Object Recognition Exercise Sheet 8

Sascha Stelling

June 18, 2018

Task 0 Code (See task8_0.py):

```
from prepare_data import load_data
import matplotlib.pyplot as plt

#Load all datasets into X (data) and Y (labels)
for d in ['toy1', 'toy2', 'mnist']:
        X_train, Y_train = load_data(d, 'train')
        print(X_train.shape, Y_train.shape)

for d in ['toy1', 'toy2', 'mnist']:
        X_test, Y_test = load_data(d, 'test')
        print(X_test.shape, Y_test.shape)

#Plot images from the dataset
for i in range(6, 10):
        plt.imshow(X_train[i])
        plt.title(Y_train[i])
        plt.show()
```

Task 3 Choose the initial centroids randomly within the dataset, stop training if a maximum of iterations is reached or the clusters do not change anymore. Code(see task8_3.py):

```
import numpy as np
from prepare_data import load_data
from matplotlib import pyplot as plt

def distance(item, mean):
        # Euclidean distance
        sum = 0.0
        dim = len(item)
        for j in range(dim):
                sum += (item[j] - mean[j]) ** 2
        return np.sqrt(sum)
```

```
def update_clustering(norm_data, clustering, means):
        # given a set of means, assign new clustering
        # return False if no change or bad clustering
        n = len(norm_data)
        k = len(means)

        new_clustering = np.copy(clustering)
        distances = np.zeros(shape=(k), dtype=np.float32)

        for i in range(n):
                for kk in range(k):
                        distances[kk] = distance(norm_data[i], means[kk]
                new_id = np.argmin(distances)
                new_clustering[i] = new_id

        if np.array_equal(clustering, new_clustering):
                return False

        # make sure that no cluster counts have gone to zero
        counts = np.zeros(shape=(k), dtype=np.int)
        for i in range(n):
                c_id = clustering[i]
                counts[c_id] += 1

        for kk in range(k):
                if counts[kk] == 0:  # bad clustering
                        return False


        for i in range(n):
                clustering[i] = new_clustering[i]
        return True

def update_means(norm_data, clustering, means):
        # given a clustering, compute new means
        # assumes update_clustering has just been called
        # to guarantee no 0-count clusters
        (n, dim) = norm_data.shape
        k = len(means)
        counts = np.zeros(shape=(k), dtype=np.int)
        new_means = np.zeros(shape=means.shape, dtype=np.float32)
        for i in range(n):
                c_id = clustering[i]
                counts[c_id] += 1
                for j in range(dim):
```

```
                        new_means[c_id,j] += norm_data[i,j]
                         # accumulate sum

        for kk in range(k):  # each mean
                for j in range(dim):
                        if(counts[kk] != 0):
                                new_means[kk,j] /= counts[kk]

        for kk in range(k):  # each mean
                for j in range(dim):
                        means[kk,j] = new_means[kk,j]

def initialize(norm_data, k):
        (n, dim) = norm_data.shape
        clustering = np.zeros(shape=(n), dtype=np.int)
         # index = item, val = cluster ID
        for i in range(k):
                clustering[i] = i
        for i in range(k, n):
                clustering[i] = np.random.randint(0, k)

        means = np.zeros(shape=(k,dim), dtype=np.float32)
        update_means(norm_data, clustering, means)
        return(clustering, means)

def cluster(norm_data, k):
        (clustering, means) = initialize(norm_data, k)

        ok = True  # if a change was made and no bad clustering
        max_iter = 100
        sanity_ct = 1
        while sanity_ct <= max_iter:
                ok = update_clustering(norm_data, clustering, means)
                 # use new means
                if ok == False:
                        break
                update_means(norm_data, clustering, means)
                 # use new clustering
                sanity_ct += 1

                return clustering


def main():
        print("\nBegin k-means clustering")
```

```
    X_test, Y_test = load_data('toy1', 'test')

    k = 3
    print("\nClustering data with k=" + str(k))
    clustering = cluster(X_test, k)

    print("\nDone. Clustering:")
    print(clustering)

    fig = plt.figure()

    plt.scatter(X_test[:,0], X_test[:,1], marker='+')
    plt.show()

if __name__ == "__main__":
    main()
```

Datapoints: