GE461 - Introduction to Data Science
Homework 4
Report

Efe Tarhan

22002840

Electrical and Electronics Engineering

Content

Introduction

The high demand on the data-driven solutions lead industry to a path that requires the intelligent analysis of the big data. For commercial and scientific products and developments, data science has become an integral part of the process. Although there are many methods that are developed for data with specific attributes, real data is quite different from the test datasets that are being used for developing these techniques and benchmarking them.

Number of features give chance to someone for analyze the data using one more spec. Although higher number of features can enable someone to analyze data using more complex relationships, it also creates one of the major problems of data science, curse of dimensionality. In this homework the data consists of 566 samples whereas each sample has 306 features. To tackle the issue of binary classification of the members of the dataset, the dataset must be analyzed and transformed using different machine learning methods like Principal Components Analysis (PCA), k-Means Clustering, Multi-Layer Perceptron (MLP) and Support Vector Machines (SVM).

In the first part of the homework, dimensionality of the dataset is reduced by applying PCA on the whole dataset. PCA is a method of transformation where the new basis vectors of the dataset became the vectors that lie along the directions where maximum variance inside the dataset is observable. Principle components of a given set of data can be found by the following operations for a dataset that has N dimensional vectors i.e. $x_i \in R^N$:

$$[u_1, u_2, \dots, u_N] = eig(X^T X)$$

The vectors u_1 to u_N are eigenvectors of the symmetric matrix $X^T X$ where each of them belongs to R^N. The eigenvalues of the corresponding to the given eigenvectors are $\lambda_1 > \lambda_2 > \cdots > \lambda_N$. Following these operations, the dimensionality reduced dataset can be found by applying the following operation to each data instance.

$$\widetilde{x_j} = \begin{bmatrix} x_j^T u_1 \\ x_j^T u_2 \\ \vdots \\ x_j^T u_k \end{bmatrix}$$

As a result of these operations dimensionality of the dataset can be reduced to k-features where $\widetilde{x_j} \in R^k$.

After applying PCA, k-means clustering will be used for the analysis of the dataset. K-means clustering is an iterative non-parametric unsupervised learning method that can be used for grouping data into K clusters. For this assignment Euclidian distance is preferred as the metric of k-means clustering algorithm. Then the loss function of the algorithm can be given as:

$$J = \sum_{i=1}^{N} \sum_{j=1}^{K} I(x_i \in c_j)(x_i - \mu_j)^2$$

where μ_j is the center of the cluster c_j and $I(.)$ is the indicator function that gives 1 if the given data belongs to the given cluster. It is hard to given optimization task for all dataset at once since it requires the optimization of many parameters at once. Rather, an iterative algorithm is applied with the given steps:

Expectation:

$$I(x_i \in c_j) = \begin{cases} 1, & c_{\underset{m}{\operatorname{argmin}}(x_i - \mu_m)^2 \ m \in \{1,2,\ldots,k\}} \\ 0, & else \end{cases}$$

Maximization:

$$\mu_j = \frac{\sum_{i=1}^{N} I(x_i \in c_j) x_i}{\sum_{i=1}^{N} I(x_i \in c_j)}$$

From this perspective it is guaranteed that the algorithm will converge to a local minimum point after required number of iterations. This algorithm will be used for partitioning the dataset into desired number of clusters in Part A.

In Part B, 2 different supervised machine learning techniques are used for creating classifiers for the dataset. First method is the SVM which is a technique that relies on the method of transforming data using a nonlinear method to achieve separability. There are several different “kernels” like “polynomial”, “radial” or linear that are being preferred for applying support vector machine algorithm on data with different structures. After transforming the data using the defined nonlinear relationships, the data is then separated using the best separating hyperplane and support vectors around it. There are several different hyperparameters of the support vector machine algorithm which will be tried during the homework.

The second algorithm that will be used for the supervised classification of the data is the multilayer perceptron (MLP). MLP is also known as the deep learning that only consists of layers with fully connected perceptrons. These networks learn the complex relationships of the data by applying and optimizer algorithm and using a gradient based optimization method like stochastic gradient descent (SGD) or adaptive momentum (ADAM). There are also hyperparameters to explore like the number of epochs, learning rate, model architecture, etc. Output of a neural network can be found by passing the input data from the architecture layer by layer by multiplying the input with corresponding weights and passing the result through nonlinear activation functions. The results of applying these methods will be investigated in the following sections.

Part A - PCA and K-Means Clustering

At the first part of the report the dataset is initially standardized before being transformed using the PCA algorithm. Following this operation, dimension of the dataset is reduced to 2 by using the 2 principal components with largest eigenvalues. Result of changing the basis of the dataset with the given 2 eigenvectors and plotting each data in 2D is shown in Figure 1.

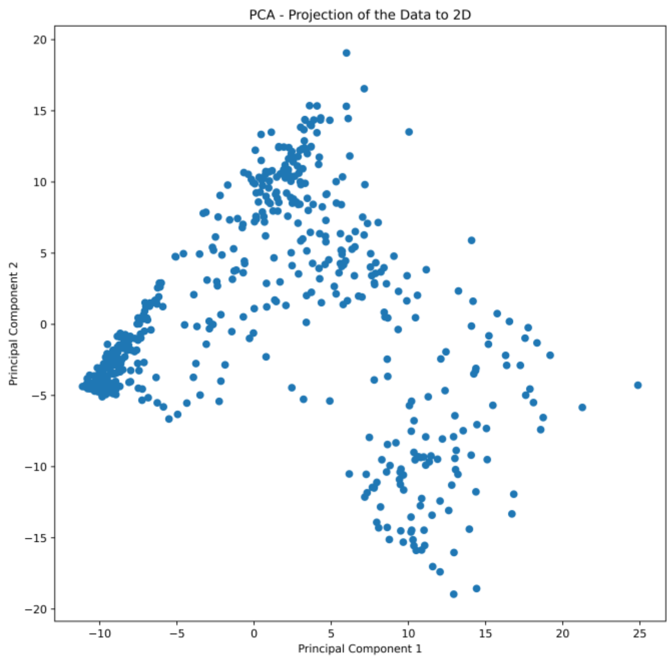


Fig.1 Transformation of the dataset to the 2D space using the eigenvectors.

After obtaining the new dataset using PCA, the data points are grouped using the k-means clustering algorithm. Since it is supervised, the data is tried to be separated using different number of clusters and the best one that fits the data is observed. Results of running the k-means algorithm for $k = 2, \ldots, 10$ is shown in Figure 2.

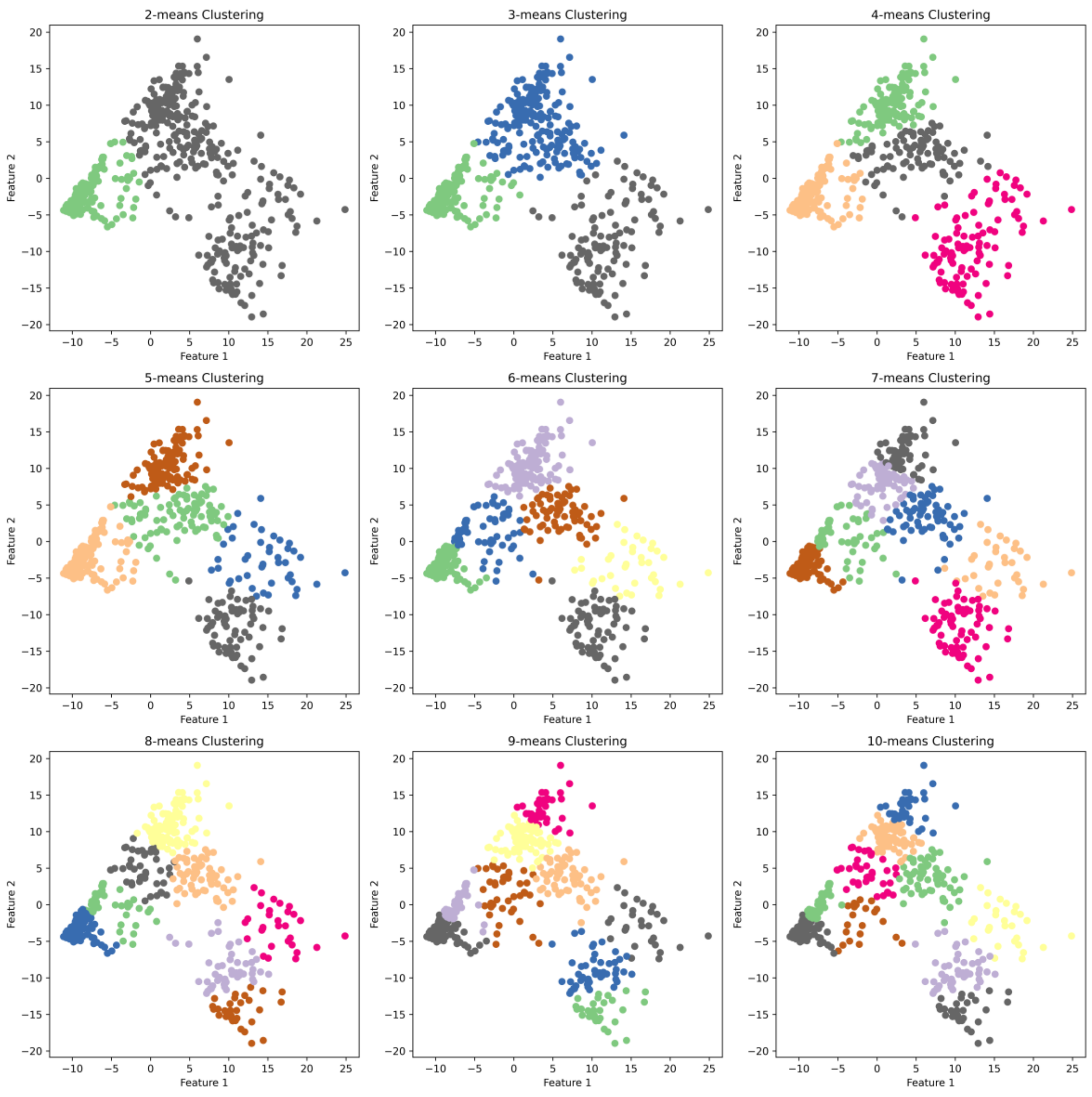


Fig.2 Clustering results of k-means algorithm for different values of k.

By observing the results, it can be stated that separating the dataset into clusters using 2 or 3 centroids give meaningful cluster boundaries whereas going beyond 4 clusters create regions uncorrelated with the inherent information of the data.

Following these observations the relationship between the classes obtained by grouping the data into 2 clusters with k-means clustering algorithm and actual labels of the data are compared. The result of 2-means clustering can be seen from Figure 3 and dataset with the actual labels is shown in Figure 4.

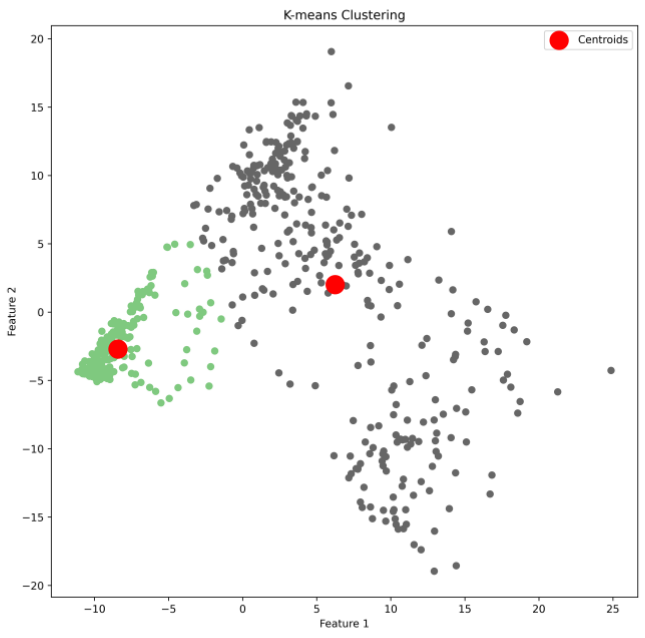


Fig.3 Result of applying k-means clustering on the dataset to obtain 2 classes.

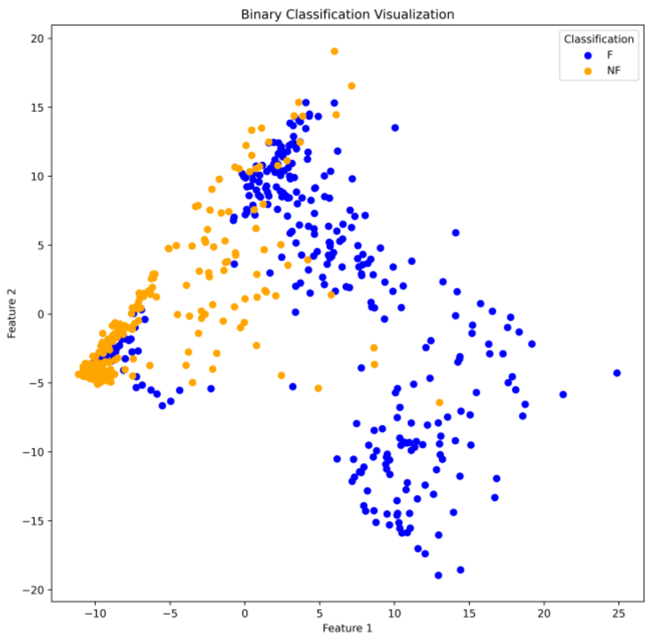


Fig.4 Result of applying k-means clustering on the dataset to obtain 2 classes.

By observing the overlapping regions between the two regions, the following plots are obtained where the class 0 corresponds to “NF” and class 1 corresponds to “F”. The confusion matrix of the results of this classification technique can be seen in Figure 5 and the corresponding performance metrics are shown in Figure 6.

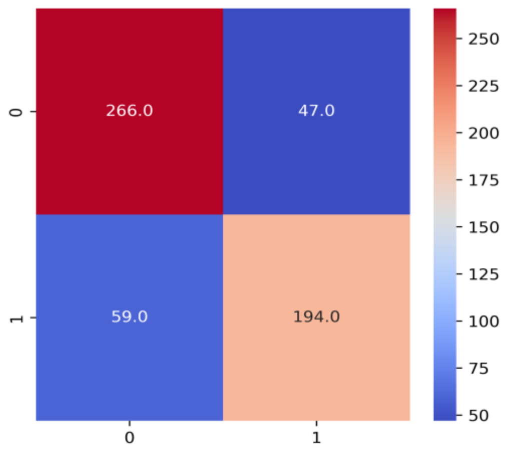
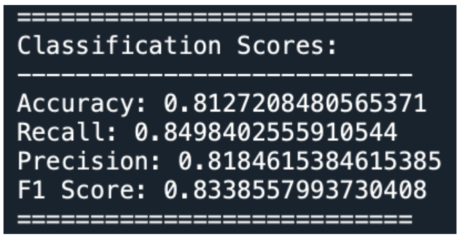


Fig.5 Confusion matrix of the proposed classification approach in homework guideline



```
===========================
Classification Scores:
---------------------------
Accuracy: 0.8127208480565371
Recall: 0.8498402555910544
Precision: 0.8184615384615385
F1 Score: 0.8338557993730408
===========================
```

Fig.6 Accuracy, recall, precision and F1 score metrics of the classification performance.

By observing the classification scores given in Figure 6. The model performs evenly because it performs good not only in terms of accuracy but also detection or misclassification errors and their average which is F1 score. From these values it can be argued that this technique can be used for fall detection because 80% accuracy is satisfying and can be considered as successful, but it also must be noted that especially in the medical sector, the algorithms might require higher accuracy promising because they can be used for safety critical systems or healthcare.

Part B – SVM and MLP for Supervised Learning

For this part, the dataset is directly used for supervised learning algorithm described in the introduction part which are the support vector machine and multilayer perceptron classifier.

First method that will be investigated is SVM where the results are demonstrated by mentioning the used hyperparameters etc. A range of hyperparameters are defined for the k-fold cross validation process. Linear, radial, polynomial and sigmoidal kernels are being tested for the kernels. Another hyperparameter of the support vector classifier is the “C” parameter. This parameter determines the trade-off between the training error and the width of the margin. Larger values of “C” leads to smaller margins. The values 0.001, 0.01, 0.1, 1, 10, 100 and 1000 are tested for the hyperparameter “C”. A 20-fold cross validation scheme has been used for the hyperparameter search. Results of the average validation accuracy can be seen from Figure 7.

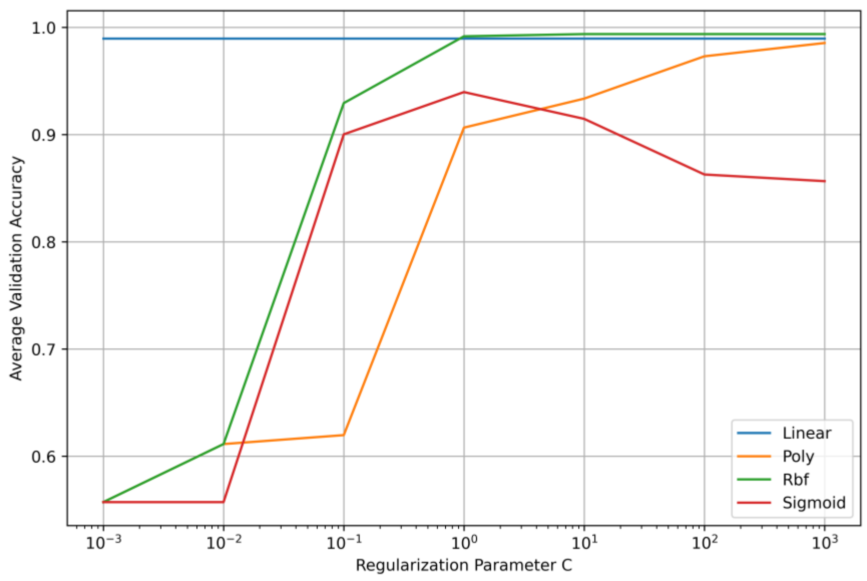


Fig.7 Validation accuracy for different kernels and regularization parameters.

From the results the highest accuracy is satisfied using a linear kernel where it outperforms other methods significantly for smaller values of the regularization parameter and for heavy regularization. According to the code that employs 20-fold cross validation the best SVC employs the parameter values “linear” for the kernel and 0.01 for the regularization parameter C. The performance of the best support vector classifier on the fixed test set given as %15 of the initial dataset is %98.82 which is a remarkable performance. Also, this result is correlated with the result that has been obtained from the PCA. According to results of the PCA, the data is quite linearly separable with projection of the data to a 2-dimensional space. The SVM uses all features and therefore can capture a better “best separating hyperplane” which might be the reason why the “linear” kernel appeared to be the best kernel.

Second method that will be used for the supervised learning task is the multilayer perceptron which is also known as the deep learning with only fully connected layers inside the architecture. Different architectures, learning rates and optimizers are tested, where the best hyperparameter set among these are selected using k-fold cross validation. Three different learning rates are tested which are 0.001, 0.01 and 0.1. For the optimizers, 2 of the common deep learning optimizers are tested, SGD and ADAM. For architectures 4 different neural network architectures are tested with layer structures 1 hidden layer with 50 neurons, 1 hidden layer with 100 neurons, 2 hidden layers with 50 neurons each and lastly 2 hidden layers with 100 neurons each. Results of the validation accuracy obtained by trying different hyperparameters can be seen from Figure 8.

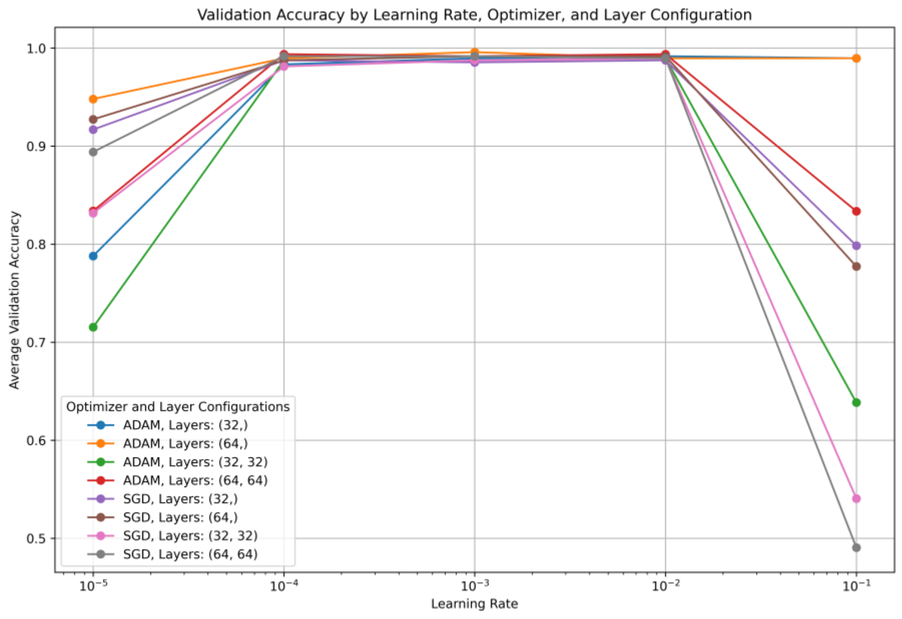


Fig.8 Validation accuracies for different optimizers, model architectures and learning rates.

Among these models the best result has been selected using a grid search method and 5-fold cross validation. The best result has been obtained by setting the learning rate to 0.001, optimizer to ADAM and model architecture to 1 hidden layer with 64 neurons. Decrease of the training loss per each epoch for the selected best model can be seen from Figure 9.

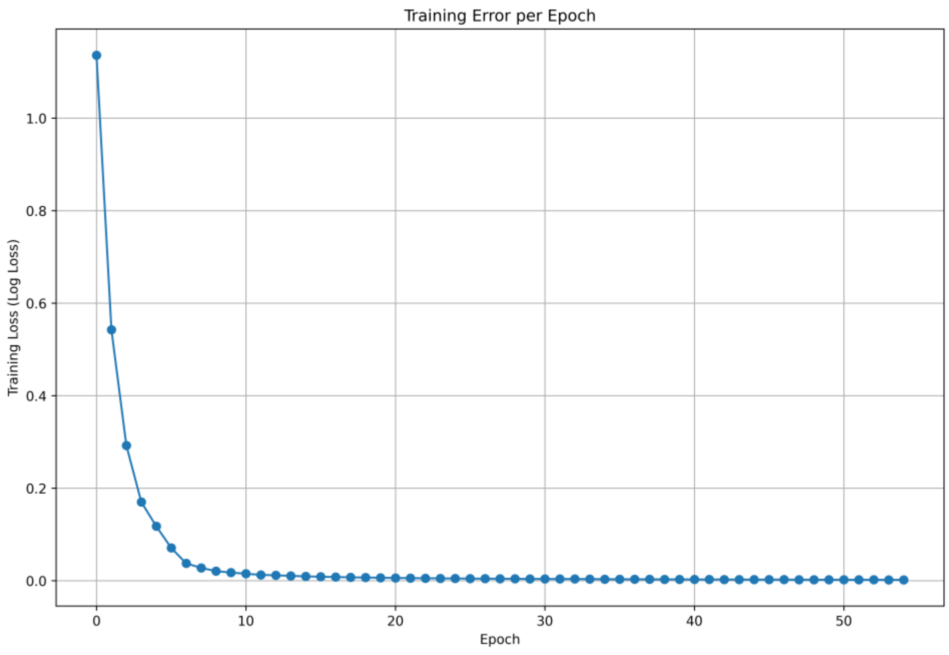


Fig.9 Training loss per epoch of the selected best model.

Test set accuracy for the best model is %98.82 like the result of the SVM.

Although results of MLP and SVM are similar this happened due to the small number of data that can be allocated for testing purposes. Also, the MLP algorithm has taken a lot more time to train and learn the data compared to SVM because of the more complex and iterative gradient based optimizers. Whereas for this problem it can be argued that MLP structure is a bit unnecessary and cause inefficient use of the resources since the data can be mostly linearly separable whereas the role of MLP is to capture the nonlinear relationships inside the dataset. This result can also be verified by observing that the best kernel for the support vector classifier was the linear kernel which does not apply and nonlinear transformation to the data.

Conclusion

Aim of this homework was to observe the real-world applications of several supervised and unsupervised learning techniques. Understanding the effects of selecting different hyperparameters for each of these methods was another outcome of the homework. From the results it is inferred that dataset with large number of features compared to the amount of data inside the dataset can be analyzed if the data contains adequate information for the classification task. Overall, it was an insightful homework and experiment to work with this data using different techniques.

Appendix

```
#%% Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split,KFold
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, log_loss
import warnings
warnings.filterwarnings("ignore")
#%% Read the data
df = pd.read_csv('falldetection_dataset.csv', header=None)
data = df.values
data = data[:,1:]
X,y = data[:,1:],data[:,0]
#%% PART A - PCA

scaler = StandardScaler()
data_scaled = scaler.fit_transform(X)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(data_scaled)

plt.figure(figsize=(10, 10),dpi = 600)
plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA - Projection of the Data to 2D')
plt.show()
#%% Creating different number of clusters using the dataset

clusters = [KMeans(n_clusters=i, random_state=42).fit_predict(X_pca) for i in
range(2,11)]
fig, axes = plt.subplots(3, 3, figsize=(15, 15),dpi = 600)
for i in range(3):
    for j in range(3):
        axes[i, j].scatter(X_pca[:, 0], X_pca[:, 1], c=clusters[i*3+j], cmap='Accent')
        axes[i, j].set_title(f'{i*3+j+2}-means Clustering')
        axes[i, j].set_xlabel('Feature 1')
        axes[i, j].set_ylabel('Feature 2')

plt.tight_layout()
plt.show()

#%% Result of the k means on PCA
kmeans = KMeans(n_clusters=2, random_state=42)
clusters = kmeans.fit_predict(X_pca)
plt.figure(dpi = 600, figsize = (10,10))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap='Accent')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300,
c='red', label='Centroids')
plt.title('K-means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

```
#%% Acutal data with PCA
y_bin = y == 'F'

plt.figure(dpi=600, figsize=(10, 10))

plt.scatter(X_pca[y_bin, 0], X_pca[y_bin, 1], color='blue', label='F')
plt.scatter(X_pca[~y_bin, 0], X_pca[~y_bin, 1], color='orange', label='NF')

plt.title('Binary Classification Visualization')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.legend(title="Classification")

plt.show()
#%% Overlapping with acutal

s11 = np.sum((y =='NF') & (clusters == 0))
s12 = np.sum((y =='NF') & (clusters == 1))
s21 = np.sum((y =='F') & (clusters == 0))
s22 = np.sum((y =='F') & (clusters == 1))

plt.figure(figsize=(5, 5),dpi = 600)
sns.heatmap([[s11,s12],[s21,s22]], annot=True, cmap='coolwarm',fmt=".1f" )
plt.show()
acc = (s11+s22)/(s11+s12+s21+s22)
recall = s22 / (s22 + s21)
precision = s22 / (s22 + s12)
f1 = 2*precision*recall / (precision + recall)
print("========================")
print("Classification Scores: ")
print("------------------------")
print("Accuracy: {}".format(acc))
print("Recall: {}".format(recall))
print("Precision: {}".format(precision))
print("F1 Score: {}".format(f1))
print("============================")

#%% PART B: SVM

# Split data into train+validation (85%) and test (15%)
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.15,
random_state=42)

# Parameters for cross-validation
n_splits = 20
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

# Initialize the SVM classifier parameters
kernel = ['linear', 'poly', 'rbf', 'sigmoid']
lr = np.array([1e-3, 1e-2, 1e-1, 1, 10, 100, 1000])
accuracy_dict = {k: np.zeros(len(lr)) for k in kernel}

# Cross-validation to find the best parameters
for train_index, val_index in kf.split(X_train_val):
    X_train, X_val = X_train_val[train_index], X_train_val[val_index]
    y_train, y_val = y_train_val[train_index], y_train_val[val_index]
    for k in kernel:
        for idx, C in enumerate(lr):
            svm = SVC(C=C, kernel=k)
            svm.fit(X_train, y_train)
            y_val_pred = svm.predict(X_val)
            acc = accuracy_score(y_val, y_val_pred)
            accuracy_dict[k][idx] += acc

# Average the accuracies over the folds
for k in kernel:
    accuracy_dict[k] /= n_splits
```

```
plt.figure(dpi=600, figsize=(9, 6))
for k in kernel:
    plt.semilogx(lr, accuracy_dict[k], label=k.capitalize())
plt.grid(True)
plt.xlabel("Regularization Parameter C")
plt.ylabel("Average Validation Accuracy")
plt.legend()
plt.show()

best_kernel, best_c_idx = max(((k, np.argmax(v)) for k, v in accuracy_dict.items()),
key=lambda x: accuracy_dict[x[0]][x[1]])
best_c = lr[best_c_idx]

best_svm = SVC(C=best_c, kernel=best_kernel)
best_svm.fit(X_train_val, y_train_val)
y_test_pred = best_svm.predict(X_test)
test_acc = accuracy_score(y_test, y_test_pred)
print(f"Test Accuracy with best C={best_c} and kernel={best_kernel}: {test_acc}")

#%% Part B: MLP

# Split data into train+validation (85%) and test (15%)
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.15,
random_state=42)

# Parameters for cross-validation
n_splits = 5
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

# Define hyperparameters to test
learning_rates = [0.00001,0.0001, 0.001, 0.01, 0.1]
optimizers = ['adam', 'sgd']
layer_sizes = [(32,), (64,), (32, 32), (64, 64)]
```

```
fixed_epochs = 400

# Initialize dictionary to store accuracies
accuracy_dict = {(lr, opt, ls): [] for lr in learning_rates for opt in optimizers for ls in
layer_sizes}

# Cross-validation to find the best parameters
for train_index, val_index in kf.split(X_train_val):
    X_train, X_val = X_train_val[train_index], X_train_val[val_index]
    y_train, y_val = y_train_val[train_index], y_train_val[val_index]
    for lr in learning_rates:
        for opt in optimizers:
            for ls in layer_sizes:
                mlp = MLPClassifier(hidden_layer_sizes=ls, learning_rate_init=lr,
max_iter=fixed_epochs, solver=opt, random_state=42)
                mlp.fit(X_train, y_train)
                y_val_pred = mlp.predict(X_val)
                acc = accuracy_score(y_val, y_val_pred)
                accuracy_dict[(lr, opt, ls)].append(acc)

# Average the accuracies over the folds
for key in accuracy_dict.keys():
    accuracy_dict[key] = np.mean(accuracy_dict[key])

# Plot Learning Rate vs. Average Validation Accuracy
plt.figure(dpi=600, figsize=(12, 8))
for opt in optimizers:
    for ls in layer_sizes:
        accuracies = [accuracy_dict[(lr, opt, ls)] for lr in learning_rates]
        plt.semilogx(learning_rates, accuracies, marker='o', label=f"{opt.upper()},
Layers: {ls}")

plt.title("Validation Accuracy by Learning Rate, Optimizer, and Layer Configuration")
plt.xlabel("Learning Rate")
plt.ylabel("Average Validation Accuracy")
plt.legend(title="Optimizer and Layer Configurations", loc='best')
plt.grid(True)
plt.show()

best_params = max(accuracy_dict, key=accuracy_dict.get)
best_lr, best_optimizer, best_layers = best_params

best_mlp = MLPClassifier(hidden_layer_sizes=best_layers,
learning_rate_init=best_lr, max_iter=fixed_epochs, solver=best_optimizer,
random_state=42, verbose=True)
best_mlp.fit(X_train_val, y_train_val)

training_losses = best_mlp.loss_curve_
plt.figure(dpi=600, figsize=(12, 8))
plt.plot(training_losses, marker='o', linestyle='-')
plt.title("Training Error per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Training Loss (Log Loss)")
plt.grid(True)
plt.show()

y_test_pred = best_mlp.predict(X_test)
test_acc = accuracy_score(y_test, y_test_pred)
print(f"Test Accuracy with best params - LR: {best_lr}, Optimizer: {best_optimizer},
Layers: {best_layers}: {test_acc}")
```