1 Clustering

In [1]:
```
# Install additional libraries
#!pip install fastcluster
```

2 Import Libraries ¶

In [2]:
```
# Import libraries
'''Main'''
import numpy as np
import pandas as pd
import os, time
import pickle, gzip

'''Data Viz'''
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
color = sns.color_palette()
%matplotlib inline

'''Data Prep and Model Evaluation'''
from sklearn import preprocessing as pp
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve, average_precision_score
from sklearn.metrics import roc_curve, auc, roc_auc_score

'''Algorithms'''
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import fastcluster
from scipy.cluster.hierarchy import dendrogram, cophenet, fcluster
from scipy.spatial.distance import pdist
```

3 Load Data

In [3]:
```
# Load the datasets
current_path = os.getcwd()
file = os.path.sep.join(['', 'datasets', 'mnist_data', 'mnist.pkl.gz'])

f = gzip.open(current_path+file, 'rb')
train_set, validation_set, test_set = pickle.load(f, encoding='latin1')
f.close()

X_train, y_train = train_set[0], train_set[1]
X_validation, y_validation = validation_set[0], validation_set[1]
X_test, y_test = test_set[0], test_set[1]
```

In [4]:
```
# Create Pandas DataFrames from the datasets
train_index = range(0,len(X_train))
validation_index = range(len(X_train), \
                         len(X_train)+len(X_validation))
test_index = range(len(X_train)+len(X_validation), \
                   len(X_train)+len(X_validation)+len(X_test))

X_train = pd.DataFrame(data=X_train,index=train_index)
y_train = pd.Series(data=y_train,index=train_index)

X_validation = pd.DataFrame(data=X_validation,index=validation_index)
y_validation = pd.Series(data=y_validation,index=validation_index)

X_test = pd.DataFrame(data=X_test,index=test_index)
y_test = pd.Series(data=y_test,index=test_index)

print('train:',X_train.shape)
print('validation:',X_validation.shape)
print('test:',X_test.shape)
```

```
train: (50000, 784)
validation: (10000, 784)
test: (10000, 784)
```

In [5]:
```
np.bincount(y_train)
```

Out[5]:
```
array([4932, 5678, 4968, 5101, 4859, 4506, 4951, 5175, 4842, 4988],
      dtype=int64)
```

In [6]:
```
print(sum(np.bincount(y_train)))
```

```
50000
```

4 Dimensionality Reduction

In [7]:
```python
# Principal Component Analysis
from sklearn.decomposition import PCA

n_components = 784

pca = PCA(n_components=n_components)

X_train_PCA = pca.fit_transform(X_train)
X_train_PCA = pd.DataFrame(data=X_train_PCA, index=train_index)
X_train_PCA.head()
```

Out[7]:

	0	1	2	3	4	5	6	7	
0	0.461484	-1.246856	0.046275	-2.151944	-0.247284	-0.925426	0.889320	0.507180	-1.54
1	3.921805	-1.252039	2.335258	-1.340842	-3.421519	-0.725720	-0.206354	-0.345258	0.13
2	-0.203586	1.547886	-0.980333	2.039061	-1.079858	0.112944	-3.312401	1.403132	-0.59
3	-3.148331	-2.296074	1.091138	0.484675	0.066836	2.778989	-1.834335	-0.174753	1.16
4	-1.442694	2.872064	0.175669	-0.976944	0.302754	0.120619	-0.376697	-1.478133	1.00

5 rows × 784 columns

In [8]:
```python
#  # Log data
# cwd = os.getcwd()
# log_dir = cwd+"/logs/05_clustering/"
# y_train[0:2000].to_csv(log_dir+'labels.tsv', sep = '\t', index=False, head
```

In [9]:
```python
# # Write dimensions to CSV
# X_train_PCA.iloc[0:2000,0:3].to_csv(log_dir+'pca_data.tsv', sep = '\t', in
```

5 K-means

5.1 Inertia

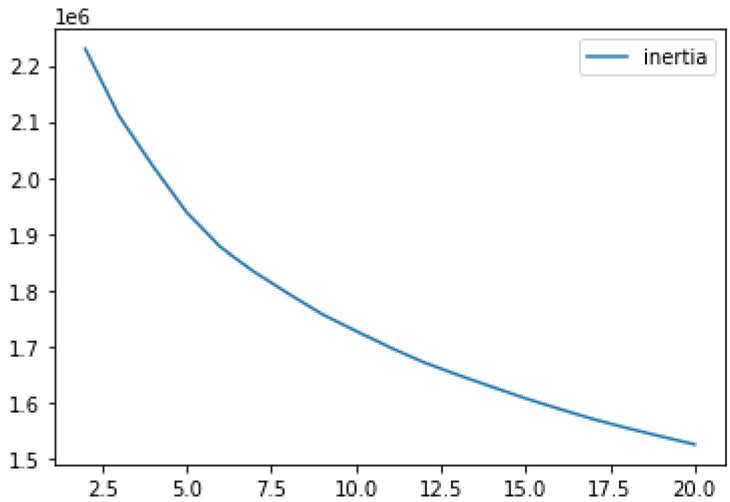
- MNIST只取前cutoff(=99)個的PCA特徵,進行k-means 並將得到的inertia存於DataFrame
- Cluster數量從2到20個, 觀察k-means inertia

In [66]:
```python
# K-means - Inertia as the number of clusters varies
from sklearn.cluster import KMeans


n_init = 10
max_iter = 300
tol = 0.0001
random_state = 2022

kMeans_inertia = pd.DataFrame(data=[],index=range(2,21), \
                              columns=['inertia'])
```

In [67]:
```python
for n_clusters in range(2,21):
    kmeans = KMeans(n_clusters=n_clusters, n_init=n_init, \
                max_iter=max_iter, tol=tol, random_state=random_state)

    cutoff = 99
    kmeans.fit(X_train_PCA.loc[:,0:cutoff])
    kMeans_inertia.loc[n_clusters] = kmeans.inertia_
```

In [68]:
```python
# Plot inertia relative to k # of clusters
kMeans_inertia.plot()
```

Out[68]: <Axes: >



5.2 Accuracy

5.2.1 如何設計分群好壞的評分指標

單群精確度: 每一群內的同質性愈高, 表示分群的愈好. 同質性=該群內類別出現最多次的數量/該群的總數 若值為1, 表示該群內為均為同一種類別, 若為0.5 表示只有最多有一半是相同類別.

整體的精確度=sum(各群內類別出現最多次的數量)/全部資料數

群的數量是為k (k=2~20)

- countByCluster:記錄了預測k群, 各群的數量分佈
- countMostFreq : 記錄預測的每一群中, 出現最多次數字的數量
- accuracyDF :為countMostFreq和countByCluster的合併

- countByLabel 記錄了Ground Truth在各群(即數字0~9)的數量分佈

countByCluster

	cluster	clusterCount
0	17	3667
1	8	3449
2	12	3006
3	10	2976
4	2	2915
5	9	2864
6	14	2826
7	4	2795
8	3	2658
9	16	2639
10	0	2560
11	11	2438
12	15	2255

countMostFreq

	cluster	countMostFrequent
0	0	1631
1	1	2000
2	2	2473
3	3	1387
4	4	1997
5	5	1745
6	6	1295
7	7	1839
8	8	1398
9	9	1435
10	10	1137
11	11	1433
12	12	1392

accuracyDF (merge)

	cluster	countMostFrequent	clusterCount
0	0	1631	2560
1	1	2000	2153
2	2	2473	2915
3	3	1387	2658
4	4	1997	2795
5	5	1745	1835
6	6	1295	1452
7	7	1839	2001
8	8	1398	3449
9	9	1435	2864
10	10	1137	2976
11	11	1433	2438
12	12	1392	3006

各群的正確率

$$\frac{countMostFreq}{clusterCount}$$

整體分群的正確率

$$\frac{sum(countMostFreq)}{sum(clusterCount)}$$

Preds

	trueLabel	cluster
0	5	4
1	0	19
2	4	18
3	1	2
4	9	10
...	...	...
49995	5	9
49996	0	13
49997	8	8
49998	4	12
49999	8	8

50000 rows × 2 columns

```
countMostFreq = \
    pd.DataFrame(data=preds.groupby('cluster').agg( \
                lambda x:x.value_counts().iloc[0]))
countMostFreq.reset_index(inplace=True,drop=False)
countMostFreq.columns = ['cluster','countMostFrequent']
```

```
In [13]: # Define analyze cluster function
def analyzeCluster(clusterDF, labelsDF):
    countByCluster = pd.DataFrame(data=clusterDF['cluster'].value_counts())
    countByCluster.reset_index(inplace=True,drop=False) #drop=False : the c
    countByCluster.columns = ['cluster','clusterCount']

    preds = pd.concat([labelsDF,clusterDF], axis=1) #merge  horizontally
    preds.columns = ['trueLabel','cluster']


    countMostFreq = \
        pd.DataFrame(data=preds.groupby('cluster').agg( \
                        lambda x:x.value_counts().iloc[0]))
    countMostFreq.reset_index(inplace=True,drop=False)
    countMostFreq.columns = ['cluster','countMostFrequent']

    accuracyDF = countMostFreq.merge(countByCluster, \
                        left_on="cluster",right_on="cluster")
    overallAccuracy = accuracyDF.countMostFrequent.sum()/ \
                        accuracyDF.clusterCount.sum()

    accuracyByLabel = accuracyDF.countMostFrequent/ \
                        accuracyDF.clusterCount

    countByLabel = pd.DataFrame(data=preds.groupby('trueLabel').count())

    return countByCluster, countByLabel, countMostFreq, \
            accuracyDF, overallAccuracy, accuracyByLabel
```

5.2.2 Review: Pandas functoin

DataFrame.agg(func=None, axis=0, *args,* *kwargs)
Aggregate using one or more operations over the specified axis.

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.agg.html
(https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.agg.html)

In [71]:
```
XClustered = kmeans.predict(X_train_PCA.loc[:,0:cutoff])
XClustered = pd.DataFrame(data=XClustered, index=X_train.index,columns=['clu
XClustered_tbl = pd.concat([y_train,XClustered], axis=1)
XClustered_tbl.columns = ['trueLabel','cluster']
XClustered_tbl
```

	trueLabel	cluster
0	5	17
1	0	16
2	4	0
3	1	12
4	9	8
...	...	...
49995	5	14
49996	0	18
49997	8	11
49998	4	19
49999	8	11

50000 rows × 2 columns

In [82]:
```
countByLabel = pd.DataFrame(data=XClustered_tbl.groupby('trueLabel').count(
countByLabel
```

Out[82]:

	cluster
trueLabel	
0	4932
1	5678
2	4968
3	5101
4	4859
5	4506
6	4951
7	5175
8	4842
9	4988

In [83]:
```
np.bincount(y_train)
```

Out[83]:
```
array([4932, 5678, 4968, 5101, 4859, 4506, 4951, 5175, 4842, 4988],
      dtype=int64)
```

In [78]:

```
#分群後每一群的數量
countByCluster = pd.DataFrame(XClustered_tbl['cluster'].value_counts())
countByCluster.reset_index(inplace=True,drop=False) #drop=False : the old i
countByCluster.columns = ['cluster','clusterCount']
countByCluster
```

Out[78]:

	cluster	clusterCount
0	10	3668
1	11	3480
2	19	3089
3	8	2936
4	12	2918
5	14	2893
6	9	2839
7	17	2804
8	1	2683
9	6	2588
10	0	2390
11	4	2377
12	3	2350
13	2	2254
14	7	2111
15	15	1997
16	13	1845
17	18	1741
18	16	1593
19	5	1444

In [73]:

```
#groupby 'cluster', 顯示每一群的label
XClustered_tbl.groupby('cluster').agg(lambda x: print(x))
```

```
49976    2
49993    2
Name: trueLabel, Length: 2254, dtype: int64
54       9
89       4
115      4
139      4
167      9
        ..
49874    4
49904    9
49940    4
49982    4
49988    9
Name: trueLabel, Length: 2350, dtype: int64
15       7
19       9
22       9
33       9
```

In [74]:

```
#groupby 'cluster', 計算每一群的label數量(預設由大小到小排序輸出)
XClustered_tbl.groupby('cluster').agg(lambda x: print(x.value_counts()))
```

```
4    1082
9     780
7     255
2      73
5      71
8      44
6      43
3      22
0      20
Name: trueLabel, dtype: int64
5    1406
4     292
8     254
0     176
2     173
6     166
3      99
7      50
9      41
```

In [75]:
```
#取出各分群的出現最多次數的那一個Label (即第0列的值)
countMostFreq=pd.DataFrame(XClustered_tbl.groupby('cluster').agg(lambda x:x
countMostFreq.reset_index(inplace=True,drop=False)
countMostFreq.columns = ['cluster','countMostFrequent']
countMostFreq
```

Out[75]:

	cluster	countMostFrequent
0	0	1082
1	1	1406
2	2	2127
3	3	1394
4	4	1486
5	5	1293
6	6	2061
7	7	1961
8	8	1106
9	9	2495
10	10	3093
11	11	1408
12	12	2474
13	13	1756
14	14	1447
15	15	1839
16	16	1487
17	17	1996
18	18	1665
19	19	1421

In [79]:
```
accuracyDF = countMostFreq.merge(countByCluster,left_on="cluster",right_on='
accuracyDF
```

Out[79]:

	cluster	countMostFrequent	clusterCount
0	0	1082	2390
1	1	1406	2683
2	2	2127	2254
3	3	1394	2350
4	4	1486	2377
5	5	1293	1444
6	6	2061	2588
7	7	1961	2111
8	8	1106	2936
9	9	2495	2839
10	10	3093	3668

In [81]:
```
#各群的正確率
accuracyByLabel = accuracyDF.countMostFrequent/accuracyDF.clusterCount
accuracyByLabel
```

Out[81]:
```
0     0.452720
1     0.524040
2     0.943656
3     0.593191
4     0.625158
5     0.895429
6     0.796368
7     0.928944
8     0.376703
9     0.878831
10    0.843239
11    0.404598
12    0.847841
13    0.951762
14    0.500173
15    0.920881
16    0.933459
17    0.711840
18    0.956347
19    0.460019
dtype: float64
```

In [80]:
```
overallAccuracy = accuracyDF.countMostFrequent.sum()/accuracyDF.clusterCount
print('overallAccuracy',overallAccuracy)
```

```
overallAccuracy 0.69994
```

5.3 K-means - Accuracy as the number of clusters varies

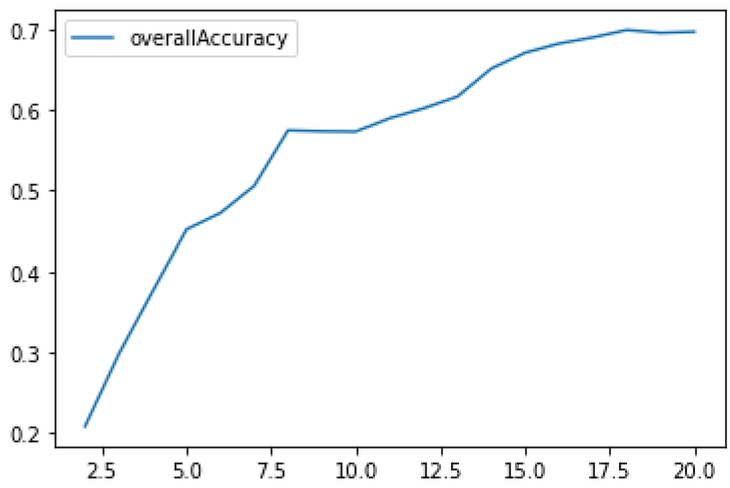
In [19]:

```
n_init = 10
max_iter = 300
tol = 0.0001
random_state = 2018

kMeans_inertia = \
    pd.DataFrame(data=[],index=range(2,21),columns=['inertia'])
overallAccuracy_kMeansDF = \
    pd.DataFrame(data=[],index=range(2,21),columns=['overallAccuracy'])

for n_clusters in range(2,21):
    kmeans = KMeans(n_clusters=n_clusters, n_init=n_init, \
                max_iter=max_iter, tol=tol, random_state=random_state)

    cutoff = 99
    kmeans.fit(X_train_PCA.loc[:,0:cutoff])
    kMeans_inertia.loc[n_clusters] = kmeans.inertia_
    X_train_kmeansClustered = kmeans.predict(X_train_PCA.loc[:,0:cutoff])
    X_train_kmeansClustered = \
        pd.DataFrame(data=X_train_kmeansClustered, index=X_train.index, \
                     columns=['cluster'])

    countByCluster_kMeans, countByLabel_kMeans, countMostFreq_kMeans, \
        accuracyDF_kMeans, overallAccuracy_kMeans, accuracyByLabel_kMeans \
        = analyzeCluster(X_train_kmeansClustered, y_train)

    overallAccuracy_kMeansDF.loc[n_clusters] = overallAccuracy_kMeans
```

In [20]:

```
# Plot accuracy
#分群愈多,accuracy愈高(理所當然. 因為群愈多,群內的同質性就會愈高)
overallAccuracy_kMeansDF.plot()
```

Out[20]: <Axes: >



In [21]:
```
# Accuracy by cluster
accuracyByLabel_kMeans
```

Out[21]:
```
0     0.634156
1     0.923767
2     0.389366
3     0.905002
4     0.463787
5     0.941964
6     0.928007
7     0.856261
8     0.954465
9     0.596520
10    0.823661
11    0.711253
12    0.526414
13    0.950378
14    0.489108
15    0.838684
16    0.802189
17    0.385685
18    0.451670
19    0.864761
dtype: float64
```

In [22]:
```
# View cluster labels
X_train_kmeansClustered
```

Out[22]:

	cluster
0	11
1	6
2	18
3	10
4	17
...	...
49995	14
49996	8
49997	2
49998	4
49999	2

50000 rows × 1 columns

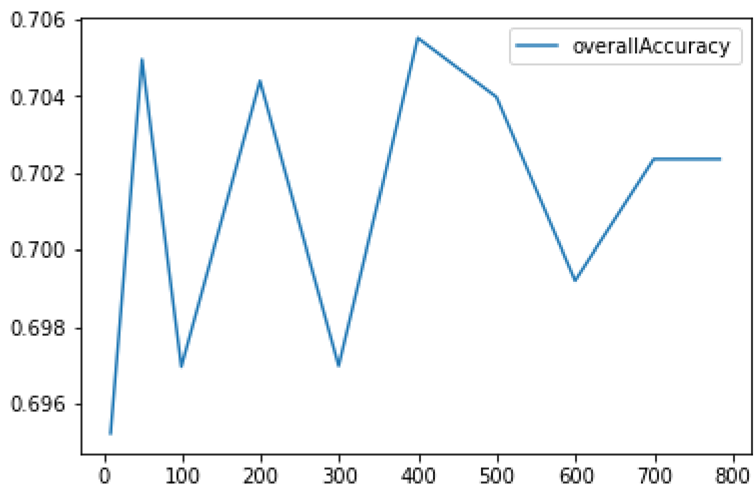
In [23]:
```
# Save cluster labels
X_train_kmeansClustered[0:2000].to_csv('kmeans_cluster_labels.tsv', sep = '\
```

5.4 Accuracy as the number of principal components varies

In [85]:
```
# K-means - Accuracy as the number of components varies

n_clusters = 20
n_init = 10
max_iter = 300
tol = 0.0001
random_state = 2018

kMeans_inertia = pd.DataFrame(data=[],index=[9, 49, 99, 199, \
                    299, 399, 499, 599, 699, 783],columns=['inertia'])

overallAccuracy_kMeansDF = pd.DataFrame(data=[],index=[9, 49, \
                    99, 199, 299, 399, 499, 599, 699, 783], \
                    columns=['overallAccuracy'])

for cutoffNumber in [9, 49, 99, 199, 299, 399, 499, 599, 699, 783]:
    kmeans = KMeans(n_clusters=n_clusters, n_init=n_init, \
                max_iter=max_iter, tol=tol, random_state=random_state)

    cutoff = cutoffNumber
    kmeans.fit(X_train_PCA.loc[:,0:cutoff])
    kMeans_inertia.loc[cutoff] = kmeans.inertia_
    X_train_kmeansClustered = kmeans.predict(X_train_PCA.loc[:,0:cutoff])
    X_train_kmeansClustered = pd.DataFrame(data=X_train_kmeansClustered, \
                                index=X_train.index, columns=['cluster'])

    countByCluster_kMeans, countByLabel_kMeans, countMostFreq_kMeans, \
        accuracyDF_kMeans, overallAccuracy_kMeans, accuracyByLabel_kMeans \
        = analyzeCluster(X_train_kmeansClustered, y_train)

    overallAccuracy_kMeansDF.loc[cutoff] = overallAccuracy_kMeans
```

In [87]:
```
# Accuracy relative to number of principal components
overallAccuracy_kMeansDF.plot()
```

Out[87]: <Axes: >



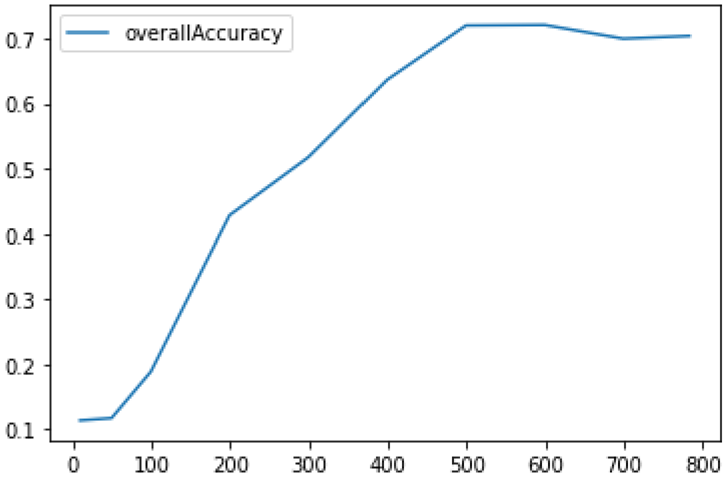
5.5 Accuracy as the number of original dimensions varies

In [88]:
```
# K-means - Accuracy as the number of components varies
# On the original MNIST data (not PCA-reduced)

n_clusters = 20
n_init = 10
max_iter = 300
tol = 0.0001
random_state = 2018

kMeans_inertia = pd.DataFrame(data=[],index=[9, 49, 99, 199, \
                    299, 399, 499, 599, 699, 783],columns=['inertia'])

overallAccuracy_kMeansDF = pd.DataFrame(data=[],index=[9, 49, \
                    99, 199, 299, 399, 499, 599, 699, 783], \
                    columns=['overallAccuracy'])

for cutoffNumber in [9, 49, 99, 199, 299, 399, 499, 599, 699, 783]:
    kmeans = KMeans(n_clusters=n_clusters, n_init=n_init, \
                max_iter=max_iter, tol=tol, random_state=random_state)

    cutoff = cutoffNumber
    kmeans.fit(X_train.loc[:,0:cutoff])
    kMeans_inertia.loc[cutoff] = kmeans.inertia_
    X_train_kmeansClustered = kmeans.predict(X_train.loc[:,0:cutoff])
    X_train_kmeansClustered = pd.DataFrame(data=X_train_kmeansClustered, \
                                index=X_train.index, columns=['cluster'])

    countByCluster_kMeans, countByLabel_kMeans, countMostFreq_kMeans, \
        accuracyDF_kMeans, overallAccuracy_kMeans, accuracyByLabel_kMeans \
        = analyzeCluster(X_train_kmeansClustered, y_train)

    overallAccuracy_kMeansDF.loc[cutoff] = overallAccuracy_kMeans
```

```
C:\Users\joseph\anaconda3\lib\site-packages\sklearn\base.py:1152: Converge
nceWarning: Number of distinct clusters (1) found smaller than n_clusters
(20). Possibly due to duplicate points in X.
  return fit_method(estimator, *args, **kwargs)
```

In [89]:
```
# Accuracy relative to number of original dimensions
overallAccuracy_kMeansDF.plot()
```

Out[89]: <Axes: >



5.6 Conculsion:

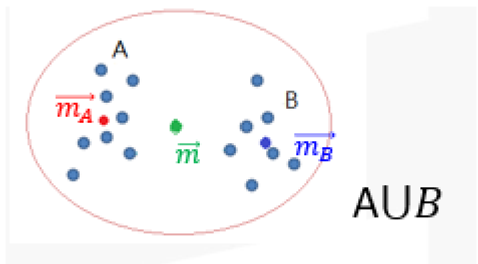
使用原始資料，資料維度在784維時的正確率是0.7左右, 而若使用PCA時,在少於100個維度的情況下, 就可以達到0.7的正確率。

6 Hierarchical clustering

[https://jbhender.github.io/Stats506/F18/GP/Group10.html (https://jbhender.github.io/Stats506/F18/GP/Group10.html)](https://jbhender.github.io/Stats506/F18/GP/Group10.html)

Ward's method says that the distance between two clusters, A and B, is how much the sum of squares will increase when we merge them.

$$\Delta(A, B) = \sum_{i \in A \bigcup B} ||\vec{x_i} - \vec{m}_{A \bigcup B}||^2 - \sum_{i \in A} ||\vec{x_i} - \vec{m}_A||^2 - \sum_{i \in B} ||\vec{x_i} - \vec{m}_B||^2 = \frac{n_A n_B}{n_A + n}$$



In [28]:
```
# Perform hierarchical clustering
import fastcluster
from scipy.cluster.hierarchy import dendrogram, cophenet
from scipy.spatial.distance import pdist

cutoff = 99
Z = fastcluster.linkage_vector(X_train_PCA.loc[:,0:cutoff], \
                               method='ward', metric='euclidean')
Z_dataFrame = pd.DataFrame(data=Z, \
    columns=['clusterOne','clusterTwo','distance','newClusterSize'])
```

In [29]:
```
# Show leaves
Z_dataFrame.iloc[:20]
```

Out[29]:

	clusterOne	clusterTwo	distance	newClusterSize
0	42194.0	43025.0	0.562811	2.0
1	28350.0	37674.0	0.590923	2.0
2	26696.0	44705.0	0.621491	2.0
3	12634.0	32823.0	0.627762	2.0
4	24707.0	43151.0	0.637646	2.0
5	20465.0	24483.0	0.662483	2.0
6	466.0	42098.0	0.664151	2.0
7	46542.0	49961.0	0.665527	2.0
8	2301.0	5732.0	0.671081	2.0
9	37564.0	47668.0	0.675107	2.0
10	3375.0	26243.0	0.685899	2.0
11	15722.0	30368.0	0.686326	2.0
12	21247.0	21575.0	0.694369	2.0
13	14900.0	42486.0	0.696735	2.0
14	30100.0	41908.0	0.699287	2.0
15	12040.0	13254.0	0.701116	2.0
16	10508.0	25434.0	0.708619	2.0
17	30695.0	30757.0	0.710044	2.0
18	31019.0	31033.0	0.712045	2.0
19	36264.0	37285.0	0.713133	2.0

In [30]:
```
# Show leaves higher on the tree
Z_dataFrame.iloc[49980:]
```

Out[30]:

	clusterOne	clusterTwo	distance	newClusterSize
49980	99938.0	99959.0	152.158499	3544.0
49981	99953.0	99974.0	177.594230	5680.0
49982	99961.0	99976.0	179.132302	3379.0
49983	99942.0	99972.0	180.320587	4781.0
49984	99962.0	99980.0	189.535103	6051.0
49985	99971.0	99979.0	198.547538	6364.0
49986	99968.0	99969.0	202.903316	4246.0
49987	99954.0	99973.0	205.242622	6070.0
49988	99956.0	99982.0	222.401645	4872.0
49989	99966.0	99987.0	246.573179	9119.0
49990	99978.0	99986.0	247.756736	7000.0
49991	99975.0	99977.0	262.852446	6133.0
49992	99984.0	99985.0	265.504907	12415.0
49993	99983.0	99990.0	267.704777	11781.0
49994	99981.0	99989.0	293.502476	14799.0
49995	99992.0	99993.0	390.062843	24196.0
49996	99991.0	99995.0	419.196650	30329.0
49997	99988.0	99996.0	469.343025	35201.0
49998	99994.0	99997.0	497.511872	50000.0

In [31]:
```
# Create clusters
from scipy.cluster.hierarchy import fcluster

distance_threshold = 160
clusters = fcluster(Z, distance_threshold, criterion='distance')
X_train_hierClustered = \
    pd.DataFrame(data=clusters,index=X_train_PCA.index,columns=['cluster'])
```

In [32]:
```
# Print number of clusters
print("Number of distinct clusters: ", \
      len(X_train_hierClustered['cluster'].unique()))
```

```
Number of distinct clusters:  19
```

In [33]:
```python
# Show overall accuracy
countByCluster_hierClust, countByLabel_hierClust, \
    countMostFreq_hierClust, accuracyDF_hierClust, \
    overallAccuracy_hierClust, accuracyByLabel_hierClust \
    = analyzeCluster(X_train_hierClustered, y_train)

print("Overall accuracy from hierarchical clustering: ", \
      overallAccuracy_hierClust)
```

```
Overall accuracy from hierarchical clustering:  0.79216
```

In [34]:
```python
# Show accuracy by cluster
print("Accuracy by cluster for hierarchical clustering")
accuracyByLabel_hierClust
```

```
Accuracy by cluster for hierarchical clustering
```

Out[34]:
```
0     0.917068
1     0.505117
2     0.503444
3     0.559322
4     0.969558
5     0.990623
6     0.986610
7     0.982719
8     0.860870
9     0.984424
10    0.960112
11    0.526806
12    0.402222
13    0.906386
14    0.952848
15    0.946633
16    0.982934
17    0.965486
18    0.701881
dtype: float64
```

In [35]:
```
# View cluster labels
X_train_hierClustered
```

Out[35]:

	cluster
0	14
1	7
2	3
3	10
4	4
...	...
49995	14
49996	7
49997	12
49998	4
49999	13

50000 rows × 1 columns

In [36]:
```
# Save cluster labels
X_train_hierClustered[0:2000].to_csv('hierarchical_cluster_labels.tsv', sep
```

7 DBSCAN

In [37]:
```
# Perform DBSCAN
from sklearn.cluster import DBSCAN

eps = 3
min_samples = 10

db = DBSCAN(eps=eps, min_samples=min_samples)

cutoff = 99
X_train_PCA_dbscanClustered = db.fit_predict(X_train_PCA.loc[:,0:cutoff])
X_train_PCA_dbscanClustered = \
    pd.DataFrame(data=X_train_PCA_dbscanClustered, index=X_train.index, \
                 columns=['cluster'])

countByCluster_dbscan, countByLabel_dbscan, countMostFreq_dbscan, \
    accuracyDF_dbscan, overallAccuracy_dbscan, accuracyByLabel_dbscan \
    = analyzeCluster(X_train_PCA_dbscanClustered, y_train)

overallAccuracy_dbscan
```

Out[37]: 0.22434

In [38]:
```
# Print overall accuracy
print("Overall accuracy from DBSCAN: ",overallAccuracy_dbscan)
```

```
Overall accuracy from DBSCAN:  0.22434
```

In [39]:
```
# View cluster labels
X_train_PCA_dbscanClustered
```

Out[39]:

	cluster
0	-1
1	-1
2	-1
3	0
4	-1
...	...
49995	-1
49996	-1
49997	-1
49998	-1
49999	-1

50000 rows × 1 columns

In [40]:
```
# Show cluster results
print("Cluster results for DBSCAN")
countByCluster_dbscan
```

```
Cluster results for DBSCAN
```

Out[40]:

	cluster	clusterCount
0	-1	42073
1	0	7259
2	5	184
3	1	100
4	4	66
5	11	36
6	12	31
7	16	29
8	3	21
9	13	21

In [41]:
```
accuracyDF_dbscan
```

Out[41]:

	cluster	countMostFrequent	clusterCount
0	-1	5095	42073
1	0	5455	7259
2	1	100	100
3	2	14	14
4	3	21	21
5	4	65	66
6	5	184	184
7	6	9	9
8	7	9	9
9	8	10	10
10	9	10	10
11	10	7	7
12	11	36	36
13	12	31	31
14	13	21	21
15	14	12	12
16	15	9	9
17	16	29	29
18	17	8	8
19	18	8	8
20	19	5	5
21	20	11	11
22	21	10	10
23	22	8	8
24	23	10	10
25	24	10	10
26	25	10	10
27	26	10	10
28	27	10	10

In [42]:
```
# Save cluster labels
X_train_PCA_dbscanClustered[0:2000].to_csv('dbscan_cluster_labels.tsv', sep
```

7.1 Conclusion:

DBSCAN 在處理MNIST 資料集表現比k-means差很多, 因為找不到 Density 的關係, 超過80%以上資料都被歸類成outlier. 扣除群0其他群僅僅是零星數量。

7.2 Conslusion

HDBSCAN 效果只有比 DBSCAN 好一點而已

In [ ]:

```

```