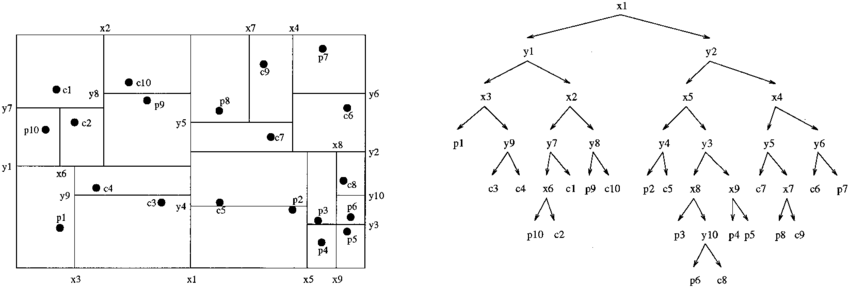
Introduction to Machine Learning Program assignment #2

Environment: Ubuntu 16.04.3 LTS

Using library: numpy、math、sys

Language: Python 2.7.12

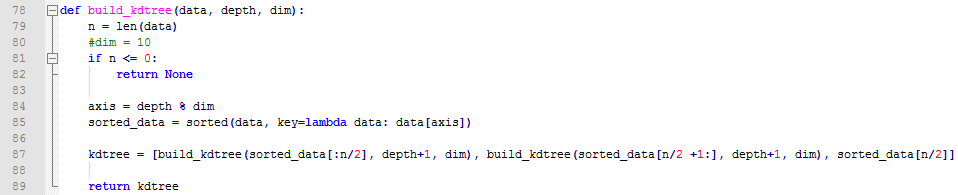
**K-D Tree Algorithm**



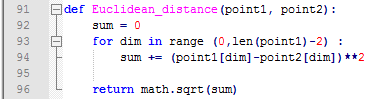
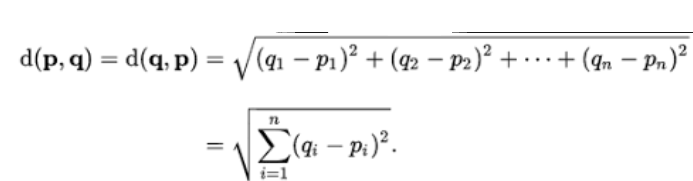
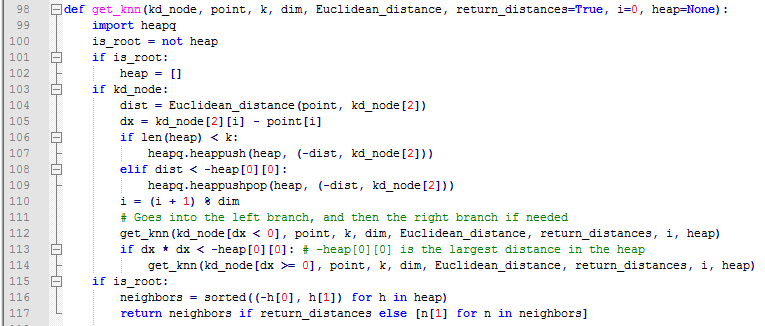
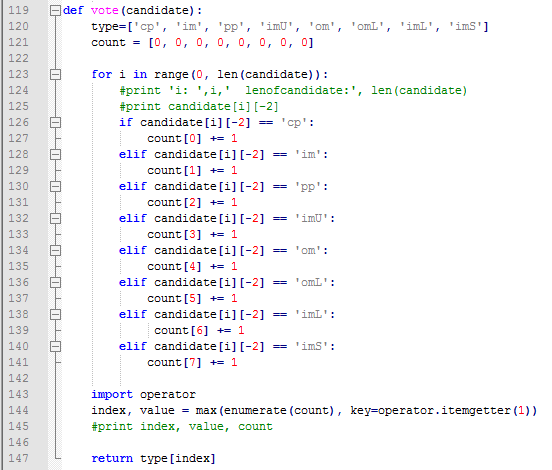
(圖片來源：https://www.researchgate.net/figure/2334587\_fig9\_Figure-11-Adaptive-k-d-tree)

K-D Tree可以用來分類，從第一個feature到最後一個feature依序切割，sort後用中位數的點當分割點，小於節點的接左子樹、大於節點的接右子樹，直到所有feature的中位數都被當成分割點去recurrsive做出K-D tree，我們的樹便種好了！

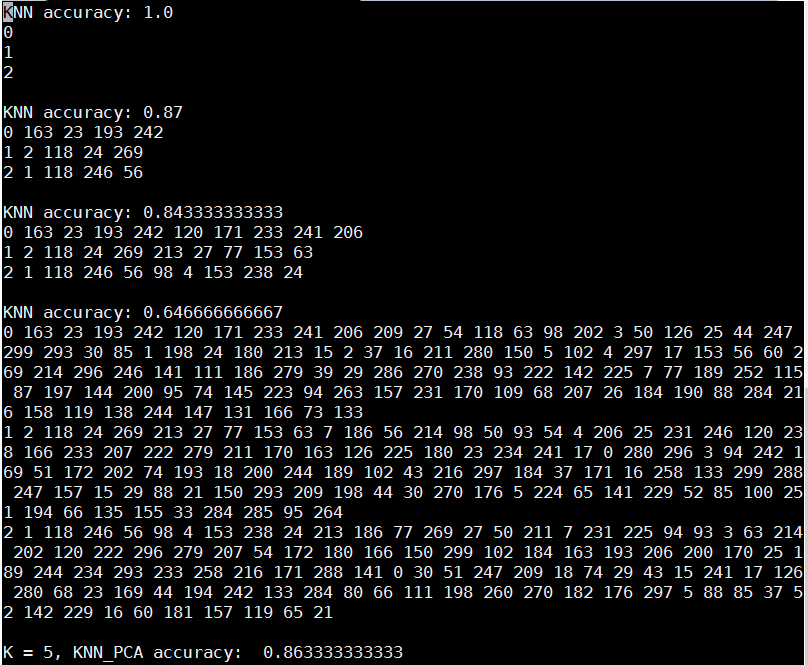
**我的作法與使用的function**

1. (這次的作業我有到youtube找tutorial多加認識K-D tree 觀看K-d Tree in Python #1.2.3─https://www.youtube.com/watch?v=u4M5rRYwRHs)  
   在這些影片裡，我學會用recurrsive function建立K-D tree、計算nearest data等等的觀念  
   而且作者的dataset是用dictionary的格式，所以我對python的dictionary有了更多的了解
2. build\_kdtree(data, depth, dim)  
   

一開始傳進depth=0，每一次的recursive call會再依序加1，depth的作用是讓我們選feature，從index為0的開始切割，並回傳樹的左子樹、右子樹、自己，並分別繼續build\_kdtree(data, depth, dim)建下面的子樹，直到Line81的條件：n<=0成立時，已經切到最小單位，return None完成那一輪的recursion。

1. Euclidean\_distance(point1, point2)  
     
   計算distance的function使用Euclidean的算法，讓兩點間的所有dim值平方相加後開根號即為所求，。
2. get\_knn(kd\_node, point, k, dim, Euclidean\_distance, return\_distances=True, i=0, heap=None)  
     
   (這個function的概念我有參考https://github.com/Vectorized/Python-KD-Tree/blob/master/kdtree.py)  
   第一次傳進這個function時kd\_node是tree、point是我們要query的點、i初值為0，即現在樹的深度(feature的index為第0個feature)、heap為None，所以Line100的is\_root會是true，代表這是樹的最頂端，並繼續往下search。  
   Line105dx為root減query point的距離，若該點在root的左邊dx為正；反之在右子樹則為負。  
   Line106如果現在heap內的點還沒到達k個就直接push那個點進去，且push進去時紀錄 –dist，heapq的function會自動按照大小排序好，因此之後pop會從小的pop，也就是pop到距離最遠的。  
   Line108如果queue內已經有k個點了，我們先將當下的點push進去，然後再把最距離最遠的pop出來。  
   Line110讓i+1，深度+1，往下一個feature去recursive計算對應到的結果。  
   Line113是防呆機制，以免另一邊有距離更近的點，所以設定當要query的點減最近的feature=dx小於heap中最的最大距離，我們就往tree的另一邊子樹試試看，因為另一端可能存在更距離接近的人選。
3. vote(candidate)  
     
    vote這個function會計算得票數最多的答案，使用時將knn找出的k個資料，他們分類到的type黏成一個list，傳入vote後取MAX找出index，回傳最有可能的type。

**Testing Result**K-D tree + KNN validation:



這裡我將助教給的300筆資料切割出280筆train、20筆test，因為只需要顯示前三筆資料就好，所以上面的output file都只印最前三筆的k個result，而這20筆testing data的accuracy結果如下：  
當k = 1時→accuracy為1.0

當k = 5時→accuracy為0.87

當k = 10時→accuracy為0.843

當k = 100時→accuracy為0.646

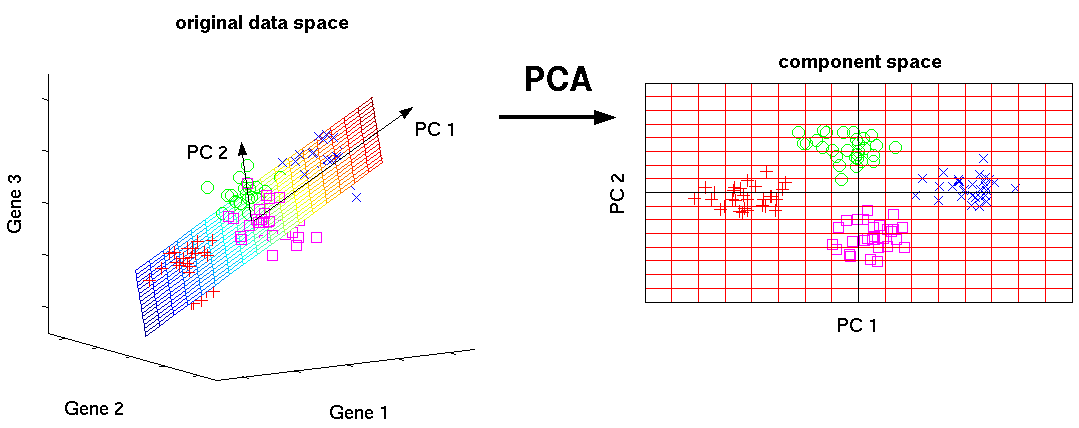
可以看出accuracy隨著k越多範圍放大而下降，可能是因為我們的dataset不算大而導致。

PCA:

當k = 5時→accuracy為0.863

其實跟降維之前的結果差不多，當dataset變大，PCA可能會有更厲害的效果體現accuracy及time complexity上。

**PCA(bonus)**



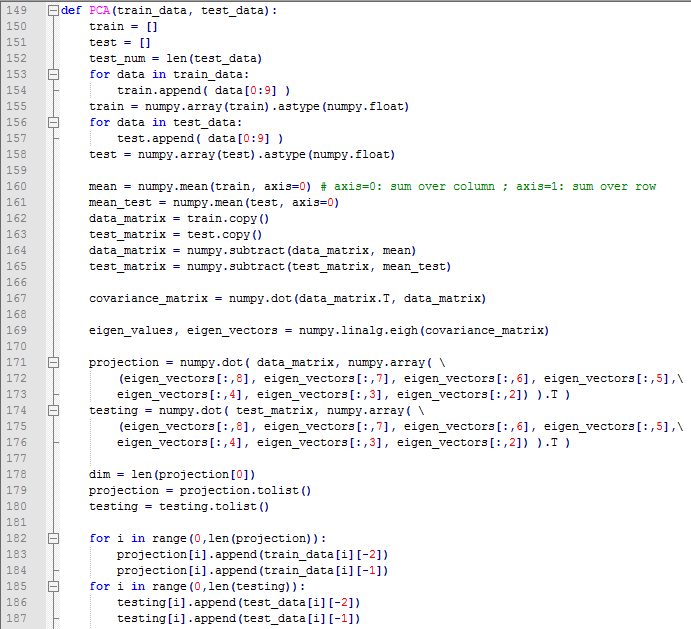
(圖片來源：https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/)

PCA利用線性變換將原始資料轉換為一組各維度線性獨立的資料，將高維向量投影到低維空間中，是一種降低數據維度的有效辦法，步驟如下：

設有m條n維數據。

1）將原始數據按列組成n行m列矩陣X  
2）將X的每一行（代表一個屬性字段）進行零均值化，即減去這一行的均值  
3）求出協方差矩陣  
4）求出協方差矩陣的特徵值及對應的特徵向量  
5）將特徵向量按對應特徵值大小從上到下按行排列成矩陣，取前k行組成矩陣P  
6）Y=PX即為降維到k維後的數據

**我的作法與使用的function**



這部分其實和前面差不多，只是training data和testing data要做前處理，我的作法是只取其中eigenvalue最高7個的features。

Line153-158是把train和test的數值部分(9個features)取出來，並轉換成numpy array float的型式，後面就能直接使用numpy的function去計算mean、矩陣的dot和subtract、eigen value、eigenvector。

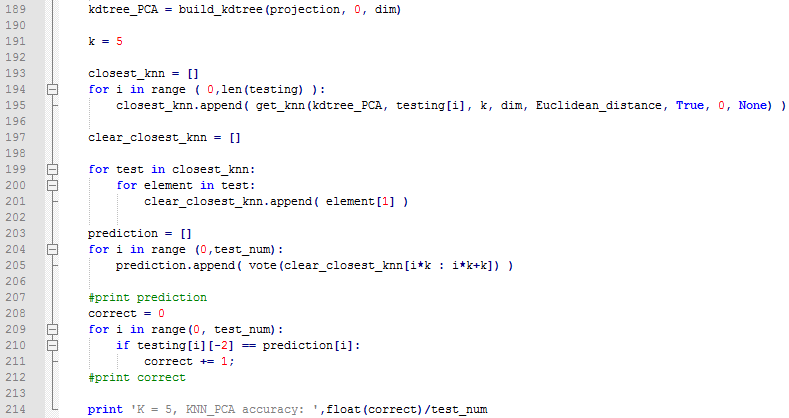
Line164-165先將資料複製到另一個矩陣，減掉該矩陣的均值。

Line167是把training data的他的transpose相乘，計算出covariance matrix。

Line169使用numpy內建計算eigenvalue和eigenvector的function，他會以ascending order的排序方式，回傳計算結果，因此我們要取的vector是從後面往前取。

Line171-176讓trainingdata和testingdata乘上eigenvector做轉換，我取了7個feature，從index8~2，data\_matrix的size是資料數\*9、我取的eigen vactor size是7\*9，所以要將後者轉置再相乘。

Line179-187計算完畢後轉回list的型式，並黏回每一筆資料的type分類和index值。



Line191設定k為5。

projection丟入build\_kdtree的function，種出屬於PCA的tree，再放入get\_knn的function得出每一筆data的k個候選人後進行投票即可對答案算accuracy。