GE 461: Introduction to Data Science

Spring 2024

Project: Fall Detection

The Dataset

The dataset consists of **566** data points labeled either "**Fall**" or "**No fall**". Each data point has **306** continuously valued features. There are **313** "Fall", **253** "No fall" data points. There are no NaN values in the dataset.

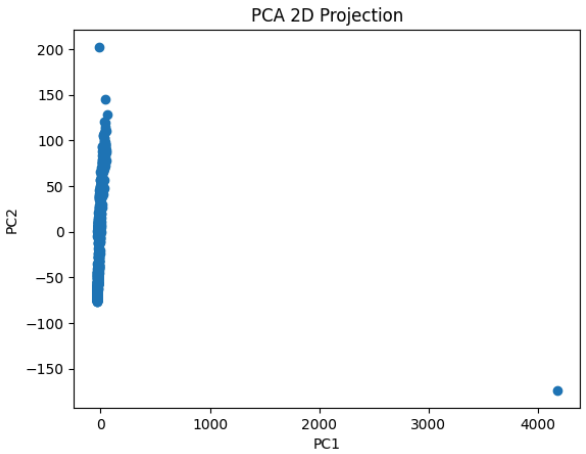

Figure 1: The dataset visualized using PCA for dimensionality reduction.

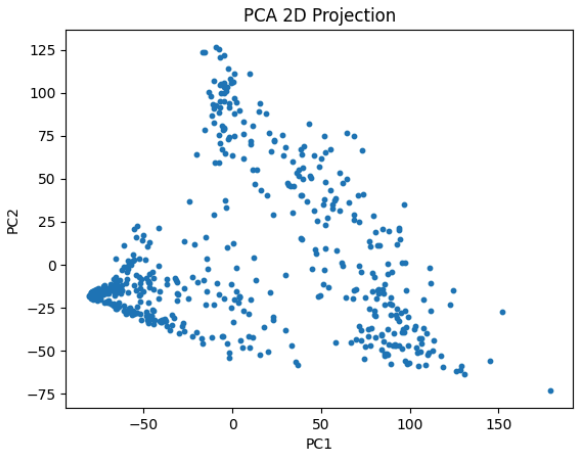

Figure 2: The modified dataset visualized using PCA for dimensionality reduction.

As can be easily observed from figure 1, the dataset contains an obvious **outlier**. Removing this outlier data point will no doubt improve our results. The index of the offending data point is **503** when using 0 based indexing. As presented in figure 2, the modified dataset without the outlier will be used for the remainder of this project.

Part A)

The two dimensional representation of the dataset obtained by using PCA can be seen in figure 2. Color coding the data points according to their labels can help us understand the distributions of the classes.

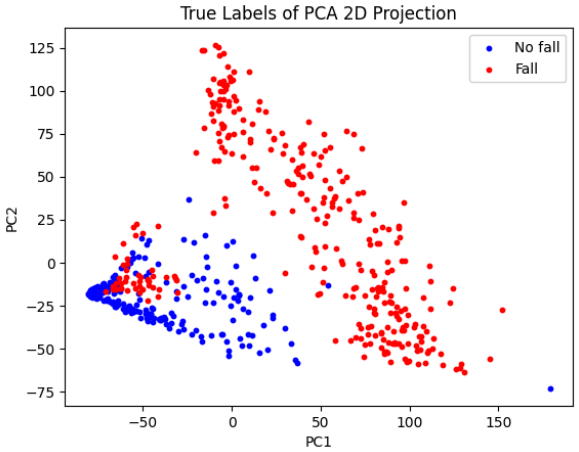

Figure 3: Color coded 2D PCA visualization.

PCA

Already visualized in figures 1, 2, and 3, PCA is a method for creating orthogonal new features from the original features while preserving the most amount of variance in the least amount of PCs. Since we are using PCA for visualization purposes, we take the first two PCs of the transformation. For transformations of figures 1 and 2, the proportions of variance explained are as follows:

Proportion of variance explained by ...	PC1	PC2	PC1 + PC2
Original Dataset	75.3%	8.5%	83.8%
Modified Dataset	36.9%	17.2%	54.1%

Table 1: Proportion of variance explained by various datasets and PCs.

From table 1, it can be seen that the first two principal components of the original dataset explain a **larger proportion** of the variance compared to the first two principal components of the modified dataset.

So are we getting worse performance by excluding the outlier from the dataset? **No**. The reported proportion of the variance explained numbers of the original dataset is **misleading**. The outlier skews the results and makes the dataset appear simpler than it really is, resulting in PCA putting too much weight on just one outlier and reporting better performance. If PCA considers the outlier point when constructing the PCs, the results are nonoptimal for the regular data points.

K-Means Clustering

K-Means clustering uses the expectation maximization algorithm to find the centers of clusters that best represent the data. K is a hyperparameter that determines the number of clusters that must be specified. Below are the results of the algorithm with **K = 5, 4, 3, 2** on the 2D PCA transformed dataset.

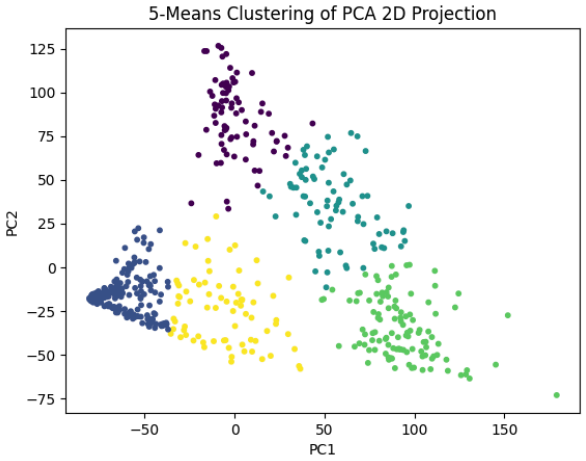

Figure 4: 5 Means Clustering

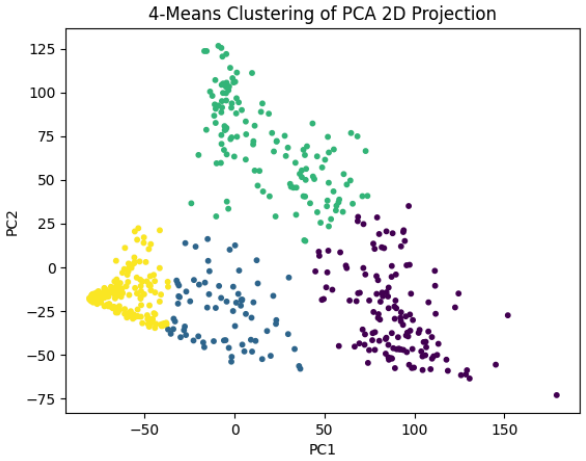

Figure 5: 4 Means Clustering

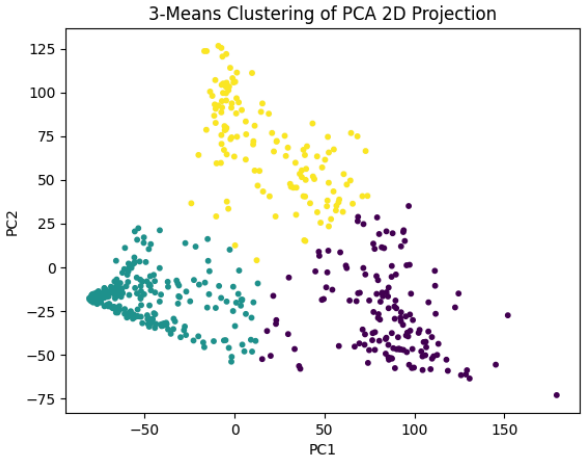

Figure 6: 3 Means Clustering

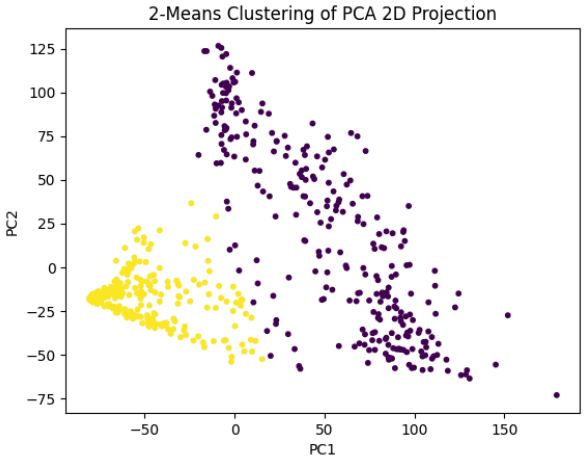

Figure 7: 2 Means Clustering

Just from the visual examination of figure 2, subjectively, the most apparent number of clusters is 2. However, none of the clusterings in figures 4, 5, 6, and 7 seem nonsensical to me. These might be useful for other interpretations of the data.

Percentage of overlap between 2 means clustering labels and the true labels is **88.5%**. Considering that random guessing would have resulted in 50%, fall detection seems to be possible based on these measurements.

Part B)

The dataset must be processed in preparation for supervised learning. This processing step includes splitting and standardization. Neural network models are especially sensitive to non standardized or non normalized data.

The dataset is divided into two sets, namely training and test sets. The training set contains **70%** of the data points while the test set contains the remaining **30%**. The split is stratified, this means that the ratio of the labels in the dataset is reflected in the training and test sets. We also shuffle the dataset before the split to prevent any bias. The validation set is not needed in this case because we will use **10-fold cross validation** for hyperparameter optimization.

The standardization procedure includes subtracting the mean and dividing by the standard deviation. The mean and standard deviation values of the training set is used for both the training and test sets to avoid data leakage.

Hyperparameter Optimization

Hyperparameter optimization is accomplished by performing grid search over a set of hyperparameters. **F1 score** is used as the 10-fold cross validation performance metric.

Support Vector Machine

The grid search is conducted using the following parameters:

C	Kernel	Degree
10^{-6}, 10^{-5}, 10^{-4}, ..., 10^{6}, 10^{7}	Linear	-
10^{-6}, 10^{-5}, 10^{-4}, ..., 10^{6}, 10^{7}	Polynomial	2, 3, 4, 5, 6, 7, 8
10^{-6}, 10^{-5}, 10^{-4}, ..., 10^{6}, 10^{7}	Radial Basis Function	-
10^{-6}, 10^{-5}, 10^{-4}, ..., 10^{6}, 10^{7}	Sigmoid	-

Table 2: SVM grid search hyperparameter values.

The C parameter is inversely proportional to regularization.

The results of the grid search procedure can be visualized by plotting one of the hyperparameters on the x axis and the F1 scores in the y axis. In the following scatter plots, every dot corresponds to a particular combination of hyperparameters.

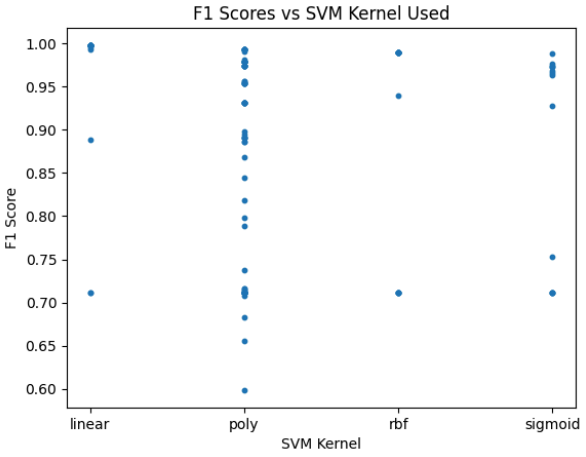

Figure 8: F1 scores vs SVM kernel used

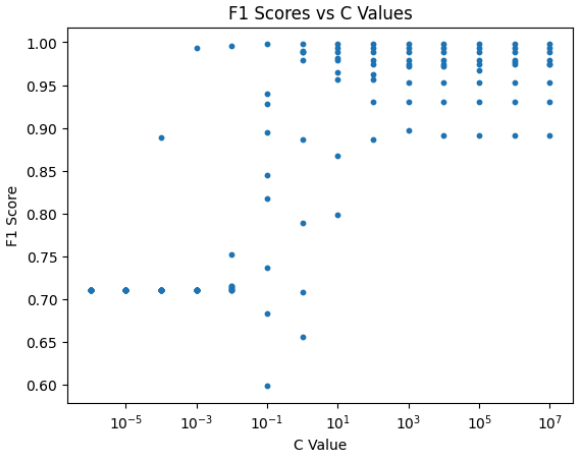

Figure 9: F1 scores vs C values

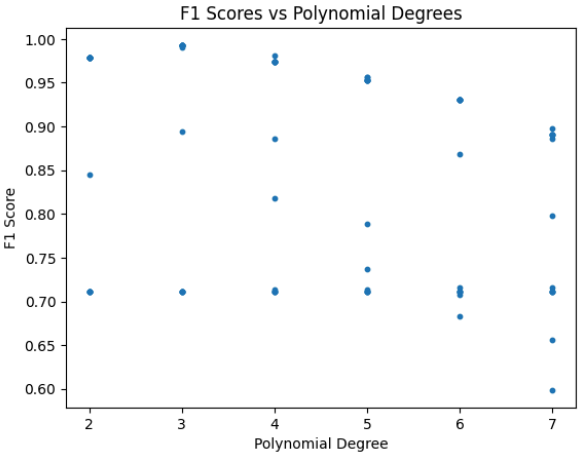

Figure 10: F1 scores vs Polynomial degrees

The best performing combination of hyperparameters according to 10 fold cross validation F1 score is: **C = 0.1, Kernel = Linear** with an F1 score of **0.998**.

Multi Layer Perceptron

Only **one hidden layer MLPs** are considered. The grid search is conducted using the following parameters:

Hidden layer unit count	Solver	Alpha	Batch size	Learning rate
1, 2, 3, 4, 5	Mini-batch gradient descent	10^{-10}, 10^{-9}, ..., 10^{2}, 10^{3}	64	10^{-5}, 10^{-4}, ..., 10^{0}, 10^{1}

Table 3: MLP grid search hyperparameter values.

The alpha parameter determines the strength of regularization applied to the model.

The results of the grid search procedure can be visualized by plotting one of the hyperparameters on the x axis and the F1 scores in the y axis. In the following scatter plots, every dot corresponds to a particular combination of hyperparameters.

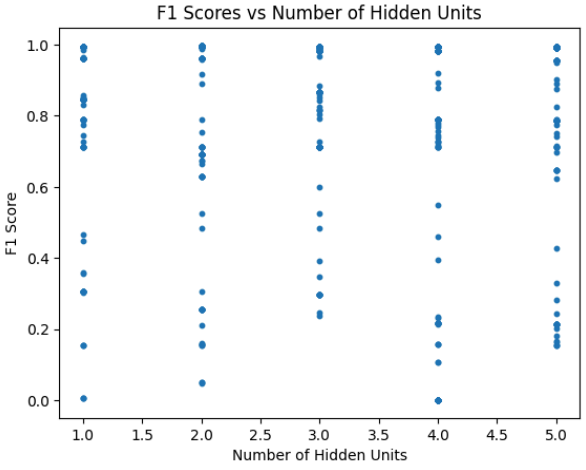

Figure 11: F1 scores vs Number of hidden units

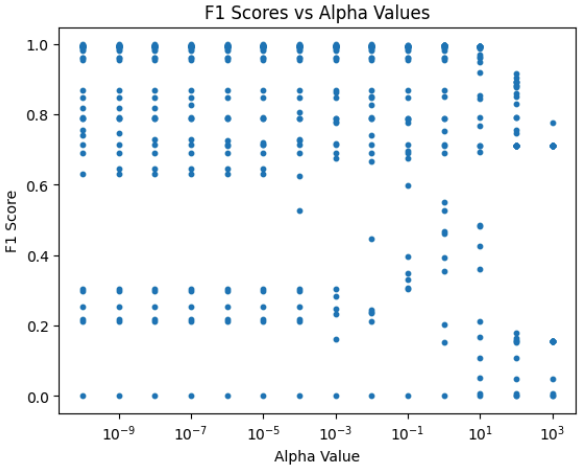

Figure 12: F1 scores vs Alpha values

Figure 13: F1 scores vs Learning rates

The best performing combination of hyperparameters according to 10 fold cross validation F1 score is: **Hidden unit count = 2, Alpha = 10^{-10}, Learning rate = 0.1** with an F1 score of **0.998**. Maximum number of iterations is 200 for all passes. Some passes did not converge before the iteration limit.

Test Set Performance

For the evaluation of test set performance, the following metrics are considered:

- Confusion matrix
- Accuracy
- Balanced accuracy
- Precision
- Recall
- F1 score

Support Vector Machine

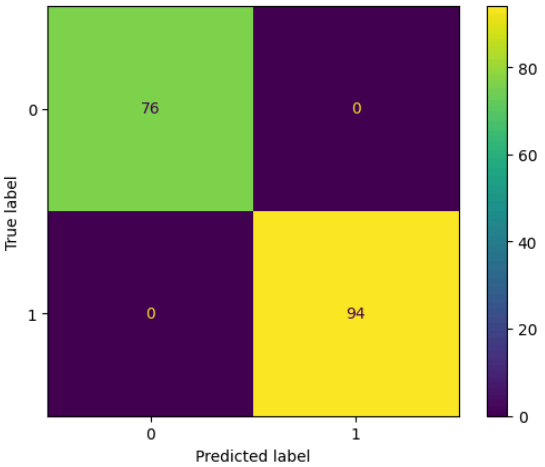

Figure 14: SVM best model test set confusion matrix.

- Accuracy: 1
- Balanced accuracy: 1
- Precision: 1

- Recall: 1
- F1 score: 1

Multi Layer Perceptron

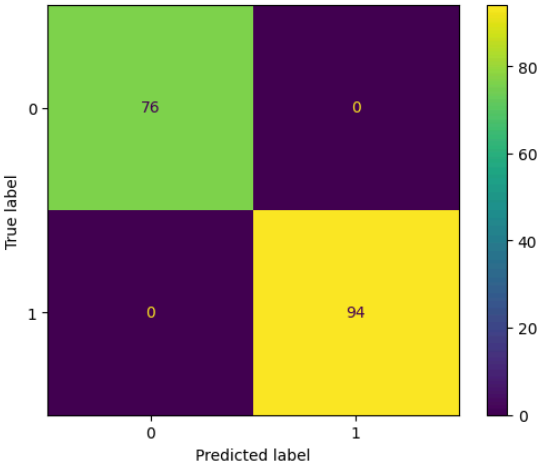

Figure 15: MLP best model test set confusion matrix.

- Accuracy: 1
- Balanced accuracy: 1
- Precision: 1
- Recall: 1
- F1 score: 1

Conclusion

Both models score perfectly on the test set. The hyperparameter optimization and training procedure for the MLP model is much more involved both complexity and time wise compared to the SVM model.

As a result of my experimentation on this project, I believe that fall detection using machine learning based on wearable sensors is a viable task.

Appendix

```
# %% [markdown]
# # Alkım Ege Akarsu | 21901461 | GE 461 | Project 4: Telehealth - Fall
Detection

# %% [markdown]
# ## Import Modules

# %%
from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score,
balanced_accuracy_score, f1_score, precision_score, recall_score,
PrecisionRecallDisplay, RocCurveDisplay

SEED = 0

# %% [markdown]
# ## Load Dataset

# %%
# Get dataset path
dataset_path = Path().resolve().joinpath("falldetection_dataset.csv")
# Load dataset into DataFrame
dataset = pd.read_csv(dataset_path, sep=",", header=None)

# Delete first column (indices)
dataset.drop(dataset.columns[0], axis=1, inplace=True)
# Convert NF, F to 0, 1 (labels)
dataset.iloc[:, 0] = dataset.iloc[:, 0].replace({"NF": 0, "F":
1}).infer_objects(copy=False)

# Check for NaN values
if dataset.isnull().values.any() == True:
      print("----- NaN values found in dataset! -----\n")
else:
      print("----- No NaN values found in dataset! -----\n")

# Convert DataFrame to ndarray
dataset = dataset.to_numpy()

# Separate X and y (labels in column index 0)
X = dataset[:, 1:]
y = dataset[:, 0]

# Get the index of the outlier
outlier_index_row = (np.argmax(X) / X.shape[1]).astype(np.uint16)
# Remove the outlier
X_no_outlier = np.delete(X, outlier_index_row, axis=0)
y_no_outlier = np.delete(y, outlier_index_row)
```

```
# Get number of data points belonging to each class
num_1 = np.count_nonzero(y == 1)
num_0 = y.shape[0] - num_1
print(f"Number of 'No Fall' data points: {num_0}\n"
      f"Number of 'Fall' data points: {num_1}\n")

# %% [markdown]
# ## Part A

# %% [markdown]
# ### Outlier

# %%
# WITH THE OUTLIER
# Get the PCA object
pca = PCA(n_components=2, random_state=SEED)
# Fit and transform
X_pca = pca.fit_transform(X)
# Get variance captured
var_exp = pca.explained_variance_ratio_
print(f"Variance explained by the first two principal components:\n"
      f"PC1: {var_exp[0] * 100:.1f}%\n"
      f"PC2: {var_exp[1] * 100:.1f}%\n"
      f"PC1 + PC2: {np.sum(var_exp) * 100:.1f}%\n")

# Visualize the results
plt.figure()
plt.title("PCA 2D Projection")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.show()

# WITHOUT THE OUTLIER
# Get the PCA object
pca = PCA(n_components=2, random_state=SEED)
# Fit and transform
X_pca = pca.fit_transform(X_no_outlier)
# Get variance captured
var_exp = pca.explained_variance_ratio_
print(f"Variance explained by the first two principal components (outlier
removed):\n"
      f"PC1: {var_exp[0] * 100:.1f}%\n"
      f"PC2: {var_exp[1] * 100:.1f}%\n"
      f"PC1 + PC2: {np.sum(var_exp) * 100:.1f}%\n")

# Visualize the results
```

```
plt.figure()
plt.title("PCA 2D Projection")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.scatter(X_pca[:, 0], X_pca[:, 1], s=10)
plt.show()

# Remove outlier from the original dataset
X = X_no_outlier
y = y_no_outlier

# %% [markdown]
# ### K-Means Clustering

# %%
# K = 5
# Get the KMeans object
kmeans = KMeans(n_clusters=5, init="random", n_init=20,
max_iter=np.uint32(1e4), random_state=SEED)
# Run K-Means on the 2D data
X_kmeans = kmeans.fit_transform(X_pca)
# Get labels
y_kmeans = kmeans.labels_

# Visualize the results
plt.figure()
plt.title("5-Means Clustering of PCA 2D Projection")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.scatter(X_pca[:, 0], X_pca[:, 1], s=10, c=y_kmeans)
plt.show()


# K = 4
# Get the KMeans object
kmeans = KMeans(n_clusters=4, init="random", n_init=20,
max_iter=np.uint32(1e4), random_state=SEED)
# Run K-Means on the 2D data
X_kmeans = kmeans.fit_transform(X_pca)
# Get labels
y_kmeans = kmeans.labels_

# Visualize the results
plt.figure()
plt.title("4-Means Clustering of PCA 2D Projection")
plt.xlabel("PC1")
plt.ylabel("PC2")
```

```
plt.scatter(X_pca[:, 0], X_pca[:, 1], s=10, c=y_kmeans)
plt.show()


# K = 3
# Get the KMeans object
kmeans = KMeans(n_clusters=3, init="random", n_init=20,
max_iter=np.uint32(1e4), random_state=SEED)
# Run K-Means on the 2D data
X_kmeans = kmeans.fit_transform(X_pca)
# Get labels
y_kmeans = kmeans.labels_

# Visualize the results
plt.figure()
plt.title("3-Means Clustering of PCA 2D Projection")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.scatter(X_pca[:, 0], X_pca[:, 1], s=10, c=y_kmeans)
plt.show()


# K = 2
# Get the KMeans object
kmeans = KMeans(n_clusters=2, init="random", n_init=20,
max_iter=np.uint32(1e4), random_state=SEED)
# Run K-Means on the 2D data
X_kmeans = kmeans.fit_transform(X_pca)
# Get labels
y_kmeans = kmeans.labels_

# Visualize the results
plt.figure()
plt.title("2-Means Clustering of PCA 2D Projection")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.scatter(X_pca[:, 0], X_pca[:, 1], s=10, c=y_kmeans)
plt.show()

# Visualize true labels
plt.figure()
plt.title("True Labels of PCA 2D Projection")
plt.xlabel("PC1")
plt.ylabel("PC2")
# Create a color map for the two classes
colors = ['blue', 'red']
labels = ['No fall', 'Fall']
```

```
# Scatter plot with custom color map
for i in range(len(colors)):
      plt.scatter(X_pca[y == i, 0], X_pca[y == i, 1], s=10, c=colors[i],
label=labels[i])
plt.legend(loc="upper right")
plt.show()


# Overlap calculation
# Initilize counter for counting overlap
counter = 0
# Loop over the labels
for i in range(y.shape[0]):
      # Increment counter if labels are the same
      if y_kmeans[i] == y[i]:
      counter += 1

# Get ratio
ratio = counter / y.shape[0]
# Get inverse of counter if less than 0.5
if ratio < 0.5:
      ratio = 1 - ratio

# Print the result
print(f"Overlap ratio between true labels and clustering labels: {ratio *
100:.1f}%")

# %% [markdown]
# ## Part B

# %% [markdown]
# ### Preprocessing

# %%
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=SEED, stratify=y)

# Get standardizer object
scaler = StandardScaler()
# Get mean and std of the TRAINING SET
scaler.fit(X_train)
# Standardize all sets
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Convert y to integer data types
```

```
y_train = np.uint8(y_train)
y_test = np.uint8(y_test)

# %% [markdown]
# ### Hyperparameter Optimization With Cross Validation

# %% [markdown]
# #### Support Vector Machine

# %%
# Prepare hyperparameter grid
param_grid = [
      {"C": np.logspace(-6, 7, 14), "kernel": ["linear"], "random_state":
[SEED]},
      {"C": np.logspace(-6, 7, 14), "kernel": ["poly"], "degree":
list(range(2, 8)), "random_state": [SEED]},
      {"C": np.logspace(-6, 7, 14), "kernel": ["rbf"], "random_state":
[SEED]},
      {"C": np.logspace(-6, 7, 14), "kernel": ["sigmoid"], "random_state":
[SEED]},
]

# Get model object
svm = SVC()

# Get grid search object
best_svm = GridSearchCV(estimator=svm, param_grid=param_grid, scoring="f1",
cv=10, n_jobs=-1)
# Perform grid search
best_svm.fit(X_train, y_train)

# Print the best parameters and score
print(f"Best parameters: {best_svm.best_params_}")
print(f"Best score: {best_svm.best_score_}")

# Visualize results
# Different kernels
plt.figure()
plt.title("F1 Scores vs SVM Kernel Used")
plt.xlabel("SVM Kernel")
plt.ylabel("F1 Score")
plt.scatter(best_svm.cv_results_["param_kernel"],
best_svm.cv_results_["mean_test_score"], s=10)
plt.show()

# Different C values
plt.figure()
```

```
plt.title("F1 Scores vs C Values")
plt.xlabel("C Value")
plt.ylabel("F1 Score")
plt.xscale("log")
plt.scatter(best_svm.cv_results_["param_C"],
best_svm.cv_results_["mean_test_score"], s=10)
plt.show()

# Diffent polynomial degrees
plt.figure()
plt.title("F1 Scores vs Polynomial Degrees")
plt.xlabel("Polynomial Degree")
plt.ylabel("F1 Score")
plt.scatter(best_svm.cv_results_["param_degree"],
best_svm.cv_results_["mean_test_score"], s=10)
plt.show()

# %% [markdown]
# #### Multi Layer Perceptron

# %%
# Hidden layer sizes for single hidden layer MLP
hidden_layer_sizes = []
for i in range(1, 6):
      hidden_layer_sizes.append((i,))

# Prepare hyperparameter grid
param_grid = [
      {"hidden_layer_sizes": hidden_layer_sizes,
      "solver": ["sgd"],
      "alpha": np.logspace(-10, 3, 14),
      "batch_size": [64],
      "learning_rate_init": np.logspace(-5, 1, 7),
      "random_state": [SEED]}
]

# Get model object
mlp = MLPClassifier()

# Get grid search object
best_mlp = GridSearchCV(estimator=mlp, param_grid=param_grid, scoring="f1",
cv=10, n_jobs=-1)
# Perform grid search
best_mlp.fit(X_train, y_train)

# Print the best parameters and score
print(f"Best parameters: {best_mlp.best_params_}")
```

```
print(f"Best score: {best_mlp.best_score_}")

# Visualize results
# Different number of hidden units
plt.figure()
plt.title("F1 Scores vs Number of Hidden Units")
plt.xlabel("Number of Hidden Units")
plt.ylabel("F1 Score")
plt.scatter([t[0] for t in best_mlp.cv_results_["param_hidden_layer_sizes"]],
best_mlp.cv_results_["mean_test_score"], s=10)
plt.show()

# Different alpha values
plt.figure()
plt.title("F1 Scores vs Alpha Values")
plt.xlabel("Alpha Value")
plt.ylabel("F1 Score")
plt.xscale("log")
plt.scatter(best_mlp.cv_results_["param_alpha"],
best_mlp.cv_results_["mean_test_score"], s=10)
plt.show()

# Diffent learning rates
plt.figure()
plt.title("F1 Scores vs Learning Rates")
plt.xlabel("Learning Rate")
plt.ylabel("F1 Score")
plt.xscale("log")
plt.scatter(best_mlp.cv_results_["param_learning_rate_init"],
best_mlp.cv_results_["mean_test_score"], s=10)
plt.show()

# %% [markdown]
# ### Test Set Performance

# %% [markdown]
# #### Support Vector Machine

# %%
# Get predictions
y_pred = best_svm.predict(X_test)
# Get confusion matrix
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.show()
# Get accuracy
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
# Get balanced accuracy
```

```
print(f"Balanced accuracy: {balanced_accuracy_score(y_test, y_pred,
adjusted=True)}")
# Get precision
print(f"Precision: {precision_score(y_test, y_pred)}")
# Get recall
print(f"Recall: {recall_score(y_test, y_pred)}")
# Get f1 score
print(f"F1 score: {f1_score(y_test, y_pred)}")
# Get precision recall curve
PrecisionRecallDisplay.from_predictions(y_test, y_pred)
plt.show()
# Get ROC curve
RocCurveDisplay.from_predictions(y_test, y_pred)
plt.show()

# %% [markdown]
# #### Multi Layer Perceptron

# %%
# Get predictions
y_pred = best_mlp.predict(X_test)
# Get confusion matrix
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.show()
# Get accuracy
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
# Get balanced accuracy
print(f"Balanced accuracy: {balanced_accuracy_score(y_test, y_pred,
adjusted=True)}")
# Get precision
print(f"Precision: {precision_score(y_test, y_pred)}")
# Get recall
print(f"Recall: {recall_score(y_test, y_pred)}")
# Get f1 score
print(f"F1 score: {f1_score(y_test, y_pred)}")
# Get precision recall curve
PrecisionRecallDisplay.from_predictions(y_test, y_pred)
plt.show()
# Get ROC curve
RocCurveDisplay.from_predictions(y_test, y_pred)
plt.show()
```