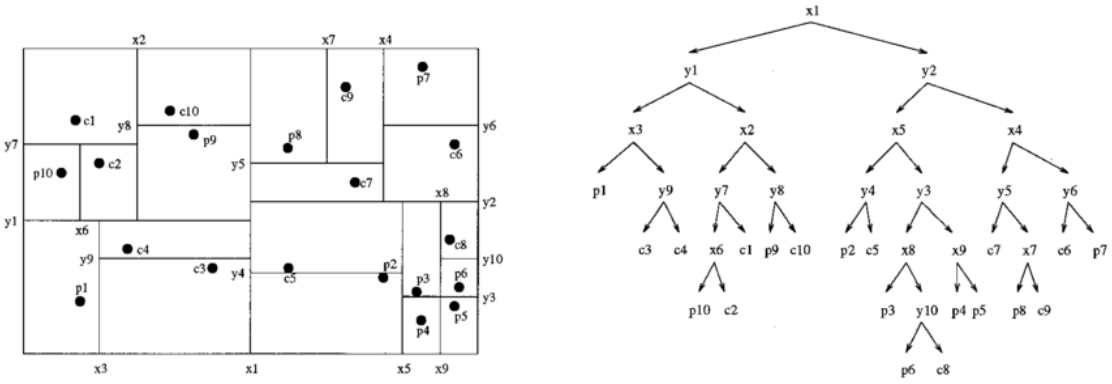
Introduction to Machine Learning Program assignment #2

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)

```
78  def build_kdtree(data, depth, dim):
79      n = len(data)
80      #dim = 10
81      if n <= 0:
82          return None
83
84      axis = depth % dim
85      sorted_data = sorted(data, key=lambda data: data[axis])
86
87      kdtree = [build_kdtree(sorted_data[:n/2], depth+1, dim), build_kdtree(sorted_data[n/2 +1:], depth+1, dim), sorted_data[n/2]]
88
89      return kdtree
```

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。

3. Euclidean_distance(point1, point2)

```
91  def Euclidean_distance(point1, point2):
92      sum = 0
93      for dim in range (0,len(point1)-2) :
94          sum += (point1[dim]-point2[dim])**2
95
96      return math.sqrt(sum)
```

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，

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\cdots+(q_n-p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

。

4. get_knn(kd_node, point, k, dim, Euclidean_distance, return_distances=True, i=0, heap=None)

```
98  def get_knn(kd_node, point, k, dim, Euclidean_distance, return_distances=True, i=0, heap=None):
99      import heapq
100     is_root = not heap
101     if is_root:
102         heap = []
103     if kd_node:
104         dist = Euclidean_distance(point, kd_node[2])
105         dx = kd_node[2][i] - point[i]
106         if len(heap) < k:
107             heapq.heappush(heap, (-dist, kd_node[2]))
108         elif dist < -heap[0][0]:
109             heapq.heappushpop(heap, (-dist, kd_node[2]))
110         i = (i + 1) % dim
111         # Goes into the left branch, and then the right branch if needed
112         get_knn(kd_node[dx < 0], point, k, dim, Euclidean_distance, return_distances, i, heap)
113         if dx * dx < -heap[0][0]: # -heap[0][0] is the largest distance in the heap
114             get_knn(kd_node[dx >= 0], point, k, dim, Euclidean_distance, return_distances, i, heap)
115     if is_root:
116         neighbors = sorted((-h[0], h[1]) for h in heap)
117         return neighbors if return_distances else [n[1] for n in neighbors]
```

(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。

5. vote(candidate)

```
119 def vote(candidate):
120     type=['cp', 'im', 'pp', 'imU', 'om', 'omL', 'imL', 'imS']
121     count = [0, 0, 0, 0, 0, 0, 0, 0]
122 
123     for i in range(0, len(candidate)):
124         #print 'i: ',i,'  lenofcandidate:', len(candidate)
125         #print candidate[i][-2]
126         if candidate[i][-2] == 'cp':
127             count[0] += 1
128         elif candidate[i][-2] == 'im':
129             count[1] += 1
130         elif candidate[i][-2] == 'pp':
131             count[2] += 1
132         elif candidate[i][-2] == 'imU':
133             count[3] += 1
134         elif candidate[i][-2] == 'om':
135             count[4] += 1
136         elif candidate[i][-2] == 'omL':
137             count[5] += 1
138         elif candidate[i][-2] == 'imL':
139             count[6] += 1
140         elif candidate[i][-2] == 'imS':
141             count[7] += 1
142 
143     import operator
144     index, value = max(enumerate(count), key=operator.itemgetter(1))
145     #print index, value, count
146 
147     return type[index]
```

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:

```
KNN accuracy: 1.0
0
1
2

KNN accuracy: 0.87
0 163 23 193 242
1 2 118 24 269
2 1 118 246 56

KNN accuracy: 0.843333333333
0 163 23 193 242 120 171 233 241 206
1 2 118 24 269 213 27 77 153 63
2 1 118 246 56 98 4 153 238 24

KNN accuracy: 0.646666666667
0 163 23 193 242 120 171 233 241 206 209 27 54 118 63 98 202 3 50 126 25 44 247
299 293 30 85 1 198 24 180 213 15 2 37 16 211 280 150 5 102 4 297 17 153 56 60 2
69 214 296 246 141 111 186 279 39 29 286 270 238 93 222 142 225 7 77 189 252 115
 87 197 144 200 95 74 145 223 94 263 157 231 170 109 68 207 26 184 190 88 284 21
6 158 119 138 244 147 131 166 73 133
1 2 118 24 269 213 27 77 153 63 7 186 56 214 98 50 93 54 4 206 25 231 246 120 23
8 166 233 207 222 279 211 170 163 126 225 180 23 234 241 17 0 280 296 3 94 242 1
69 51 172 202 74 193 18 200 244 189 102 43 216 297 184 37 171 16 258 133 299 288
 247 157 15 29 88 21 150 293 209 198 44 30 270 176 5 224 65 141 229 52 85 100 25
1 194 66 135 155 33 284 285 95 264
2 1 118 246 56 98 4 153 238 24 213 186 77 269 27 50 211 7 231 225 94 93 3 63 214
 202 120 222 296 279 207 54 172 180 166 150 299 102 184 163 193 206 200 170 25 1
89 244 234 293 233 258 216 171 288 141 0 30 51 247 209 18 74 29 43 15 241 17 126
 280 68 23 169 44 194 242 133 284 80 66 111 198 260 270 182 176 297 5 88 85 37 5
2 142 229 16 60 181 157 119 65 21

K = 5, KNN_PCA accuracy:  0.863333333333
```

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

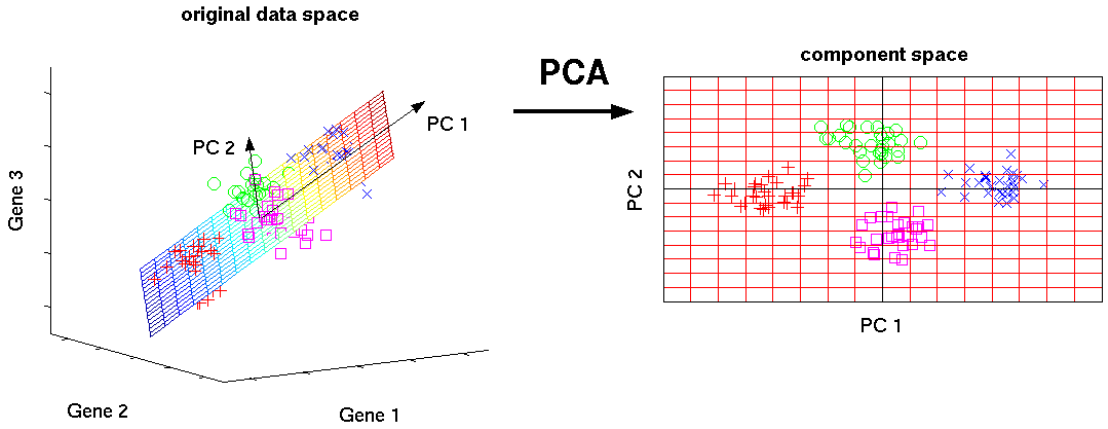
可以看出 accuracy 隨著 k 越多範圍放大而下降，可能是因為我們的 dataset 不算大而導致。

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 $C = \frac{1}{m} X X^{\mathrm{T}}$

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

```
149 def PCA(train_data, test_data):
150     train = []
151     test = []
152     test_num = len(test_data)
153     for data in train_data:
154         train.append( data[0:9] )
155     train = numpy.array(train).astype(numpy.float)
156     for data in test_data:
157         test.append( data[0:9] )
158     test = numpy.array(test).astype(numpy.float)
159 
160     mean = numpy.mean(train, axis=0) # axis=0: sum over column ; axis=1: sum over row
161     mean_test = numpy.mean(test, axis=0)
162     data_matrix = train.copy()
163     test_matrix = test.copy()
164     data_matrix = numpy.subtract(data_matrix, mean)
165     test_matrix = numpy.subtract(test_matrix, mean_test)
166 
167     covariance_matrix = numpy.dot(data_matrix.T, data_matrix)
168 
169     eigen_values, eigen_vectors = numpy.linalg.eigh(covariance_matrix)
170 
171     projection = numpy.dot( data_matrix, numpy.array( \
172         (eigen_vectors[:,8], eigen_vectors[:,7], eigen_vectors[:,6], eigen_vectors[:,5],\
173         eigen_vectors[:,4], eigen_vectors[:,3], eigen_vectors[:,2]) ).T )
174     testing = numpy.dot( test_matrix, numpy.array( \
175         (eigen_vectors[:,8], eigen_vectors[:,7], eigen_vectors[:,6], eigen_vectors[:,5],\
176         eigen_vectors[:,4], eigen_vectors[:,3], eigen_vectors[:,2]) ).T )
177 
178     dim = len(projection[0])
179     projection = projection.tolist()
180     testing = testing.tolist()
181 
182     for i in range(0,len(projection)):
183         projection[i].append(train_data[i][-2])
184         projection[i].append(train_data[i][-1])
185     for i in range(0,len(testing)):
186         testing[i].append(test_data[i][-2])
187         testing[i].append(test_data[i][-1])
```

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。

```
189     kdtree_PCA = build_kdtree(projection, 0, dim)
190
191     k = 5
192
193     closest_knn = []
194     for i in range ( 0,len(testing) ):
195         closest_knn.append( get_knn(kdtree_PCA, testing[i], k, dim, Euclidean_distance, True, 0, None) )
196
197     clear_closest_knn = []
198
199     for test in closest_knn:
200         for element in test:
201             clear_closest_knn.append( element[1] )
202
203     prediction = []
204     for i in range (0,test_num):
205         prediction.append( vote(clear_closest_knn[i*k : i*k+k]) )
206
207     #print prediction
208     correct = 0
209     for i in range(0, test_num):
210         if testing[i][-2] == prediction[i]:
211             correct += 1;
212     #print correct
213
214     print 'K = 5, KNN_PCA accuracy: ',float(correct)/test_num
```

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。