Eray Gündoğdu 22101962

GE461 Telehealth – Fall Detection Project

Part A)

In this part the aim is to perform explarotary data analysis by using clustering. Before applying k-means clustering to observe whether data is seperable or not, the data is reduced two 2 dimensions by applying PCA. The eigenvalues of the autocorrelation matrix and the corrsponding explained variance ratios can be observed in Figure 1. In Figure 2 the projected data on the first 2 PC's can be observed. It is benefitial to mention that since features are on a different scale the data is centered and standardized.

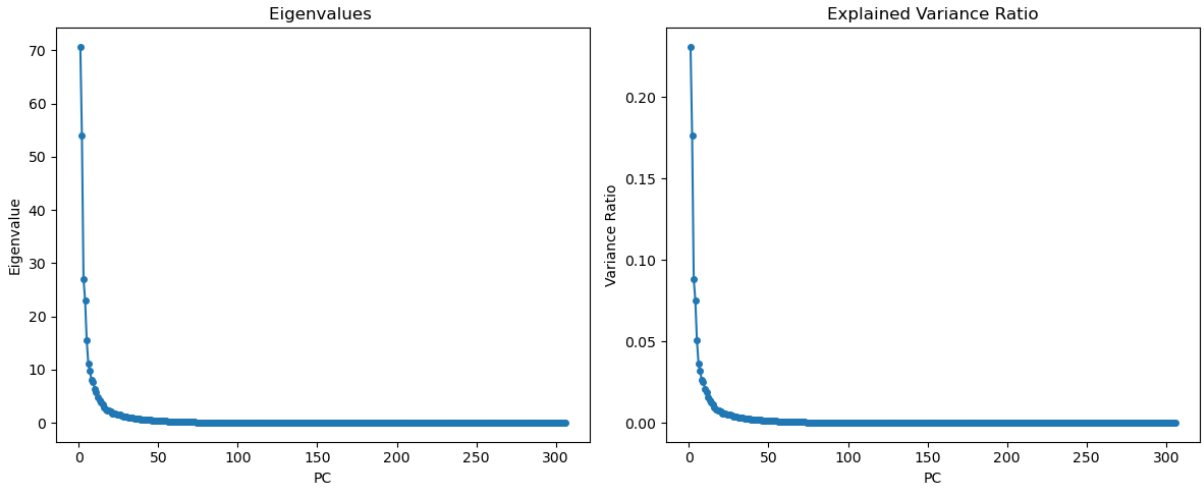

Figure 1 Demonstration of eigenvalues and corresponding variance ratios

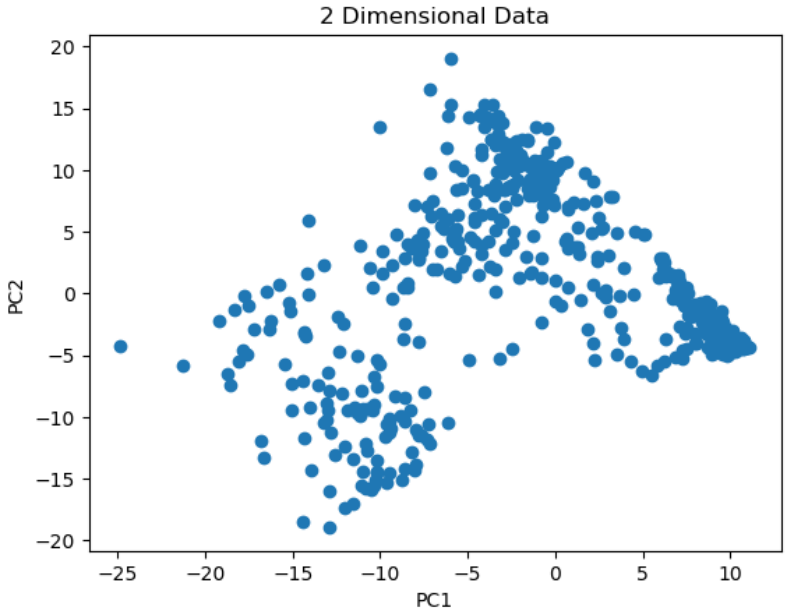

Figure 2 2 dimensional representation of the dataset

Now k-means algorithm will be applied after reducing tthe dimensionalty of the data to 566x2 space. Since there are 2 labels NF and F, if the 566x2 dimensional data can be seperated as 2 clusters, the result will be the most efficient in terms exploratory data analysis. The results for different number of clusters can be observed in Figure 3.

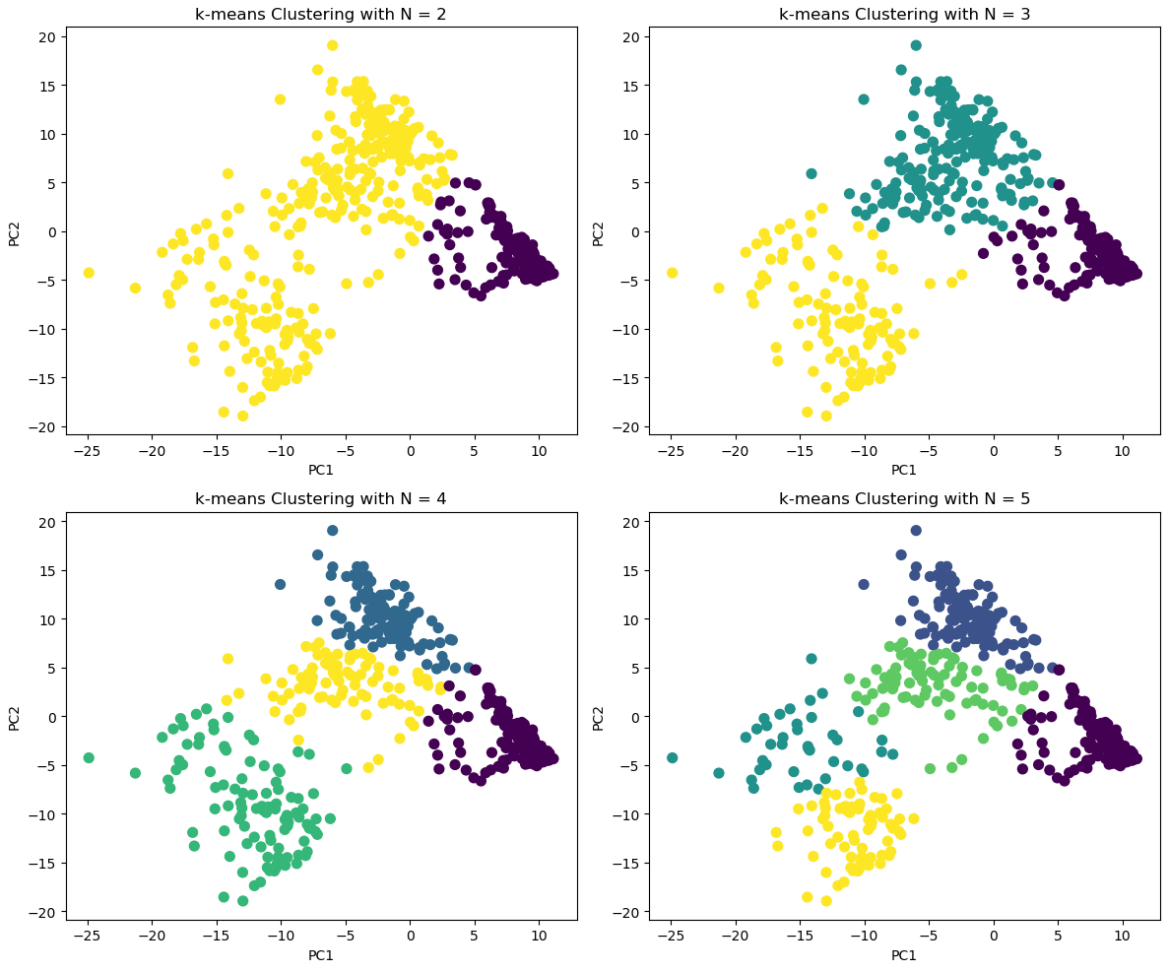

Figure 3 k-means algorithm results with different number of clusters

To obsereve the performance of the number of the clusters applied, percentage consistency between the cluster memberships and the action labels originally provided is taken into account. After implementing the relevant code to obtain the percentages, N=2 k-means clustering's consistency percentage is measured as %81.45. As the number of clusters increase the data is most likely to be seperated better if the label consistency percentage is considered since the class samples become smaller and achieving pure classes is most likely (consider the extreme where N=566, each of the data points is explained with one class hence theoratically will result in %100 percentage class overlap). As N=5 k-means is observed, the ratio increases to %87.63, in this case the original data is seperated as 5 clusters which have labels (as majority) F, F, F, NF, F which increases the label consistency percentage. Theoratically if N=566 k-means algorith will be applied %100 percent label consistency percentage is expected however this case is not practical in terms exploratory data analysis.

As a result explarotary data analysis inform the observer in some way to make a rough NF or F classification as described above the observer can make a rough classification by assigning labels to the clusters (for N=5 F,F,F,NF, F labeling of clusters results in %87.63 consistency).

However the direct observation of the data in two clusters obtains a %81.45 class overlap percentage. The performance can be improved by supervised learning which is investigated in Part B.

Part B)

After completing the explarotary data analysis, the aim is to learn and generalize the data by using SVM and MLP. Before the train test validation split again the centered and standardized data is used. The dataset is divided to train(%70) test(15) validation(%15) splits and carefully handled throughout the assignment. As the SVM model sicikit-learn's SVC (Support Vector Classifier) is used. The model consists of kernel, 'C' and gamma parameters. Kernel determines whether the model uses a linear or a non-linear decision boundry, C is the regularization parameter that determines the penalty for misclassification, gamma is only avaliable for rbf kernel and plays a role in determining the influence of a single sample on other reigons in the space. All of this values has a role in model complexity hence they determine whether the model is generalizable or model underfits/overfits. To extract the best possible model a cross-validation procedure is applied. As mentioned previosly the model is trained for different set of hyperparameters on the training set (%70). Afterwards different trained models are validated on the unseen validation set (%15). As the final step the model that achieves highest validation accuracy, is tested on test set using the unseen test data (%15). The values for kernel hyperparameter are 'linear','rbf' the values for C are 0.1, 1, 10, the values for gamma are determined to be 'auto' and 'scale' in total 9 different sets of hyperparameters are obtained.

```
SET 1: kernel=linear, C=0.1 | gamma=scale | Validation Accuracy: 0.9882
SET 2: kernel=linear, C=1 | gamma=scale | Validation Accuracy: 0.9882
SET 3: kernel=linear, C=10 | gamma=scale | Validation Accuracy: 0.9882
SET 4: kernel=rbf, C=0.1 | gamma=scale | Validation Accuracy: 0.8941
SET 5: kernel=rbf, C=0.1 | gamma=auto | Validation Accuracy: 0.7647
SET 6: kernel=rbf, C=1 | gamma=scale | Validation Accuracy: 0.9882
SET 7: kernel=rbf, C=1 | gamma=auto | Validation Accuracy: 0.8824
SET 8: kernel=rbf, C=10 | gamma=scale | Validation Accuracy: 0.9765
SET 9: kernel=rbf, C=10 | gamma=auto | Validation Accuracy: 0.8941

Best parameters (SVM): {'kernel': 'linear', 'C': 0.1, 'gamma': 'scale'}
Best validation accuracy (SVM): 0.9882352941176471
Test AccuracY (SVM): 0.9764705882352941
```

Figure 5 SVM hyperparameter tuning results

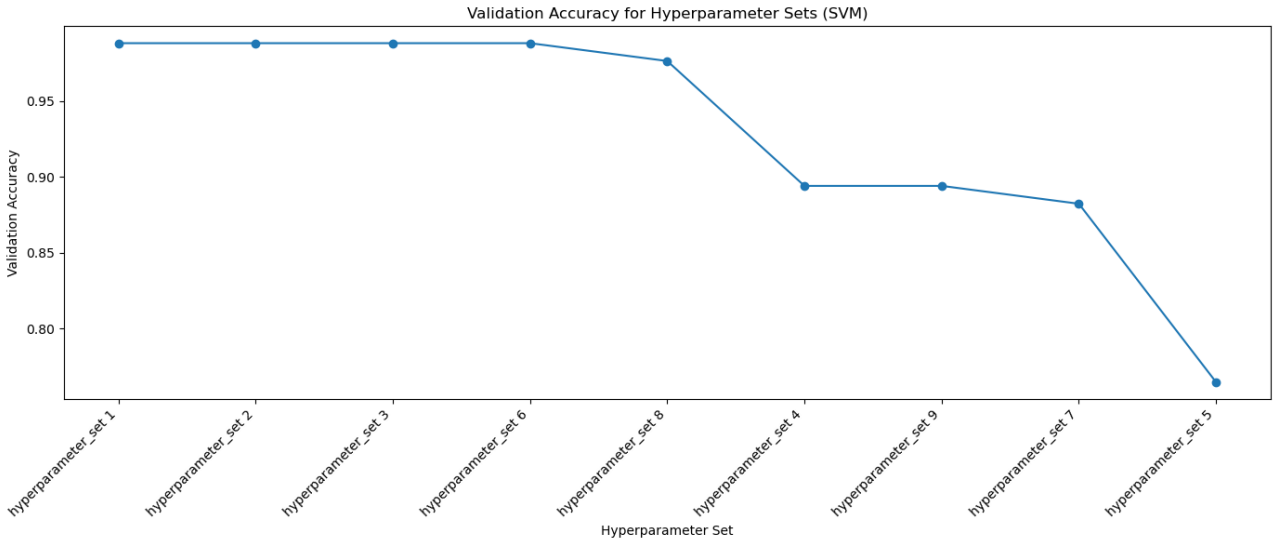

Figure 6 Validation accuracies for each SVM hyperparameter set

According to the experimentations with hyperparameters the linear kernel and 0.1 regularization parameter outputs a the most generalizable model compared to other sets of hyperparmeters for this fall detection dataset. Now as the MLP model scikit-learn's MLPClassifer model will be used. The hyperparameters or to be more accurate the model can take arguements for hidden layer sizes, activation fuction, L2 regularization parameter(alpha), aand learning rate. To mention the influences of the parameters, number of hidden layers and number of hidden unit layers directly influence the model complexity hence they are crucial in terms of obtaining a generalizable model. Activation functions are important in terms of computational effciency and overcoming vanishing/exploding gradient problems. L2 regularization parameter penalizes the weights therefore able to make the model more generalizable. As the last hyperparameter learning rate is important in terms of convergance, escaping local minima problems.

The MLPClassifier model is highly optimized model. The model automatically checks the convergence however convegance does not imply a generalizable model. Therefore before moving onto hyperparameter tuning the number of iterations (epochs) must be determined to avoid underfitting and overfitting. To test this an a sample model is trained with high complexity to make it overfit ie no regularization (100,100,100) hidden units. In Figure 7 to understand the model status in terms of overfitting and underfitting the difference between training accuracy and validation accuracy is inspected and around 100 iterations the gap starts to increase which implies overfitting whereas beforehand (up to 100) is the underfitting reigon. Also it is benefitial to mention that 100 iterations are enough for the convergence of the optimizer of MLPClassifier model since no convergance error occurs during runtime.

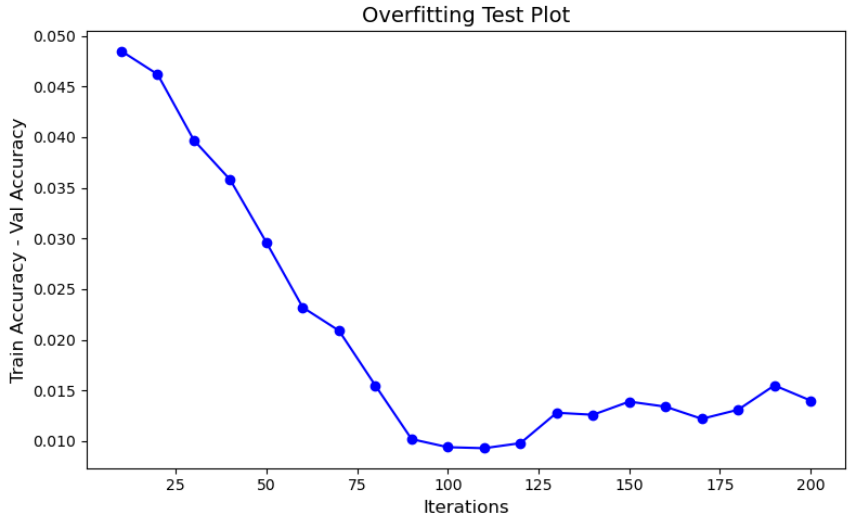

Figure 7 Demosntration of the optimal number of iterations

Now for the hyperparameter tuning secction max_iter is set to 100. The hyperparameter values are as following. For hidden layer sizes 50, 100, (50,50), (100,100) are assigned, relu and sigmoid functions are assigned as activation functions. For regularization parameter (alpha) 0.0001 and 0.001 are assigned. Finally for learning rates 0.001, 0.01 and 3 are assigned which makes a toatl of 48 sets. The hyperparramter tuning results are as following.

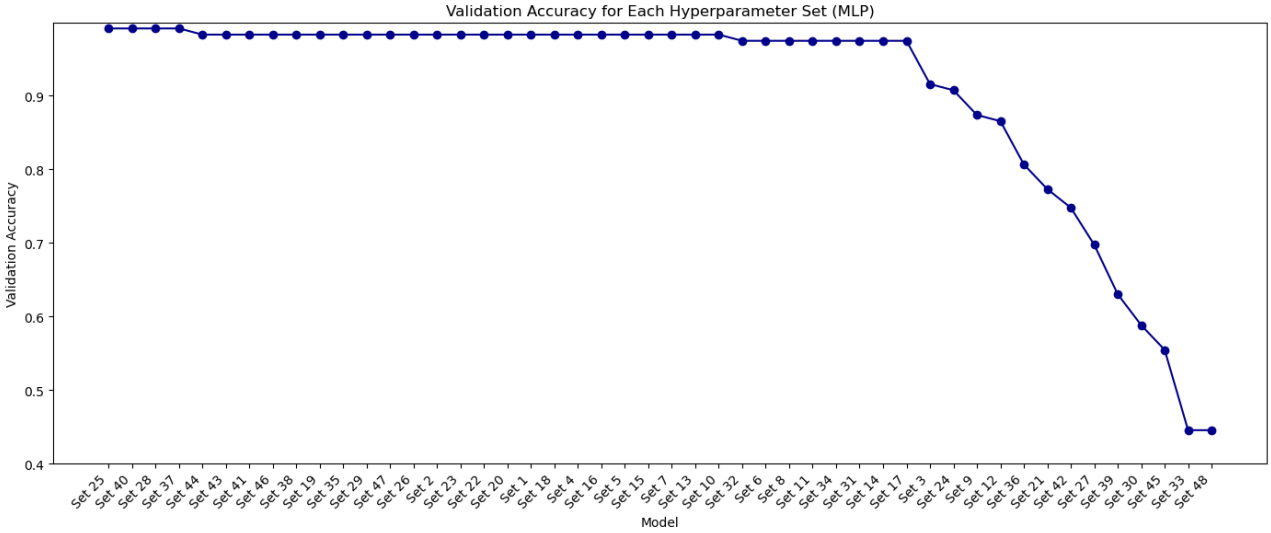

Figure 8 Validation accuracies for each MLP hyperparameter set

```
Set 1: hidden_layer_sizes=(50,) | activation=relu | alpha=0.0001 | lr=0.001 | Validation Accuracy: 0.9832
Set 2: hidden_layer_sizes=(50,) | activation=relu | alpha=0.0001 | lr=0.01 | Validation Accuracy: 0.9832
Set 3: hidden_layer_sizes=(50,) | activation=relu | alpha=0.0001 | lr=3 | Validation Accuracy: 0.9160
Set 4: hidden_layer_sizes=(50,) | activation=relu | alpha=0.001 | lr=0.001 | Validation Accuracy: 0.9832
Set 5: hidden_layer_sizes=(50,) | activation=relu | alpha=0.001 | lr=0.01 | Validation Accuracy: 0.9832
Set 6: hidden_layer_sizes=(50,) | activation=relu | alpha=0.001 | lr=3 | Validation Accuracy: 0.9748
Set 7: hidden_layer_sizes=(50,) | activation=tanh | alpha=0.0001 | lr=0.001 | Validation Accuracy: 0.9832
Set 8: hidden_layer_sizes=(50,) | activation=tanh | alpha=0.0001 | lr=0.01 | Validation Accuracy: 0.9748
Set 9: hidden_layer_sizes=(50,) | activation=tanh | alpha=0.0001 | lr=3 | Validation Accuracy: 0.8739
Set 10: hidden_layer_sizes=(50,) | activation=tanh | alpha=0.001 | lr=0.001 | Validation Accuracy: 0.9832
Set 11: hidden_layer_sizes=(50,) | activation=tanh | alpha=0.001 | lr=0.01 | Validation Accuracy: 0.9748
Set 12: hidden_layer_sizes=(50,) | activation=tanh | alpha=0.001 | lr=3 | Validation Accuracy: 0.8655
Set 13: hidden_layer_sizes=(100,) | activation=relu | alpha=0.0001 | lr=0.001 | Validation Accuracy: 0.9832
Set 14: hidden_layer_sizes=(100,) | activation=relu | alpha=0.0001 | lr=0.01 | Validation Accuracy: 0.9748
Set 15: hidden_layer_sizes=(100,) | activation=relu | alpha=0.0001 | lr=3 | Validation Accuracy: 0.9832
Set 16: hidden_layer_sizes=(100,) | activation=relu | alpha=0.001 | lr=0.001 | Validation Accuracy: 0.9832
Set 17: hidden_layer_sizes=(100,) | activation=relu | alpha=0.001 | lr=0.01 | Validation Accuracy: 0.9748
Set 18: hidden_layer_sizes=(100,) | activation=relu | alpha=0.001 | lr=3 | Validation Accuracy: 0.9832
Set 19: hidden_layer_sizes=(100,) | activation=tanh | alpha=0.0001 | lr=0.001 | Validation Accuracy: 0.9832
Set 20: hidden_layer_sizes=(100,) | activation=tanh | alpha=0.0001 | lr=0.01 | Validation Accuracy: 0.9832
Set 21: hidden_layer_sizes=(100,) | activation=tanh | alpha=0.0001 | lr=3 | Validation Accuracy: 0.7731
Set 22: hidden_layer_sizes=(100,) | activation=tanh | alpha=0.001 | lr=0.001 | Validation Accuracy: 0.9832
Set 23: hidden_layer_sizes=(100,) | activation=tanh | alpha=0.001 | lr=0.01 | Validation Accuracy: 0.9832
Set 24: hidden_layer_sizes=(100,) | activation=tanh | alpha=0.001 | lr=3 | Validation Accuracy: 0.9076
Set 25: hidden_layer_sizes=(50, 50) | activation=relu | alpha=0.0001 | lr=0.001 | Validation Accuracy: 0.9916
Set 26: hidden_layer_sizes=(50, 50) | activation=relu | alpha=0.0001 | lr=0.01 | Validation Accuracy: 0.9832
Set 27: hidden_layer_sizes=(50, 50) | activation=relu | alpha=0.0001 | lr=3 | Validation Accuracy: 0.6975
Set 28: hidden_layer_sizes=(50, 50) | activation=relu | alpha=0.001 | lr=0.001 | Validation Accuracy: 0.9916
Set 29: hidden_layer_sizes=(50, 50) | activation=relu | alpha=0.001 | lr=0.01 | Validation Accuracy: 0.9832
Set 30: hidden_layer_sizes=(50, 50) | activation=relu | alpha=0.001 | lr=3 | Validation Accuracy: 0.5882
Set 31: hidden_layer_sizes=(50, 50) | activation=tanh | alpha=0.0001 | lr=0.001 | Validation Accuracy: 0.9748
Set 32: hidden_layer_sizes=(50, 50) | activation=tanh | alpha=0.0001 | lr=0.01 | Validation Accuracy: 0.9748
Set 33: hidden_layer_sizes=(50, 50) | activation=tanh | alpha=0.0001 | lr=3 | Validation Accuracy: 0.4454
Set 34: hidden_layer_sizes=(50, 50) | activation=tanh | alpha=0.001 | lr=0.001 | Validation Accuracy: 0.9748
Set 35: hidden_layer_sizes=(50, 50) | activation=tanh | alpha=0.001 | lr=0.01 | Validation Accuracy: 0.9832
Set 36: hidden_layer_sizes=(50, 50) | activation=tanh | alpha=0.001 | lr=3 | Validation Accuracy: 0.8067
Set 37: hidden_layer_sizes=(100, 100) | activation=relu | alpha=0.0001 | lr=0.001 | Validation Accuracy: 0.9916
Set 38: hidden_layer_sizes=(100, 100) | activation=relu | alpha=0.0001 | lr=0.01 | Validation Accuracy: 0.9832
Set 39: hidden_layer_sizes=(100, 100) | activation=relu | alpha=0.0001 | lr=3 | Validation Accuracy: 0.6303
Set 40: hidden_layer_sizes=(100, 100) | activation=relu | alpha=0.001 | lr=0.001 | Validation Accuracy: 0.9916
Set 41: hidden_layer_sizes=(100, 100) | activation=relu | alpha=0.001 | lr=0.01 | Validation Accuracy: 0.9832
Set 42: hidden_layer_sizes=(100, 100) | activation=relu | alpha=0.001 | lr=3 | Validation Accuracy: 0.7479
Set 43: hidden_layer_sizes=(100, 100) | activation=tanh | alpha=0.0001 | lr=0.001 | Validation Accuracy: 0.9832
Set 44: hidden_layer_sizes=(100, 100) | activation=tanh | alpha=0.0001 | lr=0.01 | Validation Accuracy: 0.9832
Set 45: hidden_layer_sizes=(100, 100) | activation=tanh | alpha=0.0001 | lr=3 | Validation Accuracy: 0.5546
Set 46: hidden_layer_sizes=(100, 100) | activation=tanh | alpha=0.001 | lr=0.001 | Validation Accuracy: 0.9832
Set 47: hidden_layer_sizes=(100, 100) | activation=tanh | alpha=0.001 | lr=0.01 | Validation Accuracy: 0.9832
Set 48: hidden_layer_sizes=(100, 100) | activation=tanh | alpha=0.001 | lr=3 | Validation Accuracy: 0.4454

Best MLP parameters: {'hidden_layer_sizes': (50, 50), 'activation': 'relu', 'alpha': 0.0001, 'learning_rate_init': 0.001}
Best MLP validation accuracy: 0.9915966386554622
Final MLP Test Accuracy: 0.9803921568627451
```

Figure 9 MLP hyperparameter tuning results

As the final step to compare SVM and MLP results recall, precision, f1-score and confusion matrices are included because a simply higher test accuracy does not mean a better model in fact important metrics such as f1-score, confusion matrices, recall, precision values should be inspected to have an idea about false positive ratios. It is benefitial to mention that these values belong to test set which has a total of 85 samples.

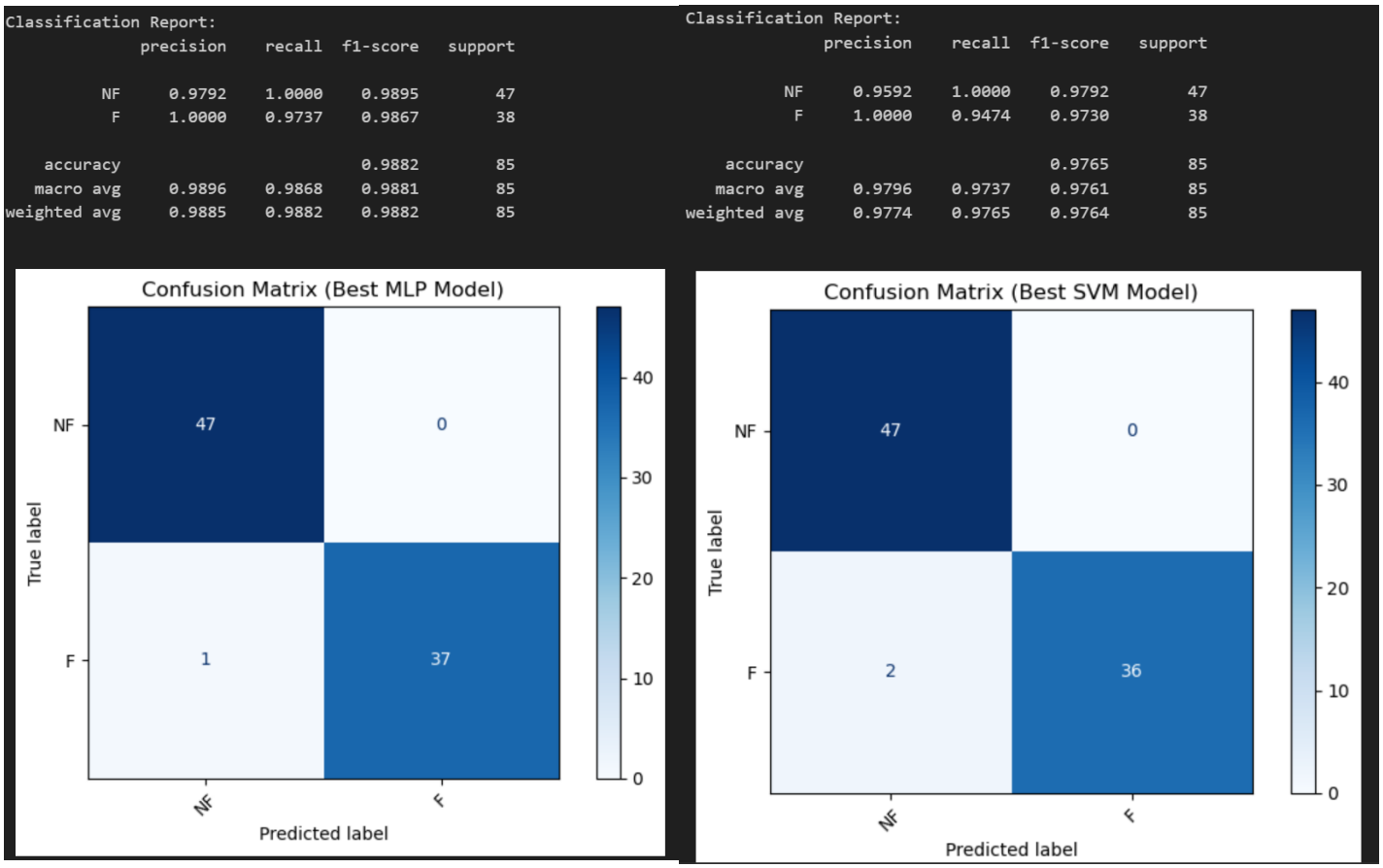

Figure 10 Model evaluation values.

As the test accuracy SVM obtained a 0.9764 test accuracy whereas MLP achieved a 0.9880 test accuracy. Both of the models are able to predict NF class correctly with 0 misclassifications however the 1v extra misclassfication of F class in SVM results in lower overall test accuracy. Precision of F is equal for both models (ratio of true positives to sum of positives) also recall for NF is also equal (true positives /true positives + false negatives). For precision of NF MLP outperformed SVM which is the same result for recall of F hence f1-scores are different. To be more clear for NF class, MLP model makes more positive predictions and also in F class MLP makes more complete positive predictions. It can be concluded that the supervised learning is able to advance the classification accuracy and both MLP and SVM achieved remarkable results. According to the analysis above MLP model is more preferable for the fall detection application in terms of the test accuracy, confusion matrices, f1, recall and precision scores.

APPENDIX

Python Codes

```
###Eray Gündoğdu 22101962
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
##PART A CODE
pd.set_option('display.precision', 16)
pd.options.display.float_format = '{:.16f}'.format
data = pd.read_csv('falldetection_dataset.csv',
header=None,float_precision='round_trip')
print("Data shape:", data.shape)
data.columns = ['SampleID', 'Label'] + [f"Feature_{i}" for i in range(1, 307)]
print("Data shape after setting column names:", data.shape)
X = data.iloc[:, 2:]
labels = data.iloc[:, 1]
print("Shape of feature matrix X (sensor features only):", X.shape)
X = X - X.mean(axis=0) ##centerted
X = (X - X.mean()) / X.std() #standardized


pca_full = PCA()
pca_full.fit(X)

eigenvalues = pca_full.explained_variance_
explained_variance_ratio = pca_full.explained_variance_ratio_

print("Top 2 eigenvalues:", eigenvalues[:2])
print("Variance explained by the top 2 PCs:", explained_variance_ratio[:2])

pca = PCA(n_components=2) ## 2D visualization of the data
X_pca_std = pca.fit_transform(X)

fig, ax = plt.subplots(1, 2, figsize=(12, 5))

ax[0].plot(np.arange(1, len(eigenvalues) + 1), eigenvalues, 'o-',
markersize=4)
ax[0].set_title("Eigenvalues ")
ax[0].set_xlabel("PC")
ax[0].set_ylabel("Eigenvalue")

ax[1].plot(np.arange(1, len(explained_variance_ratio) + 1),
explained_variance_ratio, 'o-', markersize=4)
ax[1].set_title("Explained Variance Ratio")
```

```
ax[1].set_xlabel("PC")
ax[1].set_ylabel("Variance Ratio")

plt.tight_layout()
plt.show()

plt.scatter(X_pca_std[:, 0], X_pca_std[:, 1])
plt.title("2 Dimensional Data")
plt.xlabel("PC1")
plt.ylabel("PC2")

from sklearn.cluster import KMeans  ##code for different n for kmeans
algorithm
c_m = [2, 3, 4,5]
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
axes = axes.flatten()

for i, n_c in enumerate(c_m):
    kmeans = KMeans(n_clusters=n_c, random_state=22101962)
    clusters = kmeans.fit_predict(X_pca_std)

    axes[i].scatter(X_pca_std[:, 0], X_pca_std[:, 1], c=clusters,
cmap='viridis', s=50)
    axes[i].set_title(f"k-means Clustering with N = {n_c}")
    axes[i].set_xlabel("PC1")
    axes[i].set_ylabel("PC2")

plt.tight_layout()
plt.show()

sample_kmeans = KMeans(n_clusters=5, random_state=22101962)
all_c = sample_kmeans.fit_predict(X)


overlap_ratio = pd.crosstab(labels, all_c, rownames=["True Label"],
colnames=["Cluster"])
print(overlap_ratio)

correct = 0 ##cluster label consistency results
for cluster in np.unique(all_c):
    cluster_counts = overlap_ratio.iloc[:, cluster]
    correct += cluster_counts.max()

fin_accuracy = correct / len(labels) * 100
print(f"cluster/label consistency : {fin_accuracy:.2f}%")

##PART B CODE

from sklearn.model_selection import train_test_split
```

```
from sklearn.svm import SVC

def accuracy_func(y_true, y_pred):
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)
    return np.mean(y_true == y_pred)

X_train, X_div, y_train, y_div = train_test_split(X, labels, test_size=0.30,
random_state=22101962, stratify=labels) #%70 train
X_val, X_test, y_val, y_test = train_test_split(X_div, y_div, test_size=0.50,
random_state=22101962, stratify=y_div) # %15 test %15 validation


print("Training set shape:", X_train.shape)
print("Validation set shape:", X_val.shape)
print("Test set shape:", X_test.shape)

results_svm = []  ## variables to capture the best validation accuracy and
relevant set of hyperparams
best_svm = None
best_val_acc_svm = 0
best_params_svm = None
hyp_set = 0

svm_params = {'kernel': ['linear', 'rbf'], 'C': [0.1, 1, 10], 'gamma':
['scale', 'auto']}

for kernel in svm_params['kernel']: #nested loops for ftting the model to set
of hperparams
    for c_reigon in svm_params['C']:
        if kernel == 'linear':
            gamma_values = ['scale']
        else:
            gamma_values = svm_params['gamma']
        for gamma in gamma_values:
            hyp_set += 1
            svm_model = SVC(kernel=kernel, C=c_reigon, gamma=gamma,
random_state=None)
            svm_model.fit(X_train, y_train)
            y_val_pred = svm_model.predict(X_val)
            val_acc = accuracy_func(y_val, y_val_pred)
            print(f"SET {hyp_set}: kernel={kernel}, C={c_reigon} |
gamma={gamma} | Validation Accuracy: {val_acc:.4f}")
            results_svm.append({
                "hyperparameter_set": f"hyperparameter_set {hyp_set}",
                "kernel": kernel,
                "C": c_reigon,
                "gamma": gamma,
                "val_acc": val_acc
```

```
            })
            if val_acc > best_val_acc_svm:
                best_val_acc_svm = val_acc
                best_svm = svm_model
                best_params_svm = {"kernel": kernel, "C": c_reigon, "gamma":
gamma}

print("\nBest parameters (SVM):", best_params_svm)
print("Best validation accuracy (SVM):", best_val_acc_svm)

y_test_pred_svm = best_svm.predict(X_test)
test_acc_svm = accuracy_func(y_test, y_test_pred_svm)
print("Test AccuracY (SVM):", test_acc_svm)

svm_df = pd.DataFrame(results_svm)
svm_df_sorted = svm_df.sort_values(by="val_acc", ascending=False)

plt.figure(figsize=(14, 6))
plt.plot(svm_df_sorted["hyperparameter_set"], svm_df_sorted["val_acc"],
marker='o', linestyle='-')
plt.xlabel("Hyperparameter Set")
plt.ylabel("Validation Accuracy")
plt.title("Validation Accuracy for Hyperparameter Sets (SVM)")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
from sklearn.metrics import classification_report, confusion_matrix,
ConfusionMatrixDisplay
class_names = ["NF", "F"]
print(classification_report(y_test, y_test_pred_svm, target_names=class_names,
digits=4))
cm = confusion_matrix(y_test, y_test_pred_svm)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
disp.plot(cmap='Blues', xticks_rotation=45)
plt.title("Confusion Matrix (Best SVM Model)")
plt.tight_layout()
plt.show()

from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split

mlp_params = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 100)],
    'activation': ['relu', 'tanh'],
    'alpha': [0.0001, 0.001],
    'learning_rate_init': [0.001, 0.01, 3]
}
```

```
results = [] ## variables to capture the best validation accuracy and relevant
set of hyperparams
best_mlp = None
best_val_acc_mlp = 0
best_params_mlp = None
hyp_set_mlp = 0

for hidden_layer_sizes in mlp_params['hidden_layer_sizes']: #nested loops for
ftting the model to set of hperparams
    for activation in mlp_params['activation']:
        for reg_param in mlp_params['alpha']:
            for lr in mlp_params['learning_rate_init']:
                hyp_set_mlp += 1
                mlp_model =
MLPClassifier(hidden_layer_sizes=hidden_layer_sizes,
                                        activation=activation,
                                        alpha=reg_param,
                                        learning_rate_init=lr,
                                        max_iter=100,
                                        random_state=22101962)
                mlp_model.fit(X_train, y_train)
                y_val_pred = mlp_model.predict(X_val)
                val_acc = accuracy_func(y_val, y_val_pred)
                print(f"Set {hyp_set_mlp}:
hidden_layer_sizes={hidden_layer_sizes} | activation={activation} |
alpha={reg_param} | lr={lr} | Validation Accuracy: {val_acc:.4f}")
                results.append({
                    "model_number": f"Set {hyp_set_mlp}",
                    "hidden_layer_sizes": hidden_layer_sizes,
                    "activation": activation,
                    "alpha": reg_param,
                    "learning_rate_init": lr,
                    "val_acc": val_acc
                })
                if val_acc > best_val_acc_mlp:
                    best_val_acc_mlp = val_acc
                    best_mlp = mlp_model
                    best_params_mlp = {'hidden_layer_sizes':
hidden_layer_sizes,
                                       'activation': activation,
                                       'alpha': reg_param,
                                       'learning_rate_init': lr}

print("\nBest MLP parameters:", best_params_mlp)
print("Best MLP validation accuracy:", best_val_acc_mlp)

y_test_pred_mlp = best_mlp.predict(X_test)
test_acc_mlp = accuracy_func(y_test, y_test_pred_mlp)
print("Final MLP Test Accuracy:", test_acc_mlp)
```

```
df = pd.DataFrame(results)
df_sorted = df.sort_values(by="val_acc", ascending=False)
plt.figure(figsize=(14, 6))
plt.plot(df_sorted["model_number"], df_sorted["val_acc"], marker='o',
linestyle='-', color="darkblue")
plt.xlabel("Model")
plt.ylabel("Validation Accuracy")
plt.ylim(0.40, 0.999)
plt.title("Validation Accuracy for Each Hyperparameter Set (MLP)")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

class_names = ["NF", "F"]
print("\nClassification Report:")
print(classification_report(y_test, y_test_pred_mlp, target_names=class_names,
digits=4))
cm = confusion_matrix(y_test, y_test_pred_mlp)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
disp.plot(cmap='Blues', xticks_rotation=45)
plt.title("Confusion Matrix (Best MLP Model)")
plt.tight_layout()
plt.show()
```