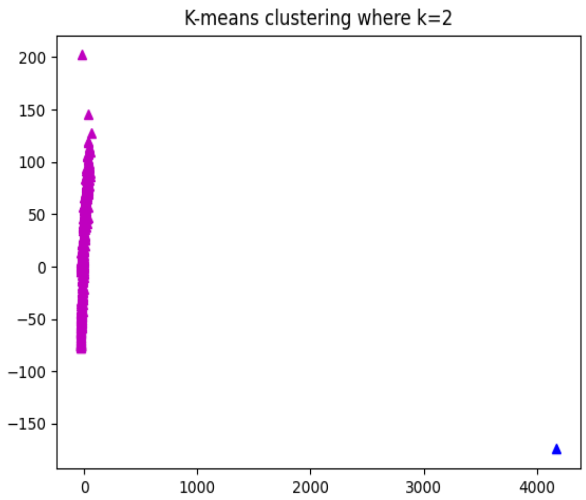
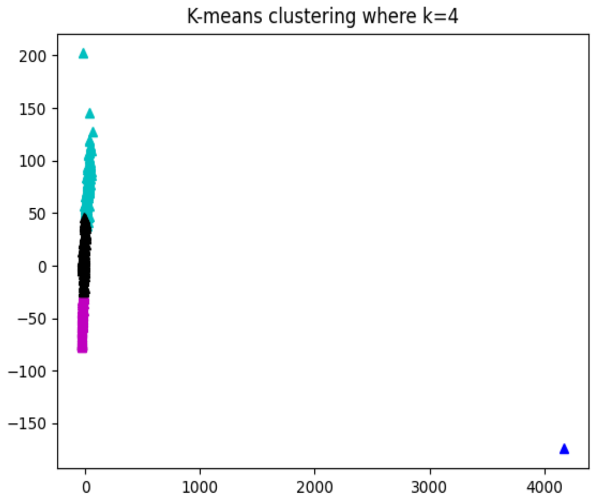
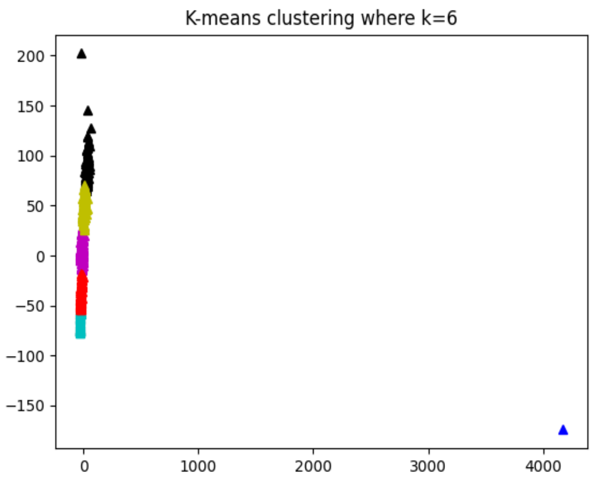
GE 461 Introduction to Data Science
Project W13: Telehealth - Fall Detection

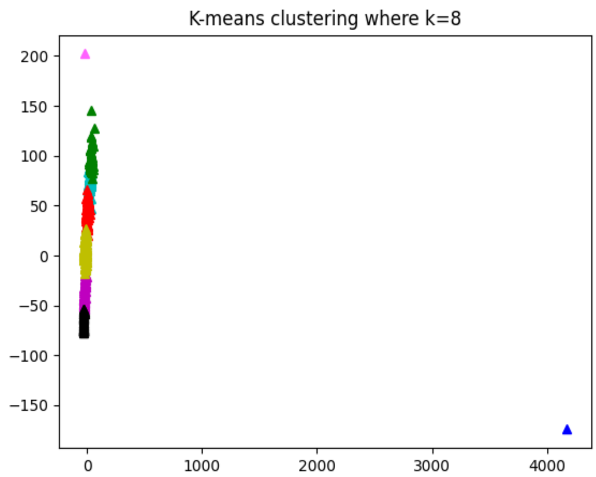
İlknur Baş
21601847

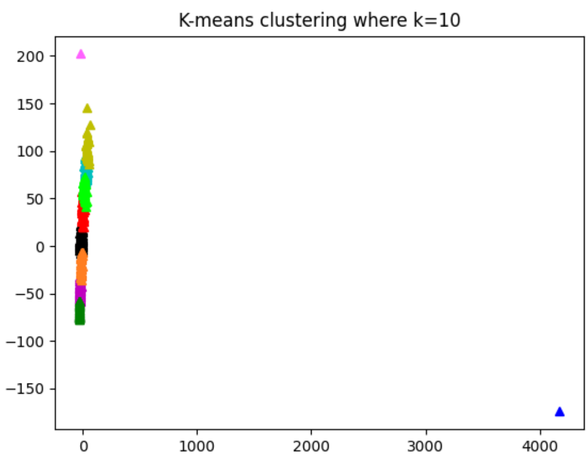
PART A)











GE 461 Introduction to Data Science
Project W13: Telehealth - Fall Detection
İlknur Baş
21601847

Variances of top two PCs (also can be seen as an output from the .py file) are as follows: 0.75307248, 0.0851159.

In the figures above, clusters where n=2,4,6,8,10 are represented. In general, 5 figures are very similar with each other. Changing n did not change anything in terms of clustering. The reason could be as follows: In the assignment it is given that we have 2 actions, fall action (F) and non-fall action (NF). For that reason, making n larger than 2 did not do any difference.

As a common thing, there is a sample in the lower right corner and also in the upper left corner which is away from the location that the other data generally are located. To consider the 1st figure specifically, we cannot say that clustering the data is worked. Because we cannot clearly see whether 2 actions are split. Samples are seemed to overlap with each other. The reason could be related to the data that is in the lower right corner or in the upper left corner. Hence, k-means clustering is failed.

PART B)

MLP a multi-layer perceptron classifier with different hyperparameters and their results

- MLPClassifier(hidden_layer_sizes=100, solver='lbfgs',alpha=0.0001, learning_rate= 'constant', learning_rate_init=0.0001,max_iter=200,random_state=None)

MLP accuracy ratio for validation: 1.0
MLP accuracy ratio for test: 0.9411764705882353

- MLPClassifier(hidden_layer_sizes=100,solver='lbfgs',alpha=0.0007, learning_rate='constant', learning_rate_init=0.001,max_iter=200,random_state=None)

MLP accuracy ratio for validation: 0.9882352941176471
MLP accuracy ratio for test: 1.0

- MLPClassifier(hidden_layer_sizes=100,solver='sgd',alpha=0.0007, learning_rate='constant',learning_rate_init=0.001,max_iter=200,random_state=None,)

MLP accuracy ratio for validation: 0.9764705882352941
MLP accuracy ratio for test: 0.9882352941176471

- MLPClassifier(hidden_layer_sizes=100,solver='sgd',alpha=0.0001, learning_rate='constant',learning_rate_init=0.0001,max_iter=200,random_state=None,)

MLP accuracy ratio for validation: 1.0
MLP accuracy ratio for test: 0.9882352941176471

GE 461 Introduction to Data Science
Project W13: Telehealth - Fall Detection
İlknur Baş
21601847

- MLPClassifier(hidden_layer_sizes=100,solver='lbfgs',alpha=0.001, learning_rate='constant',learning_rate_init=0.1,max_iter=200,random_state=None,)

MLP accuracy ratio for validation: 0.9647058823529412
MLP accuracy ratio for test: 0.9764705882352941

- MLPClassifier(hidden_layer_sizes=100,solver='lbfgs',alpha=0.001,
learning_rate='constant',learning_rate_init=0.0000001,max_iter=200,
random_state=None)

MLP accuracy ratio for validation: 1.0
MLP accuracy ratio for test: 0.9647058823529412

SVM a support-vector-machine classifier with different hyperparameters and their results

- SVC (C=1.0, kernel='rbf', degree=5, gamma='scale', shrinking=True, tol=0.001, max_iter=100, random_state=None)

SVM accuracy ratio for validation: 1.0
SVM accuracy ratio for test: 0.9764705882352941

- SVC(C=1.0,kernel='rbf',degree=50,gamma='scale',shrinking=True,tol=0.005,max_iter=100 , random_state=None)

SVM accuracy ratio for validation: 1.0
SVM accuracy ratio for test: 0.9882352941176471

- SVC(C=1.0,kernel='rbf',degree=50,gamma='scale',shrinking=True,tol=0.005,max_iter=900 ,random_state=None)

SVM accuracy ratio for validation: 0.9882352941176471
SVM accuracy ratio for test: 0.9882352941176471

- SVC(C=1.0,kernel='rbf',degree=50,gamma='scale',shrinking=True,tol=0.005,max_iter=100 0, random_state=None)

SVM accuracy ratio for validation: 1.0
SVM accuracy ratio for test: 0.9882352941176471

- SVC(C=1.0, kernel='rbf', degree=200,gamma='scale',shrinking=True,tol=0.005, max_iter=1000 ,random_state=None)

GE 461 Introduction to Data Science
Project W13: Telehealth - Fall Detection
İlknur Baş
21601847

SVM accuracy ratio for validation: 1.0
SVM accuracy ratio for test: 0.9647058823529412

The change in the parameters did not make much difference in their accuracy ratio. The ratio came up larger than approximately 0.95 for both SVM and MLP in all cases that I have tried. Hence we can say that both SVM and MLP gave succesfull results.

APPENDIX

PYTHON CODE:

```
import csv
from warnings import simplefilter
import numpy as np
import sklearn.decomposition
import numpy
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from random import shuffle
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC

# REFERENCES
# [1] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
# [2] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
# [3] https://scikit-learn.org/stable/modules/neural_networks_supervised.html
# [4] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

data_array = []
with open('falldetection_dataset.csv', 'r') as csv_file:
    csv_file = csv.reader(csv_file)
    for row in csv_file:
        data_array.append(row)

# converting numpy.array for consistency.
data_array = np.array(data_array)

# According to our dataset, features are starting from 3rd column.
features = data_array[:, 2:]
# for feature in features:
#    print(feature)
```

GE 461 Introduction to Data Science
Project W13: Telehealth - Fall Detection
İlknur Baş
21601847

```
# In 2nd column actions labels are present.
labels = data_array[:, 1]
# for label in labels:
#     print(label)

# principal components analysis (PCA) is performed on features array to extract top 2 PCAs.
pca = sklearn.decomposition.PCA(n_components=2)
pca_fit_transform = pca.fit(features).transform(features)

# Variance of top 2 PCAs.
# If n_components is not set then all components are stored and the sum of the ratios is
equal to 1.0. REF:[1]
variance = pca.explained_variance_ratio_
print("Variance:", variance)

# CLUSTERING STARTS

cluster_number = [2, 4, 6, 8, 10]
colors = ['m^', 'b^', 'c^', 'k^', 'r^', 'y^', 'g^']
new_color = ['#FF62FF', '#FF7E21', '#00FF00']
for i in cluster_number:

    # clustering where k=2,4,6,8,10 is performed.
    k_means = KMeans(n_clusters=i)
    # Compute k-means clustering for all features in the array. REF:[2]
    k_means.fit(pca_fit_transform)

    # According to my observation, k_means.labels_' values consist of numbers less than the
number of cluster,
    # which makes sense.
    k_means.labels_
    print("k_means.labels_:", k_means.labels_)
    print("pca_fit_transform:", pca_fit_transform)

    for k in range(0, i):
        # print("k:", k)
        x1 = pca_fit_transform[k_means.labels_ == k][:, 0]  # k=1,2,...i-1 for 1 case
        y1 = pca_fit_transform[k_means.labels_ == k][:, 1]
        # For the cluster where n_cluster=2, blue is fall action (F), purple is non-fall action (NF)
        # There is no such thing for where n_cluster > 2
        if k > 6:
            plt.plot(x1, y1, color=new_color[k - 7], marker='^')
        else:
            plt.plot(x1, y1, colors[k])
```

```
    plt.title("K-means clustering where k=" + str(i))
    plt.show()

# PART B

# Find the NF and F data, and their lengths
f = np.empty((0, 308))
nf = np.empty((0, 308))

for n in range(0, 566):
    if data_array[n, 1] == 'NF':
        nf = numpy.append(nf, data_array[n, :].reshape(1, 308), axis=0)
    elif data_array[n, 1] == 'F':
        f = numpy.append(f, data_array[n, :].reshape(1, 308), axis=0)

print("nf,size:", len(nf))  # 253 rows(data) 308 columns
print("f,size:", f.shape)  # 313 rows(data) 308 columns

# According to the example given in the assignment, training/validation/testing sets
# will(could) have the following ratio: 70%, 15%,15%.
# Hence, for NF: training:177, test:38, valid:38
# for NF: training:219, test:47, valid:47 number of data will be involved.

# 70 15 15 for NF
shuffle(nf)
nf_train, nf_validate, nf_test = np.split(nf, [int(.7 * len(nf)), int(.85 * len(nf))])
print("NF: train:", nf_train)
print("NF: validate:", nf_validate)
print("NF: test:", nf_test)
print("NF: Length train:", len(nf_train))
print("NF: Length validate:", len(nf_validate))

nf_train_label = nf_train[:, 1]
nf_train_data = nf_train[:, 2:]
nf_validate_label = nf_validate[:, 1]
nf_validate_data = nf_validate[:, 2:]
nf_test_label = nf_test[:, 1]
nf_test_data = nf_test[:, 2:]

# 70 15 15 for F
shuffle(f)
f_train, f_validate, f_test = np.split(f, [int(.7 * len(f)), int(.85 * len(f))])
print("F: train:", f_train)
print("F: validate:", f_validate)
print("F: test:", f_test)
```

```
print("F: Length train:", len(f_train))
print("F: Length validate:", len(f_validate))

f_train_label = f_train[:, 1]
f_train_data = f_train[:, 2:]
f_validate_label = f_validate[:, 1]
f_validate_data = f_validate[:, 2:]
f_test_label = f_test[:, 1]
f_test_data = f_test[:, 2:]

# now combine them all
train_label = numpy.append(f_train_label, nf_train_label)
train_data = numpy.append(f_train_data, nf_train_data, axis=0)
validate_label = numpy.append(f_validate_label, nf_validate_label)
validate_data = numpy.append(f_validate_data, nf_validate_data, axis=0)
test_label = numpy.append(f_test_label, nf_test_label)
test_data = numpy.append(f_test_data, nf_test_data, axis=0)

simplefilter(action='ignore', category=FutureWarning)

# MLP multi-layer perceptron classifier is performed. REF: [3] is used for that part,
specifically in order to have a
# better understanding of hyperparameters in MLPClassifier.

# hidden_layer_sizes=100, activation='relu', *, solver='adam', alpha=0.0001,
batch_size='auto',
# learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200,
shuffle=True,
# random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9,
nesterovs_momentum=True,
# early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08,
n_iter_no_change=10,
# max_fun=15000

mlp_classifier = MLPClassifier(
   hidden_layer_sizes=100,
   solver='lbfgs',
   alpha=0.001,
   learning_rate='constant',
   learning_rate_init=0.0000001,
   max_iter=200,
   random_state=None,
)
mlp_classifier.fit(train_data, train_label)
```

```
count = 0
for i in range(0, 85):
    if mlp_classifier.predict(validate_data)[i] == validate_label[i]:
        count = count + 1
print("**count-mlp-validate ", count)
print("MLP accuracy ratio for validation: ", (count / float(len(validate_label))))

count = 0
for i in range(0, 85):
    if mlp_classifier.predict(test_data)[i] == test_label[i]:
        count = count + 1
print("**count-mlp-test: ", count)
print("MLP accuracy ratio for test: ", (count / float(len(test_label))))

print("-------------------------- ")

# SVM a support-vector machine is performed. REF: [4] is used for that part, specifically in order to have a
# better understanding of hyperparameters in SVM.

# (*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=- 1, decision_function_shape='ovr', break_ties=False,
# random_state=None)
svm = SVC(C=1.0,
          kernel='rbf',
          degree=200,
          gamma='scale',
          shrinking=True,
          tol=0.005,
          max_iter=1000,
          random_state=None)
svm.fit(train_data, train_label)

count = 0
for i in range(0, 85):
    if svm.predict(validate_data)[i] == validate_label[i]:
        count = count + 1
print("**count-svm-validate ", count)
print("SVM accuracy ratio for validation: ", (count / float(len(validate_label))))

count = 0
for i in range(0, 85):
    if svm.predict(test_data)[i] == test_label[i]:
```

```
    count = count + 1
print("**count-svm-test: ", count)
print("SVM accuracy ratio for test: ", (count / float(len(test_label))))
```