GE 461 Introduction to Data Science

Project 4 Report: Fall Detection

Fahad Waseem Butt

21801356

Spring 2022

TABLE OF CONTENTS

PART A:

In this part, first, libraries were imported, such as pandas, numpy, matplotlib.pyplot, and various packages from the scikit-learn toolbox [1]. After the dataset has been uploaded onto Google Colaboratory, the features and labels are separated. The features gathered from the dataset needed to be scaled, this is an important step wherein the features are made to resemble a standard normal distribution, which has a mean value of zero and standard deviation value of one [2]. For this purpose, "sklearn.preprocessing.StandardScaler" [3] was used. With this applied, the scaled features were then checked for the mean, which was 2.0204987260782545e-18, and the standard deviation, which was 1.0; these values are very close to the desired mean and standard deviation values, which were 0 and 1 respectively.

Principal Component Analysis (PCA) projects higher dimensional data onto a lower dimensional subspace, it is a method for reducing dimensionality in data. When data is projected onto a lower dimension, Principal Components (PCs) retain most of the variance, or information, in the data [4]. To achieve this, "sklearn.decomposition.PCA" [5] was used. After performing the computation to extract the top 2 PCs, as per the requirement, it was found that Principal Component 1 held 23.101268% and Principal Component 2 held 17.62439%. Overall 59.274342% of the original information was lost by using the top 2 PCs, however other PCs held significantly lower variance individually. The top 2 PCs have also been plotted as shown in Figure 1.

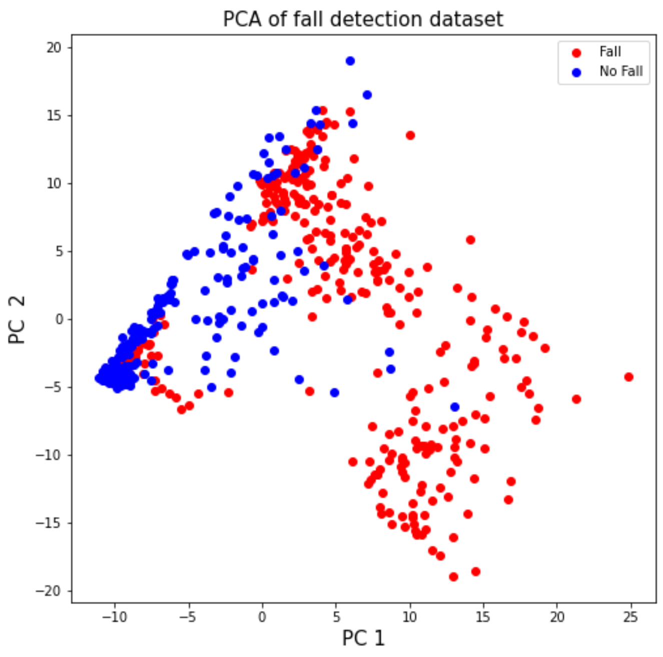

Figure 1: Scatter Plot for Top 2 Principal Components

The next step to this Exploratory Analysis was to separate the data into clusters, and this was done so by running k-means clustering with the use of "sklearn.cluster.KMeans" [6]. In k-means clustering, which is an unsupervised learning algorithm, the data is classified into a predetermined ‘k’ number of clusters of points that are closest to the optimally chosen centroids, or k centre points [7]. A different number of clusters were checked for in trial runs, and the number of clusters in each run were arbitrarily chosen to be the prime numbers 2, 3, 5, 7, 11, and 13. The results are plotted in Figure 2.

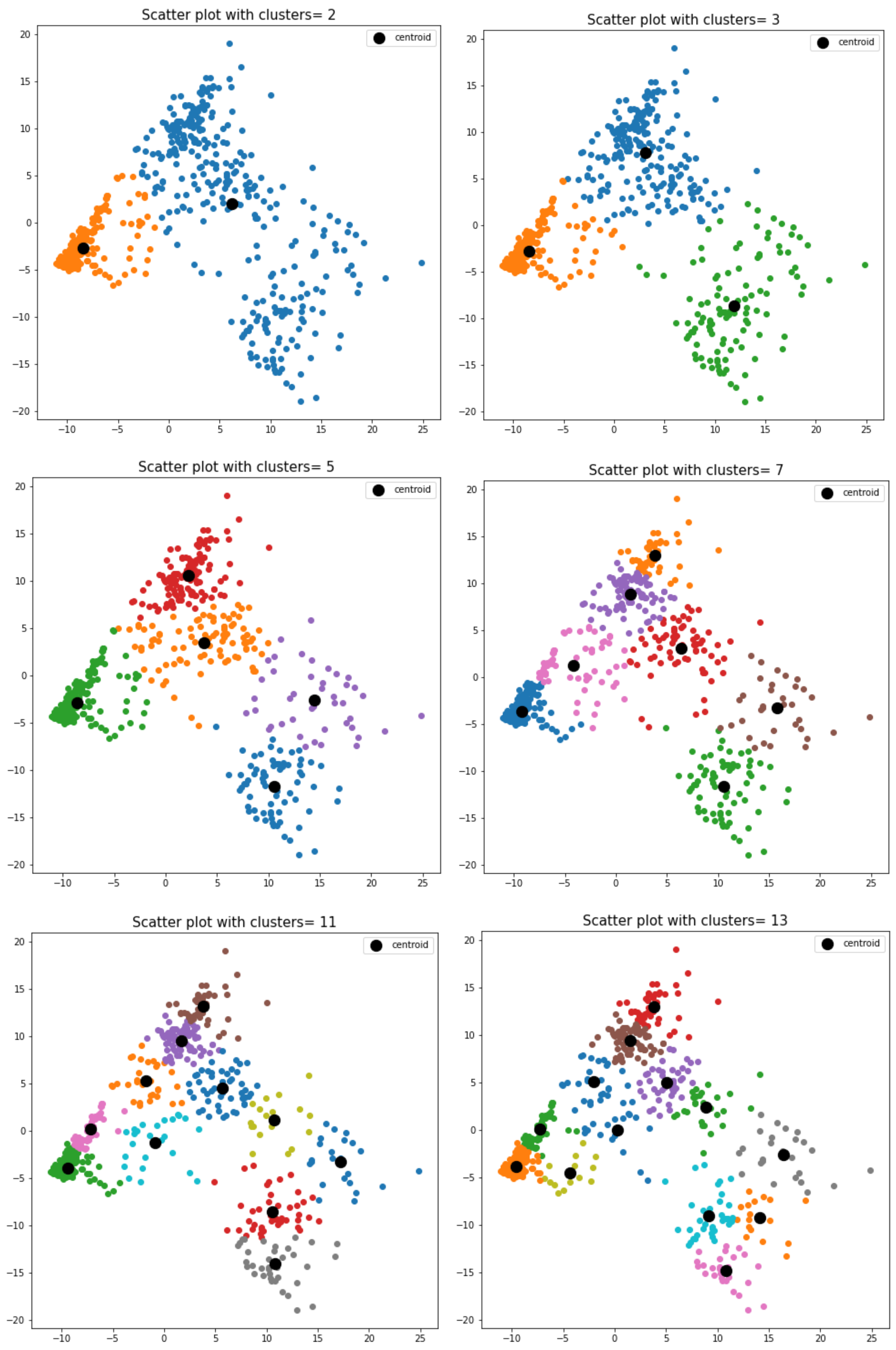

Figure 2: Scatter Plot after k-means Clustering with 2, 3, 5, 7, 11, and 13 Clusters.

To help assess these results for the different number of clusters, the Within Clusters Sum of Squares (WCSS) [7] is made use of and plotted as shown in Figure 3.

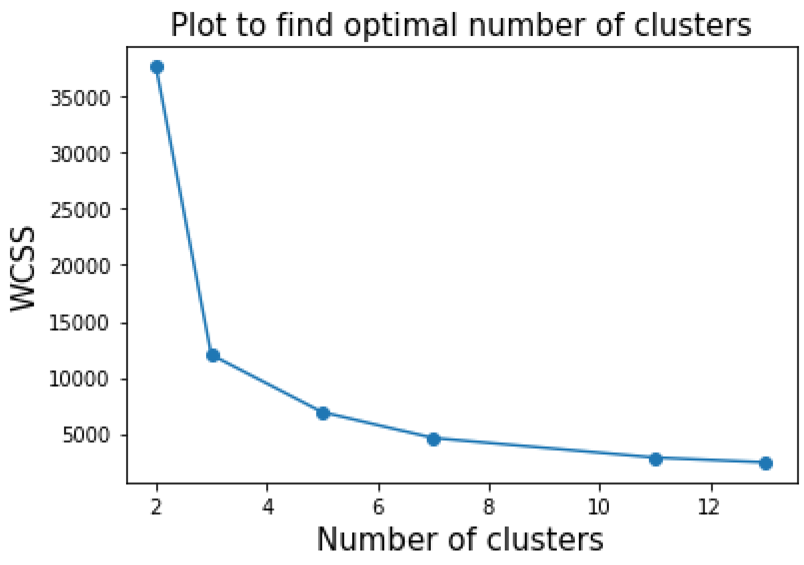

Figure 3: WCSS plot for 2, 3, 5, 7, 11, and 13 Clusters

As a way to identify which number of clusters would be optimal, a good approach is the "elbow method", wherein when the plot makes a sudden and rapid change in shape (resembling an elbow), the point at which it occurs is the optimal number of clusters [8]. Using this method, it can be seen that the optimal number of clusters for this data would be either 3 or 5, and it would be best to choose 5 clusters as the WCSS value also starts to saturate after exceeding 5 clusters. This can also be observed in Figure 2, where the 5 clusters separate the data cleanly into 5 regions.

As per the requirement, the number of clusters were chosen as N=2. Then, Percentage overlap/consistency with action labels originally provided for both F and NF cases was found to be 18.727915194346288%, which seems to be a good number considering that the number of clusters chosen was only 2.

Hence, looking at the above Exploratory Analysis and basing an expectation using it, it is most likely possible to conduct fall detection.

PART B:

This part involves supervised learning, wherein the requirement was to build a Support-Vector Machine Classifier and a Multi-Layer Perceptron Classifier. The details for both are given below. Before using the classifiers, the data was required to be split into training, validation and testing sets. For this purpose, "sklearn.model_selection.train_test_split" [9] was made use of. The data was split as follows: 70% for training, 15% for validation, and 15% for testing. In this part, the reduced dimensional data was used instead of the original data.

SUPPORT-VECTOR MACHINE CLASSIFIER:

An SVM forms a hyperplane, with the help of support vector data points, in order to separate it into classes [10]. For implementing the Support-Vector Machine (SVM) Classifier, the "Support Vector Machines" library was used from scikit-learn [11]. When selecting the model parameters in the scikit-learn SVM library, the hyperparameters were experimented on by using the training dataset. One to take note of, the RBF kernel was used, as it is better for linearly inseparable data [12], which, based on the exploratory analysis, should fit better on the data; additionally, maximum iterations were set to 1000, gamma as 0.2, and tolerance for stopping as 1e-05. When the model is trained using the hyperparameters chosen with the use of the training set, it is then tested with the validation set and an acceptable accuracy score is obtained. With parameters selected using the validation data, the model was then tested with the testing split of the data. With the test data, the test accuracy for the model was found to be 88.23529411764706%.

MULTI-LAYER PERCEPTRON CLASSIFIER:

A Multi-Layer Perceptron (MLP) Classifier is an application of a feedforward neural network, which consists of an input layer, hidden layer and output layer, and also makes use of backpropagation [13]. For implementing the Multi-Layer Perceptron (MLP) Classifier, "sklearn.neural_network.MLPClassifier" [14] was used. Using the training split of the data, the hyperparameters for the classifier were chosen. The activation function for the MLP classifier was chosen as the RELU, rectified linear activation unit, which is acknowledged as one of the best activation functions to use in a neural network in general [15]. 16 hidden layers with 128 hidden units were used. The Adam optimizer, adaptive moment estimation, is widely used as it is considered a good method of gradient descent as an optimization technique [16], hence it was chosen. The maximum iterations were 1000, early stopping was turned off, random state 0 was selected, alpha was 0.01, beta1 was 0.75, beta2 was 0.975, and epsilon was 1e-4. The plot for the loss in Figure 4 shows that the loss saturates, and so any increase in the model's accuracy would have been very small after this point. The validation set was tested with the trained model and an acceptable accuracy score was obtained, with the training being done on 629 iterations before stopping. With parameters selected using the validation data, the model was then tested with the testing split of the data. With the test data, the test accuracy for the model was found to be 87.05882352941177%.

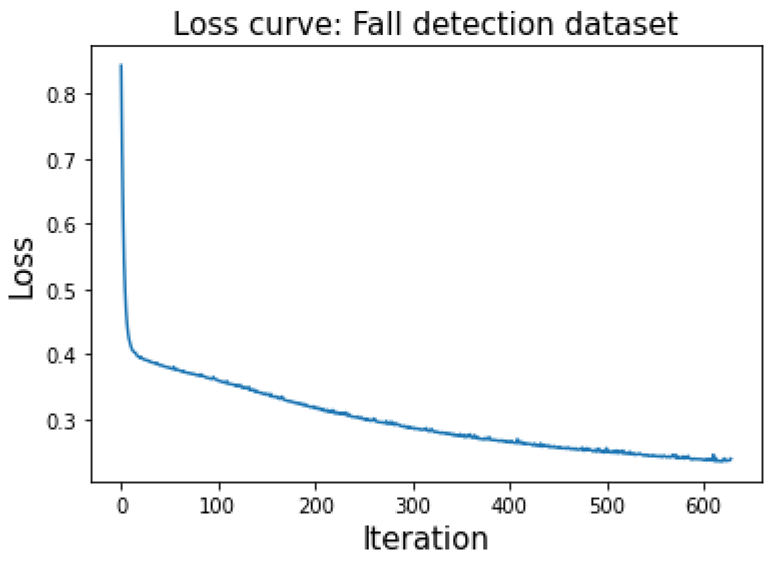

Figure 4: Loss Plot for MLP Classifier

COMPARISON OF SVM AND MLP CLASSIFIERS:

With the Support-Vector Machine Classifier, the test accuracy obtained at the end was 88.23529411764706%. With the Multi-Layer Perceptron Classifier, the test accuracy obtained at the end was 87.05882352941177%. Hence, based on these findings, overall it can be said that fall detection based on wearable sensors is very possible, as both classifiers give a high accuracy, with being close to 90% accurate. While both classifiers are good, it is noted that the SVM classifier outperformed the MLP classifier by a small margin. It can still be possible that with more data, MLP may perform better, however the goal of fall detection was accomplished.

REFERENCES

[1] "scikit-learn: machine learning in Python — scikit-learn 1.0.2 documentation", Scikit-learn.org, 2021. [Online]. Available: https://scikit-learn.org/stable/.

[2] "Importance of Feature Scaling", scikit-learn. [Online]. Available: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html.

[3] "sklearn.preprocessing.StandardScaler", scikit-learn, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[4] A. Sharma, "Principal Component Analysis (PCA) in Python Tutorial", datacamp, 2020. [Online]. Available: https://www.datacamp.com/tutorial/principal-component-analysis-in-python.

[5] "sklearn.decomposition.PCA", scikit-learn, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

[6] "sklearn.cluster.KMeans", scikit-learn, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

[7] "Clustering in Python | What is K means Clustering?", Analytics Vidhya, 2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/12/a-detailed-introduction-to-k-means-clustering-in-python/.

[8] B. Saji, "K Means Clustering | K Means Clustering Algorithm in Machine Learning", Analytics Vidhya, 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning/.

[9] "sklearn.model_selection.train_test_split", scikit-learn, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[10] A. Navlani, "Support Vector Machines with Scikit-learn Tutorial", datacamp, 2019. [Online]. Available: https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python.

[11] "1.4. Support Vector Machines", scikit-learn, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/svm.html.

[12] A. Kumar, "SVM RBF Kernel Parameters With Code Examples - DZone AI", dzone.com, 2020. [Online]. Available: https://dzone.com/articles/using-jsonb-in-postgresql-how-to-effectively-store-1.

[13] "What is a Multilayer Perceptron (MLP)?", Techopedia. [Online]. Available: https://www.techopedia.com/definition/20879/multilayer-perceptron-mlp.

[14] "sklearn.neural_network.MLPClassifier", scikit-learn, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.

[15] Great Learning Team, "What is Rectified Linear Unit (ReLU)? | Introduction to ReLU Activation Function", Great Learning, 2020. [Online]. Available: https://www.mygreatlearning.com/blog/relu-activation-function/.

[16] "Intuition of Adam Optimizer - GeeksforGeeks", GeeksforGeeks, 2020. [Online]. Available: https://www.geeksforgeeks.org/intuition-of-adam-optimizer/.

APPENDIX:

This project was done on Google Colaboratory, and the source code is as follows. To check the code, it would be best to upload the separately uploaded .ipynb file to Google Colab.

```
# -*- coding: utf-8 -*-

"""GE 461 Project 4 Code Fahad Waseem Butt 21801356.ipynb



Automatically generated by Colaboratory.

"""



# GE 461 Project 4 Fall Detection Fahad Waseem Butt 21801356



# Upload the dataset file from the device's local drive.

# Upload when prompted, from below this code block.



# Uploading dataset

from google.colab import files

uploaded = files.upload()



# Importing Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split

from sklearn import svm

from sklearn import metrics

from sklearn.neural_network import MLPClassifier



dataset = pd.read_csv('falldetection_dataset.csv', header=None)

data = dataset.iloc[:, 1:]                 # Removing first column (Left
over columns i.e. 1 ~ 566)

# print(data)

data_labels = data.iloc[:, 0].values     # Separating labels i.e F or NF
(First Column) (566,)

# print(data_labels)

data_features = data.iloc[:, 1:].values # Separating numeric data (566,
306)

# print(data_features)



scaled_features  =  StandardScaler().fit_transform(data_features)      #
Scaling the numeric input data

# print(scaled_features)



# Check whether normalized data has mean=0 & SD=1

Mean = np.mean(scaled_features)

StdDev = np.std(scaled_features)

print('Mean =', Mean , ', Standard Deviation =', StdDev)



print(dataset.shape)
print(data.shape)

print(data_features.shape)

print(data_labels.shape)

print(scaled_features.shape)



"""PART A"""



# Projecting the 306 dimensional fall detection data to ten-dimensional
principal components

pca_fall = PCA(n_components=8)

pca_fall.fit_transform(scaled_features)

# Variance each principal component holds

print('Variance per PC: {}'.format(pca_fall.explained_variance_ratio_))

pca_toptwo = PCA(n_components=2)

pca_toptwo.fit_transform(scaled_features)

print('\nVariance           for           top           2           PCs:
{}'.format(pca_toptwo.explained_variance_ratio_))



#  Projecting  306  dimensional  fall  detection  data  to  two-dimensional
principal components

pca_fall = PCA(n_components=2)

scaled_updated   =   pca_fall.fit_transform(scaled_features)   #   Apply
scaling by doing fit_transform

plt.figure(figsize=(8,8))

plt.scatter(scaled_updated[data_labels             ==             'F',0],
scaled_updated[data_labels == 'F',1],c='r', label='Fall')
```

```
plt.scatter(scaled_updated[data_labels            ==            'NF',0],
scaled_updated[data_labels == 'NF',1],c='b', label='No Fall')

plt.xticks(fontsize=10), plt.yticks(fontsize=10)

plt.xlabel('PC 1',fontsize=15),plt.ylabel('PC  2',fontsize=15)

plt.title("PCA of fall detection dataset",fontsize=15);

plt.legend();


# K-means Clustering

cluster_range = [2, 3, 5, 7, 11, 13]

inertias = []

for cr in cluster_range:

                                                        km                =
KMeans(n_clusters=cr,random_state=0,max_iter=50).fit(scaled_updated)

    inertias.append(km.inertia_)

    plt.figure(figsize=(8,8))

    plt.title('Scatter plot with clusters= ' + str(cr), fontsize=15)

    for i in range(cr):

                  plt.scatter(scaled_updated[km.labels_   ==   i,   0],
scaled_updated[km.labels_ == i, 1])

     plt.scatter(km.cluster_centers_[:,  0],  km.cluster_centers_[:,  1],
label='centroid',  c='black',marker='o',  s=150)

    plt.legend();



plt.figure()

plt.plot(cluster_range,inertias,'o-')

plt.xlabel('Number                                                      of
clusters',fontsize=15),plt.ylabel('WCSS',fontsize=15)
```

```
plt.title("Plot to find optimal number of clusters",fontsize=15);


# Percentage overlap/consistency for 2 x clusters

km              =              KMeans(n_clusters=2,            max_iter=150,
random_state=0).fit(scaled_updated)

plt.figure(figsize=(8,8))

plt.title('Scatter Plot: 2 x Clusters',fontsize=15)

plt.scatter(scaled_updated[km.labels_         ==          0,          0],
scaled_updated[km.labels_ == 0, 1],c='black')

plt.scatter(scaled_updated[km.labels_         ==          1,          0],
scaled_updated[km.labels_ == 1, 1],c='blue')

plt.scatter(km.cluster_centers_[:,    0],    km.cluster_centers_[:,    1],
marker='o',s=150,c='red', label='centroid'),plt.legend()

# Find index where one is located

index_ones=[i for i, e in enumerate(km.labels_) if e == 1]

# Check if F is also present at the found index

OL_NF=[i for i, e in enumerate(data_labels[index_ones]) if e == 'F']

NF_overlap=len(OL_NF)

# Find index where zero is located

index_zeroes=[i for i, e in enumerate(km.labels_) if e == 0]

# Check if NF is also present at the found index

OL_F=[i for i, e in enumerate(data_labels[index_zeroes]) if e == 'NF']

F_overlap=len(OL_F)

print('Percentage  overlap/consistency  with  action  labels  originally
provided      for      both       F       and       NF       cases      =',
((NF_overlap+F_overlap)/data_labels.shape[0]*100))

print('\n')
```

```
"""PART B"""


# Split to get Test data = 15 %, Train data= 85 %    # Using reduced
dimensional data

X_train, X_test, y_train, y_test = train_test_split(scaled_updated,
data_labels,test_size=0.15, shuffle = True, random_state = 8)


# Split to get Train date = 70 % (1-0.1765)*0.85, val data = 15%
(0.1765*0.85)

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.1765, random_state= 8)


clf = svm.SVC(kernel='rbf', gamma=0.2, max_iter = 1000, tol = 1e-05)

SVMclassifier = clf.fit(X_train,y_train)

#Predict the response for Test dataset

val_pred1 = SVMclassifier.predict(X_val)

#Predict the response for Validation dataset

test_pred1 = SVMclassifier.predict(X_test)

# Model Accuracy

print("Validation Accuracy:",metrics.accuracy_score(y_val, val_pred1))

print("Test Accuracy:",metrics.accuracy_score(y_test, test_pred1))


clf = MLPClassifier(hidden_layer_sizes=(16, 128), activation='relu',
solver='adam',

                                    verbose=True, max_iter=1000,
early_stopping=False, random_state=0, alpha=0.01, beta_1=0.75,
beta_2=0.975, epsilon=1e-4)
```

```
MLPclassifier = clf.fit(X_train, y_train)

#Predict the response for Test dataset

val_pred2 = MLPclassifier.predict(X_val)

#Predict the response for Validation dataset

test_pred2 = MLPclassifier.predict(X_test)

print("Validation Accuracy:",metrics.accuracy_score(y_val, val_pred2))

print("\nTest Accuracy:",metrics.accuracy_score(y_test, test_pred2))

print("\n")

plt.plot(clf.loss_curve_)

plt.xlabel('Iteration',fontsize=15),plt.ylabel('Loss',fontsize=15)

plt.title("Loss curve: Fall detection dataset",fontsize=15);
```