homework13

December 9, 2019

NAME: Zhi Li
SECTION: 113523595
CS 5970: Machine Learning Practices

1 Homework 13: Clustering

1.1 Assignment Overview

Follow the TODOs and read through and understand any provided code.
For all plots, make sure all necessary axes and curves are clearly and accurately labeled. Include figure/plot titles appropriately as well. Post any questions to the Canvas Discussion.

For this assignment you will be exploring clustering. Clustering is an unsupervised learning technique that can be utilized to discover interesting patterns or groupings amongst data.

Select one of the two tasks below

1.1.1 Task 1

Explore clustering for the Human Activity Recognition dataset. Recordings come from 30 subjects performing activities of daily living while carrying a waist-mounted smartphone with embedded inertial sensors.

1.1.2 Task 2

Explore clustering for a few synthetic datasets.

1.1.3 Objectives

- Clustering
- Dimensionality Reduction

1.1.4 Notes

- Do not save work within the ml_practices folder

1.1.5 General References

- Guide to Jupyter
- Python Built-in Functions
- Python Data Structures
- Numpy Reference
- Numpy Cheat Sheet
- Summary of matplotlib
- DataCamp: Matplotlib
- Pandas DataFrames
- Sci-kit Learn Linear Models
- Sci-kit Learn Ensemble Models
- Sci-kit Learn Metrics
- Sci-kit Learn Model Selection
- Sci-kit Learn Pipelines
- Sci-kit Learn Preprocessing

[1]:
```python
import sys
sys.path.insert(0, "ml_practices/imports/hws/hw9/visualize.py")

# THESE 4 IMPORTS ARE CUSTOM .py FILES AND CAN BE FOUND
# ON THE SERVER AND GIT
import visualize
import metrics_plots
from pipeline_components import DataSampleDropper, DataFrameSelector
from pipeline_components import DataScaler, DataLabelEncoder

import pandas as pd
import numpy as np
import seaborn as sns
import scipy.stats as stats
import os, re, fnmatch
import pathlib, itertools
import time as timelib
import matplotlib.pyplot as plt
import matplotlib.patheffects as peffects

from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import explained_variance_score, confusion_matrix
from sklearn.metrics import mean_squared_error, roc_curve, auc, f1_score
```

```
from sklearn import cluster
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.manifold import Isomap
from sklearn.neighbors import NearestNeighbors
from sklearn.externals import joblib

FIGWIDTH = 5
FIGHEIGHT = 5
FONTSIZE = 12

plt.rcParams['figure.figsize'] = (FIGWIDTH, FIGHEIGHT)
plt.rcParams['font.size'] = FONTSIZE

plt.rcParams['xtick.labelsize'] = FONTSIZE
plt.rcParams['ytick.labelsize'] = FONTSIZE

%matplotlib inline
```

[2]:
```
"""
Display current working directory of this notebook. If you are using
relative paths for your data, then it needs to be relative to the CWD.
"""
HOME_DIR = pathlib.Path.home()
pathlib.Path.cwd()
```

[2]:
```
PosixPath('/home/jovyan')
```

2 TASK 1 DATASET: UCI_HAR_Dataset

2.0.1 LOAD DATA

[3]:
```
"""
https://archive.ics.uci.edu/ml/datasets/
↪Human+Activity+Recognition+Using+Smartphones

Abstract: Human Activity Recognition database built from the recordings of 30␣
↪subjects
performing activities of daily living (ADL) while carrying a waist-mounted␣
↪smartphone
with embedded inertial sensors.

Number of Attributes: 561

Source:
```

```
Jorge L. Reyes-Ortiz(1,2), Davide Anguita(1), Alessandro Ghio(1), Luca Oneto(1)␣
↪and
Xavier Parra(2)
1 - Smartlab - Non-Linear Complex Systems Laboratory
DITEN - Università degli Studi di Genova, Genoa (I-16145), Italy.
2 - CETpD - Technical Research Centre for Dependency Care and Autonomous Living
Universitat Politècnica de Catalunya (BarcelonaTech). Vilanova i la Geltrú␣
↪(08800),
Spain activityrecognition '@' smartlab.ws


Data Set Information:
The experiments have been carried out with a group of 30 subjects. Each person
performed six activities (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS,␣
↪SITTING,
STANDING, LAYING) wearing a smartphone (Samsung Galaxy S II) on the waist.

The sensor signals (accelerometer and gyroscope) were pre-processed by applying␣
↪noise
filters and then sampled in fixed-width sliding windows of 2.56 sec and 50%␣
↪overlap
(128 readings/window).

Check the README.txt file for further details about this dataset.

Attribute Information:
For each record in the dataset it is provided:
- Triaxial acceleration from the accelerometer (total acceleration) and the
  estimated body acceleration.
- Triaxial Angular velocity from the gyroscope.
- A 561-feature vector with time and frequency domain variables.
- Its activity label.
- An identifier of the subject who carried out the experiment.

Citation:
Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra and Jorge L.␣
↪Reyes-Ortiz. A Public Domain Dataset for Human Activity Recognition Using␣
↪Smartphones. 21th European Symposium on Artificial Neural Networks,␣
↪Computational Intelligence and Machine Learning, ESANN 2013. Bruges, Belgium␣
↪24-26 April 2013.
"""
# TODO: Set any data file paths appropriately
data = pd.read_csv('ml_practices/imports/datasets/UCI_HAR_Dataset/
↪uci_har_Xy_train.csv')
data.shape
```

[3]: (7352, 563)

[4]:
```
data.head()
```

[4]:
```
   tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  tBodyAcc-std()-X  \
0           0.288585          -0.020294          -0.132905         -0.995279
1           0.278419          -0.016411          -0.123520         -0.998245
2           0.279653          -0.019467          -0.113462         -0.995380
3           0.279174          -0.026201          -0.123283         -0.996091
4           0.276629          -0.016570          -0.115362         -0.998139

   tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  tBodyAcc-mad()-Y  \
0         -0.983111         -0.913526         -0.995112         -0.983185
1         -0.975300         -0.960322         -0.998807         -0.974914
2         -0.967187         -0.978944         -0.996520         -0.963668
3         -0.983403         -0.990675         -0.997099         -0.982750
4         -0.980817         -0.990482         -0.998321         -0.979672

   tBodyAcc-mad()-Z  tBodyAcc-max()-X  …  fBodyBodyGyroJerkMag-kurtosis()  \
0         -0.923527         -0.934724  …                        -0.710304
1         -0.957686         -0.943068  …                        -0.861499
2         -0.977469         -0.938692  …                        -0.760104
3         -0.989302         -0.938692  …                        -0.482845
4         -0.990441         -0.942469  …                        -0.699205

   angle(tBodyAccMean,gravity)  angle(tBodyAccJerkMean),gravityMean)  \
0                    -0.112754                              0.030400
1                     0.053477                             -0.007435
2                    -0.118559                              0.177899
3                    -0.036788                             -0.012892
4                     0.123320                              0.122542

   angle(tBodyGyroMean,gravityMean)  angle(tBodyGyroJerkMean,gravityMean)  \
0                         -0.464761                             -0.018446
1                         -0.732626                              0.703511
2                          0.100699                              0.808529
3                          0.640011                             -0.485366
4                          0.693578                             -0.615971

   angle(X,gravityMean)  angle(Y,gravityMean)  angle(Z,gravityMean)  \
0             -0.841247              0.179941             -0.058627
1             -0.844788              0.180289             -0.054317
2             -0.848933              0.180637             -0.049118
3             -0.848649              0.181935             -0.047663
4             -0.847865              0.185151             -0.043892

   activity_label  subj_number
```

```
0                5            1
1                5            1
2                5            1
3                5            1
4                5            1

[5 rows x 563 columns]
```

[5]:
```
""" PROVIDED
Check for any NaNs
"""
data.isna().any().any()
```

[5]:
```
False
```

[6]:
```
data.describe()
```

[6]:
```
       tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  \
count        7352.000000        7352.000000        7352.000000
mean            0.274488          -0.017695          -0.109141
std             0.070261           0.040811           0.056635
min            -1.000000          -1.000000          -1.000000
25%             0.262975          -0.024863          -0.120993
50%             0.277193          -0.017219          -0.108676
75%             0.288461          -0.010783          -0.097794
max             1.000000           1.000000           1.000000

       tBodyAcc-std()-X  tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  \
count       7352.000000       7352.000000       7352.000000       7352.000000
mean          -0.605438         -0.510938         -0.604754         -0.630512
std            0.448734          0.502645          0.418687          0.424073
min           -1.000000         -0.999873         -1.000000         -1.000000
25%           -0.992754         -0.978129         -0.980233         -0.993591
50%           -0.946196         -0.851897         -0.859365         -0.950709
75%           -0.242813         -0.034231         -0.262415         -0.292680
max            1.000000          0.916238          1.000000          1.000000

       tBodyAcc-mad()-Y  tBodyAcc-mad()-Z  tBodyAcc-max()-X  …  \
count       7352.000000       7352.000000       7352.000000  …
mean          -0.526907         -0.606150         -0.468604  …
std            0.485942          0.414122          0.544547  …
min           -1.000000         -1.000000         -1.000000  …
25%           -0.978162         -0.980251         -0.936219  …
50%           -0.857328         -0.857143         -0.881637  …
75%           -0.066701         -0.265671         -0.017129  …
max            0.967664          1.000000          1.000000  …
```

```
       fBodyBodyGyroJerkMag-kurtosis()  angle(tBodyAccMean,gravity)  \
count                      7352.000000                  7352.000000
mean                         -0.625294                     0.008684
std                           0.307584                     0.336787
min                          -0.999765                    -0.976580
25%                          -0.845573                    -0.121527
50%                          -0.711692                     0.009509
75%                          -0.503878                     0.150865
max                           0.956845                     1.000000

       angle(tBodyAccJerkMean),gravityMean)  angle(tBodyGyroMean,gravityMean)  \
count                           7352.000000                       7352.000000
mean                               0.002186                          0.008726
std                                0.448306                          0.608303
min                               -1.000000                         -1.000000
25%                               -0.289549                         -0.482273
50%                                0.008943                          0.008735
75%                                0.292861                          0.506187
max                                1.000000                          0.998702

       angle(tBodyGyroJerkMean,gravityMean)  angle(X,gravityMean)  \
count                           7352.000000           7352.000000
mean                              -0.005981             -0.489547
std                                0.477975              0.511807
min                               -1.000000             -1.000000
25%                               -0.376341             -0.812065
50%                               -0.000368             -0.709417
75%                                0.359368             -0.509079
max                                0.996078              1.000000

       angle(Y,gravityMean)  angle(Z,gravityMean)  activity_label  subj_number
count           7352.000000           7352.000000     7352.000000  7352.000000
mean               0.058593             -0.056515        3.643362    17.413085
std                0.297480              0.279122        1.744802     8.975143
min               -1.000000             -1.000000        1.000000     1.000000
25%               -0.017885             -0.143414        2.000000     8.000000
50%                0.182071              0.003181        4.000000    19.000000
75%                0.248353              0.107659        5.000000    26.000000
max                0.478157              1.000000        6.000000    30.000000

[8 rows x 563 columns]
```

[7]:
```
""" TODO
Class counts
use pd.value_counts(data['activity_label']) to determine how many instances␣
 ↪there
are for each activity class
```

```
"""
cnt = pd.value_counts(data['activity_label'])# TODO
cnt
```

[7]:
```
6    1407
5    1374
4    1286
1    1226
2    1073
3     986
Name: activity_label, dtype: int64
```

[8]:
```
""" PROVIDED
Feature names for each column in the data
"""
features = pd.read_csv('ml_practices/imports/datasets/UCI_HAR_Dataset/meta/
 ↪features.txt', sep='\s+', header=None)
features.columns = ['num', 'feature_name']
features.shape
```

[8]:
```
(561, 2)
```

[9]:
```
features.head()
```

[9]:

	num	feature_name
0	1	tBodyAcc-mean()-X
1	2	tBodyAcc-mean()-Y
2	3	tBodyAcc-mean()-Z
3	4	tBodyAcc-std()-X
4	5	tBodyAcc-std()-Y

[10]:
```
""" PROVIDED
Activity Class Label names
"""
activity_labels = pd.read_csv('ml_practices/imports/datasets/UCI_HAR_Dataset/
 ↪meta/activity_labels.txt', sep='\s+', header=None)
activity_labels.columns = ['num', 'activity_name']
nclasses, ncols = activity_labels.shape
nclasses, ncols
```

[10]:
```
(6, 2)
```

[11]:
```
# Display class names and corresponding number
activity_labels
```

[11]:

	num	activity_name
0	1	WALKING

```
1    2    WALKING_UPSTAIRS
2    3  WALKING_DOWNSTAIRS
3    4             SITTING
4    5            STANDING
5    6              LAYING
```

[12]:
```python
# Separate out just the class names
activity_names = list(activity_labels['activity_name'].values)
activity_names
```

[12]:
```
['WALKING',
 'WALKING_UPSTAIRS',
 'WALKING_DOWNSTAIRS',
 'SITTING',
 'STANDING',
 'LAYING']
```

2.0.2 PARTITION DATA

[13]:
```python
""" TODO
Separate the data into X and y. Use the features variable to pull out the
appropriate feature data. For y we are predicting the 'activity_label'
column from the data.

Hold out a subset of the data, using train_test_split, a test_size
fraction of .2, and shuffle to False
"""
# Feature Names
feature_names = features['feature_name'].values

# TODO: Separate the data into X and y
X = data.drop('activity_label', axis=1)# TODO

# Substract 1 from the label number for convenience, such that number matches
# the list index. i.e. changing the label numbers from 1 to 6 to 0 to 5
y = data['activity_label'].copy().values - 1

# TODO: Split into train and validation
Xtrain, Xval, ytrain, yval= train_test_split(X, y, test_size=0.2, shuffle=False)


nsamples_train = Xtrain.shape[0]

Xtrain.shape, Xval.shape, ytrain.shape, yval.shape
```

[13]:
```
((5881, 562), (1471, 562), (5881,), (1471,))
```

2.0.3 CLUSTERING

[14]:
```
def group_scatter_plot(X, labels, feature_names, label_names, centers=None,
                       leg_on=False, FIGSIZE=(15,10), elev=35, angle=310):
    '''
    Plot 2D or 3D scatter plots of selected sets of features
    PARAMS:
        X: full feature space as a dataframe
        labels: labels for each example in X
        feature_names: subset of features to plot from X
        label_names: contains nclass elements, where each element is the name
                     for each class (Note: only viable for classes not clusters)
        centers: nclass-by-2 or nclass-by-3 matrix of group centers.
        leg_on: flag whether to display the legend (Note: only set True when
                plotting the actual class groupings. False when displaying␣
↪clusters)
        FIGSIZE: tuple of figure width and height
        elev: 3D plot view elevation
        angle: 3D plot view angle
    '''
    # Select a subset of features
    data = X[feature_names].copy().values

    # Create the figure
    fig = plt.figure(figsize=FIGSIZE)

    # 2D Plots
    if data.shape[1] == 2:
        ax0 = fig.add_subplot(111)
        # Plot the points by class or cluster
        for i, name in enumerate(label_names):
            inds = labels == i
            ax0.scatter(data[inds,0], data[inds,1], label=name)
        if leg_on: ax0.legend()

    # 3D Plots
    elif data.shape[1] > 2:
        ax0 = fig.add_subplot(111, projection='3d')
        # Plot the points by class or cluster
        for i, name in enumerate(label_names):
            inds = labels == i
            ax0.scatter(data[inds,0], data[inds,1], data[inds,2], label=name)
        ax0.view_init(elev, angle)
        if leg_on: ax0.legend()

    if centers is not None:
        # Plot the group centers
```

```
        mn = np.min(labels)
        mx = np.max(labels)
        if data.shape[1] == 2:
            ax0.scatter(centers[:,0], centers[:,1], c=np.arange(mn, mx+1),
                        marker='D', cmap=plt.cm.rainbow)
        elif data.shape[1] > 2:
            ax0.scatter(centers[:,0], centers[:,1], centers[:,2],
                       c=np.arange(mn, mx+1), marker='D', cmap=plt.cm.rainbow)
```

[15]:
```
def compute_class_centers(X, y, feature_names, classes):
    '''
    Compute group centers within the selected sub-feature space
    PARAMS:
        X: full feature space
        y: labels for each example in X
        classes: contains nclass elements, where each element is the index for␣
 ↪each class
    '''
    data = X[feature_names].copy().values

    nclasses = len(classes)
    nfeats = len(feature_names)
    centers = np.empty((nclasses, nfeats))

    for c in classes:
        centers[c, :] = np.mean(data[y == c, :], axis=0)
    return centers
```

[16]:
```
data.head()
```

[16]:
```
   tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  tBodyAcc-std()-X  \
0           0.288585          -0.020294          -0.132905         -0.995279
1           0.278419          -0.016411          -0.123520         -0.998245
2           0.279653          -0.019467          -0.113462         -0.995380
3           0.279174          -0.026201          -0.123283         -0.996091
4           0.276629          -0.016570          -0.115362         -0.998139

   tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  tBodyAcc-mad()-Y  \
0         -0.983111         -0.913526         -0.995112         -0.983185
1         -0.975300         -0.960322         -0.998807         -0.974914
2         -0.967187         -0.978944         -0.996520         -0.963668
3         -0.983403         -0.990675         -0.997099         -0.982750
4         -0.980817         -0.990482         -0.998321         -0.979672

   tBodyAcc-mad()-Z  tBodyAcc-max()-X  …  fBodyBodyGyroJerkMag-kurtosis()  \
0         -0.923527         -0.934724  …                        -0.710304
1         -0.957686         -0.943068  …                        -0.861499
```

```
2         -0.977469          -0.938692  …                        -0.760104
3         -0.989302          -0.938692  …                        -0.482845
4         -0.990441          -0.942469  …                        -0.699205

   angle(tBodyAccMean,gravity)  angle(tBodyAccJerkMean),gravityMean)  \
0                    -0.112754                              0.030400
1                     0.053477                             -0.007435
2                    -0.118559                              0.177899
3                    -0.036788                             -0.012892
4                     0.123320                              0.122542

   angle(tBodyGyroMean,gravityMean)  angle(tBodyGyroJerkMean,gravityMean)  \
0                         -0.464761                             -0.018446
1                         -0.732626                              0.703511
2                          0.100699                              0.808529
3                          0.640011                             -0.485366
4                          0.693578                             -0.615971

   angle(X,gravityMean)  angle(Y,gravityMean)  angle(Z,gravityMean)  \
0             -0.841247              0.179941             -0.058627
1             -0.844788              0.180289             -0.054317
2             -0.848933              0.180637             -0.049118
3             -0.848649              0.181935             -0.047663
4             -0.847865              0.185151             -0.043892

   activity_label  subj_number
0               5            1
1               5            1
2               5            1
3               5            1
4               5            1

[5 rows x 563 columns]
```

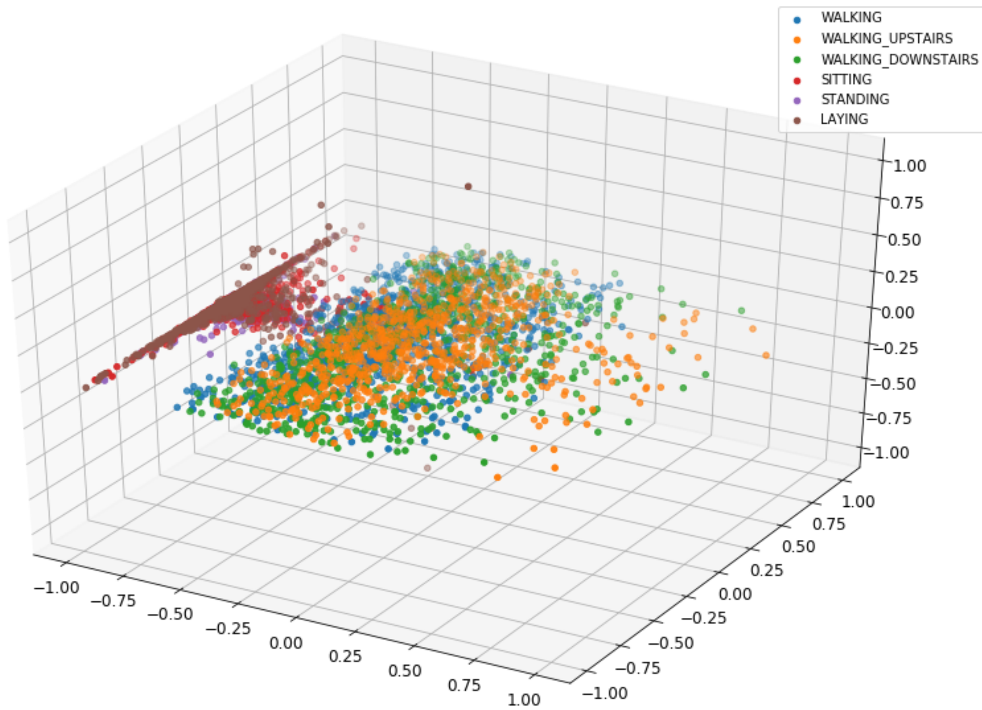
[18]:
```
model= KMeans(n_clusters=6, )
model.fit_predict(Xtrain)
```

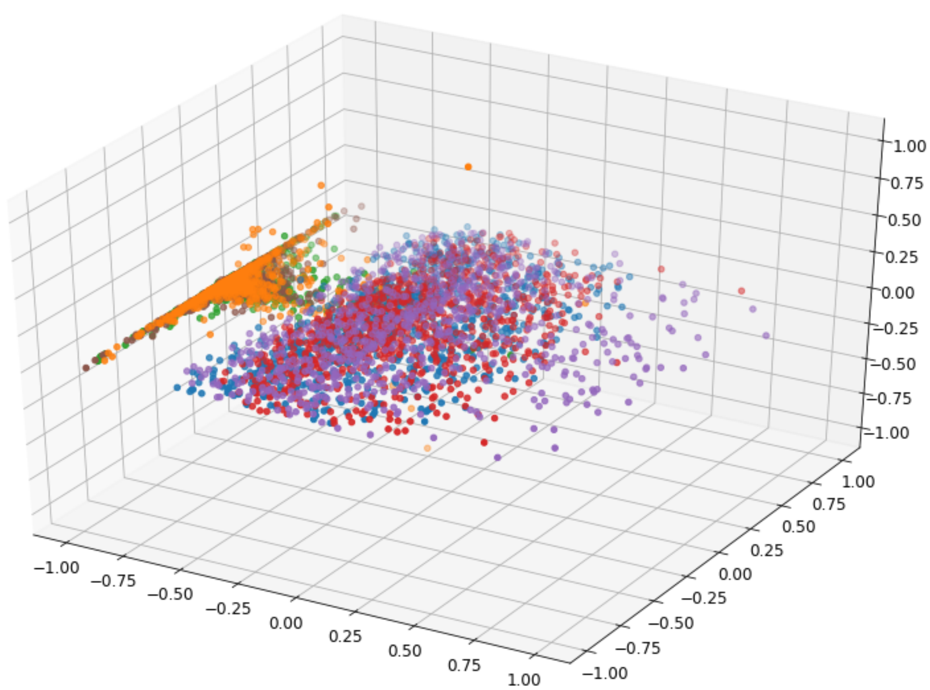
[18]: array([2, 2, 2, …, 1, 1, 1], dtype=int32)

[19]:
```
""" TODO
Use the following two cells.

Observe and analyze 2 feature subspaces of 2 or 3 features. To do this select␣
↪sets
of 2 or 3 features. For example, consider the feature subspace defined by the␣
↪features
'tBodyAcc-entropy()-X', 'tBodyAcc-entropy()-Y' and 'tBodyAcc-entropy()-Z'.
```

```
First plot the actual classifications in this subspace using:
    group_scatter_plot(Xtrain, ytrain, selected_feats, activity_names,␣
 ↪leg_on=True, angle=300)

Second, construct a KMeans model for unsupervised learning of various␣
 ↪clusterings of
the data in the selected feature subspaces. Use predict on the KMeans model to␣
 ↪determine
the set of 6 clusters. Use group_scatter_plot(leg_on=False) with the predicted␣
 ↪clusters as the
labels, and not the real classifications. Display the model's interia (i.e. the␣
 ↪sum of squared
distances of samples to their closest cluster center)
"""
angle = 300

feats = ['tBodyAcc-std()-Z', 'angle(tBodyAccMean,gravity)',␣
 ↪'tBodyAcc-mean()-Y']# TODO: list of subset of selected features
# TODO: Plot Actual classifications. use group_scatter_plot with leg_on=True
group_scatter_plot(Xtrain, ytrain, feats, activity_names, leg_on=True,␣
 ↪angle=300)


# TODO: Determine clusters. Create KMeans model and predict the clusters
model= KMeans(n_clusters=6, )
ypreds= model.fit_predict(Xtrain)


# TODO: Plot determined clusters. use group_scatter_plot with leg_on=False
group_scatter_plot(Xtrain, ypreds, feats, activity_names, leg_on=False,␣
 ↪angle=300)


# Sum of squared distances of samples to their closest cluster center
model.inertia_
```

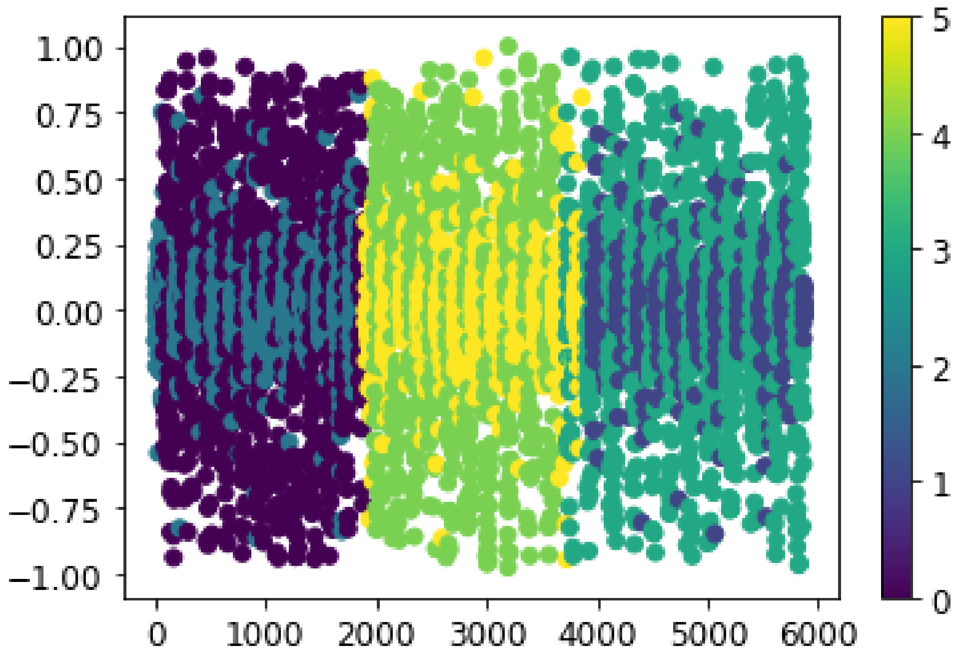
[19]: 172627.63686784907


WALKING
WALKING_UPSTAIRS
WALKING_DOWNSTAIRS
SITTING
STANDING
LAYING
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

[27]:
```
""" TODO
Observe the class groupings in the second selected feature subspace:
"""
plt.scatter(range(len(Xtrain[feats[1]])), Xtrain[feats[1]], c=ypreds)
plt.colorbar()
```

[27]: <matplotlib.colorbar.Colorbar at 0x7f15bea81e10>

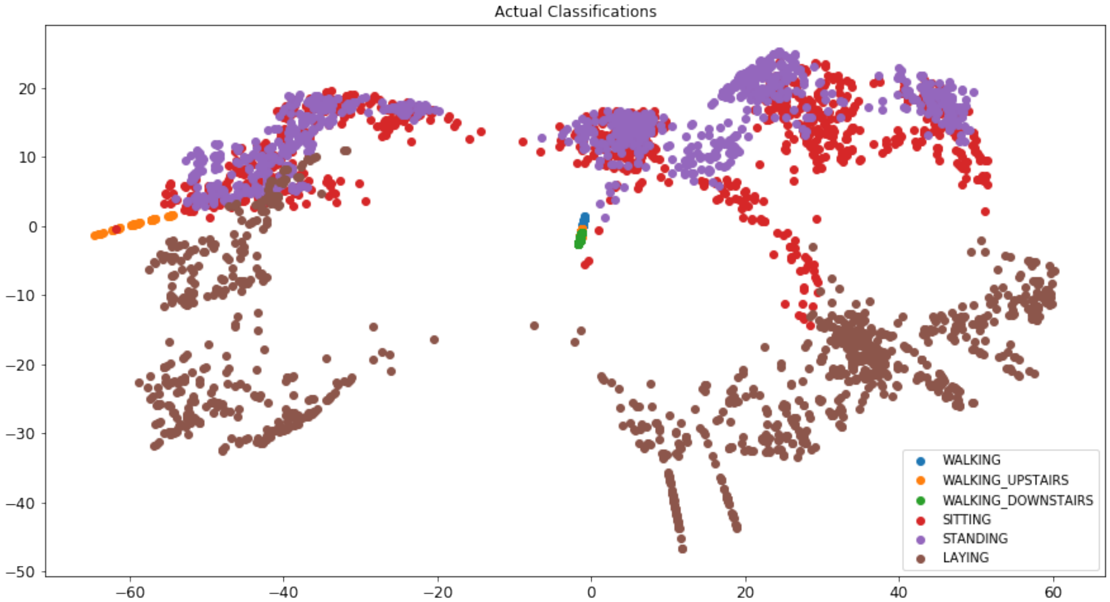


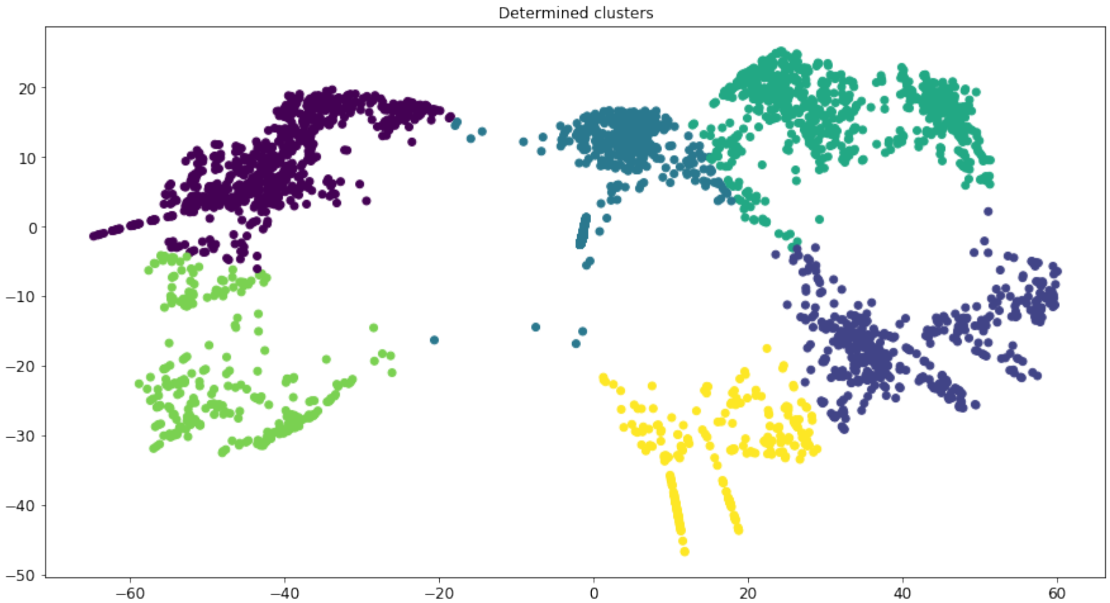
2.0.4 IsoMap

[31]:
```
""" TODO
Reduce the full feature space (i.e. all 561 features) down to
2 features (i.e n_components) using Isomap. Also, make sure to
determine a goodchoice for the number of neighbors.

Display the classes in the new feature space.

Then construct a KMeans model to locate a set of 6 clusters
in this new feature subspace. Display the determined clusters in
this new feature subspace.
"""
# TODO: Create the Isomap object and transform the training data
isomap2 = Isomap(n_neighbors= 5, n_components= 2, eigen_solver='dense',␣
 ↪n_jobs=-1)
```

```python
Xmap2 = isomap2.fit_transform(Xtrain)# TODO: transform the training data

# TODO: Plot actual classifications in the new feature space
fig = plt.figure(figsize=(15,8))
ax0 = fig.add_subplot(111)
for i, name in enumerate(activity_names):
    # Mask of examples belonging to the current class
    inds = ytrain == i
    _x= Xmap2[inds,0]
    _y= Xmap2[inds,1]
    # TODO: use scatter to plot the selected examples in the isomap
    # subspace. set the label to the class name
    # see the API pages for matplotlib scatter
    ax0.scatter(_x, _y, label=name)

ax0.set_title('Actual Classifications')
ax0.legend()

# TODO: Construct a KMeans Model. fit it to the Isomap features
iso2_model = KMeans(n_clusters=6)# TODO: create and fit the model
# TODO: determine the cluster groupings using predict
pred = iso2_model.fit_predict(Xmap2)# TODO



# TODO: Plot determined predicted clusters
fig = plt.figure(figsize=(15,8))
ax0 = fig.add_subplot(111)
# TODO: use scatter to plot all the examples in the isomap subspace.
# do NOT set the label, instead set the parameter c to the predicted clusters
# see the API pages for matplotlib scatter
ax0.scatter(Xmap2[:,0], Xmap2[:,1], c=pred)
ax0.set_title('Determined clusters')

# Sum of squared distances of samples to their closest cluster center
iso2_model.inertia_
```

[31]: 487277.45912254526

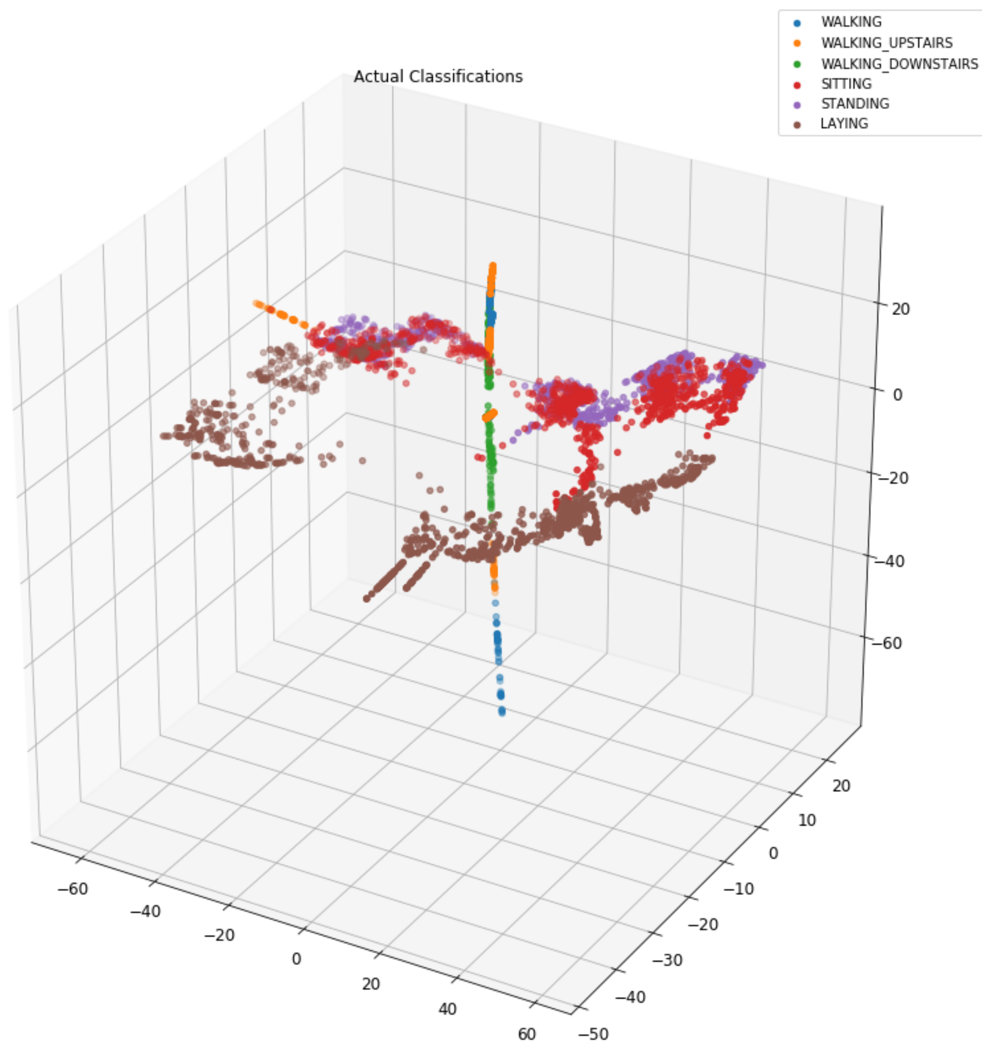


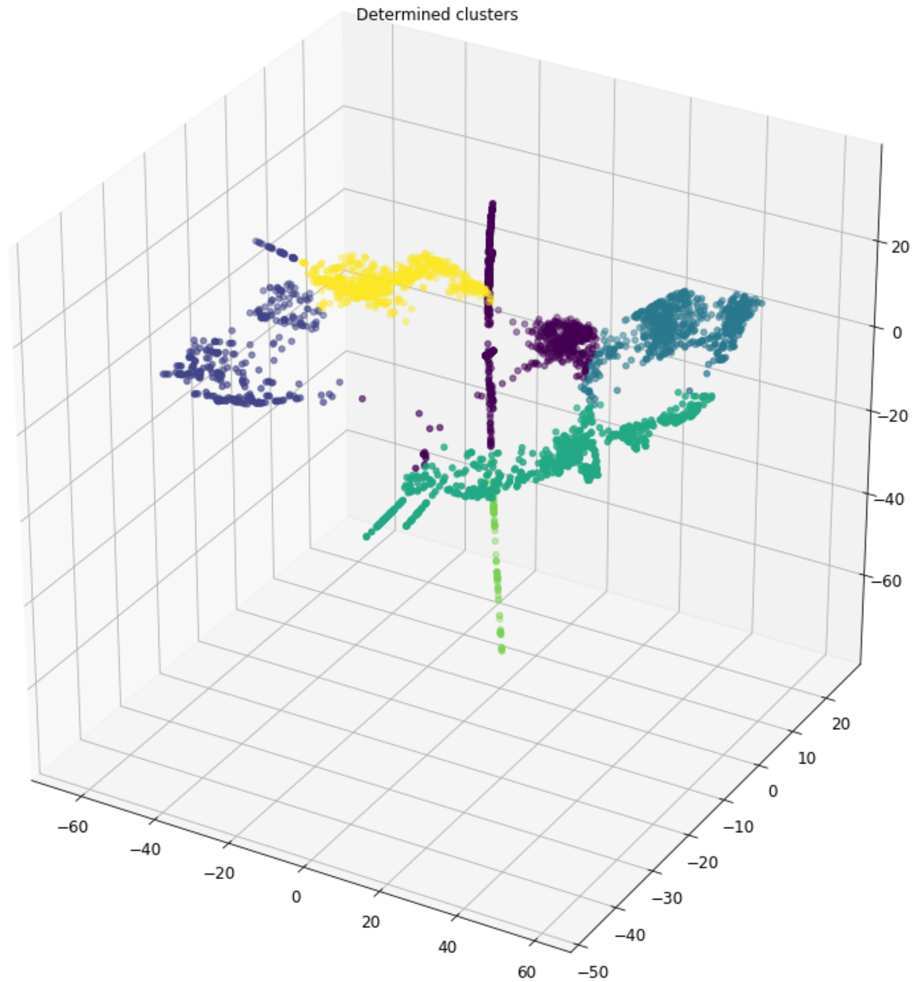


[45]:
```
""" TODO
Reduce the full feature space (i.e. all 561 features) down to
3 features using Isomap. Also, make sure to determine a good
choice for the number of neighbors.

Display the classes in the new feature space.
```

```
Then construct a KMeans model to locate a set of 6 clusters
in this new feature space. Display the determined clusters in
this new feature space.
"""
# TODO: Create the Isomap object and transform the training data
isomap3 = Isomap(n_neighbors= 5, n_components= 3, eigen_solver='dense',␣
 ↪n_jobs=-1)# TODO
Xmap3 = isomap3.fit_transform(Xtrain)# TODO

# TODO: Plot actual classifications in the new feature space
fig = plt.figure(figsize=(15,15))
ax0 = fig.add_subplot(111, projection='3d')
for i, name in enumerate(activity_names):
    # Mask of examples belonging to the current class
    inds = ytrain == i
    # TODO: use scatter to plot the selected examples in the isomap
    # subspace. set the label to the class name
    # see the API pages for matplotlib scatter
    ax0.scatter(Xmap3[inds, 0], Xmap3[inds,1],Xmap3[inds,2], label=name)


ax0.set_title('Actual Classifications')
ax0.legend()


# TODO: Construct a KMeans Model
iso3_model = KMeans(n_clusters=6)# TODO
# TODO: determine the cluster groupings
pred = iso3_model.fit_predict(Xmap3)# TODO



# TODO: Plot determined clusters
fig = plt.figure(figsize=(15,15))
ax0 = fig.add_subplot(111, projection='3d')
# TODO: use scatter to plot all the examples in the isomap subspace.
# do NOT set the label, instead set the parameter c to the predicted clusters
ax0.scatter(Xmap3[:,0], Xmap3[:,1],Xmap3[:,2], c=pred)
ax0.set_title('Determined clusters')

iso3_model.inertia_
```

[45]: 922735.6801754719


Actual Classifications
WALKING
WALKING_UPSTAIRS
WALKING_DOWNSTAIRS
SITTING
STANDING
LAYING
20
0
−20
−40
−60
20
10
0
−10
−20
−30
−40
−50
−60
−40
−20
0
20
40
60

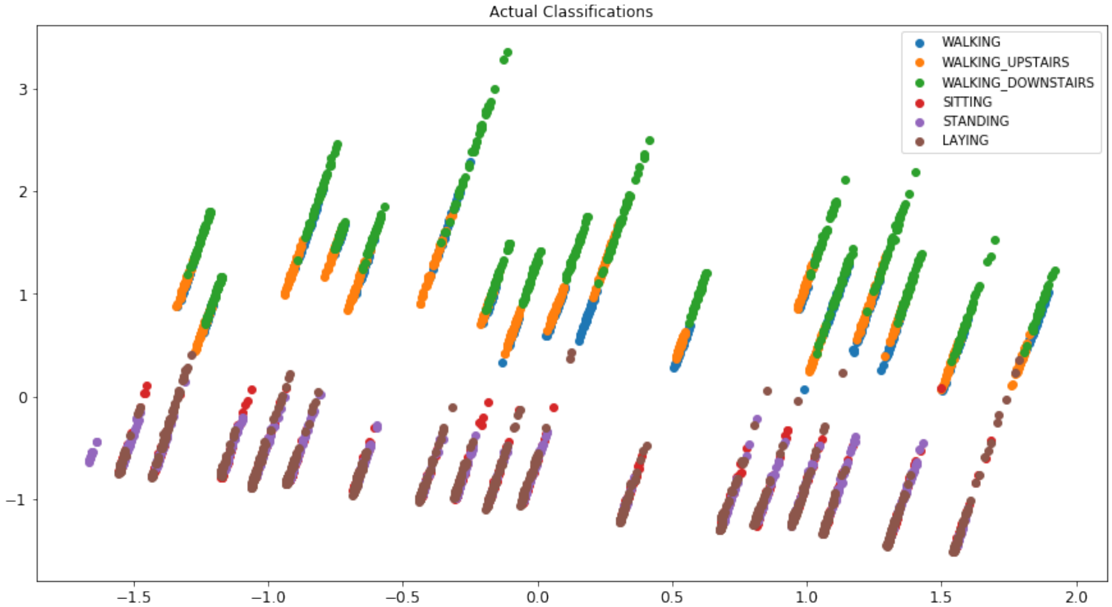


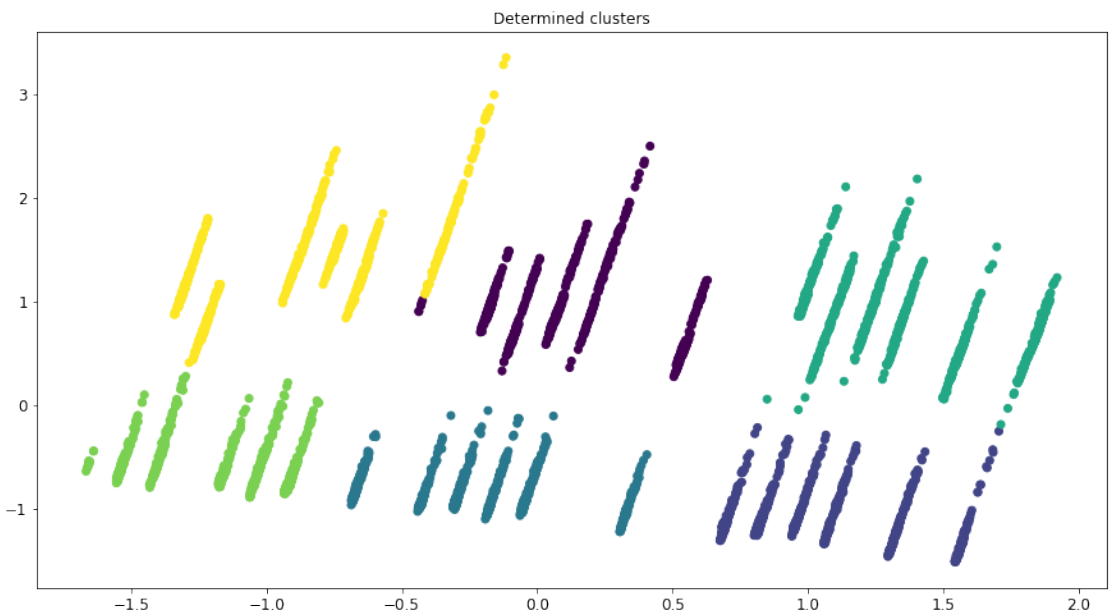
2.0.5 PCA

[39]:
```
""" TODO
Reduce the full feature space (i.e. all 561 features) down to
2 features using PCA. Also, set whiten to True.

Display the classes in the new feature space.

Then construct a KMeans model to locate a set of 6 clusters
in this new feature space. Display the determined clusters in
this new feature space.
"""
# TODO: Create the PCA object and transform the training data
```

```
pca2 = PCA(n_components= 2, whiten=True)# TODO
Xpca2 = pca2.fit_transform(Xtrain)# TODO

# TODO: Plot actual classifications in the new feature space
fig = plt.figure(figsize=(15,8))
ax0 = fig.add_subplot(111)
for i, name in enumerate(activity_names):
    # Mask of examples belonging to the current class
    inds = ytrain == i
    ax0.scatter(Xpca2[inds, 0], Xpca2[inds, 1], label=name)
    # TODO: use scatter to plot the selected examples in the PCA
    # subspace. set the label to the class name
    # see the API pages for matplotlib scatter

ax0.set_title('Actual Classifications')
ax0.legend()




# TODO: Construct a KMeans Model. fit the model to the PCA features
pca2_model = KMeans(n_clusters=6)# TODO
# TODO: determine the cluster groupings
pred = pca2_model.fit_predict(Xpca2)# TODO

# TODO: Plot determined clusters
fig = plt.figure(figsize=(15,8))
ax0 = fig.add_subplot(111)
# TODO: use scatter to plot all the examples in the isomap subspace.
# do not set the label, instead set the parameter c to the predicted clusters
# see the API pages for matplotlib scatter
ax0.scatter(Xpca2[:,0], Xpca2[:,1], c=pred)
ax0.set_title('Determined clusters')

pca2_model.inertia_
```

[39]: 1028.2883720175291

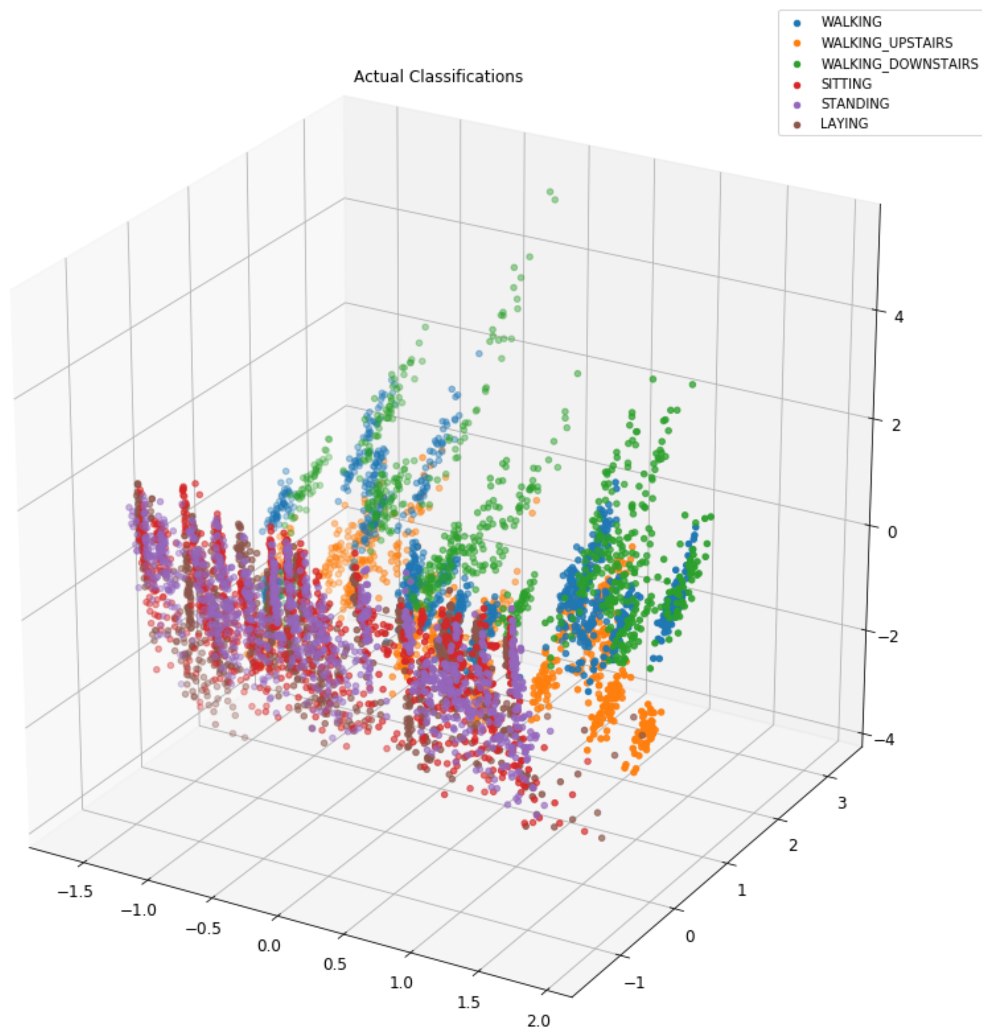




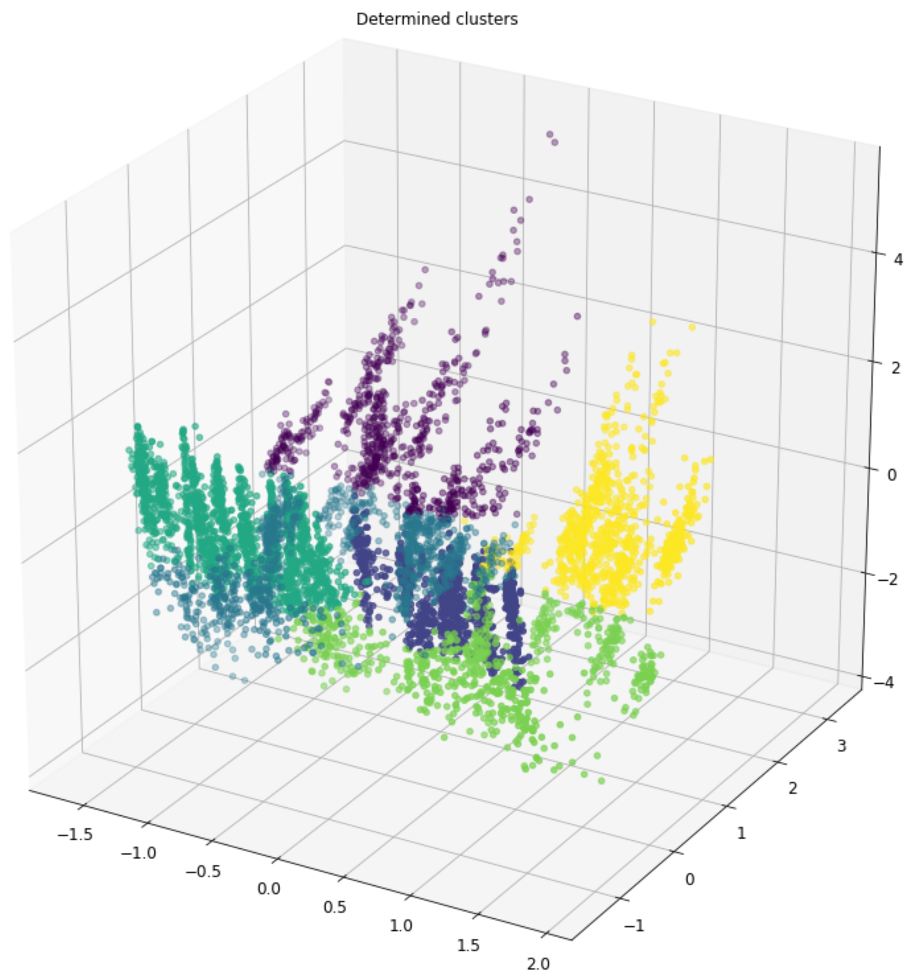
[43]:
```
""" TODO
Reduce the full feature space (i.e. all 561 features) down to
3 features using PCA. Also, set whiten to True.

Display the classes in the new feature space.

Then construct a KMeans model to locate a set of 6 clusters
```

```
in this new feature space. Display the determined clusters in
this new feature space.
"""
# TODO: Create the PCA object and transform the training data
pca3 = PCA(n_components= 3, whiten=True)# TODO
Xpca3 = pca3.fit_transform(Xtrain)# TODO

# TODO: Plot actual classifications in the new feature space
elev = 25
angle = 300
fig = plt.figure(figsize=(15,15))
ax0 = fig.add_subplot(111, projection='3d')
for i, name in enumerate(activity_names):
    # Mask of examples belonging to the current class
    inds = ytrain == i
    ax0.scatter(Xpca3[inds, 0], Xpca3[inds, 1], Xpca3[inds,2], label=name)
    # TODO: use scatter to plot the selected examples in the PCA
    # subspace. set the label to the class name
    # see the API pages for matplotlib scatter

ax0.view_init(elev, angle)
ax0.set_title('Actual Classifications')
ax0.legend()


# TODO: Construct a KMeans Model. fit the model on the PCA features
pca3_model = KMeans(n_clusters=6)# TODO
# TODO: determine the cluster groupings
pred = pca3_model.fit_predict(Xpca3)# TODO

# TODO: Plot determined clusters
fig = plt.figure(figsize=(15,15))
ax0 = fig.add_subplot(111, projection='3d')
# TODO: use scatter to plot all the examples in the isomap subspace.
# do not set the label, instead set the parameter c to the predicted clusters
# see the API pages for matplotlib scatter
ax0.scatter(Xpca3[:,0], Xpca3[:,1], Xpca3[:,2], c=pred)
ax0.view_init(elev, angle)
ax0.set_title('Determined clusters')

pca3_model.inertia_
```

[43]: 4772.366174411744



Actual Classifications
WALKING
WALKING_UPSTAIRS
WALKING_DOWNSTAIRS
SITTING
STANDING
LAYING
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
−1
0
1
2
3
−4
−2
0
2
4



[ ]:

```

```

3 TASK 2 DATASETS: SYNTHETIC DATA

3.0.1 D31

[ ]:

```python
""" PROVIDED
Load the dataset
"""
D31 = pd.read_csv('synthetic/D31.txt', sep='\s+', header=None)
D31.columns = ['x', 'y', 'cluster']
D31['cluster'] = D31['cluster'] - 1
```

```
D31.shape
```

```
[ ]: # Display first few examples
D31.head()
```

```
[ ]: """ TODO
Display class counts using pd.value_counts() on the clusters column
"""
d31_cnt = # TODO
d31_cnt
```

```
[ ]: """ TODO
Display the actual classifications and the predicted clusters
"""
# TODO: Plot true classifications. use group_scatter_plot.
# the feature names are ['x', 'y']. the label names are d31_cnt.index


# TODO: Determine a set of clusters using KMeans




# TODO: Plot the determined clusters. use group_scatter_plot.



model.inertia_
```

```
[ ]:
```

3.0.2 AGGREGATION

```
[ ]: """ PROVIDED
Load the dataset
"""
Aggregation = pd.read_csv('synthetic/Aggregation.txt', sep='\s+', header=None)
Aggregation.columns = ['x', 'y', 'cluster']
Aggregation['cluster'] = Aggregation['cluster'] - 1
Aggregation.shape
```

```
[ ]: # Display first few examples
Aggregation.head()
```

```
[ ]: """ TODO
Display class counts
"""
agg_cnt = # TODO
```

```
agg_cnt
```

[ ]:
```
""" TODO
Display the actual and predicted clusters
"""
# TODO: Plot true classifications. use group_scatter_plot.
# the feature names are ['x', 'y']. the label names are agg_cnt.index



# TODO: Determine clusters using KMeans



# TODO: Plot the determined clusters. use group_scatter_plot.


model.inertia_
```

[ ]:
```

```

3.0.3 R15

[ ]:
```
""" PROVIDED
Load the dataset
"""
R15 = pd.read_csv('synthetic/R15.txt', sep='\s+', header=None)
R15.columns = ['x', 'y', 'cluster']
R15['cluster'] = R15['cluster'] - 1
R15.shape
```

[ ]:
```
# Display first few examples
R15.head()
```

[ ]:
```
""" TODO
Display class counts
"""
r15_cnt = # TODO
r15_cnt
```

[ ]:
```
"""
Display the actual and predicted clusters
"""
# TODO: Plot true classifications. use group_scatter_plot.
# the feature names are ['x', 'y']. the label names are r15_cnt.index
```

```
# TODO: Determine clusters using KMeans



# TODO: Plot the determined clusters. use group_scatter_plot.


model.inertia_
```

[ ]:

```

```

4 DISCUSSION

For which ever task you selected, answer the following question:

In several paragraphs describe the original clusters and compare them to the clusters learned by the KMeans model. What are the limitations or issues with the learned clusters? Please be clear and concise in your response.

The original data space is quite noisy and random, which affect the performance of KMeans quite a bit. When we try to use KMeans to separate them out with six classes, it may encounter some problems.

The original data without dimentionality reduction learned by clusters show large inertion which indicates the sum of distance to their nearest nodes. When applying dimentionality reduction, Isomap even gives worse result, meaning that the dataset is not clear showing manifolds features. But with PCA, the inertial reduced to 4772, almost 1/100 of the original model, indicating the benifits of reducing 100 features into 3.