In [1]:

```
# Do the imports, so that we write less code
import scipy.io
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from time import time
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
from PIL import Image
```

In [2]:

```
# Prepare and load the data for analysis
mat = scipy.io.loadmat('2013MT60597.mat')
raw_data = mat['data_image']
target = mat['data_labels']

data = scale(raw_data)
true_labels = target.flatten()
n_samples, n_features = data.shape
n_digits = len(np.unique(target))
# print(n_digits)
```

```
/home/hd/anaconda2/lib/python2.7/site-packages/sklearn/utils/v
alidation.py:420: DataConversionWarning: Data with input dtype
int64 was converted to float64 by the scale function.
  warnings.warn(msg, DataConversionWarning)
```

In [38]:

```
# # run K-Means clustering on
# model = KMeans(init='k-means++', n_clusters=n_digits, n_init=10);
# model.fit_predict(data)
model_raw_data = KMeans(init='k-means++', n_clusters=10, n_init=10)
model_raw_data.fit_predict(raw_data)
```

Out[38]:

```
array([3, 3, 3, ..., 6, 2, 5], dtype=int32)
```

In [46]:

```
def getPredictions(labels):
    # store the cluster indexes assigned by the model in an array
    result = labels
    # build matrix whose ith row corresponds to ith cluster_index
    # matrix[i][j] denotes no. of samples of true_label j assigned to clus
ter i
    matrix = np.zeros((5,10))
    for i in range(1,n_samples):
        matrix[result[i]][true_labels[i]] += 1
    # take argmax of each row to assign the label to that cluster
    cluster_label = np.zeros(5)
    for i in range(0,5):
        cluster_label[i] = np.argmax(matrix[i,:])
    # after assigning the cluster labels, get the label predicted by the m
odel using the cluster index assigned
    predicted = np.zeros(n_samples)
    for i in range(1,n_samples):
        predicted[i] = cluster_label[result[i]]
    return predicted, cluster_label
```

In [11]:

```
# get the no. of misclassified samples
def getAccuray(true_labels,predicted_labels,samples):
    accuracy = 0.0;
    for i in range(1,samples):
        if(true_labels[i] == predicted_labels[i]):
            accuracy +=1
    return accuracy/samples
def getImage(x):
    img =  Image.fromarray(255 - (x.reshape(28, 28)).astype('uint8'))
#     plt.imshow(img,cmap='Greys_r')
#     plt.show()
    return img
```

In [39]:

```
# predicted_scaled_data, cluster_label_scaled_data = getPredictions(model.
labels_)
predicted_raw_data, cluster_label_raw_data = getPredictions(model_raw_dat
a.labels_)
getAccuray(true_labels,predicted_raw_data,n_samples)
```

Out[39]:

0.5795

In [31]:

```
for i in range(0,5):
    name = cluster_label_raw_data[i]
    name = str(name)+ str(i)  + "_.png"
    img = getImage(model_raw_data.cluster_centers_[i])
    img.save(name)
```

Observations on raw data

K = 10

- The accuracy achieved is 57%. Though it fluctuates a little bit every time due to the random initilizations.
- The cluster centers are saved in a separate folder. Analysis reveals that 0 and 9 are assigned the same label. This might be due to the similarity in their bitmap representations.
- Also we see that 4 and 5 have the same cluster center. So our model cannot distinguish between images of 4 and 5 correctly.
- Also , we see that the cluster centers of 4 and 7 have a bit of resemblance to 9 in them. This might be because of the similarity in their structure.
- We see that the class 0 has two cluster centers. My interpretation is that in reality 0 has a single cluster center, but due to highly similar resemblance to 9, it has occupied its cluster center.

K = 5

- The accuracy achieved is 43%, which varies a little bit every time.
- The cluster centers are saved in a separate folder. Analysis reveals that the cluster center of 0 is exceptionaly well captured, but all the other cluster centers are mixed and highly overlap with different digits.
- Observations reveals that cluster centers of 1 ,5 and 7 are combined. Similarly centers of 3, 8 and 9,7 are combined
- So, in this case, our cluster centers do not make much sense.

In [ ]:

```
```

In [ ]:

```
U, s, V = np.linalg.svd(data)


total_variance = sum(s)
total_variance
```

In [ ]:

```
```

In [25]:

```
pca = PCA(n_components=0.9)
pca.fit(raw_data)
```

Out[25]:

```
PCA(copy=True, n_components=0.9, whiten=False)
```

In [26]:

```
transformed_data = pca.transform(raw_data)
sum(pca.explained_variance_ratio_)
```

Out[26]:

```
0.90021308635218056
```

In [50]:

```
def getResidualVariance():
    variance_residual = np.zeros(pca.n_components_)
    var_explained = pca.explained_variance_ratio_
    temp = 0;
    for i in range(0,pca.n_components_):
        temp += var_explained[i]
        variance_residual[i] = 1 -temp
    return variance_residual


pca.n_components_
```

Out[50]:

```
82
```

In [29]:

```
residual_variance = getResidualVariance()
X = np.linspace(0, pca.n_components_,pca.n_components_)
plt.xlabel('No. of Principal Components Taken')
plt.ylabel('Residual Variance')
plt.title('Residual Variance Plot')
# plt.xticks(np.linspace(0, pca.n_components_,1), X)
plt.plot(X, residual_variance,'ro')
plt.grid(True)
plt.show()
```

In [37]:

```
model_transformed_data = KMeans(init='k-means++', n_clusters=10, n_init=1
0)
model_transformed_data.fit_predict(transformed_data)
```

Out[37]:

```
array([2, 9, 2, ..., 8, 1, 7], dtype=int32)
```

In [40]:

```
predicted_tranformed_data, cluster_label_transformed_data = getPredictions
(model_transformed_data.labels_)
getAccuray(true_labels,predicted_tranformed_data,n_samples)
```

Out[40]:

```
0.5735
```

In [42]:

```
original_data = pca.inverse_transform(transformed_data)
```

In [47]:

```
model = KMeans(init='k-means++', n_clusters=5, n_init=10)
model.fit_predict(original_data)
predicted_data, cluster_label_data = getPredictions(model.labels_)
getAccuray(true_labels,predicted_data,n_samples)
```

Out[47]:

```
0.441
```

In [49]:

```
for i in range(0,5):
    name = cluster_label_data[i]
    name = str(name)+ str(i)  + "_.png"
    img = getImage(model.cluster_centers_[i])
    img.save(name)
```

Observations for reduced data

K = 10

- The accuracy achieved is 57.9%, which is only 0.9% more than the previous case. So we can say that running PCA doesnot offer much advantage in this scenario.
- We project the data back to its original space to see that cluster centers we obtain of run the k means on this space.
- We see that the cluster centers of 4 and 7 have a bit of resemblance to 9 in them. This might be because of the similarity in their structure.
- We see that the class 7 has two cluster centers. My interpretation is that in reality 7 has a single cluster center, but due to highly similar resemblance to 9, it has occupied its cluster center, and both the cluster centers have high resemblance to both the digits.

K = 5

- The accuracy achieved is 43%, which varies a little bit every time.
- The cluster centers are saved in a separate folder. Analysis reveals that the cluster center of 0 is exceptionaly well captured, but all the other cluster centers are mixed and highly overlap with different digits.
- The cluster center labelled 1 has a high resemblance of 5 in it
- Similarly cluster center labelled 3 has a high resemblance of 8 in it, same for the case of 9, which resembles 7 to a high extent

We see that using PCA doesnot offer much advantage. This might be due to the fact the for capturing 90% of the variance, we need 82 dimensions . Now 82 dimensions in itself is a very high dimensional space and the curse of dimensionality is playing its part there too. So we donot get a significant advantage.