SUPPORT VECTOR MACHINES

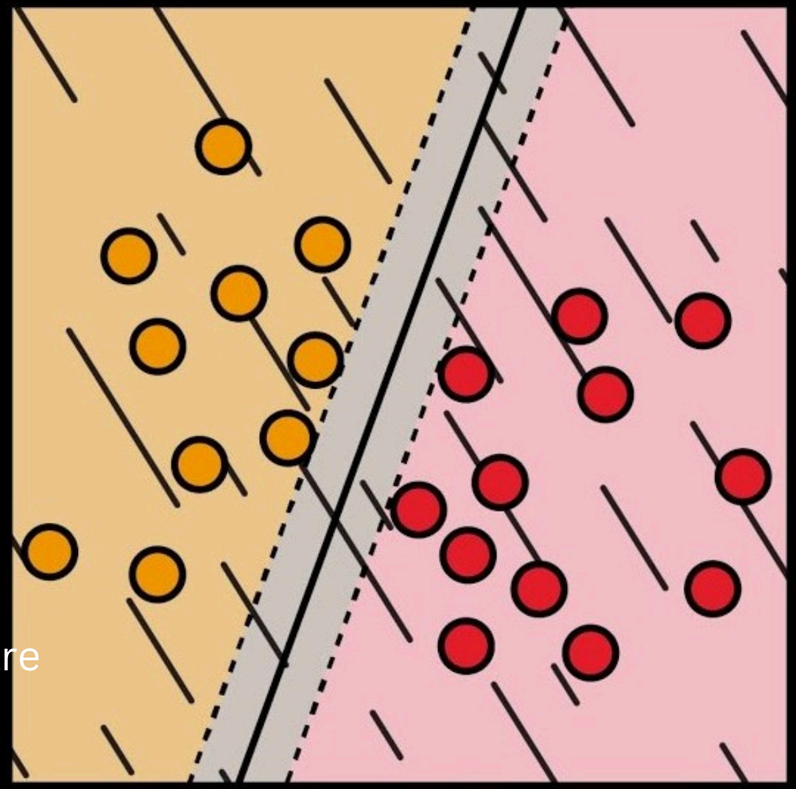
Ramavarapu Dhanush

Dept. of Civil Engineering,

IIT Kharagpur

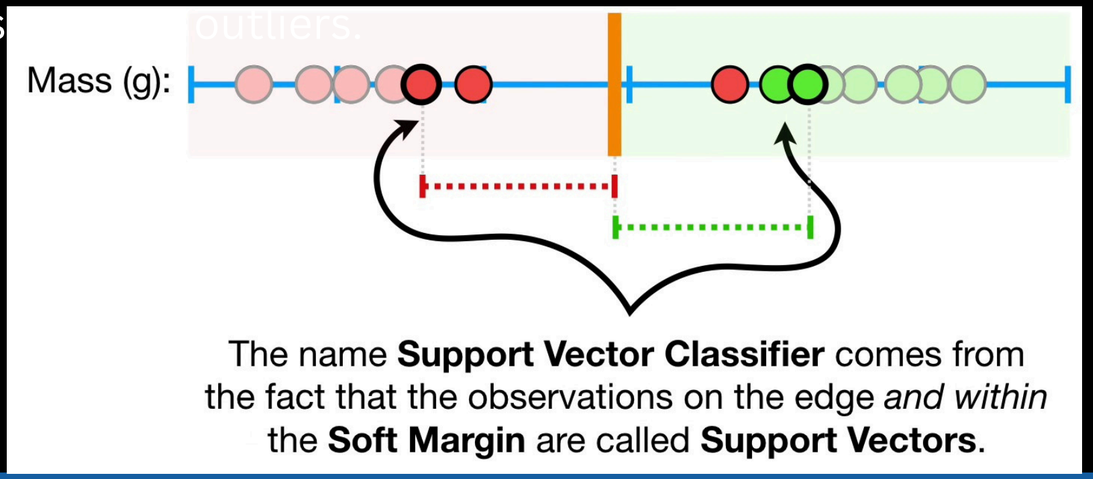
Introduction

Support Vector Machines (SVM) represent a potent tool in supervised learning, proficient in both classification and regression tasks. These models excel in navigating high-dimensional spaces, making them ideal for handling complex datasets. SVMs operate by identifying the optimal hyperplane that separates data points into distinct classes, maximizing the margin of the decision boundary for enhanced robustness. Their adaptability extends to linear and non-linear data patterns, achieved through various kernel functions like linear, polynomial, radial basis function (RBF), and sigmoid. Widely employed across domains such as image classification, bioinformatics, text categorization, and financial forecasting, SVMs are renowned for their versatility and efficacy in diverse applications, cementing their status as a popular choice in machine learning endeavors.

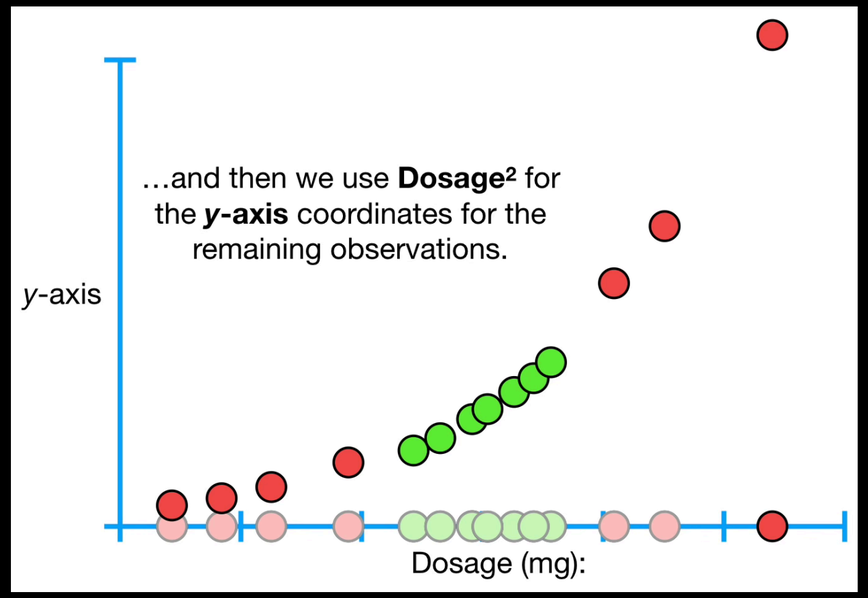


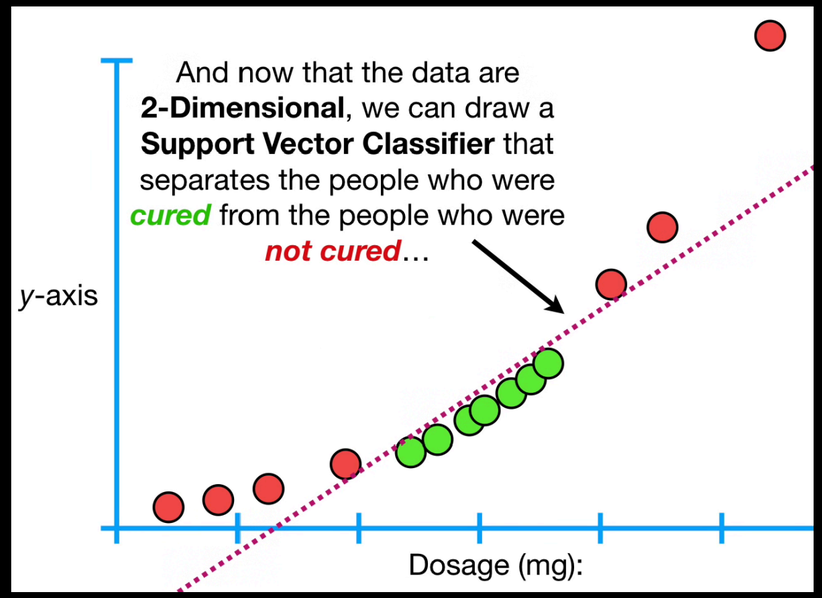
How does SVM work?

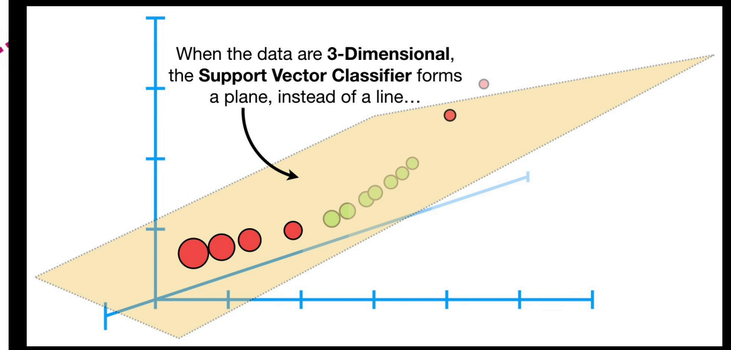
Support Vector Machines(SVMs) operateby identifying theoptimalhyperplane thatseparatesdatapointsinto different classes while maximizing the margin between classes.Mathematically, thisoptimization problemcan be formulated as finding the hyperplane that maximizes the margin, subject to the constraint thatall data points are classified correctly. The margin is defined as the distance between the hyperplane and the closest data points of each class, known as support vectors. The decision boundary is determined solely by these support vectors, making SVMs memory efficient and robus



Support Vector Machines (SVMs) achieve remarkable versatility by transforming input data into higher-dimensional feature spaces using kernel functions. This transformation enables SVMs to delineate complex decision boundaries, crucial for handling non-linearly separable data. Mathematically, kernel functions map input vectors into a higher-dimensional space, where a linear hyperplane can effectively separate classes. For instance, the polynomial kernel introduces higher-order interactions between features by computing polynomial combinations. Similarly, the radial basis function (RBF) kernel maps data into an infinite-dimensional space, capturing intricate relationships through distance-based similarity measures. This conversion process enhances SVMs' ability to classify intricate patterns, providing robust solutions across various domains.







Kernel Functions : RBF

Support Vector Machines (SVMs) employ various kernel functions to accommodate diverse data structures. Common kernels include the linear, polynomial, sigmoid, and the Radial Basis Function (RBF). Among these, the Radial Basis Function (RBF) kernel is notable for its ability to handle complex patterns by mapping data into an infinite-dimensional space. This infinite feature space enables the RBF kernel to capture intricate relationships between data points, facilitating the delineation of non-linear decision boundaries with remarkable flexibility and accuracy. The formula for the RBF kernel is as follows:

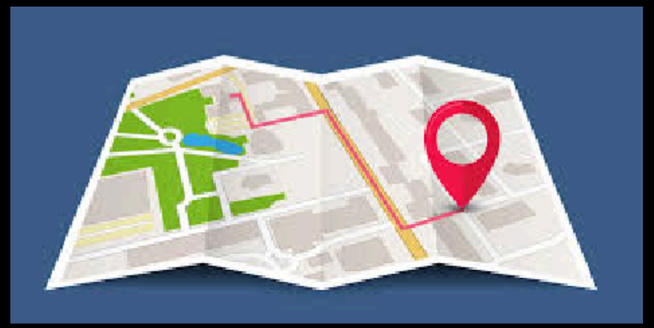
$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}$$

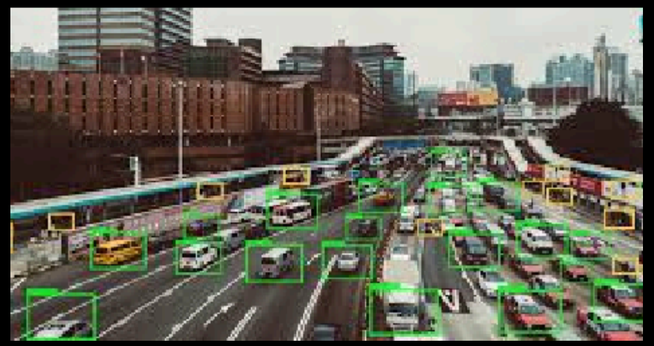
Here, γ determines the spread of the kernel, influencing the shape of the decision boundary. Adjusting γ appropriately is crucial for achieving optimal SVM performance. Cross-validation techniques are commonly employed to select suitable values for γ, ensuring robust model generalization across diverse datasets.

SVM in Transportation

Transportation Mode Detection

SVMs classify transportation modes based on sensor data like GPS and accelerometers, aiding in monitoring vehicle movement patterns.





Anomaly Detection in Vehicles

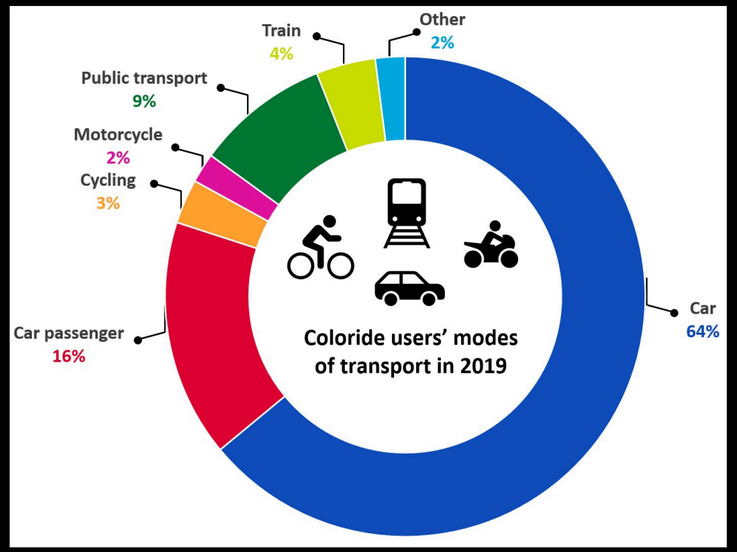
SVMs analyze vehicle sensor data to detect deviations from normal behavior, enabling proactive maintenance to minimize downtime and ensure fleet reliability.

Transportation Mode Detection

Problem Statement: Introduction

Transportation mode detection refers to the process of accurately identifying the mode of transportation a user is employing, such as walking, biking, driving, or using public transit, based on sensor data. This analysis plays a pivotal role in optimizing urban mobility, enhancing **transportation efficiency**, and enabling **smarter city planning initiatives.**

In today's world of rapidly growing urbanization and increasing reliance on diverse transportation modes, accurate mode detection is crucial for several reasons. It enables the development of **intelligent transportation systems, facilitates personalized mobility services, improves traffic management, reduces congestion, promotes sustainable transportation choices,** and ultimately contributes to building more livable and efficient cities.



1

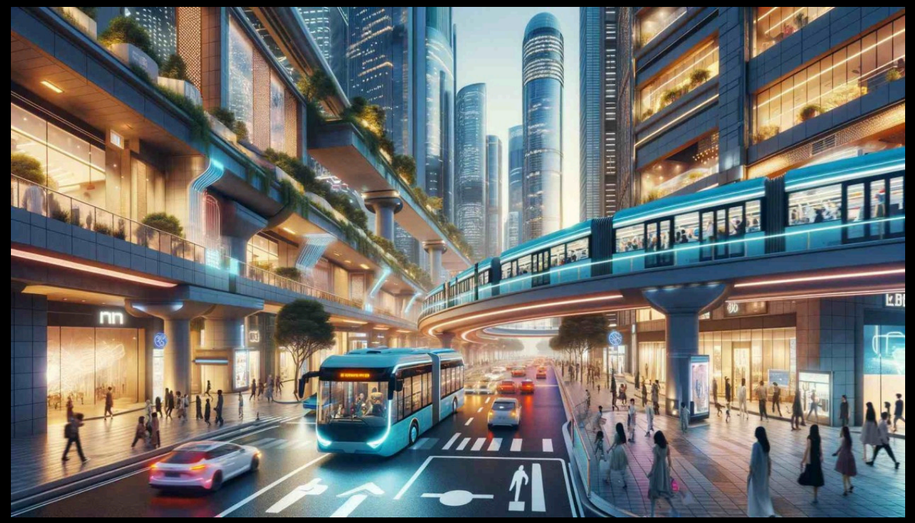
FEATURES IN DATASET DESCRIPTION :

- Time(minutes)
- Accelerometer (mean, stdv)
- Game rotation vector (mean, stdv)
- Gyroscope (mean, stdv)
- Gyroscope uncalibrated (mean, stdv)
- Linear acceleration (mean, stdv)
- Orientation (mean, stdv)
- Rotation vector (mean, stdv)
- Sound (dB) (mean, stdv)
- Speed (km/h) (mean, stdv)

2

HISTORICAL DATA

The dataset comprises historical data collected from various sensors installed in transportation devices. This data provides insights into the behavior and patterns of different transportation modes over time. Leveraging this historical perspective, we gain valuable insights for enhancing transportation efficiency and planning.

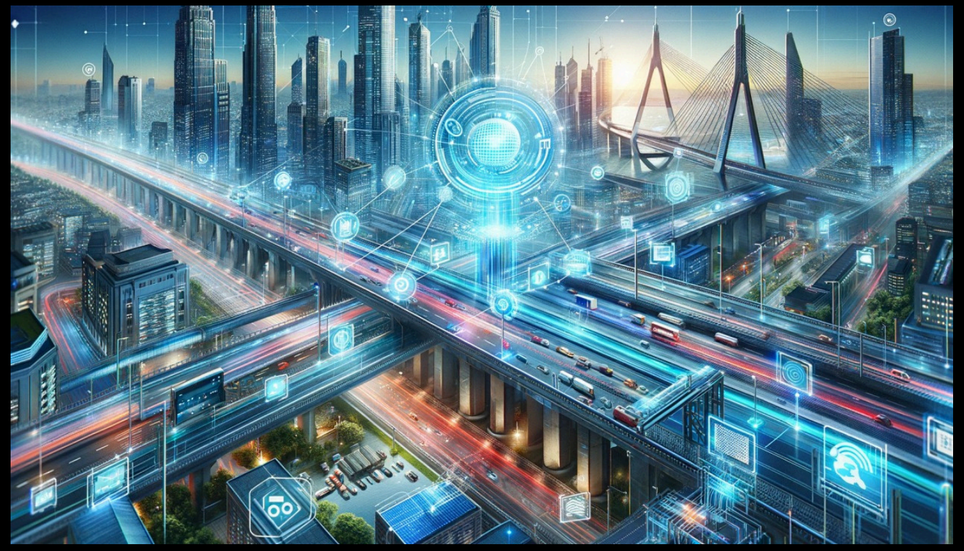


3 TRANSPORTATION MODE DETECTION :

Different transportation modes are detected by inputting the sensor data features into algorithms, such as Support Vector Machines (SVMs), which analyze the patterns and characteristics unique to each mode to predict the target transportation mode.

4 REAL-TIME USE :

Real-time utilization ofthe dataset involves feeding sensor data into predictive models continuously, enabling instantaneous classification of transportation modes. This facilitates timely interventions, dynamic traffic management, and responsive urban planning, ultimately enhancing commuter experiences and optimizing transportation systems in our ever-evolving cities.



Transportation Mode Detection Model Code

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tabulate import tabulate
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import cross_val_score

# Load your CSV file (manually enter the file path)
df = pd.read_csv("tmd_data3.csv")

# Remove specified columns
columns_to_remove = ['android.sensor.accelerometer#min', 'android.sensor.accelerometer#max',
                     'android.sensor.game_rotation_vector#min', 'android.sensor.game_rotation_vector#max',
                     'android.sensor.gyroscope#min', 'android.sensor.gyroscope#max',
                     'android.sensor.gyroscope_uncalibrated#min', 'android.sensor.gyroscope_uncalibrated#max',
                     'android.sensor.linear_acceleration#min', 'android.sensor.linear_acceleration#max',
                     'android.sensor.orientation#min', 'android.sensor.orientation#max',
                     'android.sensor.rotation_vector#min', 'android.sensor.rotation_vector#max',
                     'sound#min', 'sound#max', 'speed#min', 'speed#max']

df.drop(columns_to_remove, axis=1, inplace=True)

# General Description of the Dataset
print("General Description of the Dataset:")
print(df.describe())
```

[Removing some columns from the dataset to reduce correlation between features]

```
# Class-wise Description of Features
class_names = df['target'].unique()

for class_name in class_names:
    class_df = df[df['target'] == class_name]
    print(f"\nClass: {class_name}")
    print("Description of Features:")
    print(class_df.drop('target', axis=1).describe())

# Count the number of rows for each class in the dataset
class_counts = df['target'].value_counts()

# Print the count for each class
print("Count of rows for each class:")
print(class_counts)

# Splitting features and target variable
X = df.drop('target', axis=1)
y = df['target']

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Selecting specific features for plotting
selected_features = ['android.sensor.accelerometer#mean', 'android.sensor.gyroscope#mean', 'sound#mean']
selected_indices = [X.columns.get_loc(feature) for feature in selected_features]

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25, random_state=42)
```

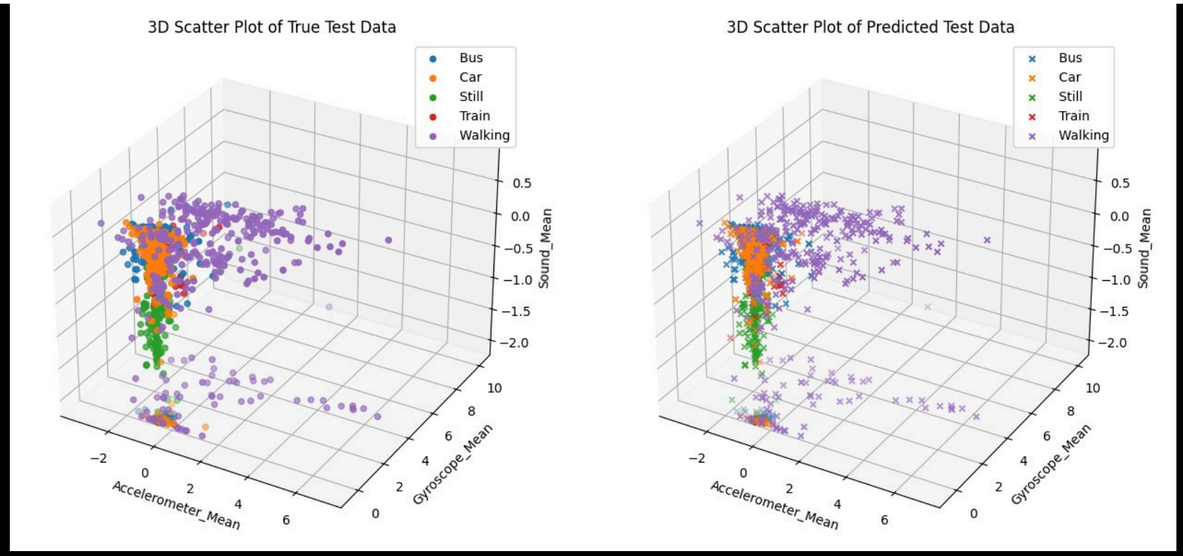
```
# Define the parameter grid including additional parameters and kernel options
param_grid = {
    'C': [1, 10,100],
    'gamma': [1,0.1,0.01],
    'kernel': [ 'rbf'],  # Add more kernel options

}

# Create a GridSearchCV object with the specified parameters
grid = GridSearchCV(SVC(probability=True), param_grid, refit=True, verbose=3, cv=5)

# Fit the grid search object to the training data
grid.fit(X_train, y_train)

# Printing the best hyperparameters
print()
print("Best hyperparameters:", grid.best_params_)

# Training the model with best hyperparameters
svm_model = grid.best_estimator_
svm_model.fit(X_train, y_train)

# Making predictions
y_pred_train = svm_model.predict(X_train)
y_pred_test = svm_model.predict(X_test)
print()

# Accuracy of training and testing data
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)
print("Accuracy of training data:", train_accuracy)
print("Accuracy of testing data:", test_accuracy)
print()
```

```python
# Predicting on the test set
Y_pred = svm_model.predict(X_test)

# Evaluating the model
accuracy = svm_model.score(X_test, y_test)
print("Accuracy:", accuracy*100,"%")
print()

# Make predictions on the test set
y_pred = svm_model.predict(X_test)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 8), subplot_kw={'projection': '3d'}, gridspec_kw={'width_ratios': [1, 1]})

# Plot true data
for label in np.unique(y_test):
    indices = np.where(y_test == label)
    ax1.scatter(X_test[indices, selected_indices[0]], X_test[indices, selected_indices[1]], X_test[indices, selected_indices[2]], label=f' {label}')

# Plot predicted data
for label in np.unique(y_pred):
    indices = np.where(y_pred == label)
    ax2.scatter(X_test[indices, selected_indices[0]], X_test[indices, selected_indices[1]], X_test[indices, selected_indices[2]], marker='x', label=f' {label}')

# Plot settings for true data
ax1.set_xlabel('Accelerometer_Mean')
ax1.set_ylabel('Gyroscope_Mean')
ax1.set_zlabel('Sound_Mean')
ax1.legend()
ax1.set_title('3D Scatter Plot of True Test Data')

# Plot settings for predicted data
ax2.set_xlabel('Accelerometer_Mean')
ax2.set_ylabel('Gyroscope_Mean')
ax2.set_zlabel('Sound_Mean')
ax2.legend()
ax2.set_title('3D Scatter Plot of Predicted Test Data')

plt.show()
```

Process Runoff after this code :

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV 1/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.863 total time=   5.2s
[CV 2/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.842 total time=   3.8s
[CV 3/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.853 total time=   3.9s
[CV 4/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.864 total time=   3.8s
[CV 5/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.830 total time=   4.0s
[CV 1/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.825 total time=   1.7s
[CV 2/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.854 total time=   1.8s
[CV 3/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.855 total time=   1.7s
[CV 4/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.868 total time=   1.7s
[CV 5/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.837 total time=   1.7s
[CV 1/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.753 total time=   2.2s
[CV 2/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.776 total time=   2.1s
[CV 3/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.770 total time=   2.1s
[CV 4/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.781 total time=   2.1s
[CV 5/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.751 total time=   2.1s
[CV 1/5] END .........C=10, gamma=1, kernel=rbf;, score=0.888 total time=   4.1s
[CV 2/5] END .........C=10, gamma=1, kernel=rbf;, score=0.871 total time=   4.1s
[CV 3/5] END .........C=10, gamma=1, kernel=rbf;, score=0.880 total time=   4.1s
[CV 4/5] END .........C=10, gamma=1, kernel=rbf;, score=0.877 total time=   4.2s
[CV 5/5] END .........C=10, gamma=1, kernel=rbf;, score=0.865 total time=   4.0s
[CV 1/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.889 total time=   1.5s
[CV 2/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.895 total time=   1.5s
[CV 3/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.894 total time=   1.6s
[CV 4/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.906 total time=   1.5s
[CV 5/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.881 total time=   1.6s
[CV 1/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.795 total time=   1.6s
[CV 2/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.819 total time=   1.6s
[CV 3/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.822 total time=   1.6s
[CV 4/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.833 total time=   1.6s
[CV 5/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.814 total time=   1.6s
[CV 1/5] END ........C=100, gamma=1, kernel=rbf;, score=0.886 total time=   4.1s
[CV 2/5] END ........C=100, gamma=1, kernel=rbf;, score=0.870 total time=   4.1s
[CV 3/5] END ........C=100, gamma=1, kernel=rbf;, score=0.878 total time=   4.1s
[CV 4/5] END ........C=100, gamma=1, kernel=rbf;, score=0.873 total time=   4.0s
[CV 5/5] END ........C=100, gamma=1, kernel=rbf;, score=0.864 total time=   3.8s
```

```
[CV 1/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.910 total time=   1.4s
[CV 2/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.919 total time=   1.5s
[CV 3/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.900 total time=   1.5s
[CV 4/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.926 total time=   1.5s
[CV 5/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.907 total time=   1.5s
[CV 1/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.826 total time=   1.5s
[CV 2/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.855 total time=   1.5s
[CV 3/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.852 total time=   1.5s
[CV 4/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.871 total time=   1.5s
[CV 5/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.854 total time=   1.5s

Best hyperparameters: {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}

Accuracy of training data: 0.9893641095270423
Accuracy of testing data: 0.9314789687924017

Accuracy: 93.14789687924016 %
```



Code for plotting Confusion Matrix :

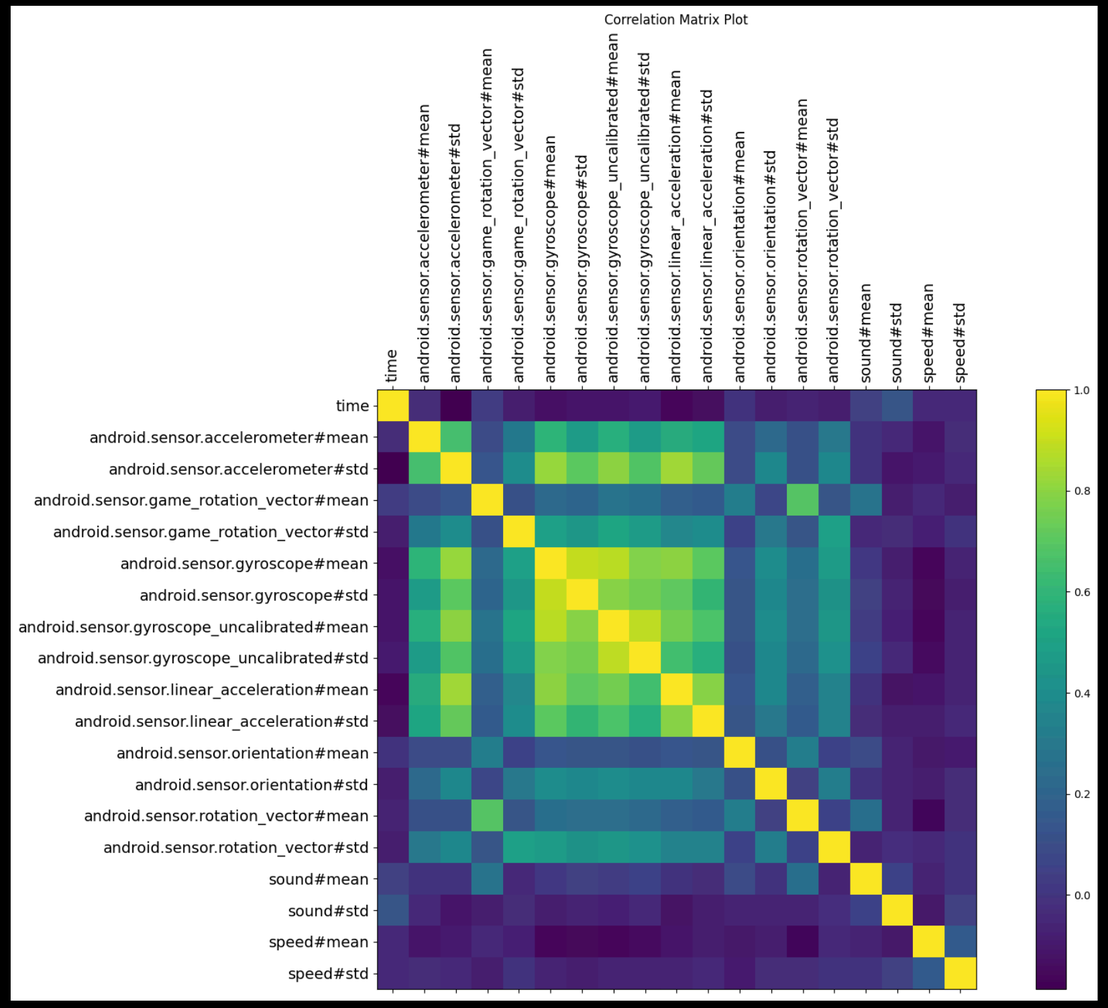
```
y_pred222 = svm_model.predict(X_scaled)

# Count the total number of values in each column
total_values = len(df)
total_matched = sum(df['target'] == y_pred222)
total_mismatched = total_values - total_matched

# Calculate the percentages
matched_percentage = (total_matched / total_values) * 100
mismatched_percentage = (total_mismatched / total_values) * 100

# Print the results
print("Fitting whole dataset into the model for cross check : ")
print(f"Total values: {total_values}")
print(f"Matched targets: {total_matched}")
print(f"Mismatched targets: {total_mismatched}")
print(f"Matched percentage: {matched_percentage:.2f}%")
print(f"Mismatched percentage: {mismatched_percentage:.2f}%")
print()

# Find the indices of mismatched data points
mismatched_indices = np.where(y != y_pred222)[0]

# Extract the row numbers of the mismatched data points
mismatched_row_numbers = df.iloc[mismatched_indices].index.tolist()

# Print the row numbers of the mismatched data points
print("Row numbers of mismatched data points:")
print(mismatched_row_numbers)

# Make predictions on the whole dataset
Y_pred_all = svm_model.predict(X_scaled)
```

```python
# Generate confusion matrix
conf_matrix = confusion_matrix(y, Y_pred_all)

# Normalize confusion matrix and convert to percentages
conf_matrix_norm_percent = (conf_matrix.astype('float') / conf_matrix.sum(axis=1)[:, np.newaxis]) * 100

# Create a DataFrame for normalized confusion matrix in percentage format
conf_df_norm_percent = pd.DataFrame(conf_matrix_norm_percent, index=np.unique(y), columns=np.unique(y))

# Print confusion matrix in table format with percentages
print()
print("Confusion Matrix (Table Format - Normalized, Percentages out of 100):")
print(tabulate(conf_df_norm_percent, headers='keys', tablefmt='grid', floatfmt=".2f"))
print()

# Plot confusion matrix in table format
conf_df = pd.DataFrame(conf_matrix_norm_percent, index=np.unique(y), columns=np.unique(y))
plt.figure(figsize=(10, 7))
sns.heatmap(conf_df, annot=True, fmt='g', cmap='Blues')
plt.title('Confusion Matrix (Heatmap Format)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

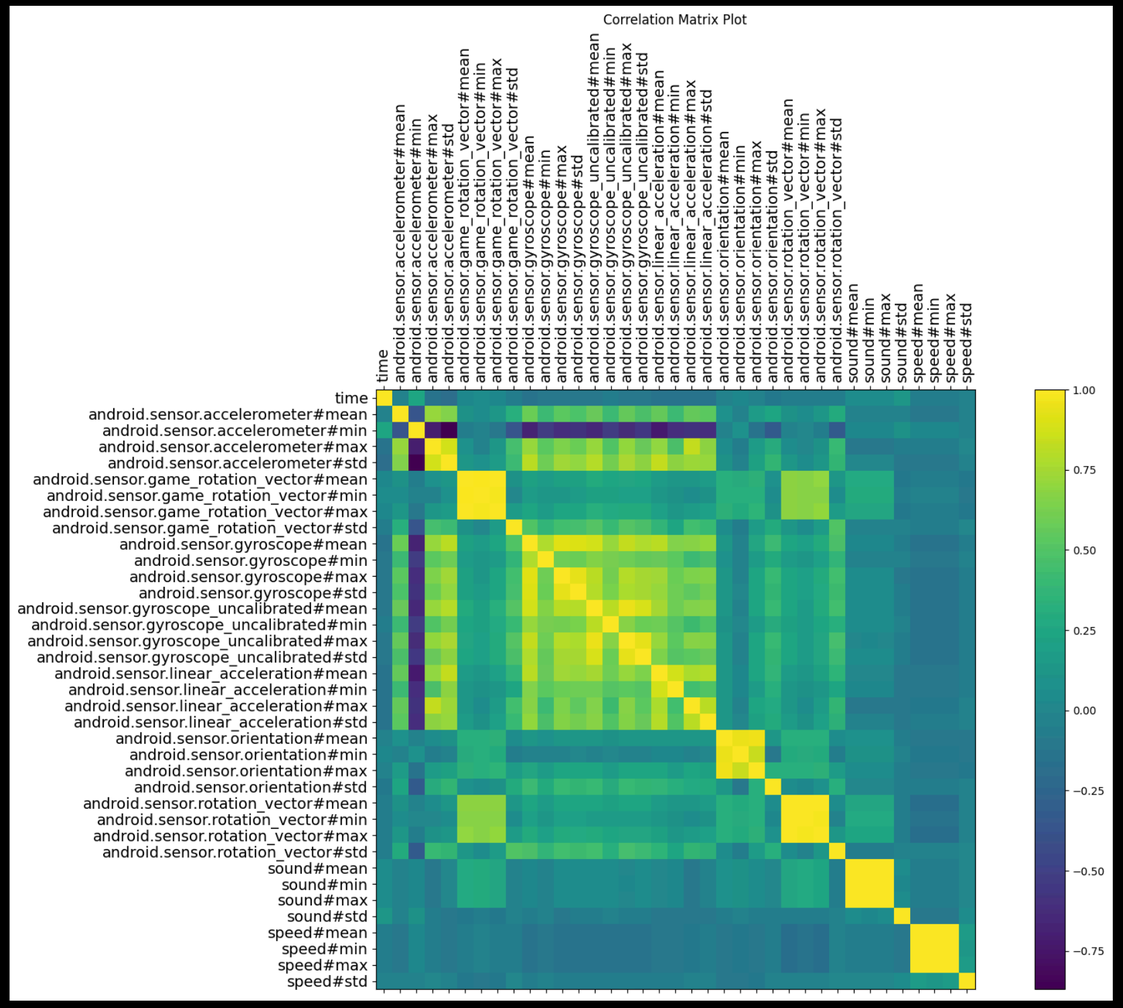
Correlation Matrix plot :

```
# Generate correlation matrix plot
numeric_df = df.select_dtypes(include=np.number)  # Select only numeric columns
plt.figure(figsize=(20, 10))
plt.matshow(numeric_df.corr(), fignum=1)
plt.yticks(np.arange(numeric_df.shape[1]), numeric_df.columns, fontsize=14)
plt.xticks(np.arange(numeric_df.shape[1]), numeric_df.columns, rotation='vertical', fontsize=14)
plt.colorbar()
plt.title('Correlation Matrix Plot')
plt.show()
```

Correlation Matrix plot :



Correlation Matrix plot before removing some columns:



CODE FOR THE OUTPUT :

```
# Plot settings for predicted data
ax2.set_xlabel('Accelerometer_Mean')
ax2.set_ylabel('Gyroscope_Mean')
ax2.set_zlabel('Sound_Mean')
ax2.legend()
ax2.set_title('3D Scatter Plot of Predicted Test Data')

plt.show()
```

```
# Take input as a single string with space-separated double elements
input_string = input("Enter the list of 19 double elements separated by spaces: ")
print()

# Split the input string by spaces and convert each substring to a float
input_list = [float(x) for x in input_string.split()]

print("Input list:", input_list)
```

```
# Convert the input features to a DataFrame
input_df = pd.DataFrame([input_list], columns=X.columns)

# Scale the input features
input_scaled = scaler.transform(input_df)

# Predict the target probabilities
predicted_probabilities = svm_model.predict_proba(input_scaled)

# Get the predicted target class
predicted_target = svm_model.predict(input_scaled)

# Print the predicted probabilities for each class
print("Predicted probabilities for each class:", predicted_probabilities)
print()
```

THE OUTPUT :

```
Enter the list of 19 double elements separated by spaces:  41   9.789771293     0.013488593     0.295890065     4.22E-05        0.004099865     0.0018011
23      0.042990844     0.001017863     0.027440834     0.003641547     336.8915069     0.047299785     0.201198417     0.000168688     65.15836901     0
0       0

Input list: [41.0, 9.789771293, 0.013488593, 0.295890065, 4.22e-05, 0.004099865, 0.001801123, 0.042990844, 0.001017863, 0.027440834, 0.003641547, 336.891
5069, 0.047299785, 0.201198417, 0.000168688, 65.15836901, 0.0, 0.0, 0.0]
```

```
Predicted probabilities for each class: [[0.00491571 0.01172498 0.0718977  0.90309912 0.00836249]]

Predicted percentage for class Bus: 0.4915707061%
Predicted percentage for class Car: 1.1724981446%
Predicted percentage for class Still: 7.1897699054%
Predicted percentage for class Train: 90.3099122865%
Predicted percentage for class Walking: 0.8362489574%

Predicted target: ['Train']
```


Thank
You