Ege Aktan
GE - 461
Project IV

Fall Detection Through PCA and Classifiers

We start by splitting our data into three division. 70% of the data stored in train set, 15% of the data into validation set and the remaining 15% into test set. Train set to be utilize for training, validation set to be utilized for parameter selection and test set to be used for retrieving accuracy of the model of our choice.

First method to implement is Support Vector Classifier (SVC) and we start by training in through the train set. First we will fix the degrees (d = 3), marginal parameter C=1 and test it for three different kernels which are polynomial, radial and sigmoid. The results obtained from validation set shows polynomial and Radial (changing between 98-100% regarding to the divisions of the data) kernels much more accurate than sigmoid (around 50%). Since polynomial and radial kernels provide similar results I tried increasing (decreasing C) and decreasing (increasing C) margins both of the kernels provide good results. Increasing degrees after d=3 of both kernels improving the model by a very small fraction (1.2%) therefore it may not be necessary to do so. Note that coefficient constant (coef0) is necessary for polynomial kernels the be non-zero entry. Accuracy of the polynomial model slightly (1.2%(improves as absolute value of coefficient constant increases.

As a result of SVM applications both radial and polynomial kernels provide accurate results through test set(97.6%). However, the accuracy of the selected models changes regarding to the training, validation and testing divisions. Indicating that there is a few outlier points in the dataset which are hard to identify.

Second method for us to implement is Multi Layer Perceptron. I fixed parameters of learning rate LR=0.001 since it was an effective initialization for catching local or global minimums from project 3. Additionally I fixed hidden layer units (H=100) and proceeded with testing different activation functions. As sigmoid has similar characteristics with tangent I picked sigmoid among the two. Sigmoid is also more effective to define gradients in the output. Similarly, Relu is a similar but better option compared to identity function. Thus, I started my comparison among sigmoid and rely activation functions only.

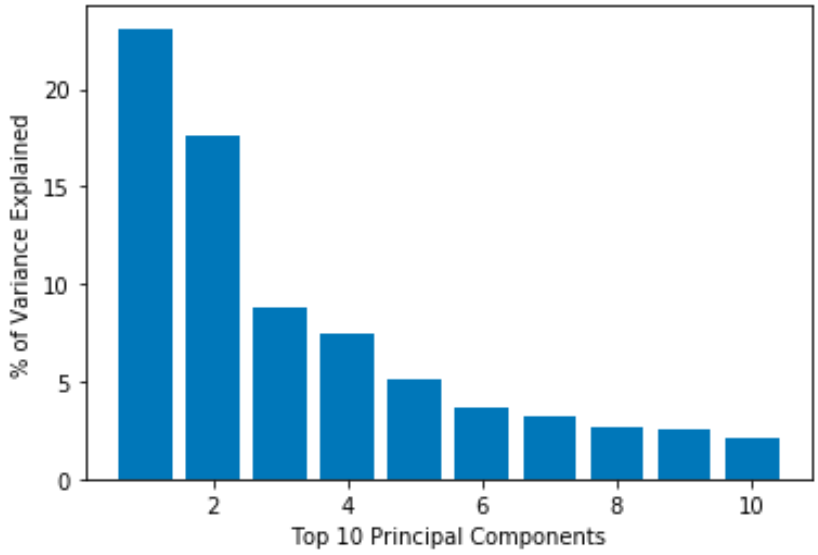
Both of them returned accurate results on the validation set (100%). However, Relu was a better choice for the sake of simplicity as the optimization converges at 144th iteration (sigmoid took more than 200). Thus, I started to simplify my model for efficiency by reducing parameters. Through the tests I made on validation set, model managed to obtain 100% accuracy by reducing H=5 and LR=0.01. Iteration count of the model also reduced to 70 after the new parameter selection and yet it retrieved 97-100% accuracy based on testing set. However, I spotted changes in the outputs regarding to the changing divisions of the sets (training, validation and testing.

Conclusion

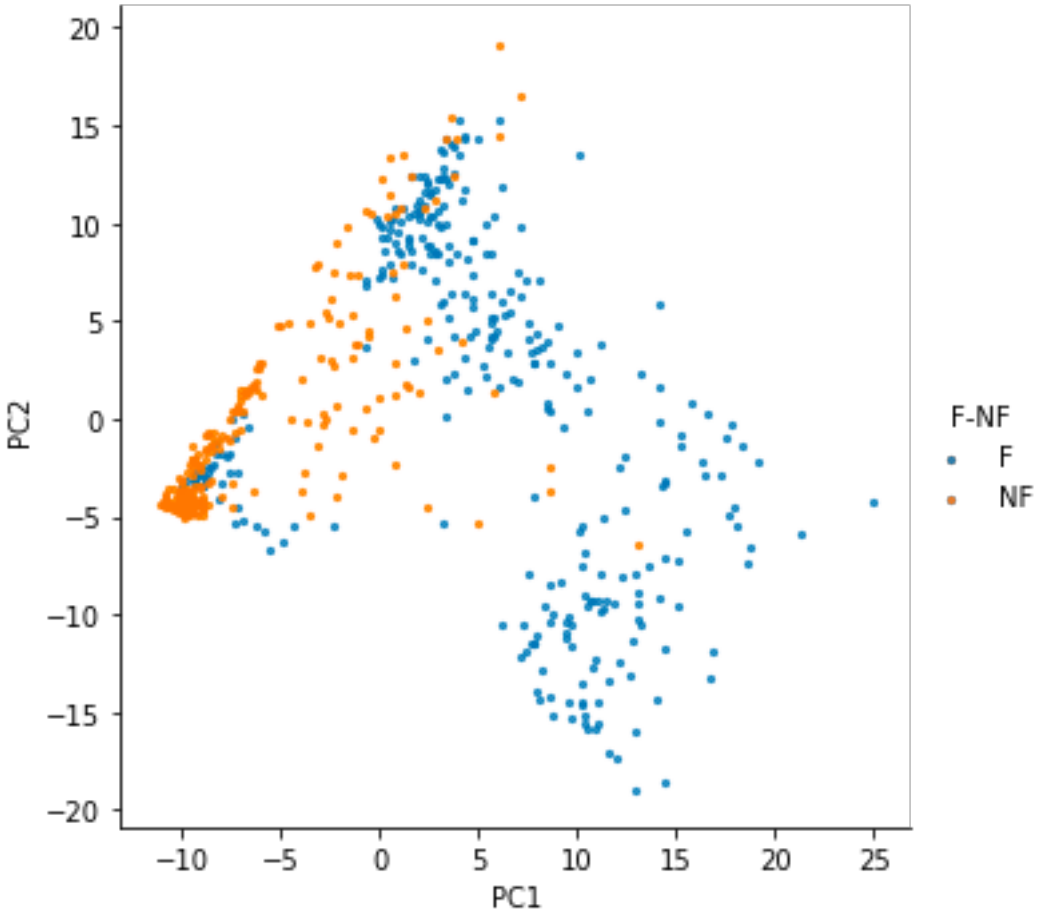
It is fair to conclude that, the data received from the wearable sensors are quite useful for detection of Falling and Not Falling actions. If the proper models utilized, both MLP and SVC retrieves at least 97.6% accuracy.

Ege Aktan
GE - 461
Project IV

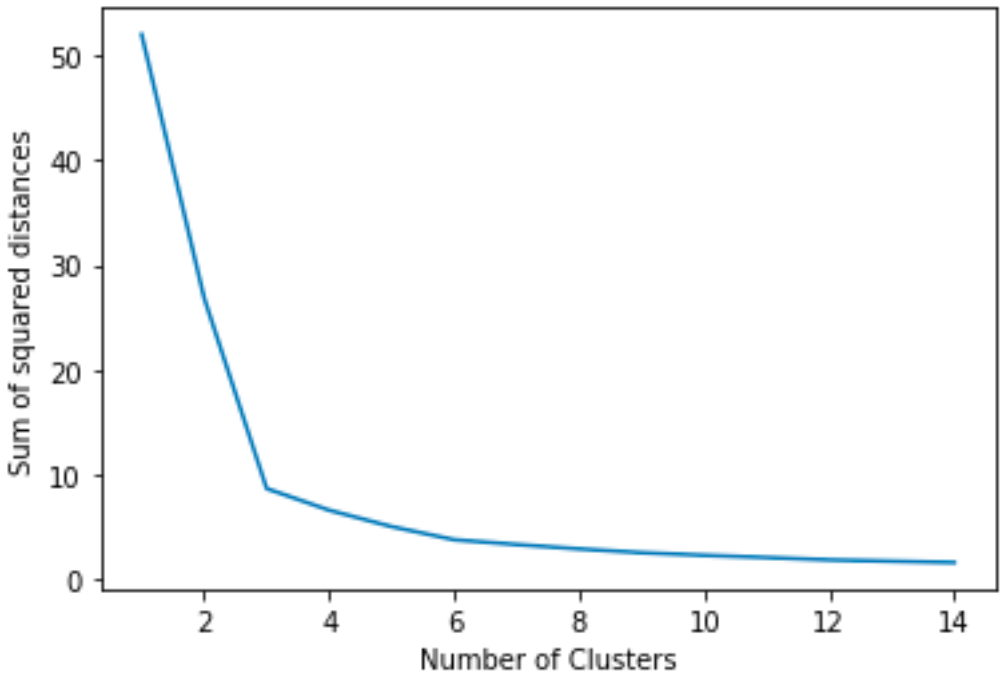
Appendix 1



Appendix 2



Appendix 3



Appendix 4 (Source Code)

```
#!/usr/bin/env python
# coding: utf-8

# In[140]:


import pandas as pd
import numpy as np


# In[141]:


from sklearn.decomposition import PCA


# In[142]:


import matplotlib.pyplot as plt
```

 # In[143]:

```
data = pd.read_csv('falldetection_dataset.csv', index_col=0, header = None )
```

 # In[144]:

```
data_m = np.matrix(data)
data.iloc[:,0:]
```

 # In[145]:

```
data_x = data.iloc[:,1:]
data_y = data.iloc[:,0]
```

 # In[146]:

```
#PART 1
```

 # In[147]:

```
#Standardizing the data and storing it in (scaled_data) to fit PCA function
from sklearn import preprocessing
scaled_data = preprocessing.scale(data_x)
pca = PCA()
pca.fit(scaled_data)
pca_data = pca.transform(scaled_data)
```

 # In[149]:

```
#Variance percentages of the features and sorted accordingly
perc_var = np.round(pca.explained_variance_ratio_* 100, decimals = 1 )
print(perc_var)
```

 # In[150]:

```
#firts two components explains 40.7% of the total variation
plt.bar(x=range(1, 11), height=perc_var[0:10])
plt.xlabel('Top 10 Principal Components')
plt.ylabel('% of Variance Explained')
plt.show()


# In[151]:


#Gathering the datas of the fitted principal components (PC1-PC2) and the response variable (F-NF)
y_df = pd.DataFrame(np.array(data_y), columns = None)
PCs_df = pd.DataFrame(pca_data, columns = None )
PCs_df = PCs_df.iloc[:,0:2]
PCs_all_df = pd.concat([PCs_df, y_df], axis = 1, ignore_index = True)
PCs_all_df.columns = ['PC1', 'PC2', 'F-NF']
print(PCs_all_df)


# In[152]:


import seaborn as sns
sns.lmplot('PC1', 'PC2', data = PCs_all_df, fit_reg=False,
        scatter_kws={"s":5},
        hue='F-NF')


# In[153]:


from sklearn.cluster import KMeans
#K-means clustering with n = 2
km = KMeans(n_clusters=2)

K = km.fit_predict(PCs_all_df[['PC1', 'PC2']])

PCs_all_df['cluster0'] = K


# In[154]:


#Visualizing the PC1 vs PC2 (colors indicating the clusters)
sns.lmplot('PC1', 'PC2', data = PCs_all_df, fit_reg = False,
```

```
        scatter_kws={'s':5},
        hue= 'cluster0')


# In[155]:


from sklearn.preprocessing import MinMaxScaler
#Scaling data to receive more accurate results
MMS = MinMaxScaler()
MMS.fit(PCs_all_df[['PC1']])
PCs_all_df[['PC1']] = MMS.transform(PCs_all_df[['PC1']])

MMS.fit(PCs_all_df[['PC2']])
PCs_all_df[['PC2']] = MMS.transform(PCs_all_df[['PC2']])


# In[156]:


km = KMeans(n_clusters=2)

K = km.fit_predict(PCs_all_df[['PC1', 'PC2']])

PCs_all_df['cluster2'] = K


# In[157]:


sns.lmplot('PC1', 'PC2', data = PCs_all_df, fit_reg = False,
        scatter_kws={'s':5},
        hue= 'cluster2')


# In[158]:


print(PCs_all_df.drop('cluster0', axis = 'columns'))


# In[160]:


#note that cluster indexes change every time runing the KMeans classifier so,
#data should be manually checked to adapt the if conditions according to the
#clusters of the majority (0 or 1)
```

```
cluster2 = PCs_all_df['cluster2']
F_NF = PCs_all_df['F-NF']
cnt = 0
positive = 0
for i in range(len(PCs_all_df)):
    if F_NF[i] =='F':
        if cluster2[i] == 1:
            positive += 1
    if F_NF[i] == 'NF':
        if cluster2[i] == 0:
            positive +=1
    cnt += 1
print("K-means(N=2) classification accuracy:")
print(positive/cnt)


# In[161]:


#retrieving Sum of square distance within
#the data points and the centroids for Ns in range(1,15)
k_rng = range(1,15)
sse_k = []
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(PCs_all_df[['PC1', 'PC2']])
    sse_k.append(km.inertia_)



# In[162]:


print(sse_k)


# In[205]:


plt.xlabel('Number of Clusters')
plt.ylabel('Sum of squared distances')
plt.plot(k_rng, sse_k)


# In[165]:
```

```
#Notice that elbow is at N=3 so we test K-means(N=3)
km = KMeans(n_clusters=3)
K = km.fit_predict(PCs_all_df[['PC1', 'PC2']])

PCs_all_df['cluster3'] = K


# In[166]:


sns.lmplot('PC1', 'PC2', data = PCs_all_df, fit_reg = False,
        scatter_kws={'s':5},
        hue= 'cluster3')


# In[169]:


#note that cluster indexes change every time runing the KMeans classifier so,
#data should be manually checked to adapt the if conditions according to the
#clusters of the majority (0 or 1)
cluster3 = PCs_all_df['cluster3']
F_NF = PCs_all_df['F-NF']
cnt = 0
positive = 0
for i in range(len(PCs_all_df)):
   if F_NF[i] =='F':
      if cluster3[i] == (0 or 2):
         positive += 1
   if F_NF[i] == 'NF':
      if cluster3[i] == 1:
         positive +=1
   cnt += 1
print("K-means(N=3) classification accuracy:")
print(positive/cnt)


# In[79]:


#We select KMeans(N=2) over KMeans(N=3) since the
#accuracy is 81.2%, 53% respectively.


# In[ ]:
```

#PART2

 # In[170]:

```
from sklearn.model_selection import train_test_split

#train and test-validation data split 70% and 30% respectively
X_train, X_tesVal, y_train, y_tesVal = train_test_split(data_x, data_y, test_size=0.3)

#Test and Validation Data split by half (originally 15% and 15%)
X_val, X_test, y_val, y_test = train_test_split(X_tesVal, y_tesVal, test_size=0.5)
```

 # In[171]:

```
from sklearn.svm import SVC
# Switching Kernels (linear, poly, sigmoid and rbf) will be experimented for accuracy
svc1 = SVC(kernel='poly', degree=3, coef0=5, C=1)
svc1.fit(X_train, y_train)
print(svc1.score(X_val, y_val))
```

 # In[172]:

```
svc2 = SVC(kernel='rbf', degree=3, C=1)
svc2.fit(X_train, y_train)
print(svc2.score(X_val, y_val))
```

 # In[173]:

```
svc3 = SVC(kernel='linear', degree=3, C=1)
svc3.fit(X_train, y_train)
print(svc3.score(X_val, y_val))
```

 # In[174]:

```
svc4 = SVC(kernel='sigmoid', degree=3, coef0=5, C=1)
svc4.fit(X_train, y_train)
```

```
print(svc4.score(X_val, y_val))


# In[175]:


#proceeding with polynomial and rbf kernels and decreasing margins (increasing C)


# In[176]:


svc1_1 = SVC(kernel='poly', degree=3, coef0=5, C=10)
svc1_1.fit(X_train, y_train)
print(svc1_1.score(X_val, y_val))


# In[177]:


svc2_1 = SVC(kernel='rbf', degree=3, C=10)
svc2_1.fit(X_train, y_train)
print(svc2_1.score(X_val, y_val))


# In[178]:


#Decreasing margin boundaries worsened the model with the polynomial kernel
#returns the same accuracy indicating that there will be no further improvements.


# In[179]:


#Increasing the margins for rbf (decreasing C)
svc2_1 = SVC(kernel='rbf', degree=3, C=0.1)
svc2_1.fit(X_train, y_train)
print(svc2_1.score(X_val, y_val))


# In[180]:


#Since it has worsened, we will experiment on rbf model while fixing C=1
```

 # In[181]:

```
svc2_1 = SVC(kernel='rbf', degree=3, C=10)
svc2_1.fit(X_train, y_train)
print(svc2_1.score(X_val, y_val))
```

 # In[182]:

```
#Increasing degree didn't make any difference accept the current model and test it.
#Retrieved 100% accuracy through validation
svc2_1 = SVC(kernel='rbf', degree=3, C=10)
svc2_1.fit(X_train, y_train)
print(svc2_1.score(X_test, y_test))
```

 # In[183]:

```
#Polynomial with constant coefficient also retrieved 100% accuracy from validation
svc1 = SVC(kernel='poly', degree=3, coef0=5, C=1)
svc1.fit(X_train, y_train)
print(svc1.score(X_test, y_test))
```

 # In[184]:

```
#The current model retrieves 97.6% accuracy but it can slightly and
#randomly change regarding to the train, validation and test divisions.
```

 # In[185]:

```
#MLP classifiers

from sklearn.neural_network import MLPClassifier
#Testing and comparing relu and sigmoid functions fixing other hyper parameters
```

 # In[186]:

```
MLP_r = MLPClassifier(hidden_layer_sizes=100, activation='relu', learning_rate_init=0.001)
```

```
MLP_r.fit(X_train, y_train)
print(MLP_r.score(X_val, y_val))


# In[187]:


MLP_s = MLPClassifier(hidden_layer_sizes=100, activation='logistic',
learning_rate_init=0.001)
MLP_s.fit(X_train, y_train)
print(MLP_s.score(X_val, y_val))


# In[188]:


#Both of the models retrieved 100% accuracy on the validation set.
#However sigmoid function is quite costly as 200 iterations is not enough
#to converge optimization.
#MLP_r is our current model
print("Iteration count of Relu MLP")
MLP_r.n_iter_


# In[189]:


#Dividing learning rate and hidden layer size parameters by 10
MLP_r1 = MLPClassifier(hidden_layer_sizes=10, activation='relu', learning_rate_init=0.01)
MLP_r1.fit(X_train, y_train)
print(MLP_r1.score(X_val, y_val))


# In[202]:


#Initializing hidden layer sizes parameter as 3
MLP_r2 = MLPClassifier(hidden_layer_sizes=5, activation='relu', learning_rate_init=0.01)
MLP_r2.fit(X_train, y_train)
print(MLP_r2.score(X_val, y_val))


# In[203]:


print("Iteration count of Relu MLP")
print(MLP_r2.n_iter_)
```

 # In[204]:

```
print("Test results for the selected Relu Model")
print(MLP_r2.score(X_test, y_test))
```