料平均取 k 個點，有可能我在 k=10 取的點比較好，所以收斂的速度較為快速。

其中各個不同 k 最後得到的中心點與數量結果我都額外存在 3_kmeans_output.txt 檔案中，可以透過以下公式，來將 kmeans 求出的 k 點中心來做為 GMM 的初值，進行後續 EM 計算：

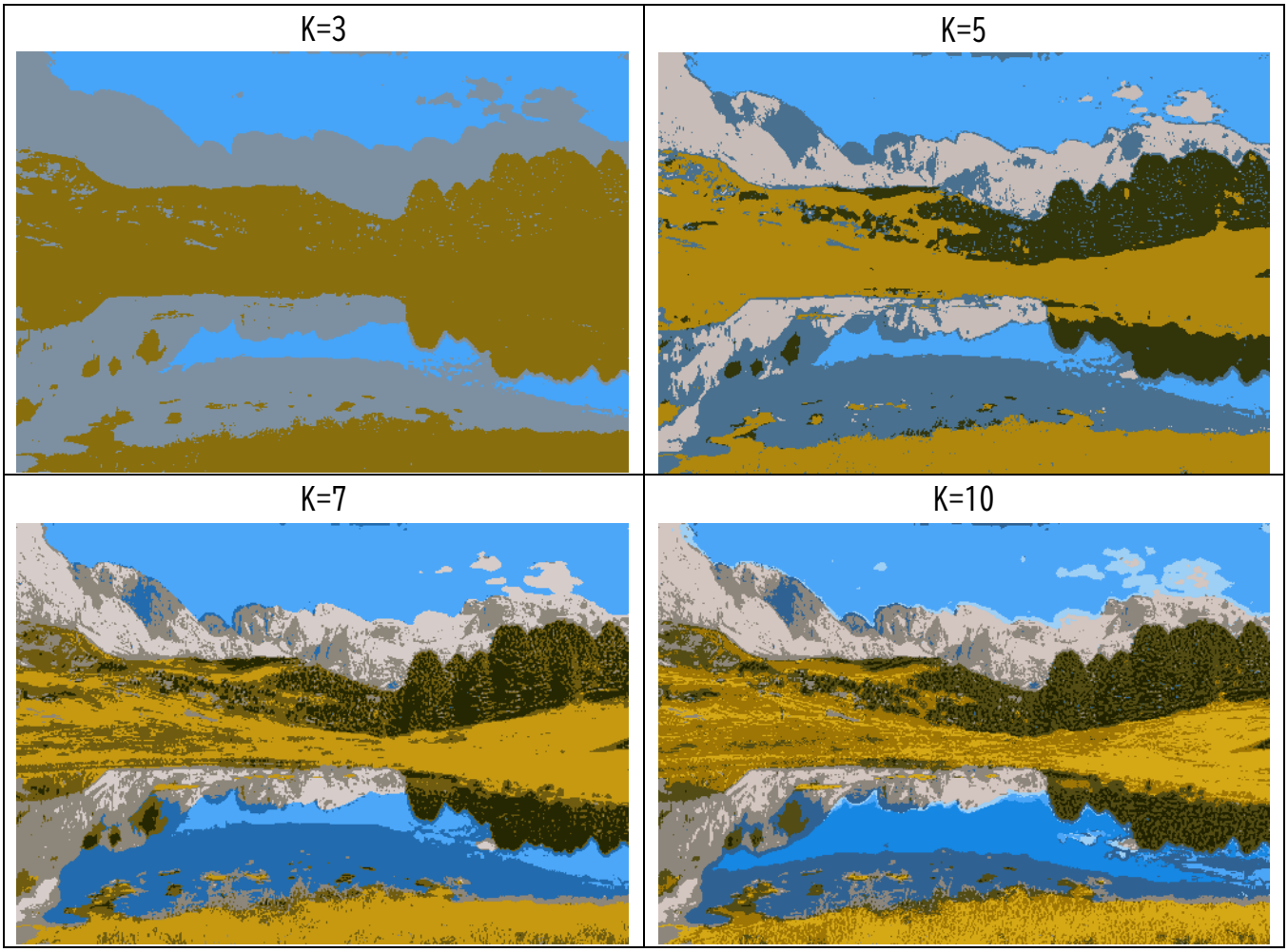
$$\mu_k = \text{kmeans centers}$$

$$\pi_k = \frac{N_k}{N}$$

$$\Sigma_k = \frac{1}{N_k}\sum_{n\epsilon k}(x_n - \mu_k)(x_n - \mu_k)^T$$

比較麻煩的點是，我的μ_k是三維的向量，所以便藝術的部分是一個矩陣，後續在 GMM 中還需要進行一些矩陣的反矩陣與行列式，會消耗掉比較多計算資源。

b. GMM with kmeans initialization



根據 kmeans 所求得的 k 個中心點來做為 GMM 的起始條件，接著使用以下的 EM 方法來做計算：

E step：

$$r(Z_{nk}) = \frac{\pi_k N(X_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{k} \pi_j N(X_n|\mu_j, \Sigma_j)}$$

M step：

$$\mu_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} r(Z_{nk}) X_n$$

$$\Sigma_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} r(Z_{nk})(x_n - \mu_k^{new})(x_n - \mu_k^{new})^T$$

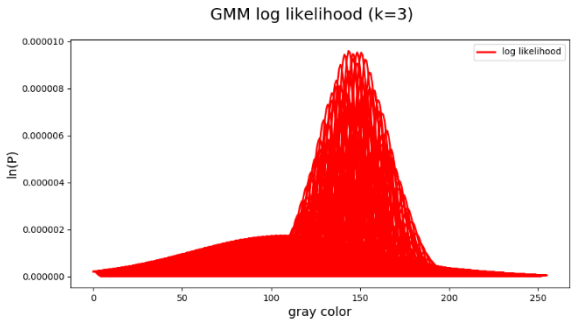
$$\pi_k^{new} = \frac{N_k}{N}$$

$$N_k = \sum_{n=1}^{N} r(Z_{nk})$$

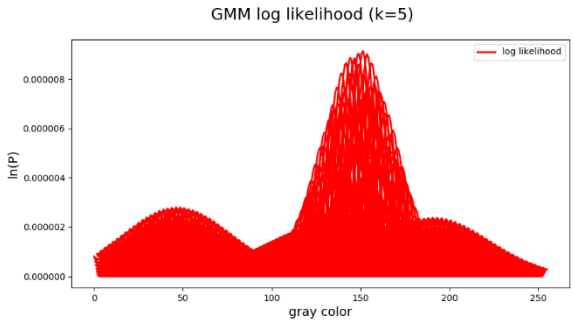
透過不斷更新μ_k、π_k、Σ_k，我們可以得到 GMM 後的新群集中心。比較特別的是 GMM 考量的都是機率，不像 Kmeans 是直接將資料分給特定群體中心，GMM 討論的是在每個類別中可能的機率。

c. Likelihood

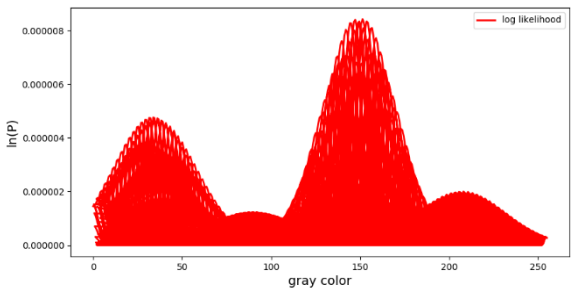
K=3



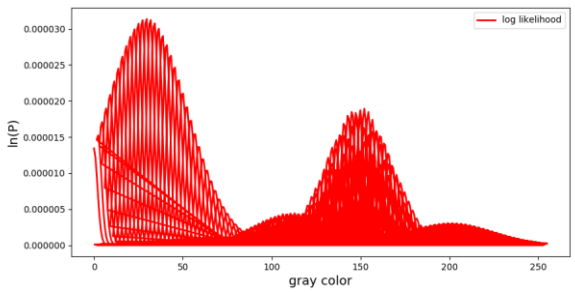
K=5



K=7



K=10



因為我的輸入資料是三維資料(rgb 三個顏色)，一直找不到可以畫出 4D 空間方法。因此，我將原本的顏色 rgb 先算出 likelihood 的值後，再將 rgb 轉成灰階值，藉此畫出來。

所以會看到相同灰階值會有不同的區間值，是因為同一個灰階值有可能來自不同的 rgb，所以才會得到此結果。