Machine Learning (HW03)

Course： NCTU-ECM5094-ML

*ID：309505002 *Name：鄭紹文

Q1：Gaussian Process for Regression

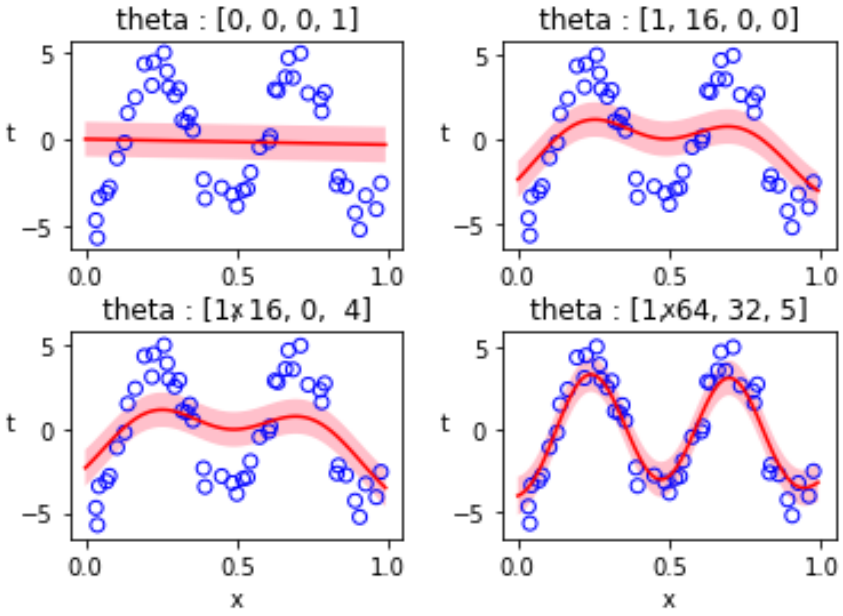

Figure 1

```
Training Set RMS error
theta set : [0, 0,  0, 1]: 3.1292014298222437
theta set : [1, 4,  0, 0]: 2.423927927831219
theta set : [1, 4,  0, 5]: 2.4105764871252062
theta set : [1, 64, 32,5]: 1.0407524150477088
Test Set RMS error
theta set : [0, 0,  0, 1]: 3.3201767255281416
theta set : [1, 4,  0, 0]: 2.4656937876934766
theta set : [1, 4,  0, 5]: 2.4557650874208927
theta set : [1, 64, 32,5]: 1.0906935313340005
```

Figure 2

$$k(x_n, x_m) = \theta_0 \exp\left\{-\frac{\theta_1}{2}\|x_h - x_m\|^2\right\} + \theta_2 + \theta_3 x_n T x_m$$

Kernel function 可以計算兩個向量($\mathbf{x}n$、$\mathbf{x}m$)之間的相似程度，透過 data point 並使用 kernel function 可以得到 model 的 mean、covariance。由 Figure 1 可知，當 theta=[0, 0, 0, 1](linear kernel model)時，明顯錯誤相對於其他得來的高，猜測原因在於它只加入 linear $x_n^T x_n$，變化相對少，model 過於簡單，無法 fit data point，由 Figure2 的 root-mean-square error 也可得到相同結果，在 training set 與 testing set 的 root-mean-square error 皆遠高於其他的 theta 組合，為 underfitting。然而，當 kernel function 加入 exponential term($\theta 1 \neq 0$)後，可由 Figure 1 看出，model 已經 fit 大部分的 data points，

Figure 2 中在 training set 與 testing set 的 root-mean-square error 也明顯下降許多，慢慢依序加入$x_n^T x_n$項($\theta 3 \neq 0$)與 constant 項($\theta 2 \neq 0$)，model 複雜度增加，由 Figure 2 可得在 training set 與 testing set 的 root-mean-square error 有些微下降。

因此可得知 exponential kernel 可以 fit 大部分的 data point，加入 linear$x_n^T x_n$與 constant 可以使結果更好，但成效似乎並不大。

```
*---------------------------*
*  tune hyperparameters...  *
*---------------------------*
```

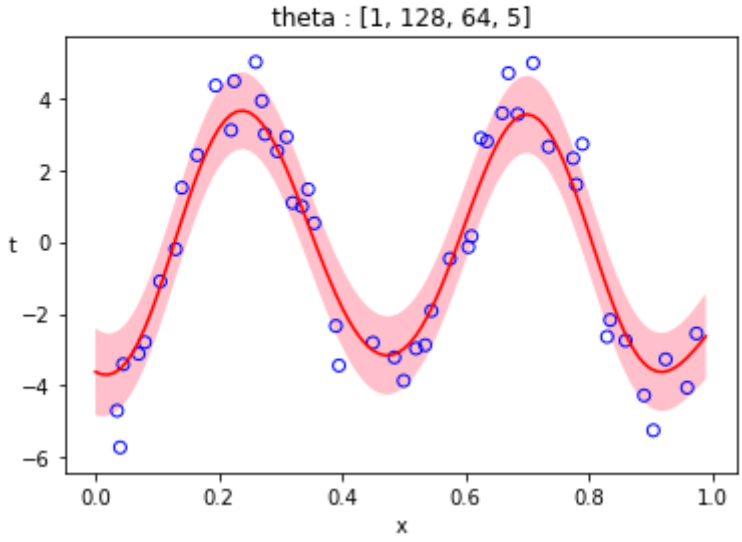

Figure 3

```
Training Set RMS error
  after tuning......
theta set : [1, 128, 64, 10]: 0.897770446187698
Test Set RMS error
after tuning......
theta set : [1, 128, 64, 10]: 0.9844080560106174
```

Figure 4

Figure 3 與 Figure 4 為透過 trial and error 調整 hyperparameter θ，得到當 θ= [1, 128, 64, 10]時，在 training set 與 testing set 上的 root-mean-square error 皆較其他的 θ 組合低。

Q2：Support Vector Machine

■ **One-versus-the-rest classifier :**

訓練時依序將某個類別的 data point 歸為一類，其他的 data point 歸為一類。如此，若有 k 個類別，就會有 k 個 classifier。分類時將未知的 data point 代入 k 個 classifier，並將該 data point 分類為具有最大分類函數值的類別。

■ **One-versus-one classifier :**

在 k 個類別中，任選兩類得到一個 classifier。因此，若有 k 個類別，就會得到 k(k-1)/2 個 classifier，分類時將未知的 data point 分類為得票最多的類別。

兩者不同地方為，one-versus-one classifier 每一次 binary classification，都是用正確類別的 data，沒有加入其他類別的 data。而且 one-versus-the-rest classifier 會有 training set imbalanced 的問題。

```
*-------------------------*
*  plot linear result...  *
*-------------------------*
```

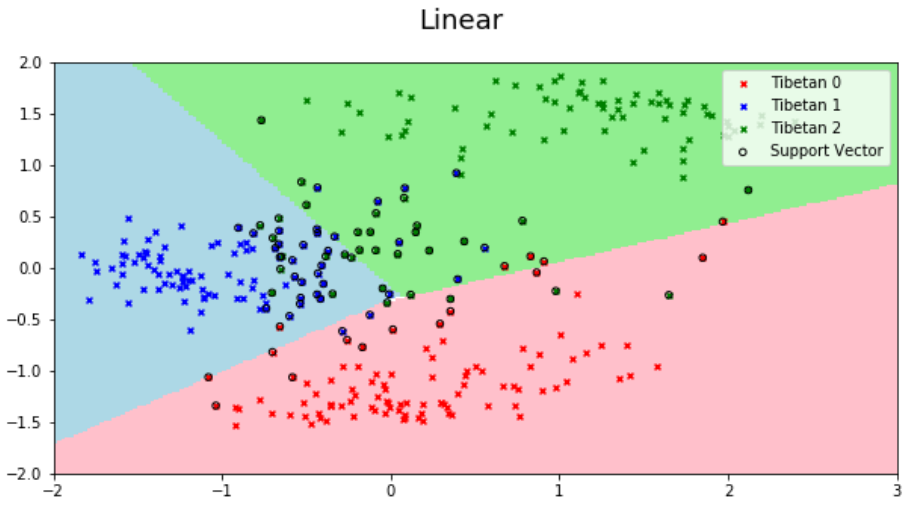

Figure 5

```
*----------------------------*
*  plot Polynomial result...  *
*----------------------------*
```

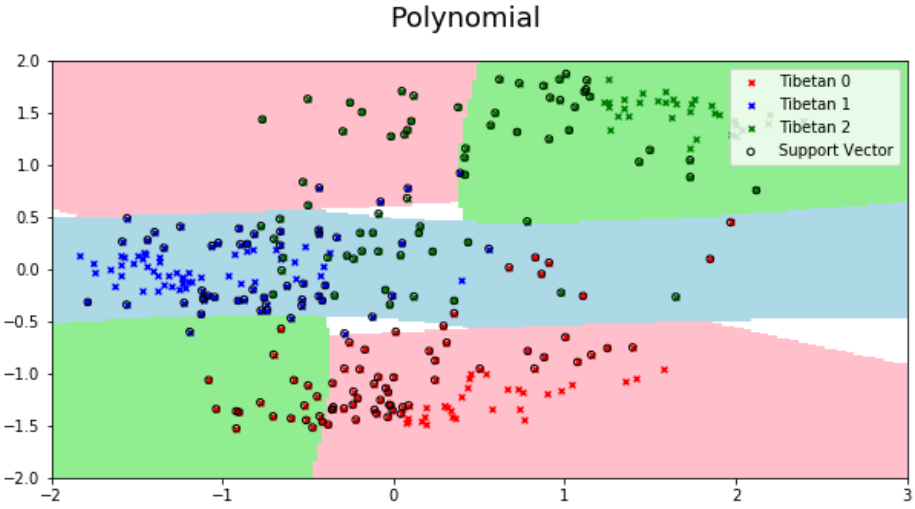

Figure 6

這部分使用了 scikit-learn 套件來求出該題 SVM 所要計算的 Lagrange Multipliers 與 support vectors，此題 SVM 的求法，有先對資料進行 PCA 降為後，做標準化，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，須進行標準化，此方面採用了 one-versus-one 來解決 multiclass，並利用 scikit-learn 裡的 fit_data 求得 coefficient，分配 multiplier 給對應的分類器，在算出每個分類器的 weight 和 bias 進行預測，最後採用投票作為分類結果。
在得到 multiplier 後，根據:

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

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

可以求出三條 decision boundary，定義↓

$$\begin{cases} y_0(x): \ class\ 0\ v.s.\ class1 \\ y_1(x): \ class\ 0\ v.s.\ class2 \\ y_2(x): \ class\ 1\ v.s.\ class2 \end{cases}$$

再帶入 x 計算出於這三條邊界上的正負來判斷屬於何種類別，
皆為正時：分類至 class 0；y2(x)為負且其餘為正時：分類至 class0；
y0(x)為負且其餘為正時：分類至 class1；y0(x)為負、其餘為正時：分類至 class1；
y1(x)為負、y2(x)為負且於為正時：分類至 class2；皆為負時：分類至 class2；

分佈上大致可以分為三群，overlap 的狀況並非很嚴重。而單從實驗的分類結果來看，我認為 linear kernel 分類的還行，雖說 overlap 的點無法分類正確，但由於資料本身的重疊狀況就不多，故結果還不錯。反觀 polynomial kernel，可以明顯發現明顯有分類到錯誤的類別，猜測可能原因是 polynomial kernel function 將資料投影到 feature space 後的情況無法明顯將不同類別的資料區隔出來。最後，觀察發現，linear kernel 可以用三調線將資料分開，所以 support vector 單純分佈在分隔線的附近；

polynomial kernel用了七條線去分隔，所以 support vector 的數量大幅增加，也同時增加記憶體的用量。所以透過這個實驗，應該要觀察資料實際分佈的特性，而去選用適合的 kernel function 去訓練。

Q3：Gaussian Mixture Model:

K-means :

在 dataset 中隨機找出 K 筆作為初始群集中心，並計算每一筆 data 到 K 個群集中心的距離，並將該筆 data 分類給與之最接近的群集中心，並產生一個群集邊界，再計算每個群集的新質量中心，作為新的該群集中心。再利用新群集中心計算距離，以此循環，直到群集成員不再變動。

Expectation maximization (EM) algorithm 可用來解決存在隱藏資訊最佳化問題，將 K-means 計算出的 mean 與 variance、mixing coefficient 等參數，再經由 EM algorithm 最佳化。

```
mean
[[88.03466906726219, 124.49392958553435, 136.2564261226166], [122.4747033385077, 130.1957630324162, 126.72625514985363], [59.0100375547858, 67.57000742499315,
 80.56001246999203]]
Pi
[0.30227462518621634, 0.4247402979672585, 0.27298507684652695]
covariance
[array([[1316.74746662, 1238.22307163, 1388.66203673],
       [1238.22307163, 1743.05946844, 1701.41965489],
       [1388.66203673, 1701.41965489, 1937.58647456]]), array([[3720.91346459, 3296.72437372, 3070.51114167],
       [3296.72437372, 3067.03296335, 2924.42267463],
       [3070.51114167, 2924.42267463, 2890.25680381]]), array([[444.23977195, 458.10136837, 497.02530707],
       [458.10136837, 495.98129641, 546.61831418],
       [497.02530707, 546.61831418, 638.71701376]])]
```

Figure 7. (Kmeans=3)

```
mean
[[59.227667127516575, 70.17696064779186, 73.33079672660621], [111.70274851241342, 150.1521702336231, 141.0282096236039], [58.59414540852439, 66.04760895694228,
 79.58893625141792], [164.11450107561635, 165.18403074718853, 159.78689594318365], [86.05982911771754, 110.63426863124442, 125.84696547356472]]
Pi
[0.16185487585275507, 0.14144361024662053, 0.19161544794147878, 0.1877960303290723, 0.31729003563007346]
covariance
[array([[688.580187  , 554.95355313, 463.69643235],
       [554.95355313, 511.07266176, 457.61502873],
       [463.69643235, 457.61502873, 478.66588474]]), array([[2464.12392005, 1356.97606486, 1127.94026536],
       [1356.97606486, 1621.43388305, 1650.56569276],
       [1127.94026536, 1650.56569276, 1801.96429334]]), array([[406.66255503, 423.72215281, 456.36420284],
       [423.72215281, 454.87031008, 495.49959665],
       [456.36420284, 495.49959665, 563.50012243]]), array([[2278.06712388, 2208.72470243, 2215.71963882],
       [2208.72470243, 2225.04640061, 2253.28894746],
       [2215.71963882, 2253.28894746, 2314.02956026]]), array([[1260.35403148, 1252.40640272, 1423.29944265],
       [1252.40640272, 1415.70427826, 1549.58760751],
       [1423.29944265, 1549.58760751, 1892.2061863 ]])]
```

Figure 8. (Kmeans=5)

```
mean
[[118.05285165324734, 145.09244343330778, 156.87728558733716], [185.66993813390778, 189.24695010379753, 183.9634674207229], [57.54419741184006, 64
.74960476965322, 78.01822984836787], [79.22836140599905, 132.12923357460159, 128.49918890475476], [128.09900944708238, 127.04536011290507, 122.19104986396187],
[75.52625960021501, 98.94733208828336, 114.81800324094841], [54.20247780164716, 65.72175345488851, 68.36209041533203]]
Pi
[0.12694520548729668, 0.11743943405856237, 0.17857066841312283, 0.10313368327610789, 0.13244191327516536, 0.18886259469664926, 0.1526065007930824]
covariance
[array([[ 978.47060884, 1108.32404155, 1083.93977199],
       [1108.32404155, 1481.95440461, 1464.40909378],
       [1083.93977199, 1464.40909378, 1821.1476681 ]]), array([[1612.41303082, 1463.5730624 , 1510.61441067],
       [1463.5730624 , 1414.27048895, 1466.02065074],
       [1510.61441067, 1466.02065074, 1541.62931077]]), array([[359.91716698, 373.27680403, 402.18931903],
       [373.27680403, 399.24270635, 434.73933646],
       [402.18931903, 434.73933646, 496.7887436 ]]), array([[1006.86958398,  882.8243269 ,  897.94878383],
       [ 882.8243269 , 1659.35279583, 1686.19908375],
       [ 897.94878383, 1686.19908375, 1740.450317  ]]), array([[2057.47804822, 1746.72270407, 1301.92615427],
       [1746.72270407, 1587.99337136, 1287.20161737],
       [1301.92615427, 1287.20161737, 1183.70439798]]), array([[ 846.44259053,  725.19343307,  913.24246345],
       [ 725.19343307,  745.77622778,  881.50363676],
       [ 913.24246345,  881.50363676, 1172.04691112]]), array([[620.60338273, 516.25899241, 398.87961249],
       [516.25899241, 488.84469384, 405.78070178],
       [398.87961249, 405.78070178, 425.18370437]])]
```

Figure 9.(Kmeans=7)

```
mean
[[88.1372626188783, 110.75389138924055, 130.50778079605038], [226.23375037336552, 227.2415793534999, 226.82373533150943], [150.95550334533132, 153
.93053427878917, 150.86583476434052], [58.255707258544305, 65.51975522630161, 78.5126607940024], [36.04006097304803, 57.9086370352042, 64.33023274992304], [72
.33005690932617, 130.14270278285963, 126.73152922030292], [167.9453244508826, 178.51252541744807, 172.4678817588247], [121.63625775852391, 118.87124824901531,
 113.6738314352459], [118.61580167089853, 150.41716262963146, 164.35477300171823], [71.15643134222104, 84.03529773599553, 87.24454188869211]]
Pi
[0.12907651380500992, 0.019562947743628573, 0.08293686277295438, 0.19132078793083598, 0.08906760860519582, 0.087281866847325, 0.08765474662174136, 0
.09446693965231379, 0.0838596710766169, 0.13477205510695012]
covariance
[array([[582.60204932, 450.53973096, 548.52197322],
       [450.53973096, 478.5714772 , 521.99724698],
       [548.52197322, 521.99724698, 692.16414986]]), array([[560.98133415, 542.79571978, 547.88820579],
       [542.79571978, 546.46795866, 548.91140929],
       [547.88820579, 548.91140929, 552.88085288]]), array([[1454.9313826 , 1212.09512681,  812.03766955],
       [1212.09512681, 1278.41791521, 1172.84224278],
       [ 812.03766955, 1172.84224278, 1430.28550235]]), array([[288.62084378, 296.31545999, 320.47752839],
       [296.31545999, 317.11729257, 347.00440312],
       [320.47752839, 347.00440312, 404.88827893]]), array([[316.48353487, 353.6829327 , 320.87682911],
       [353.6829327 , 497.27320824, 443.76862069],
       [320.87682911, 443.76862069, 470.52725406]]), array([[ 860.42160262,  882.03544746,  901.65038312],
       [ 882.03544746, 1729.93660625, 1753.61195451],
       [ 901.65038312, 1753.61195451, 1805.00948606]]), array([[1998.56309932, 1611.75347603, 1616.65216811],
       [1611.75347603, 1403.460753  , 1414.24169291],
       [1616.65216811, 1414.24169291, 1445.98456072]]), array([[2242.66461652, 2092.39905187, 1947.18158084],
       [2092.39905187, 1985.41426921, 1862.37973283],
       [1947.18158084, 1862.37973283, 1773.38258532]]), array([[ 845.11883742, 1028.01946085,  982.82542439],
       [1028.01946085, 1500.92858888, 1386.34478285],
       [ 982.82542439, 1386.34478285, 1679.92679819]]), array([[690.32012731, 649.41075368, 551.34939917],
       [649.41075368, 674.35718764, 626.54805827],
       [551.34939917, 626.54805827, 742.55909839]])]
```

Figure 10. (Kmeans=10)

將每個 pixel 的 rgb 取出來，取法是把所有資料平均取 k 個點，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，其中各個不同 k 最後得到的中心點與數量結果，再透過以下公式，來將 kmeans 求出的 k 點中心來做為 GMM 的初值，進行後續 EM 計算：

μk = kmeans centers

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

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

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EM方法來做計算：

E step：

$$r(z_{n_k}) = \frac{\pi_k N(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{k}\pi_j N(x_n|\mu_j, \Sigma_j)}$$

M step：

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

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

$$\pi_k^{new} = \frac{1v_k}{iv}$$

$$N_k = \sum_{n=1}^{1^v} r(z_{nk})$$

透過不斷更新μk、πk、Σk，可以得到GMM後的新群集中心，和Kmeans直接將資料分給特定群體中心不同，GMM偏重於每個類別可能的機率，GMM的收斂隨著K越大，達到收斂所需要的iteration更多。