Mustafa Cankan BALCI

22101761

11 May 2025

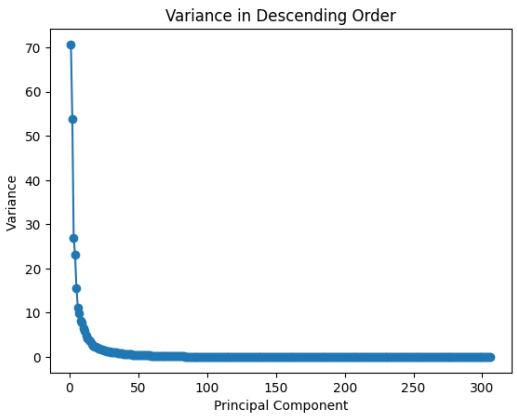
GE461: Introduction to Data Science Project 4

Introduction

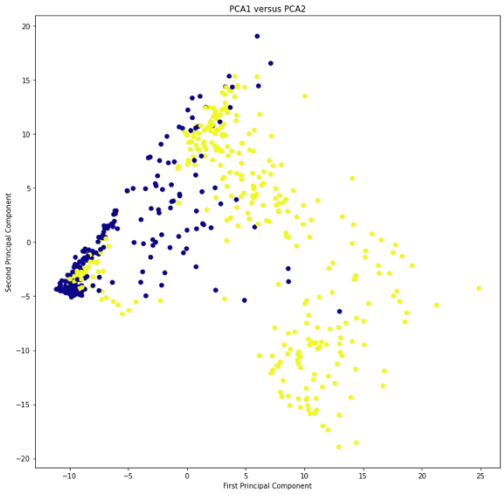
This assignment focuses on analyzing various machine learning models using a wearable sensor data collected from a group of subjects performing either a fall action (F) or non-fall action (NF). The dataset consists of 566 sample of motor actions along, each labeled as either F and NF, and includes 306 sensor-derived features capturing different properties such as velocity acceleration, and temperature. In the first part of assignment, clustering method is applied to the dataset after reducing its dimensionality using principal component analysis (PCA). In the final part, classification is performed using supervised learning methods, support vector machine (SVM) and multilayer perceptron (MLP).

Part A

The first part of assignment, exploratory data is analyzed by clustering. The dimension of data set is large to visualize, so that PCA is performed to dimension reduction. In the first step of PCA, dataset is standardized. Graph 1 shows the absolute variance in descending order. Top 2 PCs' variance captures are in table 1. Totally, 40.73% of variance is captured.



Graph 1: Variance in Descending Order

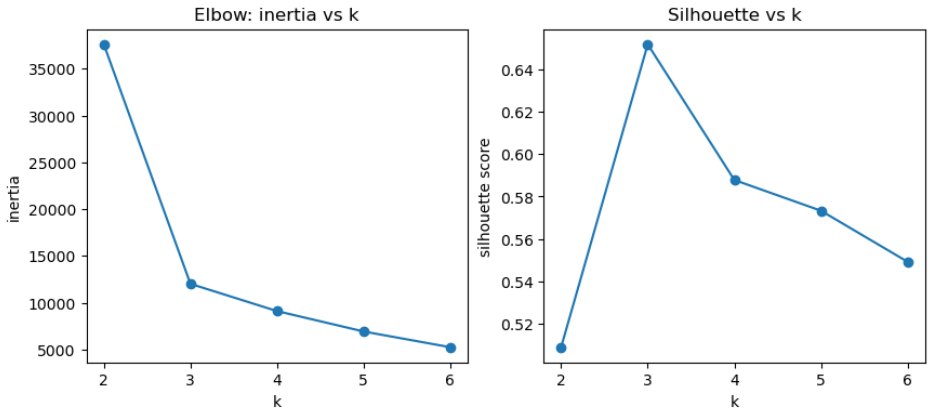


Graph 2: PCA Visualization

PCA 1	23.10 %
PCA 2	17.62%

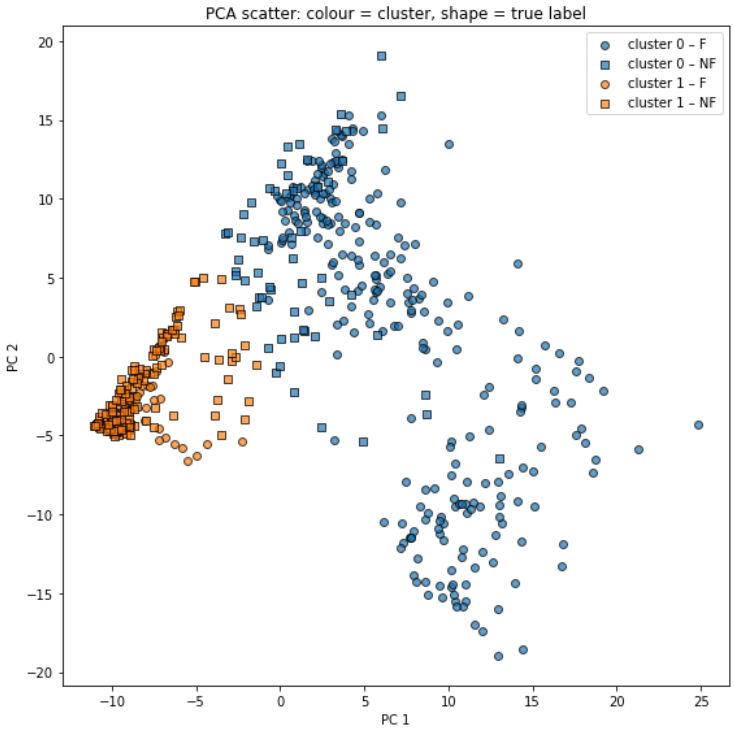
Table 1: Top 2 PCA Variance Captures

Moreover, k-means clustering performed on the PCA. For finding optimal k value, elbow and silhouette method is applied. Elbow method's purpose is to determine the optimal number of K by looking within-cluster sum of squares (WCSS), which is also called inertia. Additionally, silhouette method evaluates the quality of clustering by measuring how well each point lies within its cluster compared to other clusters. From the results, k = 3 will be best performing model among other k values.



Graph 3: Elbow and Silhoutte Graph

The result of the k-means clustering when N=2 performs promising yet imperfect fall detection performance. Graph _ shows the prediction outputs. The overall overlap is 81.27 %. In the cluster 0 and 1, purities are 81.85 % and 80.50 % respectively. While model shows potential for unsupervised separation of falls, its current accuracy may be insufficient for critical applications.



Graph 4: K-Means Result

Overall Overlap	81.27 %
Cluster 0: Purity	81.85%
Cluster 1: Purity	80.50 %

Table 2: Percantage / Overlap Result

Part B

In this part, the supervised learning models is used to make classifier model. The goal of machine learning models in part b is to build a classifier that detects the action labels such as fall or non-fall with high accuracy. 2 different approaches are implemented in part B: support vector machine and multi-layer perceptron (MLP) classifier. The data separated into training, validation and test 70%, 15% ,15% in orderly without overlapping.

Training Set Size	(396, 306)
Validation Set Size	(85,306)
Test Set Size	(85,306)

Table 3: Train/Validation/Test Size

Each model with various different hyperparameters are fit on the training set and selected based on the performance on validation set. The hyperparameters for SVM and MLP are shown table 4 and 5.

C	[0.1, 1]
Kernel	[‘rbf’]
Gamma	[‘scale’, ‘auto’]

Table 4: SVM Hyperparameters

Hidden Layer Number	[(2,), (10,)]
Activation Function	[‘logistic’,’relu’]
Alpha	[1e-4, 1e-3, 1e-2]
Initialized Learning Rate	[1e-3, 1e-2]

Table 5: MLP Hyperparameters

Moreover, the cross validation performed on both models. The models with various hyperparameters are fit on the training set, the parameter selection is performed based on the validation set. All validation accuracy values high, the least complex model is selected.

C	Kernel	Gamma	Validation Accuracy
1.0	rbf	Scale	1.00
1.0	Rbf	Auto	1.00
0.1	Rbf	Scale	0.9647
0.1	Rbf	auto	0.9647

Table 6: Train Result for SVM

Hidden Layer Number	Activation	Alpha	Initial Learning Rate	Validation Accuracy
(2,)	Logistic	0.0001	0.001	1.00
(2,)	Logistic	0.0001	0.010	1.00
(10,)	Relu	0.0100	0.001	1.00
(10,)	Relu	0.0010	0.010	1.00
(10,)	Relu	0.0010	0.001	1.00
(10,)	Relu	0.0001	0.010	1.00
(10,)	Logistic	0.0100	0.010	1.00
(10,)	Logistic	0.0100	0.001	1.00
(10,)	Logistic	0.0010	0.010	1.00
(10,)	Logistic	0.0010	0.001	1.00
(10,)	Logistic	0.0001	0.010	1.00
(10,)	Logistic	0.0001	0.001	1.00
(2,)	Relu	0.0100	0.010	1.00
(2,)	Relu	0.0010	0.010	1.00
(2,)	Relu	0.0001	0.010	1.00
(2,)	Logistic	0.0100	0.010	1.00
(2,)	Logistic	0.0100	0.010	1.00
(2,)	Logistic	0.0010	0.001	1.00
(2,)	Logistic	0.0010	0.010	1.00
(10,)	Relu	0.0100	0.001	0.9765
(2,)	Relu	0.0100	0.010	0.9765
(2,)	Relu	0.0010	0.001	0.9765
(2,)	Relu	0.0001	0.001	0.9765

Table 7: Train Result for MLP

Afterwards, best performed models selected from cross-validation and ran on the test set. Table 8 and 9 shows the performance valuation scores on test data. Depending on the result, both models worked very well on unseen data. However, MLP work slightly better than SVM.

Accuracy	98.92 %
Precision	99 %
Recall	99 %
F1 Score	99 %

Table 8: Test Parameters Result for SVM

Accuracy	100 %
Precision	100 %
Recall	100 %
F1 Score	100 %

Table 9: Test Parameters Result for MLP

Appendix

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report

from sklearn.metrics import silhouette_score
from sklearn.model_selection import train_test_split

from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

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

# Droping the first column
df = df.drop(columns=[0])
df.shape

dataset_y = df[1]

dataset_x = df.drop(columns=[1])

dataset_y_changed = dataset_y.replace({"F":1,"NF":0})

features_dataset_mean = dataset_x.mean(axis=0)
features_dataset_std = dataset_x.std(axis=0)
X_centered =( dataset_x - features_dataset_mean) / features_dataset_std

# Extract N =  2
pca_kmeans = PCA(n_components=2)
pca_x = pca_kmeans.fit_transform(X_centered)

# how much variance do PC1 and PC2
ratios = pca_kmeans.explained_variance_ratio_

print(f"PC1 explains {ratios[0] * 100:.2f}% of the total variance")
print(f"PC2 explains {ratios[1] * 100:.2f}% of the total variance")
print(f"Cumulative (PC1 + PC2): {ratios.sum() * 100:.2f}% of the total variance")

# giving a larger plot


plt.figure(figsize=(8, 6))
```

```
plt.scatter(pca_x[:, 0], pca_x[:, 1],
            c=dataset_y_changed,
            cmap='plasma')

# giving a larger plot
plt.figure(figsize=(12, 12))

plt.scatter(pca_x[:, 0], pca_x[:, 1],
            c=dataset_y_changed,
            cmap='plasma')

# labeling x and y axes
plt.title("PCA1 versus PCA2")
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.show()


inertias = []
silhouettes = []
Ks = range(2, 7)

for k in Ks:
    km = KMeans(n_clusters=k, random_state=0).fit(pca_x)
    inertias.append(km.inertia_)
    silhouettes.append(silhouette_score(pca_x, km.labels_))

fig, ax = plt.subplots(1,2, figsize=(10,4))
ax[0].plot(Ks, inertias, '-o');      ax[0].set(title="Elbow: inertia vs k",
xlabel="k", ylabel="inertia")
ax[1].plot(Ks, silhouettes, '-o');  ax[1].set(title="Silhouette vs k", xlabel="k",
ylabel="silhouette score")
plt.show()


km2   = KMeans(n_clusters=2, random_state=42, n_init=10)
cl2   = km2.fit_predict(pca_x)          # cluster memberships (0 or 1)
labels_true = dataset_y                         # your 'F' / 'NF' vector

cluster_to_label = {}
for cid in np.unique(cl2):
    majority = pd.Series(labels_true[cl2 == cid]).mode()[0]
    cluster_to_label[cid] = majority

labels_pred = np.vectorize(cluster_to_label.get)(cl2)
```

```
overall_acc = accuracy_score(labels_true, labels_pred) * 100
print(f"Overall overlap = {overall_acc:.2f} %")


cm = confusion_matrix(labels_true, labels_pred, labels=['F', 'NF'])
print("\nConfusion matrix (rows = true, cols = predicted):")
print(pd.DataFrame(cm, index=['F', 'NF'], columns=['F', 'NF']))

for cid in np.unique(cl2):
    mask      = cl2 == cid
    purity    = (labels_true[mask] == labels_pred[mask]).mean() * 100
    print(f"Cluster {cid}: purity = {purity:.2f} % "
          f"({mask.sum()} samples, majority = '{cluster_to_label[cid]}')")


marker_dict = {'F': 'o', 'NF': 's'}   # fall = circle, non-fall = square
color_dict  = {0: 'C0', 1: 'C1'}      # cluster ID → colour

plt.figure(figsize=(8, 8))
for cid in np.unique(cl2):
    for lab in ['F', 'NF']:
        mask = (cl2 == cid) & (labels_true == lab)
        plt.scatter(pca_x[mask, 0], pca_x[mask, 1],
                    c=color_dict[cid],
                    marker=marker_dict[lab],
                    edgecolor='k', alpha=0.7,
                    label=f'cluster {cid} – {lab}')

plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.title('PCA scatter: colour = cluster, shape = true label')
# de-duplicate identical legend entries
handles, labels = plt.gca().get_legend_handles_labels()
uniq = dict(zip(labels, handles))
plt.legend(uniq.values(), uniq.keys())
plt.tight_layout()
plt.show()

m = dataset_x.shape[0]
train_size = 0.7* m

validation_size = test_size = 0.15*m


print("Train Size" ,train_size)
print("Validation Size" ,validation_size)
print("Test Size" ,test_size)
```

```
X_train, X_tmp, y_train, y_tmp = train_test_split(
    X_centered, dataset_y_changed, test_size=0.30, random_state=42,
stratify=dataset_y_changed
)

X_val, X_test, y_val, y_test = train_test_split(
    X_tmp, y_tmp, test_size=0.50, random_state=22, stratify=y_tmp
)

print("Sizes:",  X_train.shape, X_val.shape, X_test.shape)

# Hyperparameters
svm_param_grid = {
    'C':       [0.1, 1],
    'kernel':  ['rbf'],
    'gamma':   ['scale','auto']
}
mlp_param_grid = {
    'hidden_layer_sizes': [(2,), (10,)],
    'activation': ['logistic','relu'],
    'alpha':              [1e-4, 1e-3, 1e-2],
    'learning_rate_init': [1e-3, 1e-2],

}

def tune_model(ModelClass, param_grid, X_tr, y_tr, X_va, y_va):
    records = []
    for params in (dict(zip(param_grid, v)) for v in
                   __import__('itertools').product(*param_grid.values())):
        model = ModelClass(**params, random_state=0)
        model.fit(X_tr, y_tr)
        y_pred = model.predict(X_va)
        acc = accuracy_score(y_va, y_pred)
        records.append({**params, 'val_acc': acc})
    return pd.DataFrame.from_records(records)


svm_results = tune_model(SVC,  svm_param_grid, X_train, y_train, X_val, y_val)
mlp_results = tune_model(MLPClassifier, mlp_param_grid, X_train, y_train, X_val,
y_val)

print("Top SVM configs:\n", svm_results.sort_values('val_acc',
ascending=False).head())
print("Top MLP configs:\n", mlp_results.sort_values('val_acc', ascending=False))
```

```
best_svm = svm_results.loc[svm_results['val_acc'].idxmax()].to_dict()
best_mlp = mlp_results.loc[mlp_results['val_acc'].idxmax()].to_dict()


best_svm_model = SVC(**{k:best_svm[k] for k in svm_param_grid}, random_state=0)
best_mlp_model = MLPClassifier(**{k:best_mlp[k] for k in mlp_param_grid},
random_state=0)


best_svm_model.fit(X_train, y_train)
best_mlp_model.fit(X_train, y_train)


for name, model in [('SVM', best_svm_model), ('MLP', best_mlp_model)]:
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print(f"\n=== {name} on TEST ===")
    print(f"Accuracy: {acc:.2%}")
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))
```