Project W13: Telehealth – Fall Detection

Muhammad Abdullah S. Mulkana – 21801075

06.05.2022

Introduction

In this project, we are implementing classification on sensor data in order to classify if there has been a fall action or a non-fall action. We are provided with a dataset of 566 observations, each having 306 features/measurements and a label (F for fall, NF for non-fall). Initially, we use unsupervised learning methods in order to visualize the dataset. Then we use two classification algorithms, Support Vector Machine (SVM) and Multi-Layer Perceptron (MLP), in order to classify the data.

Part A

In this part, we visualize the dataset by first implementing Principal Component Analysis (PCA) and projecting the data to two dimensions. Using [1], we perform PCA and find out that the first two Principal Components (PCs) capture **23.1%** and **17.6%** of the data, respectively.

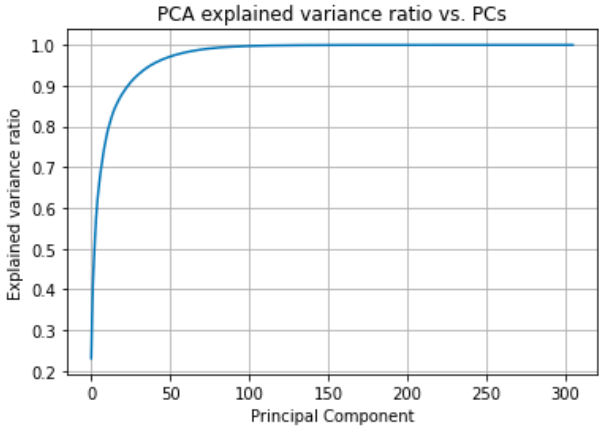

Figure 1: Graph showing the cumulative explained variance ratio vs. PCs.

Figure 1 shows the graph of the cumulative explained variance and the number of PCs. If we wanted to reduce dimensions, we could choose approximately the first 30 PCs that capture more than 90% of the explained variance. Next, we project the data to the first two PCs, and we get the scatter plot as shown in Figure 2.

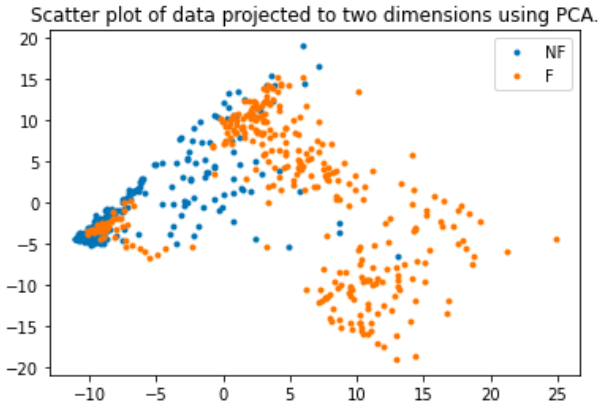

Figure 2: Scatter plot of the projected data.

From the scatter plot, we can see that the classes have a lot of overlap. Next, we implement Kmeans clustering using [2]. The hyperparameter that we change for the algorithm is the number of clusters N. We sweep through values of N ranging from 1 to 20. The metric we are looking for is inertia which is the loss function that the K-means algorithm aims to minimize.

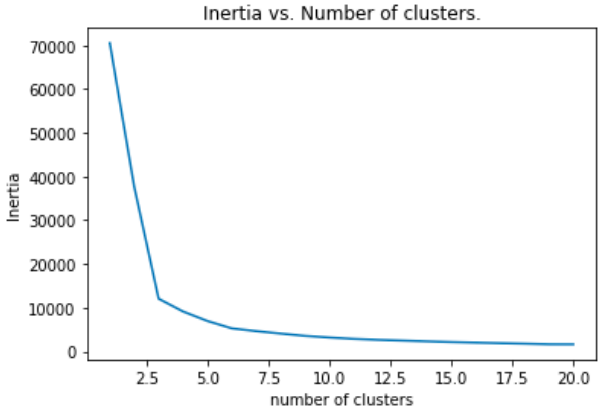

Figure 3: Plot showing the inertia versus the number of clusters for K-means.

Figure 3 shows the elbow plot for the inertia versus the number of clusters. As we can see, the optimum number of clusters to choose would be 3, 4, or 5. We choose the number of clusters to be two as we have two classes.

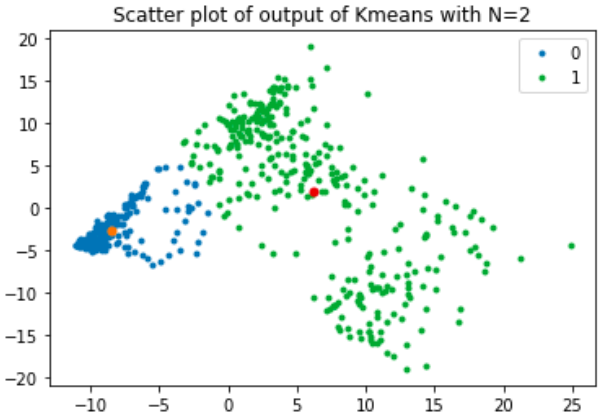

Figure 4: Output of the Kmeans clustering algorithm.

Figure 4 shows the clustering performed by the Kmeans algorithm. The blue markers represent class 0 (NF), and the green markers represent class 1 (F). The orange and red markers show the cluster centers. We check the percentage overlap between the K-means output and the actual

data, which turns out to be **18.7%.** Fall detection is possible but as we can see in Figure 2, there is no apparent decision boundary between the two classes, i.e. there is an overlap.

Part B

In this part, we use supervised classification techniques to classify our data. First, we will analyze the classification of the original data and then the PCA transformed data. Initially, we split the data into train, test, and validation splits of 75%, 15%, and 15%, respectively. The algorithms we will use are SVM and MLP.

Original Data - SVM

The hyperparameter that we can change for SVM is the kernel function used. The function we are using [3] gives us the option of four kernels: radial basis function (RBF), linear, sigmoid, and polynomial. Then if we use the polynomial kernel, we can change the polynomial degree.

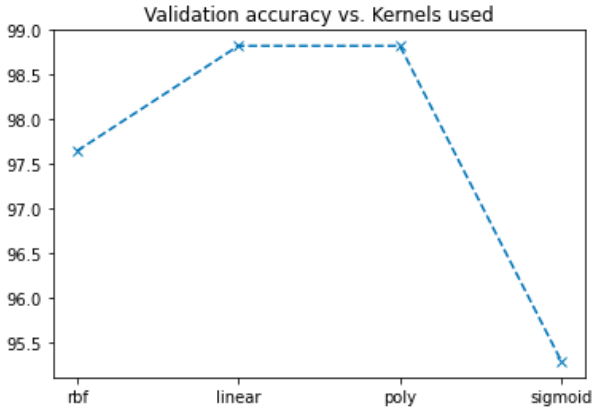

Figure 5: Validation accuracy versus kernel used (original data).

Figure 5 shows the performance of the different kernels. Linear and polynomial kernels give the same validation accuracy i.e. approximately 99%. Figure 6 shows how the degree of the polynomial kernel effects the performance. Polynomial with degree 3 (which is the default) gives the highest accuracy.

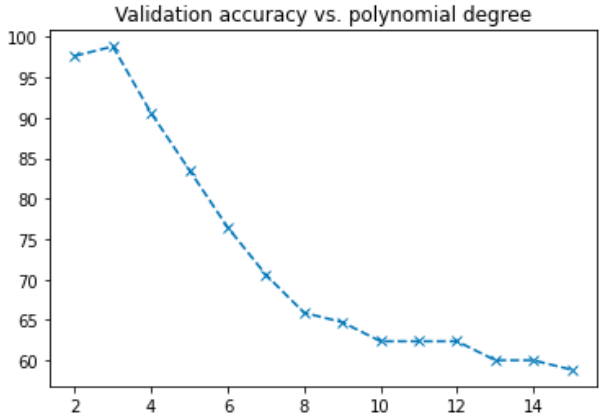

Figure 6: Validation accuracy versus the polynomial degree (original data).

Test accuracy on the two optimum models, linear and polynomial with degree 3, is **96.5%** and **98.8%,** respectively.

Original Data - MLP

In this part, we apply the MLP classifier from [4]. There are many hyperparameters that can be changed, such as the number of hidden layers, number of neurons in each hidden layer, number of epochs, learning rate, momentum rate, regularization parameter, etc. Figure 7 shows the loss plot for the MLP algorithm. Using this, we get a validation accuracy of **100%** and a test accuracy of **98.8%**.

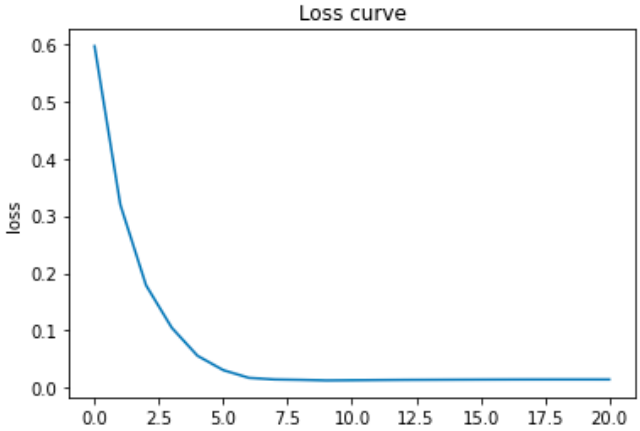

Figure 7: Loss plot for MLP on original data.

We have two hidden layers of sizes 16 and 8, respectively. All hidden layers use the ReLU activation function, and the output layer has a sigmoid activation function. Table 1 shows the parameters of the best model.

Table 1: Summary of the parameters of the best model.

Parameter	Value
Hidden_layer_sizes	(16,8,)
Activation	ReLU
Max_iter	1000
solver	Adam
Learning_rate_init	0.04
alpha	0.02
shuffle	True
momentum	0.9
Learning_rate	adaptive
Tol (tolerance)	1e-6

While changing the hyperparameters, we see that most of the models give approximately the same accuracy for the validation and test sets.

PCA Data – SVM

In this part, we are implementing the SVM algorithm on the data transformed into two dimensions using PCA. Similar to before, we will try all the possible kernels that we can use. Figure 8 shows the plot of the accuracy of the validation set versus the kernels used. As we can see, the RBF kernel gives the best accuracy, i.e., approximately 89%.

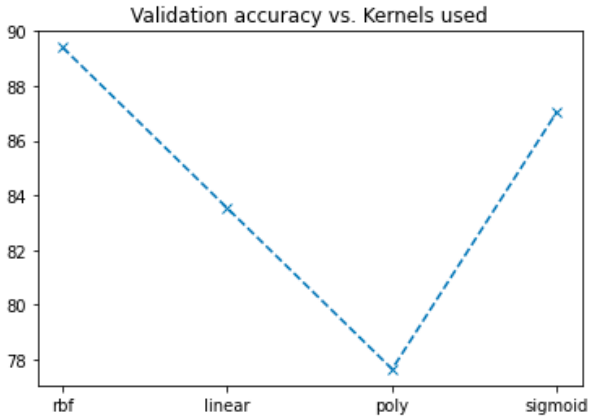

Figure 8: Plot of validation accuracies versus kernel used.

We can iterate over the degree of the polynomial for the polynomial kernel (Figure 9).

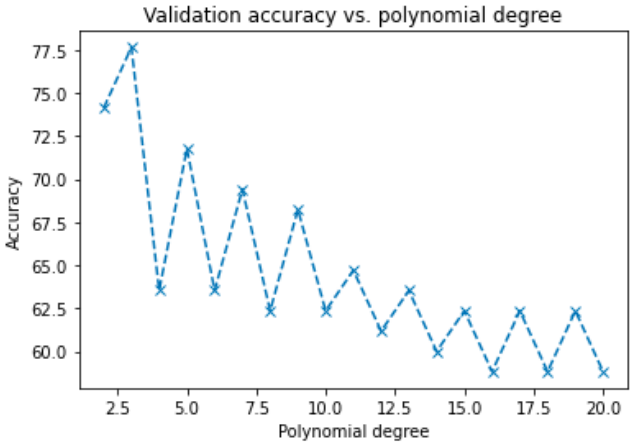

Figure 9: Validation accuracy versus the polynomial degree.

Next, we can also iterate over the possible values for the regularization parameter. From Figure 10 we see that the accuracy increases at specific values of the regularization parameter. Setting this parameter to 350, we get a validation accuracy of **91.8%** and a test accuracy of **87.1%.**

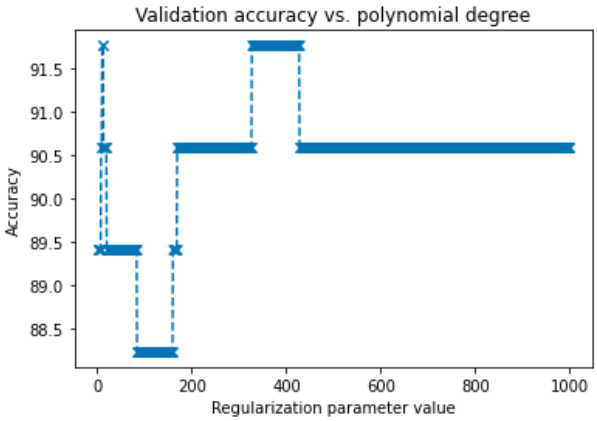

Figure 10: Validation accuracy versus the regularization parameter.

PCA Data – MLP

Similar to before, in this part, we find a configuration that gives the highest accuracy on the PCA data. Figure 11 shows the loss plot of multiple configurations of the MLP model. The validation accuracies that we get for configurations (4,2), (4,4), (2,2), (8,4) are **90.6%**, 88.2%, 88.2%, and 89.4%. Therefore, the best model is the one with two hidden layers with 4 and 2 neurons, respectively. In this model, the test accuracy is **88.2%.**

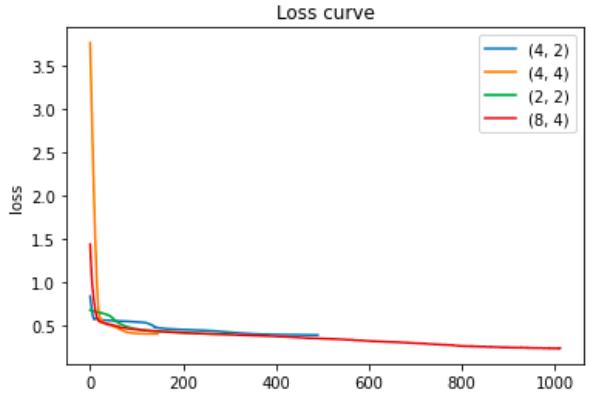

Figure 11: loss plot of MLP classifier configurations training on PCA data.

Table 2 gives the hyperparameter values for the best MLP model.

Table 2: Summary of the configuration of the best MLP model.

Parameter	Value
Hidden_layer_sizes	(4,2)
Activation	ReLU
Max_iter	1000
solver	Adam
Learning_rate_init	0.005
alpha	0.0001
shuffle	True
momentum	0.9
Learning_rate	adaptive
Tol (tolerance)	1e-4

Conclusion

In this project, we look at both unsupervised algorithms for data visualization and dimensionality reduction and supervised algorithms for classification purposes. From the result of the classification algorithms, we see that the original data gives very high accuracies on both SVM and MLP. When we apply PCA and project the data to two dimensions, we see a drop in accuracy. This is because we are losing information during the process of dimensionality reduction as we are only retaining approximately 40% of the variability, and thus the models are not able to learn from the features in order to give a higher classification accuracy. We can conclude that using these models is a good way of implementing fall detection as they give high accuracy.

References

[1] "sklearn.decomposition.PCA -- scikit-learn 1.0.2 documentation," sklearn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

[2] "sklearn.cluster.KMeans -- scikit-learn 1.0.2 documentation," sklearn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

[3] "sklearn.svm.SVC -- scikit-learn 1.0.2 documentation," sklearn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

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

Appendix

```
# Sklearn functions used and References
# [1] https://scikit-
learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
# [2] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
# [3] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
# [4] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# [5] https://scikit-
learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_ne
twork.MLPClassifier

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.cluster import KMeans
from sklearn.neural_network import MLPClassifier



data = pd.read_csv('falldetection_dataset.csv', header = None)
data = data.drop(0, axis = 1)
data = data.replace('F', 1)
data = data.replace('NF', 0)
labels = data[1]
cl = ['NF','F']
data = data.drop(1, axis = 1)
data.head()

scaler = StandardScaler()# [1]
data_norm = scaler.fit(data).transform(data)

pca = PCA()# [2]
pca.fit(data_norm)


plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Principal Component')
plt.ylabel('Explained variance ratio')
plt.title('PCA explained variance ratio vs. PCs')
plt.grid()
plt.show()
print(pca.explained_variance_ratio_[0],pca.explained_variance_ratio_[1])

pca2 = PCA(2)# [2]
```

```
data_pca = pca2.fit_transform(data_norm)

for i in range(2):
  ind = np.where(labels == i)[0]
  plt.plot(data_pca[ind,0],data_pca[ind,1], 'o',label = cl[i],markersize=3)

plt.title('Scatter plot of data projected to two dimensions using PCA.')
plt.legend()
plt.show()

"""#Kmeans"""


N = 2
kmeans = KMeans(n_clusters=N).fit(data_pca)# [3]
for i in range(N):
  ind = np.where(kmeans.labels_ == i)[0]

  plt.plot(data_pca[ind,0],data_pca[ind,1], 'o',label = i, markersize = 3)
  plt.plot(kmeans.cluster_centers_[i,0],kmeans.cluster_centers_[i,1],'o', markersize = 5)


plt.title('Scatter plot of output of Kmeans with N=2')
plt.legend()
plt.show()
sum(kmeans.labels_ == labels.to_numpy())/len(labels)

j = []
for N1 in range(1,21):
  kmeans1 = KMeans(n_clusters=N1).fit(data_pca)# [3]
  j.append(kmeans1.inertia_)

plt.plot(np.arange(1,21),j)
plt.title('Inertia vs. Number of clusters.')
plt.xlabel('number of clusters')
plt.ylabel('Inertia')
plt.show()

"""#PCA Data

##SVM
"""

data_train, data_test, labels_train, labels_test = train_test_split(data_pca, labels,
test_size=0.15)
data_train, data_val, labels_train, labels_val = train_test_split(data_train, labels_train,
test_size=0.176)

kernels = ['rbf','linear', 'poly',  'sigmoid']
```

```
score_list = []
for k in kernels:
  svm = SVC(kernel = k)# [4]
  svm.fit(data_train, labels_train)
  score_list.append(svm.score(data_val, labels_val)*100)
plt.plot(kernels, score_list, 'x--')
plt.title('Validation accuracy vs. Kernels used')
plt.show()

dim = np.arange(2,21)
score_list1 = []
for k in dim:
  svm = SVC(kernel = "poly", degree = k, C=1)# [4]
  svm.fit(data_train, labels_train)
  score_list1.append(svm.score(data_val, labels_val)*100)
plt.plot(dim, score_list1, 'x--')
plt.title('Validation accuracy vs. polynomial degree')
plt.xlabel('Polynomial degree')
plt.ylabel('Accuracy')
plt.show()

dim = np.arange(1,1000)
score_list1 = []
for k in dim:
  svm = SVC(kernel = "rbf",  C=k)# [4]
  svm.fit(data_train, labels_train)
  score_list1.append(svm.score(data_val, labels_val)*100)
plt.plot(dim, score_list1, 'x--')
plt.title('Validation accuracy vs. polynomial degree')
plt.xlabel('Regularization parameter value')
plt.ylabel('Accuracy')
plt.show()

svm = SVC(kernel = "rbf",  C=350)# [4]
svm.fit(data_train, labels_train)
score_list1.append(svm.score(data_val, labels_val)*100)
score_list1[-1]

# Best model is the rbf model
svm = SVC(kernel = "rbf",  C=350)
svm.fit(data_train, labels_train)
score_lin = svm.score(data_test, labels_test)
score_lin = score_lin*100
print(score_lin)



"""##MLP"""
```

```
layers = [(4,2),(4,4),(2,2),(8,4)]
mlp_pca = []
mlp_loss = []

for l in layers:
  clf = MLPClassifier(hidden_layer_sizes=l, activation = 'relu', max_iter = 5000,
                      solver='adam',learning_rate_init = 0.005,alpha = 0.0001,
                      shuffle = True, momentum = 0.9, learning_rate =
'adaptive').fit(data_train, labels_train)# [5]
  print(clf.score(data_val, labels_val))
  mlp_pca.append(clf.score(data_val, labels_val))
  mlp_loss.append(clf.loss_curve_)
  plt.plot(clf.loss_curve_, label = str(l))
plt.title('Loss curve')
plt.ylabel('loss')
plt.legend()
plt.show()

print(clf.score(data_test, labels_test))

"""#Original Data"""

data_train, data_test, labels_train, labels_test = train_test_split(data_norm, labels,
test_size=0.15)
data_train, data_val, labels_train, labels_val = train_test_split(data_train, labels_train,
test_size=0.176)

"""##SVM"""

kernels = ['rbf','linear', 'poly',  'sigmoid']
score_list = []
for k in kernels:
  svm = SVC(kernel = k)# [4]
  svm.fit(data_train, labels_train)
  score_list.append(svm.score(data_val, labels_val)*100)
plt.plot(kernels, score_list, 'x--')
plt.title('Validation accuracy vs. Kernels used')
plt.show()

dim = np.arange(2,16)
score_list1 = []
for k in dim:
  svm = SVC(kernel = "poly", degree = k, C=1)# [4]
  svm.fit(data_train, labels_train)
  score_list1.append(svm.score(data_val, labels_val)*100)
plt.plot(dim, score_list1, 'x--')
```

```
plt.title('Validation accuracy vs. polynomial degree')
plt.show()

# Best model is the linear model or the poly w/ deg 3
svm = SVC(kernel = "linear")# [4]
svm.fit(data_train, labels_train)
score_lin = svm.score(data_test, labels_test)
score_lin = score_lin*100
print(score_lin)


svm = SVC(kernel = "poly", degree = 3)# [4]
svm.fit(data_train, labels_train)
score_poly = svm.score(data_test, labels_test)
score_poly = score_poly * 100
print(score_poly)

"""##MLP"""



clf = MLPClassifier(hidden_layer_sizes=(16,8,), activation = 'relu', max_iter = 1000,
                    solver='adam',learning_rate_init = 0.04,alpha = 0.02,
                    shuffle = True, momentum = 0.9, learning_rate = 'adaptive', tol = 1e-
6).fit(data_train, labels_train)# [5]
clf.score(data_val, labels_val)

clf.score(data_test, labels_test)

plt.plot(clf.loss_curve_)
plt.title('Loss curve')
plt.ylabel('loss')
plt.show()
```