0310120 陳家煒

Introduction to Machine Learning HW2

- Task：Implement Kd-Tree and use K-NN classifier to analyze a data set.
- Environment：OSX MAC、Ubuntu 16.04.3 LTS
- Language：Python 2.7.12
- Library：numpy, pandas
- How does my code work：

1. K-d Tree
   1) Find the median point in the current axis.
   2) Bisect along the hyperplane going through the current median.
   3) Go one step deeper, same thing with next axis.
   4) Until every leaf is created.

```python
def create_kd_tree(points, dim, depth=0):
    if len(points) > 1:
        points.sort(key=lambda x: x[depth])
        depth = (depth + 1) % dim
        half = len(points) / 2
        return (
            create_kd_tree(points[: half], dim, depth),
            create_kd_tree(points[half + 1:], dim, depth),
            points[half])
    elif len(points) == 1:
        return (None, None, points[0])
```

2. KNN
   1) Traverse k-d Tree according to distance of query point and current point.
   2) Push candidate into priority queue until meet number of neighbors.
   3) Go one step deeper, same thing with next axis.
   4) Until all candidates have been test.

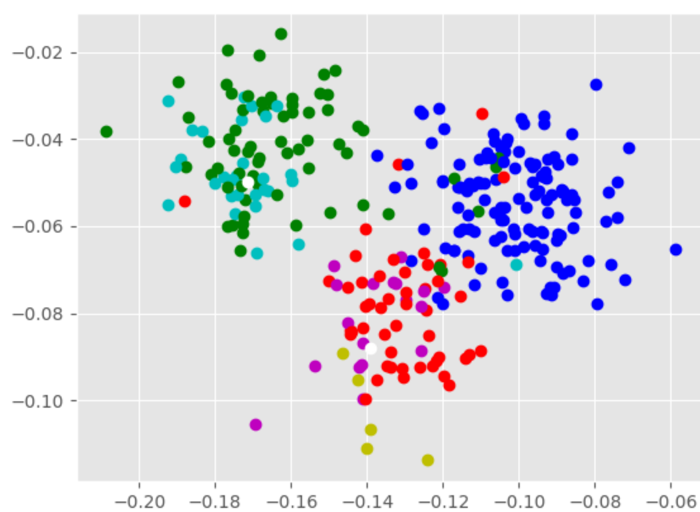
```python
def naive_knn(kd_node, point, k, dim, dist_func, return_distances=True, depth=0, heap=None):
    root_or_not = not heap
    if root_or_not:
        heap = list()
    if kd_node:
        dist = dist_func(point, kd_node[2])
        dx = kd_node[2][depth] - point[depth]
        if len(heap) < k:
            heapq.heappush(heap, (-dist, kd_node[2]))
        elif dist < -heap[0][0]:
            heapq.heappushpop(heap, (-dist, kd_node[2]))
        depth = (depth + 1) % dim
        # traverse all node in k-d tree
        naive_knn(kd_node[0], point, k, dim, dist_func, return_distances, depth, heap)
        naive_knn(kd_node[1], point, k, dim, dist_func, return_distances, depth, heap)
    # After traverse all the candidates, back to root
    if root_or_not:
        neighbors = sorted((-h[0], h[1]) for h in heap)
        return neighbors
```

3. PCA (Bonus)
    1) Compute the dimensional mean vector (i.e., the means for every dimension of the whole dataset) and then subtract it.
    2) Compute the covariance matrix of the whole dataset.
    3) Compute eigenvectors and corresponding eigenvalues.
    4) Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a d×k dimensional matrix W(where every column represents an eigenvector)
    5) Use this d×k eigenvector matrix to transform the samples onto the new subspace.
    6) Create a k-d Tree base on new features.
    7) Apply same KNN algorithm to make prediction.

```
### START PCA ###
dim = 7
features = np.asarray(train_features)
mean = np.mean(features, axis=0)
data_matrix = features.copy()
data_matrix = np.subtract(data_matrix, mean)

covariance_matrix = np.dot(data_matrix.T, data_matrix)
w, v = np.linalg.eigh(covariance_matrix)
projection = np.dot(features, np.array([v[:,-1],v[:,-2],v[:,-3],v[:,-4],
                                        v[:,-5],v[:,-6],v[:,-7]]).T)
```

Visualize the training data set with PCA (dimension from 9 -> 2)



Note that, I find the complete dataset of this homework on UCI's website and find out that only 7 features are used as training features. As a result, I use PCA to reduce dimension from 9 to 7 in order to achieve a better accuracy. Further more, it is obvious that the value of 4^{th} feature is always 0.5 which means that this is a redundant features for each data.