HW2Prob5

April 23, 2020

1 Handwritten digits

[43]:
```
import numpy as np
data=np.float64(np.load('MNIST.npy'))
labels=np.float32(np.load('MNIST_labels.npy'))
```

[44]:
```
data = data/255
Ntot,d = data.shape
Ntot,d
```

[44]: (70000, 784)

[45]:
```
np.random.seed(1)
select = np.arange(Ntot)
np.random.shuffle(select)
train, dev, test = np.split(select, [int(.6 * Ntot), int(.8 * Ntot)])
```

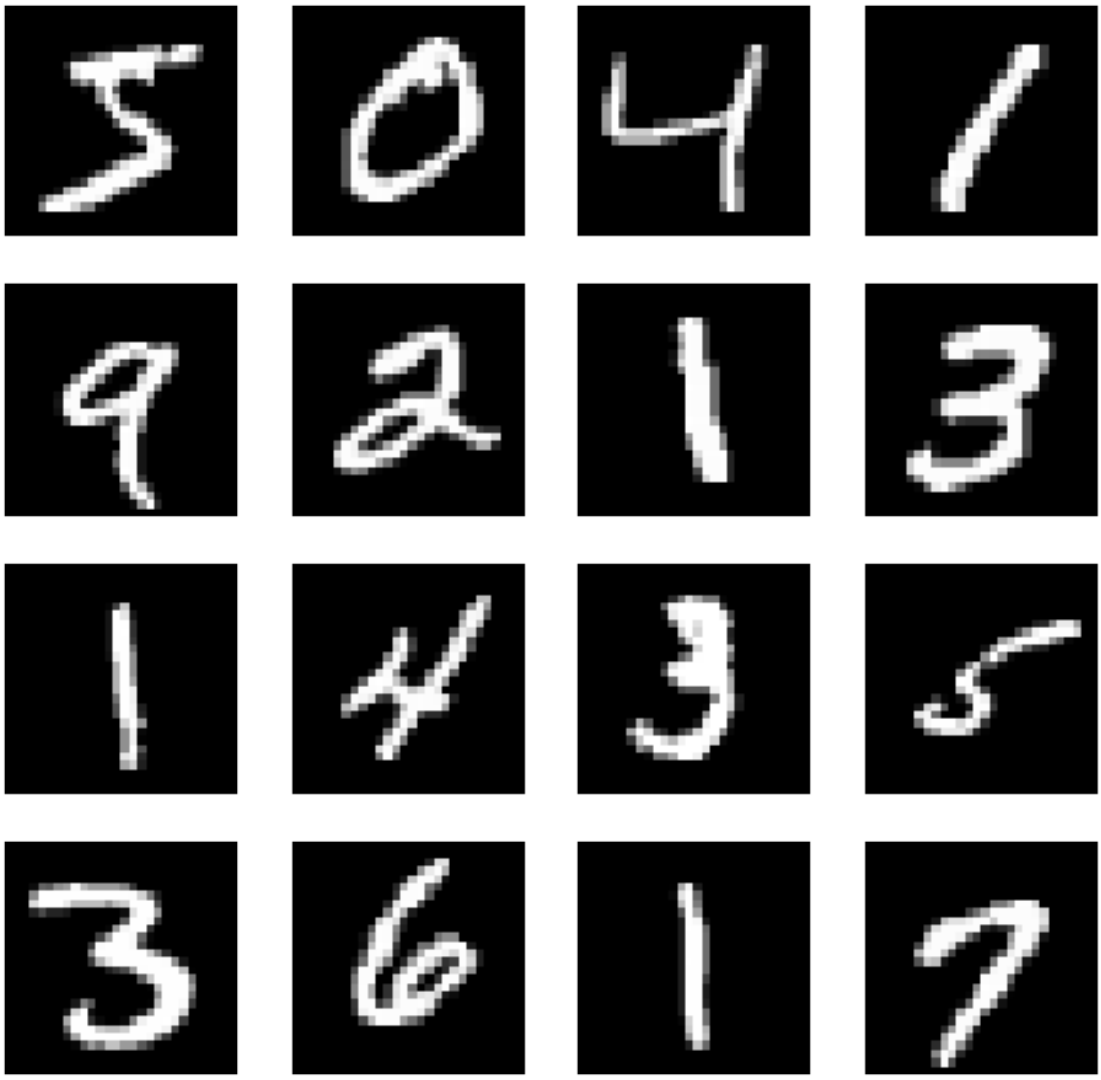
[46]:
```
train.shape, dev.shape, test.shape
```

[46]: ((42000,), (14000,), (14000,))

[47]:
```
%matplotlib inline
import matplotlib.pyplot as plt

def plot_rows(subset, nrows, ncols):
    plt.figure(figsize=(ncols*2, nrows*2))
    for i in range(nrows*ncols):
        plt.subplot(nrows, ncols, i+1)
        plt.imshow(subset[i].reshape((28,28)), cmap='gray')
        plt.axis('off')
    plt.axis('off')
    plt.show()
```

[48]:
```
subset = data[:16,]
plot_rows(subset, 4, 4)
```



1.1 Part 1: PCA

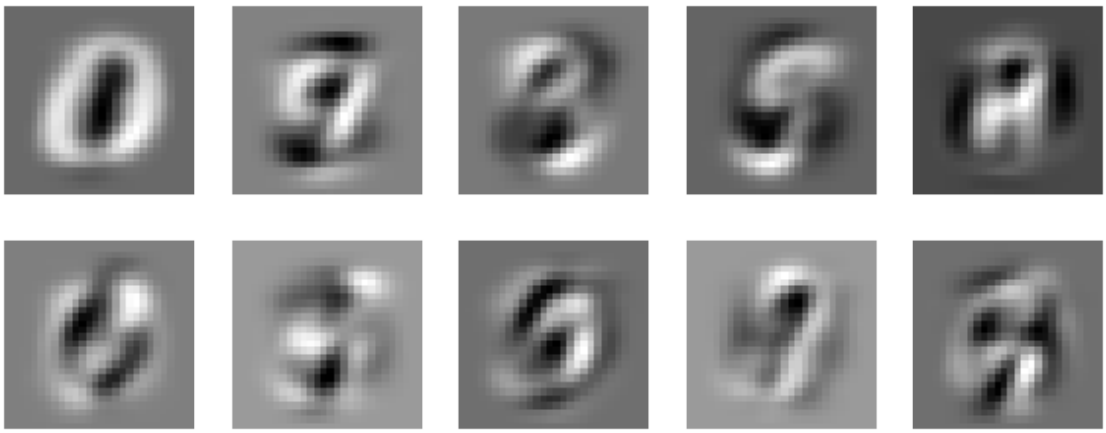
1.1.1 (a) Extract principal components

[49]:
```
trainDt = data[train]
testDt = data[test]
```

[50]:
```
trainMean = np.mean(trainDt, axis=0)
trainDt_c = trainDt-trainMean
u, s, vh = np.linalg.svd(trainDt_c, full_matrices=False)
```

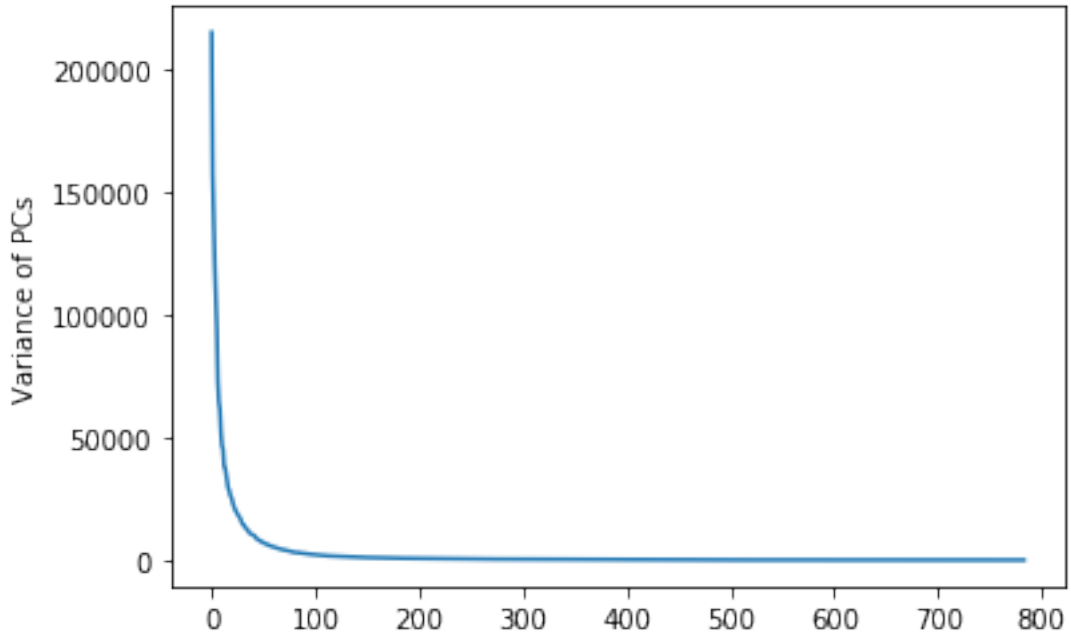
First 10 principal component

[51]:
```
plot_rows(vh[:10], 2, 5)
```



1.1.2 (b) Plot variance

[52]:
```
plt.plot(s**2)
plt.ylabel('Variance of PCs')
plt.show()
```



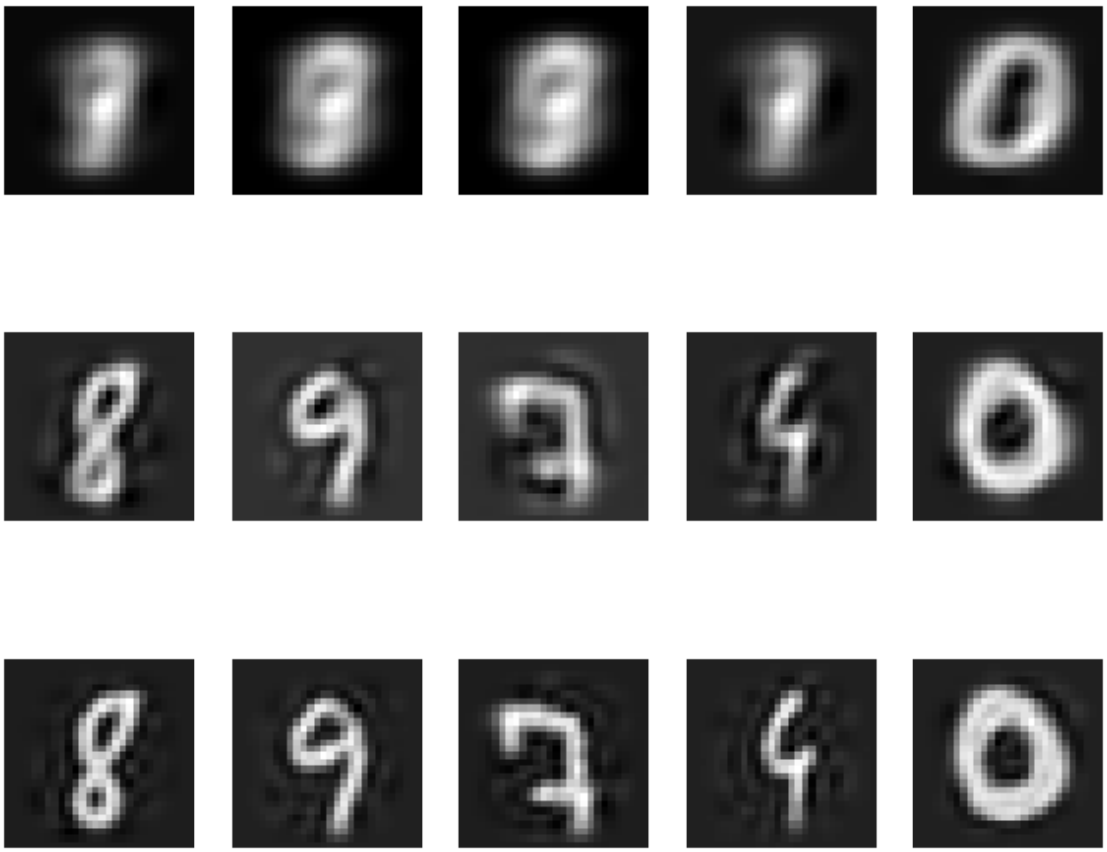
1.1.3 (c) Dimension reduction

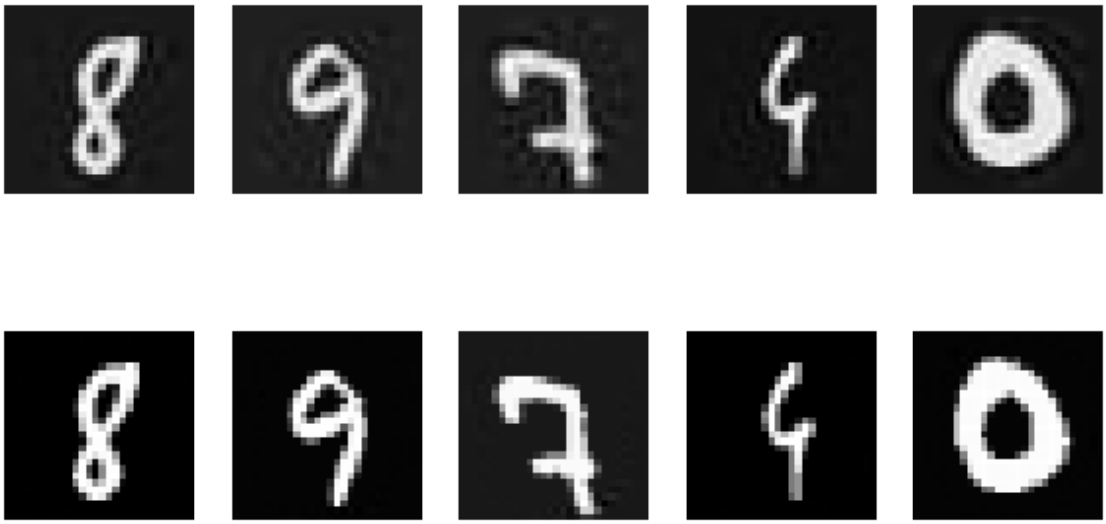
[54]:
```
def calc_proj_rows(dt_rows, m, vh, mean):
    dt_rows_c = dt_rows-mean
    PC_row = vh[:m,]
    coef = np.dot(dt_rows_c, PC_row.T)
    dt_proj_rows = np.dot(coef,PC_row)+mean
    return(dt_proj_rows)
```

Figure below shows projection of 5 data points in the test data set onto the first m PCs, where $m = 1, 50, 100, 200, 500$. Each row is the projection image for an m from small to large.

[62]:
```
ms = [1,50,100,200,500]
dts = [12,46,85,248,394]
#plot_rows([trainMean],1,1)

for m in ms:
    proj_rows = calc_proj_rows(testDt[dts], m, vh, trainMean)
    plot_rows(proj_rows, 1, len(dts))
```





Observe that with m increasing, the image gets more and more accurate. With m equals or larger than 100, the projection gives an adequte image where we can easily recognize the number. The top principal components capture where there is number written and where is the background.

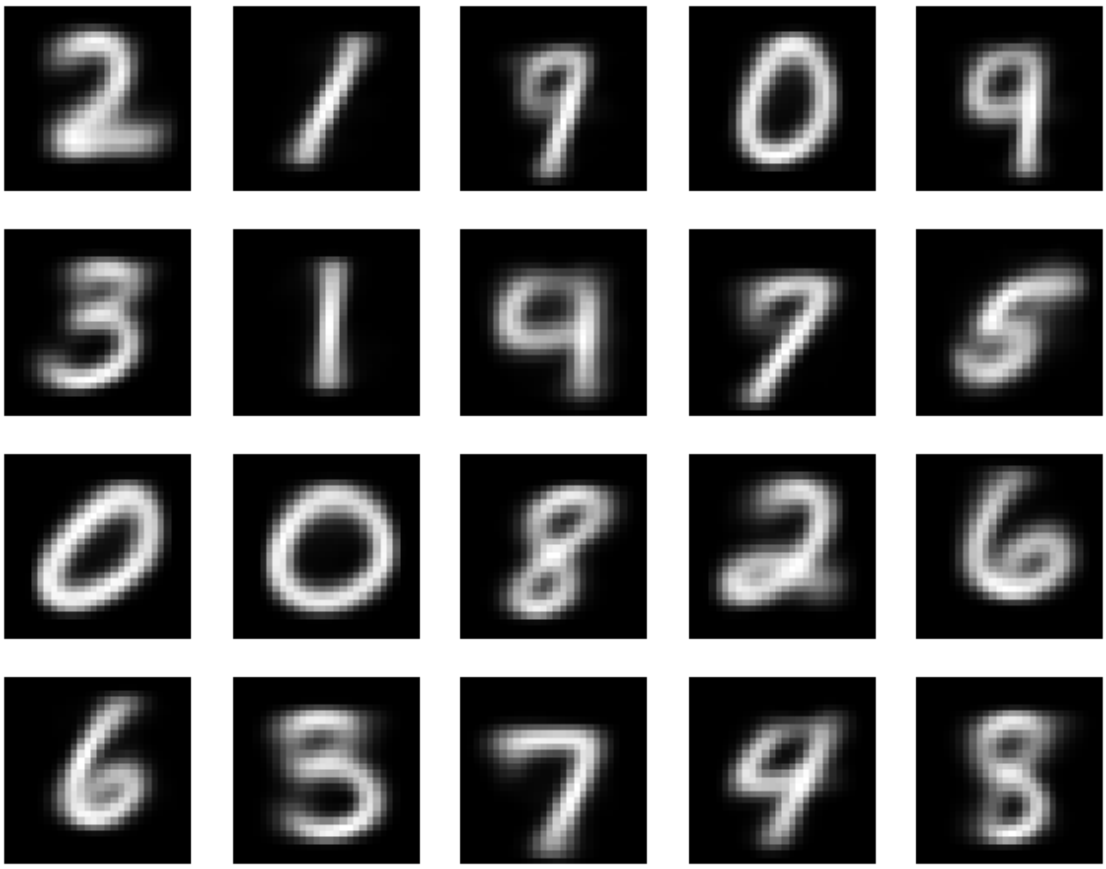
1.2 Part 2: k-means

[63]:
```
from sklearn.cluster import KMeans
```

[64]:
```
n_cluster = 20
kmeans=KMeans(n_clusters=n_cluster,random_state=12343).fit(data)
```

Use k-means to form 20 cluster. The center of each cluster is shown below.

[65]:
```
plot_rows(kmeans.cluster_centers_, 4, 5)
```



[66]:
```
labels_int = labels.astype(int)
```

[67]:
```
y = kmeans.predict(data)
cluster_label = []
for i in range(n_cluster):
    cluster_label.append(np.bincount(labels_int[y==i]))
```

Below is the error rate for each cluster when associatng each cluster with the majority label.

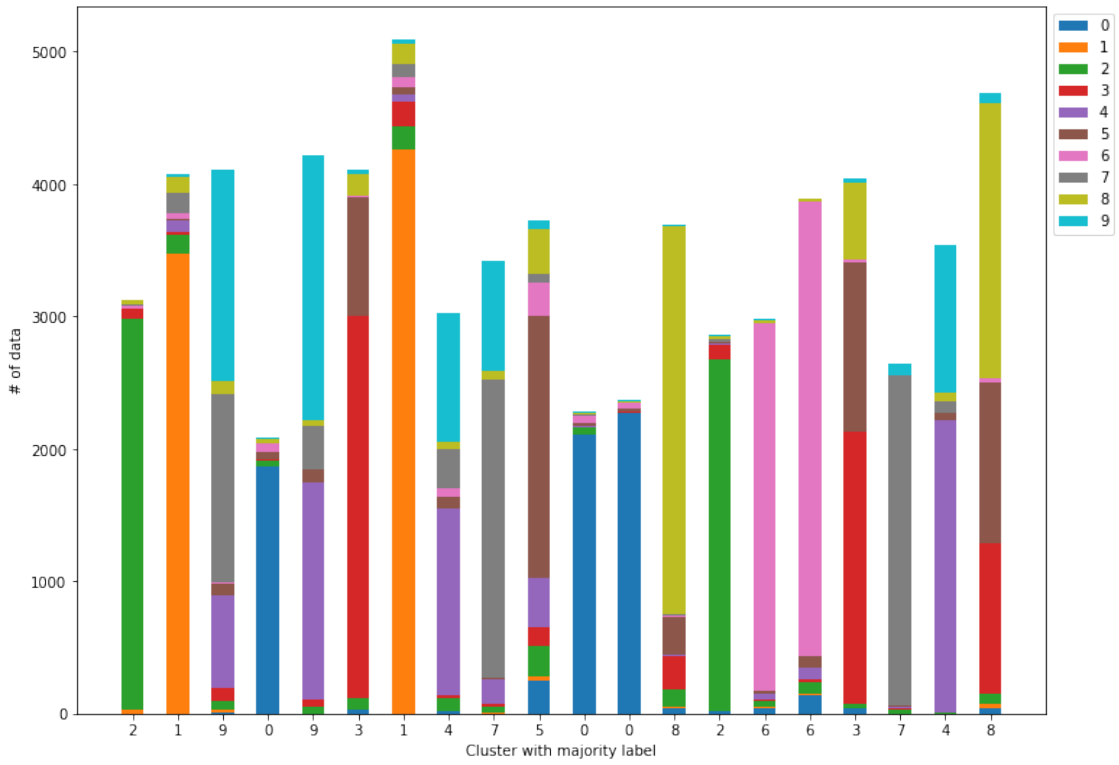
[77]:
```
cluster_label = np.array(cluster_label)
1-np.max(cluster_label, axis=1)/np.sum(cluster_label, axis=1)
```

[77]:
```
array([0.05312   , 0.14958313, 0.61240876, 0.10455635, 0.52464455,
       0.29649464, 0.1631331 , 0.53533686, 0.33966676, 0.46806227,
       0.0777972 , 0.04382638, 0.20844613, 0.07265107, 0.06912752,
       0.11896197, 0.49209095, 0.05526117, 0.37567034, 0.55742955])
```

[78]:
```
def plot_stacked_bar(cluster_label_t):
    n_group = cluster_label_t.shape[1]
    ind = np.arange(n_group)    # the x locations for the groups
```

```
    width = 0.5       # the width of the bars: can also be len(x) sequence
    bottom = np.zeros(n_group)
    plt.figure(figsize=(12, 9))
    for i in range(10):
        plt.bar(ind, cluster_label_t[i], width, bottom = bottom)
        bottom += cluster_label_t[i]
    plt.xlabel('Cluster with majority label')
    plt.ylabel('# of data')
    plt.xticks(range(n_group),np.argmax(cluster_label_t,axis=0))
    plt.legend(loc="upper left", labels = range(10), bbox_to_anchor=(1,1))
    plt.show()
```

[79]:
```
plot_stacked_bar(cluster_label.T)
```

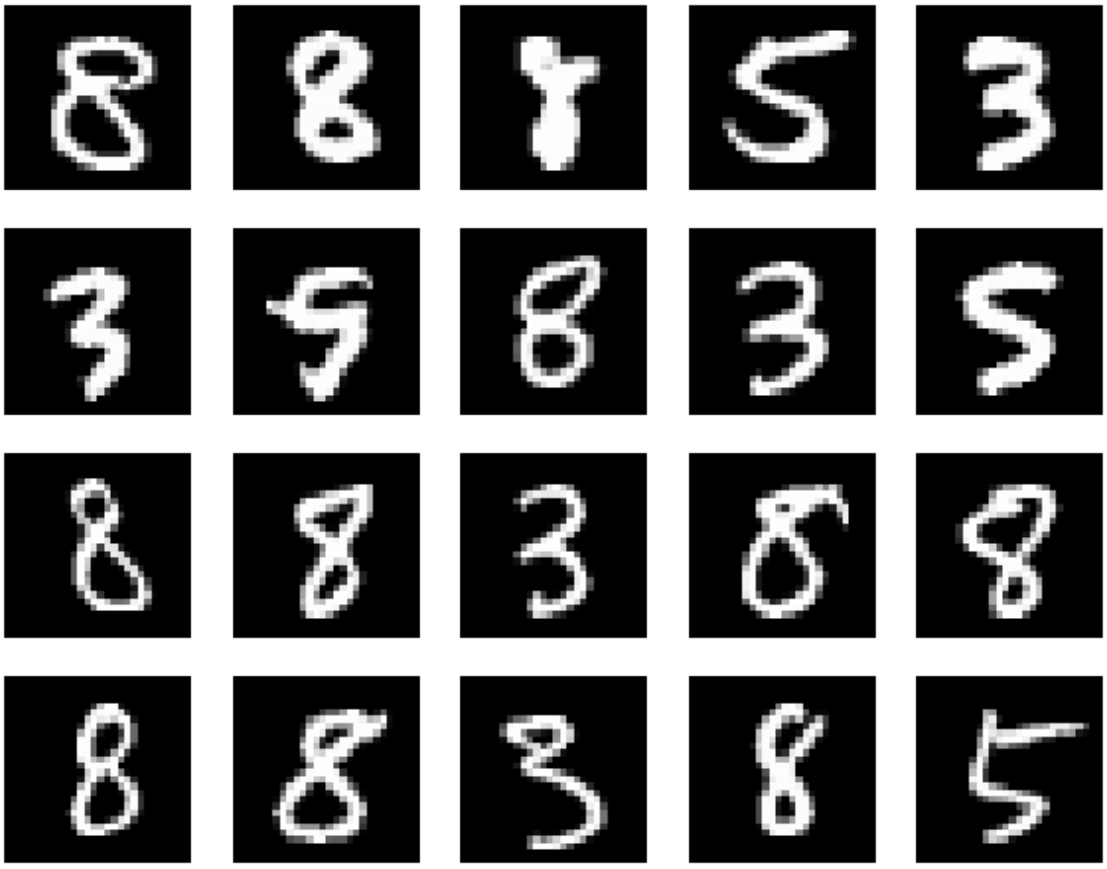


The stacked bar plot shows the number of the true labels within each cluster. The number on x axis shows the majority label of that cluster. We can observe that some cluster are relatively accurate while others are combinations of two or three digits.

20 samples of the digits from the last cluster in the graph above.

[81]:
```
cluster = 19
n_sample = 20
sample = np.random.choice(range(sum(y==cluster)),n_sample)
```

```
plot_rows(data[y==cluster][sample], 4, 5)
```



Observe that data points in a cluster do look somewhat similar visually.

The cluster to a certain degree can effectively distinguish images of digits, but are not accurate enough to sepereate images of different digits that are alike.

1.3 Part 3: Spectral clustering

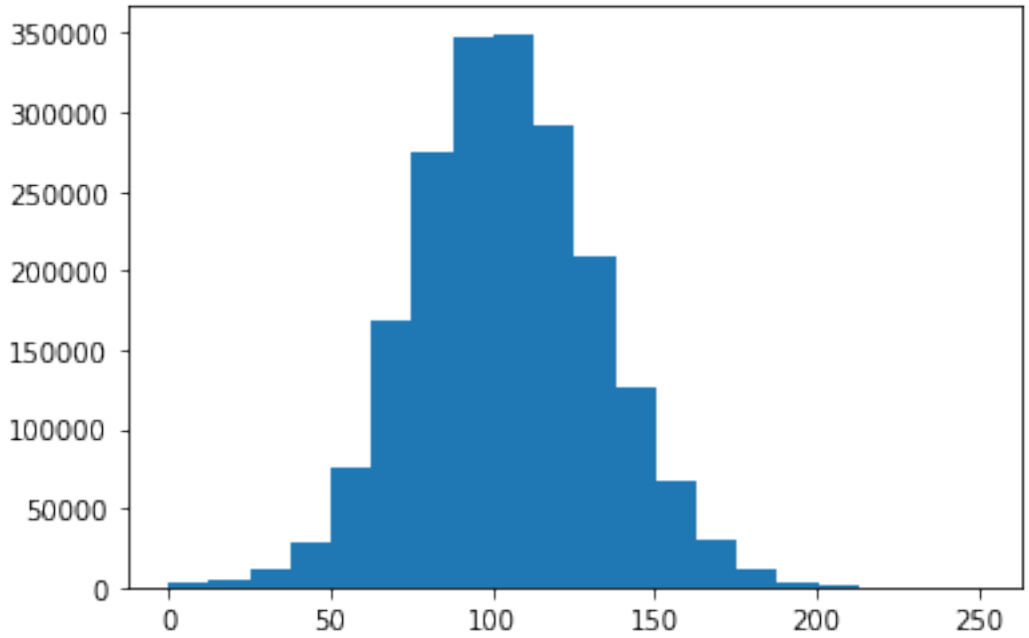
1.3.1 (a) Construct the similarity matrix

[84]:
```
n_spec = 2000
train_spec = range(n_spec)
specDt = data[train_spec,]
#specDt.shape
```

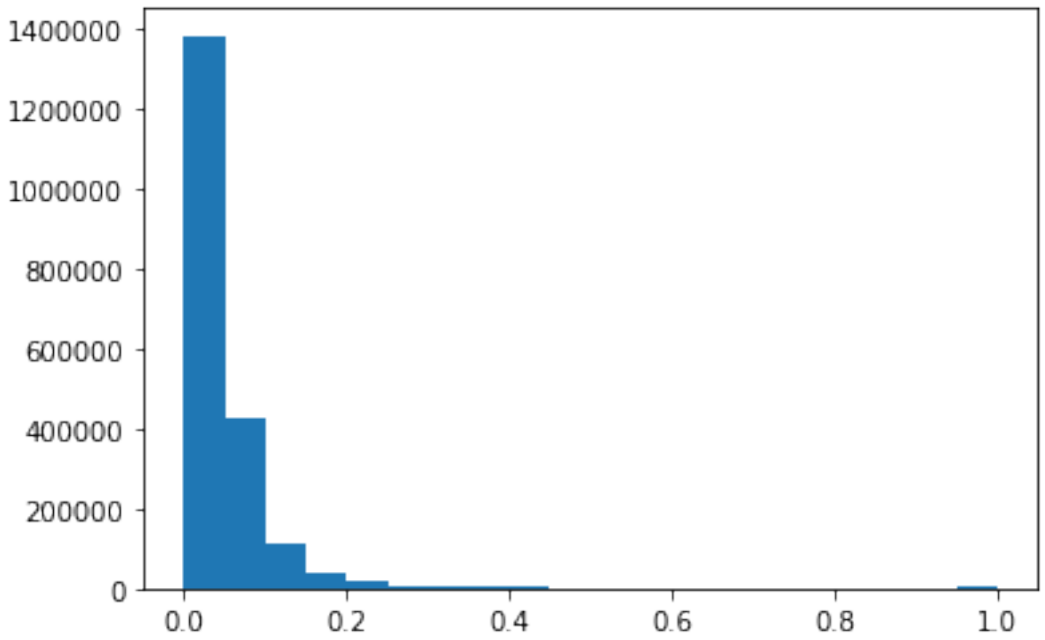
[85]:
```
from scipy.spatial.distance import pdist, squareform
pairwise_dists = squareform(pdist(specDt, 'euclidean'))
#pairwise_dists.shape
dists = pairwise_dists[np.triu_indices(n_spec)]
```

```
#dists.shape
```

[86]:
```
plt.hist(dists**2, bins=20)
plt.show()
```



[87]:
```
bandwidth = 30
expdist = np.exp(-dists**2/bandwidth)
plt.hist(expdist, bins=20)
plt.show()
```



[88]:
```
W = np.exp(-pairwise_dists**2/bandwidth)
#W.shape
inv_sqrt_rowsum = 1/np.sqrt(np.sum(W, axis=1))
D_msqrt = np.diag(inv_sqrt_rowsum)
#D_msqrt.shape
L = np.identity(n_spec)-D_msqrt@W@D_msqrt
L.shape
```

[88]:
```
(2000, 2000)
```

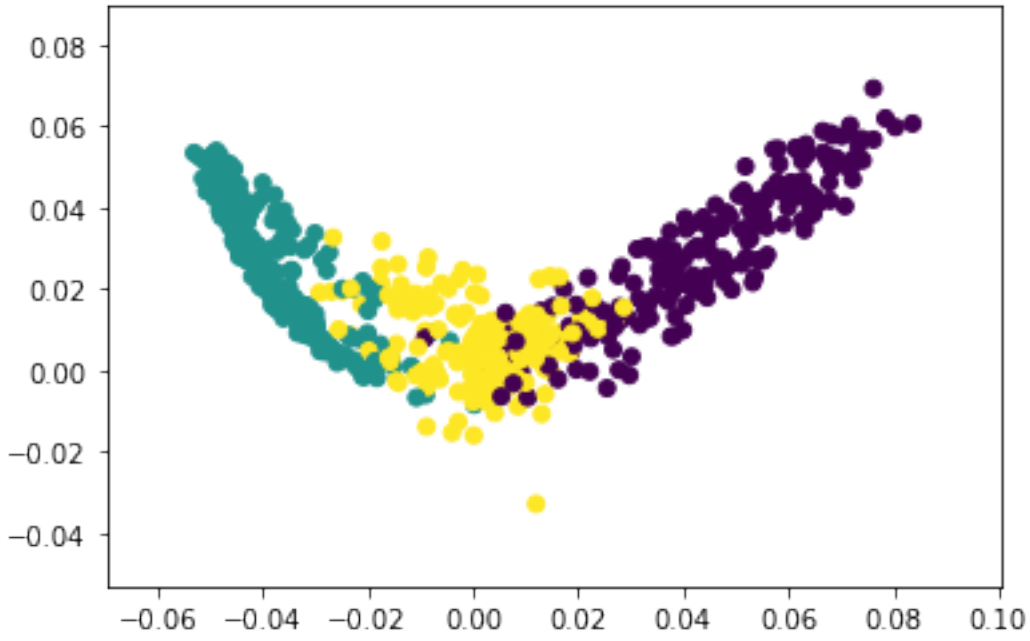
1.3.2 (b) Embed and run K-means

[90]:
```
w, v = np.linalg.eig(L)
#w.shape, v.shape
fr3dig = (labels[train_spec]<3)
#fr3dig.shape
```

[91]:
```
r=2
embed_3dig = v[fr3dig,1:(r+1)]
embed_3dig.shape
```

[91]:
```
(609, 2)
```

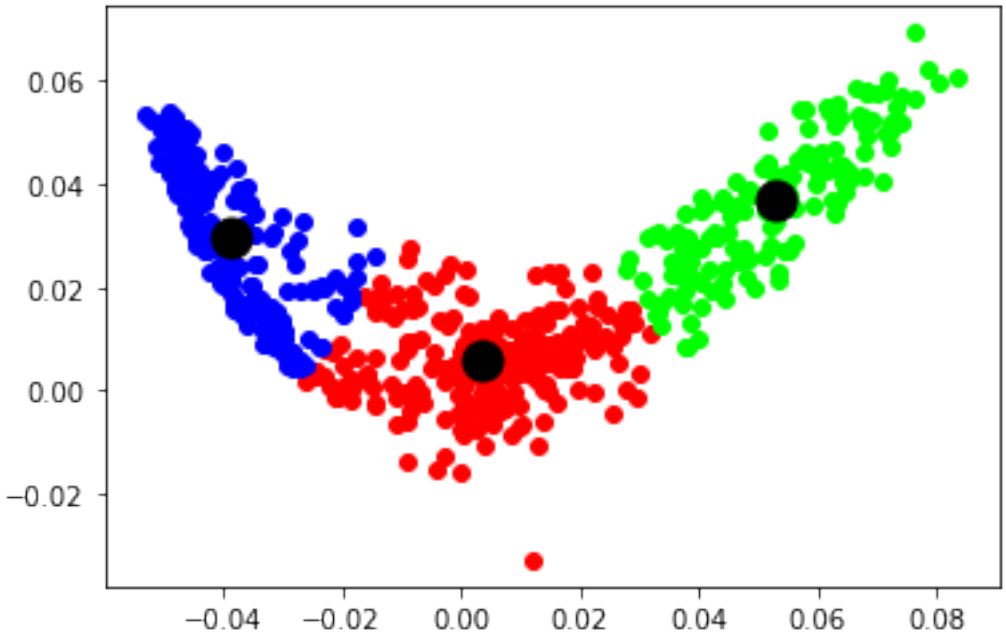
[92]:
```
plt.scatter(embed_3dig[:,0],embed_3dig[:,1], c=labels[train_spec][fr3dig])
plt.show()
```



[93]:
```
spec_cluster = 3
spec_kmeans=KMeans(n_clusters=spec_cluster,random_state=12343).fit(embed_3dig)
```

[94]:
```
def show_clusters(kmeans,X):
    y=kmeans.predict(X)
    fig=plt.figure()
    ax=fig.add_subplot(1,1,1)
    for i in set(y):
        ec=[0.,0.,0.,1.]
        ec[i]=1.
        ax.plot(X[y==i,0],X[y==i,1],'o',color=ec)
    ax.plot(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:
↪,1],'o',color='black',markersize=15)
```

[95]:
```
show_clusters(spec_kmeans, embed_3dig)
```



1.3.3 (c) Embed new data

[96]:
```
def embed_new(x, specDt, ws, vs, inv_sqrt_rowsum):
    norm = np.linalg.norm(specDt-x, axis=1)
    K = np.exp(-norm**2/bandwidth)
    tld_K_xX = K/np.sqrt(sum(K))*inv_sqrt_rowsum
    embed = np.sum(tld_K_xX*vs.T, axis=1)*(1/(1-ws))
    # w is eigenvalue of L, lambda = 1-w, lambda is eigenvalue of W
    return(embed)
```

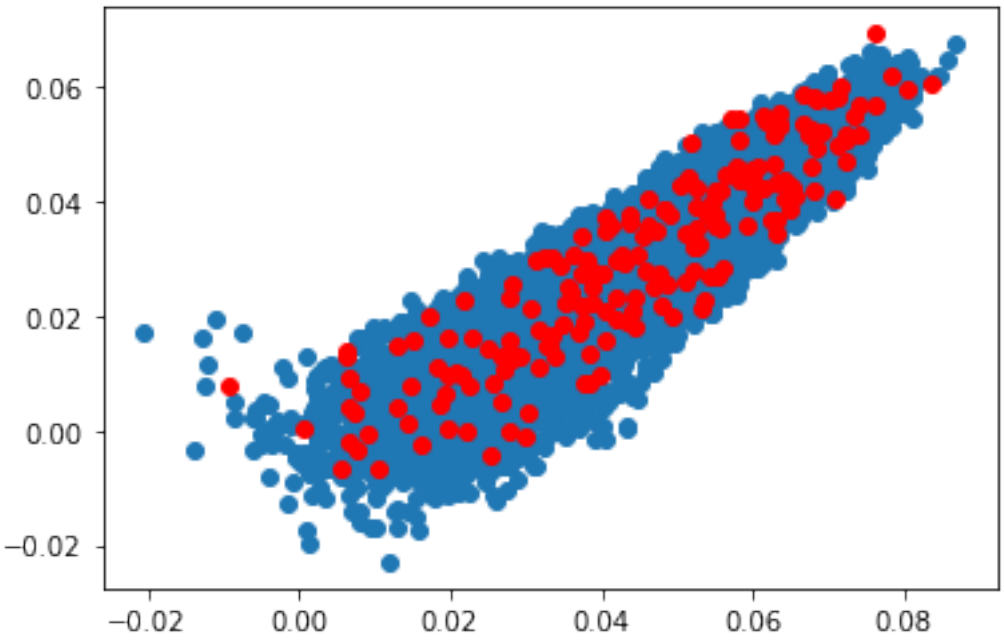
[97]:
```
zeros = (labels[n_spec:]==0)
zeroDt = data[n_spec:][zeros]
ws = w[1:r+1]
vs = v[:,1:r+1]
```

[98]:
```
embed_zero = np.apply_along_axis(embed_new, 1, zeroDt,specDt=specDt, ws=ws,␣
↪vs=vs, inv_sqrt_rowsum=inv_sqrt_rowsum)
```

[99]:
```
train_zero = (labels[train_spec]==0)
embed_train0 = v[train_zero,1:(r+1)]
```

In the figure below, the red points are zeros in the training set for spectral clustering, the blue points are zeros not in the training set.

[100]:
```
plt.plot(embed_zero[:,0], embed_zero[:,1], marker = "o", ls="")
plt.plot(embed_train0[:,0], embed_train0[:,1], marker = "o", ls="", c='red')
plt.show()
```



Notice that all embeds are in the same region and are close to each other.

1.4 Part 4: Classiffcation

[101]:
```
from sklearn.linear_model import LogisticRegression
```

[103]:
```
trainLb = labels_int[train]
#trainDt.shape, trainLb.shape
lg=LogisticRegression(fit_intercept=True, C=100000, penalty='l2',␣
 ↪multi_class='multinomial', solver='lbfgs')
```

[104]:
```
import warnings
warnings.filterwarnings("ignore")
```

[106]:
```
lg1 = lg.fit(trainDt, trainLb)
```

[107]:
```
devDt = data[dev]
devLb = labels_int[dev]
#devDt.shape, devLb.shape
lg1_pred_dev = lg1.predict(devDt)
#lg1_pred_dev.shape
```

[111]:
```
print("The error rate for multinomial logistic regression is",round(np.
 ↪mean(lg1_pred_dev!=devLb),5),".")
```

```
The error rate for multinomial logistic regression is 0.07993 .
```

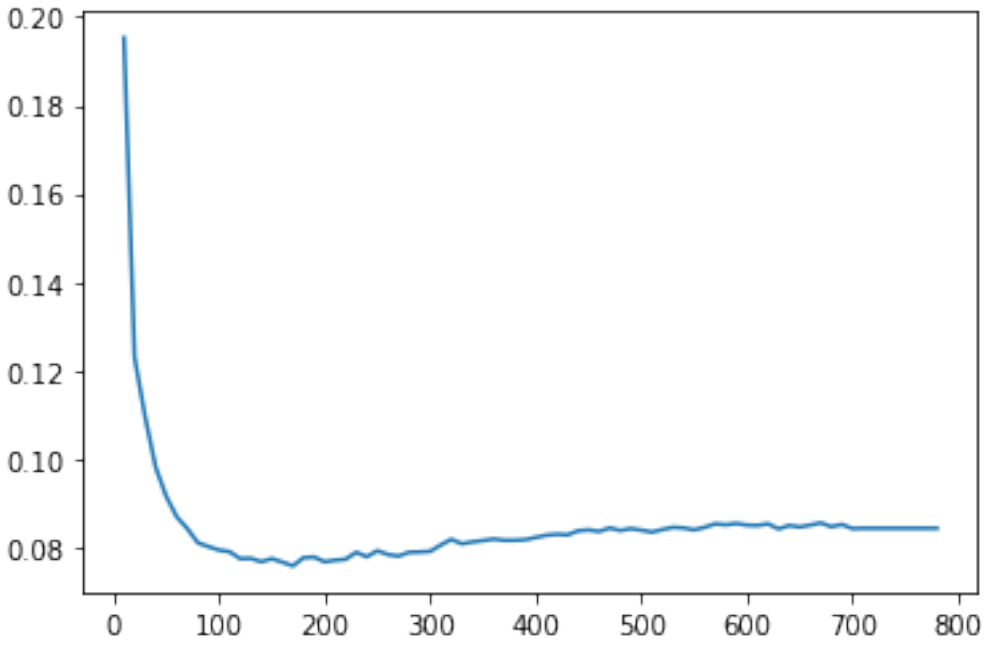
[112]:
```
PC_train = np.dot(trainDt-trainMean, vh.T)
#PC_train.shape
lg_pc = lg.fit(PC_train, trainLb)
PC_dev = np.dot(devDt-trainMean, vh.T)
lg_pc_pred_dev= lg_pc.predict(PC_dev)
print("The error rate for multinomial logistic regression with all PCs␣
 ↪is",round(np.mean(lg_pc_pred_dev!=devLb),5),".")
```

```
The error rate for multinomial logistic regression with all PCs is 0.0845 .
```

[114]:
```
# k = 100
# lg_pck = lg.fit(PC_train[:,:k], trainLb)
# lg_pck_pred_dev = lg_pck.predict(PC_dev[:,:k])
# np.mean(lg_pck_pred_dev!=devLb)
```

[115]:
```
ks = range(10,784,10)
err_rate = []
for k in ks:
    lg_pck = lg.fit(PC_train[:,:k], trainLb)
    lg_pck_pred_dev = lg_pck.predict(PC_dev[:,:k])
    err_rate.append(np.mean(lg_pck_pred_dev!=devLb))
```

[116]:
```
plt.plot(ks, err_rate)
plt.show()
```



As k increases, the error rate first decreases and then slightly increases.

[120]:
```
k_min = ks[np.argmin(err_rate)]
print("Achieve lowest error rate", np.min(err_rate), "at k =",k_min,".")
```

```
Achieve lowest error rate 0.076 at k = 170 .
```

Retrain the model using both training and development data.

[121]:
```
PC_td = np.concatenate((PC_train, PC_dev), axis=0)
LB_td= np.concatenate((trainLb, devLb), axis=0)
lg_pc2 = lg.fit(PC_td[:,:k_min], LB_td)
testLb = labels[test]
PC_test = np.dot(testDt-trainMean, vh.T)
#PC_test.shape
```

[123]:
```
lg_pc2_pred_test = lg_pc2.predict(PC_test[:,:k_min])
print("The error rate for first 170 PCs is",
      round(np.mean(lg_pc2_pred_test!=testLb),5),".")
```

```
The error rate for first 170 PCs is 0.07886 .
```

[124]:
```
lg2 = lg.fit(np.concatenate((trainDt, devDt)), np.concatenate((trainLb, devLb)))
lg2_pred_test = lg2.predict(testDt)
print("The error rate for logistic regression on raw features is",
      round(np.mean(lg2_pred_test!=testLb),5),".")
```

```
The error rate for logistic regression on raw features is 0.078 .
```

There are not significant diference on error rate between multinomial logistic regression on raw features and selected top PCs. This can show that top PCs reduce the dimension without lossing much information.