In [1]:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
from sklearn.metrics import recall_score, f1_score, confusion_matrix
from sklearn.metrics import accuracy_score, precision_score
```

In [2]:

```python
dataset_path = "D:/MTP/Mid Term/MId Term 256 models.csv"
data = pd.read_csv(dataset_path)
data.head()
```

Out[2]:

	Sl No	x1	y1	x2	y2	area	Model horizontal Class	Model Vertical class	Model Box Class	natural frequency of Mode 1	natural frequency of Mode 2	natural frequency of Mode 3
0	1	0.00	0.0	0.06	0.06	0.0036	1	1	1	6.8818	25.361	27.925
1	2	0.06	0.0	0.12	0.06	0.0036	1	1	1	6.8854	25.535	27.926
2	3	0.12	0.0	0.18	0.06	0.0036	1	1	1	6.9176	25.713	27.932
3	4	0.18	0.0	0.24	0.06	0.0036	2	1	1	6.9214	25.759	27.926
4	5	0.24	0.0	0.30	0.06	0.0036	2	1	1	6.9401	25.767	27.916

In [3]:

```python
X = data.iloc[:, 9:]  # Features: natural frequencies
y_horizontal = data['Model horizontal Class']
y_vertical = data['Model Vertical class']
y_box = data['Model Box  Class']
```

In [4]:

```python
def classify_svm(X, y, label):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    selector = SelectKBest(f_classif, k='all')
    X_train_selected = selector.fit_transform(X_train_scaled, y_train)
    X_test_selected = selector.transform(X_test_scaled)
    param_grid = {
        'C': [0.1, 1, 10],
        'gamma': [0.1, 0.01, 0.001],
        'kernel': ['linear', 'rbf', 'poly']
    }
    clf = GridSearchCV(SVC(), param_grid, cv=5)
    clf.fit(X_train_selected, y_train)
    y_train_pred = clf.predict(X_train_selected)
    y_test_pred = clf.predict(X_test_selected)
    return y_train, y_train_pred, y_test, y_test_pred
```

In [5]:

```
train_actual_horizontal, train_predicted_horizontal, test_actual_horizonta
train_actual_vertical, train_predicted_vertical, test_actual_vertical, tes
train_actual_box, train_predicted_box, test_actual_box, test_predicted_box
```

In [6]:

```
train_results_df = pd.DataFrame({
    'Actual Horizontal Class': train_actual_horizontal,
    'Predicted Horizontal Class': train_predicted_horizontal,
    'Actual Vertical Class': train_actual_vertical,
    'Predicted Vertical Class': train_predicted_vertical,
    'Actual Box Class': train_actual_box,
    'Predicted Box Class': train_predicted_box
})

test_results_df = pd.DataFrame({
    'Actual Horizontal Class': test_actual_horizontal,
    'Predicted Horizontal Class': test_predicted_horizontal,
    'Actual Vertical Class': test_actual_vertical,
    'Predicted Vertical Class': test_predicted_vertical,
    'Actual Box Class': test_actual_box,
    'Predicted Box Class': test_predicted_box
})
```

In [7]:

```
train_results_df['Horizontal Class Match'] = np.where(train_results_df['Ac
train_results_df['Vertical Class Match'] = np.where(train_results_df['Actu
train_results_df['Box Class Match'] = np.where(train_results_df['Actual Bo

test_results_df['Horizontal Class Match'] = np.where(test_results_df['Actu
test_results_df['Vertical Class Match'] = np.where(test_results_df['Actual
test_results_df['Box Class Match'] = np.where(test_results_df['Actual Box

train_results_df['Cell Number'] = range(1, len(train_results_df) + 1)
test_results_df['Cell Number'] = range(len(train_results_df) + 1, len(trai

```

In [8]:

```
# Counting 'Yes' and 'No' in training results
train_yes_no_counts = {
    'Horizontal Class Match': train_results_df['Horizontal Class Match'].v
    'Vertical Class Match': train_results_df['Vertical Class Match'].value
    'Box Class Match': train_results_df['Box Class Match'].value_counts()
}

print("Training Results 'Yes' and 'No' Counts:")
for key, value in train_yes_no_counts.items():
    print(f"{key}:")
    print(f"Yes: {value['Yes']}, No: {value['No']}")
    print()
```

```
Training Results 'Yes' and 'No' Counts:
Horizontal Class Match:
Yes: 179, No: 25

Vertical Class Match:
Yes: 177, No: 27

Box Class Match:
Yes: 195, No: 9
```

In [9]:

```
# Counting 'Yes' and 'No' in testing results
test_yes_no_counts = {
    'Horizontal Class Match': test_results_df['Horizontal Class Match'].va
    'Vertical Class Match': test_results_df['Vertical Class Match'].value_
    'Box Class Match': test_results_df['Box Class Match'].value_counts()
}

print("\nTesting Results 'Yes' and 'No' Counts:")
for key, value in test_yes_no_counts.items():
    print(f"{key}:")
    print(f"Yes: {value['Yes']}, No: {value['No']}")
    print()

```

```

Testing Results 'Yes' and 'No' Counts:
Horizontal Class Match:
Yes: 39, No: 13

Vertical Class Match:
Yes: 38, No: 14

Box Class Match:
Yes: 43, No: 9
```

In [10]:

```
combined_results_df = pd.concat([train_results_df, test_results_df], ignor
```

In [11]:

```
def calculate_metrics_and_counts_df(actual, predicted):
    report = classification_report(actual, predicted, output_dict=True)
    counts = {'Yes': 0, 'No': 0}
    for key, value in report.items():
        if key not in ['accuracy', 'macro avg', 'weighted avg']:
            counts['Yes'] += value['precision'] * value['support']
            counts['No'] += (1 - value['precision']) * value['support']
    metrics_df = pd.DataFrame(report).transpose()
    metrics_df = metrics_df[metrics_df.index.isin(['0', '1'])]  # Consider
    counts_df = pd.DataFrame([counts], index=['Counts'])
    metrics_counts_df = pd.concat([metrics_df, counts_df])
    return metrics_counts_df
```

In [12]:

```
# Calculate metrics for entire dataset
accuracy_total_horizontal = accuracy_score(train_results_df['Actual Horizo
precision_total_horizontal = precision_score(train_results_df['Actual Hori
recall_total_horizontal = recall_score(train_results_df['Actual Horizontal
f1_total_horizontal = f1_score(train_results_df['Actual Horizontal Class']
confusion_matrix_total_horizontal = confusion_matrix(train_results_df['Act

accuracy_total_vertical = accuracy_score(train_results_df['Actual Vertical
precision_total_vertical = precision_score(train_results_df['Actual Vertic
recall_total_vertical = recall_score(train_results_df['Actual Vertical Cla
f1_total_vertical = f1_score(train_results_df['Actual Vertical Class'], tr
confusion_matrix_total_vertical = confusion_matrix(train_results_df['Actua

accuracy_total_box = accuracy_score(train_results_df['Actual Box Class'],
precision_total_box = precision_score(train_results_df['Actual Box Class']
recall_total_box = recall_score(train_results_df['Actual Box Class'], trai
f1_total_box = f1_score(train_results_df['Actual Box Class'], train_result
confusion_matrix_total_box = confusion_matrix(train_results_df['Actual Box
```

In [13]:

```python
# Print results for entire dataset - Horizontal Class
print("Metrics for Entire Dataset - Horizontal Class:")
print("Accuracy:", accuracy_total_horizontal)
print("Precision:", precision_total_horizontal)
print("Recall:", recall_total_horizontal)
print("F1 Score:", f1_total_horizontal)
print("Confusion Matrix:\n", confusion_matrix_total_horizontal)

# Print results for entire dataset - Vertical Class
print("\nMetrics for Entire Dataset - Vertical Class:")
print("Accuracy:", accuracy_total_vertical)
print("Precision:", precision_total_vertical)
print("Recall:", recall_total_vertical)
print("F1 Score:", f1_total_vertical)
print("Confusion Matrix:\n", confusion_matrix_total_vertical)

# Print results for entire dataset - Box Class
print("\nMetrics for Entire Dataset - Box Class:")
print("Accuracy:", accuracy_total_box)
print("Precision:", precision_total_box)
print("Recall:", recall_total_box)
print("F1 Score:", f1_total_box)
print("Confusion Matrix:\n", confusion_matrix_total_box)
```

```
Metrics for Entire Dataset - Horizontal Class:
Accuracy: 0.8774509803921569
Precision: 0.8976060794688245
Recall: 0.8774509803921569
F1 Score: 0.8788953331408574
Confusion Matrix:
 [[37  0  1  0  0]
 [11 26  1  0  0]
 [ 4  2 40  0  0]
 [ 2  0  3 39  0]
 [ 0  0  0  1 37]]

Metrics for Entire Dataset - Vertical Class:
Accuracy: 0.8676470588235294
Precision: 0.8976364052209754
Recall: 0.8676470588235294
F1 Score: 0.8710278005817745
Confusion Matrix:
 [[30  1  1  1  2]
 [ 2 38  0  0  2]
 [ 0  0 48  0  7]
 [ 0  0  2 24  9]
 [ 0  0  0  0 37]]

Metrics for Entire Dataset - Box Class:
Accuracy: 0.9558823529411765
Precision: 0.9610955493308435
Recall: 0.9558823529411765
F1 Score: 0.9565698934964224
Confusion Matrix:
 [[18  0  0  0  0  0  0  0  0]
 [ 0 20  0  0  0  0  0  0  0]
```

```
[ 0  0 24  0  0  0  0  0  0]
[ 0  0  0 19  0  0  3  0  0]
[ 0  1  0  1 17  0  0  0  0]
[ 0  0  0  0  0 29  0  0  0]
[ 0  0  0  1  0  0 21  0  0]
[ 0  0  0  0  0  0  3 18  0]
[ 0  0  0  0  0  0  0  0 29]]
```

In [14]:

```
# Calculate metrics for training data
accuracy_train_horizontal = accuracy_score(train_results_df['Actual Horizo
precision_train_horizontal = precision_score(train_results_df['Actual Hori
recall_train_horizontal = recall_score(train_results_df['Actual Horizontal
f1_train_horizontal = f1_score(train_results_df['Actual Horizontal Class']
confusion_matrix_train_horizontal = confusion_matrix(train_results_df['Act

accuracy_train_vertical = accuracy_score(train_results_df['Actual Vertical
precision_train_vertical = precision_score(train_results_df['Actual Vertic
recall_train_vertical = recall_score(train_results_df['Actual Vertical Cla
f1_train_vertical = f1_score(train_results_df['Actual Vertical Class'], tr
confusion_matrix_train_vertical = confusion_matrix(train_results_df['Actua

accuracy_train_box = accuracy_score(train_results_df['Actual Box Class'],
precision_train_box = precision_score(train_results_df['Actual Box Class']
recall_train_box = recall_score(train_results_df['Actual Box Class'], trai
f1_train_box = f1_score(train_results_df['Actual Box Class'], train_result
confusion_matrix_train_box = confusion_matrix(train_results_df['Actual Box

```

In [15]:

```python
# Print results for training data - Horizontal Class
print("\nMetrics for Training Data - Horizontal Class:")
print("Accuracy:", accuracy_train_horizontal)
print("Precision:", precision_train_horizontal)
print("Recall:", recall_train_horizontal)
print("F1 Score:", f1_train_horizontal)
print("Confusion Matrix:\n", confusion_matrix_train_horizontal)

# Print results for training data - Vertical Class
print("\nMetrics for Training Data - Vertical Class:")
print("Accuracy:", accuracy_train_vertical)
print("Precision:", precision_train_vertical)
print("Recall:", recall_train_vertical)
print("F1 Score:", f1_train_vertical)
print("Confusion Matrix:\n", confusion_matrix_train_vertical)

# Print results for training data - Box Class
print("\nMetrics for Training Data - Box Class:")
print("Accuracy:", accuracy_train_box)
print("Precision:", precision_train_box)
print("Recall:", recall_train_box)
print("F1 Score:", f1_train_box)
print("Confusion Matrix:\n", confusion_matrix_train_box)
```

```
Metrics for Training Data - Horizontal Class:
Accuracy: 0.8774509803921569
Precision: 0.8976060794688245
Recall: 0.8774509803921569
F1 Score: 0.8788953331408574
Confusion Matrix:
 [[37  0  1  0  0]
 [11 26  1  0  0]
 [ 4  2 40  0  0]
 [ 2  0  3 39  0]
 [ 0  0  0  1 37]]

Metrics for Training Data - Vertical Class:
Accuracy: 0.8676470588235294
Precision: 0.8976364052209754
Recall: 0.8676470588235294
F1 Score: 0.8710278005817745
Confusion Matrix:
 [[30  1  1  1  2]
 [ 2 38  0  0  2]
 [ 0  0 48  0  7]
 [ 0  0  2 24  9]
 [ 0  0  0  0 37]]

Metrics for Training Data - Box Class:
Accuracy: 0.9558823529411765
Precision: 0.9610955493308435
Recall: 0.9558823529411765
F1 Score: 0.9565698934964224
Confusion Matrix:
 [[18  0  0  0  0  0  0  0  0]
```

```
[ 0 20  0  0  0  0  0  0  0]
[ 0  0 24  0  0  0  0  0  0]
[ 0  0  0 19  0  0  3  0  0]
[ 0  1  0  1 17  0  0  0  0]
[ 0  0  0  0  0 29  0  0  0]
[ 0  0  0  1  0  0 21  0  0]
[ 0  0  0  0  0  0  3 18  0]
[ 0  0  0  0  0  0  0  0 29]]
```

In [16]:

```
# Calculate metrics for testing data
accuracy_test_horizontal = accuracy_score(test_results_df['Actual Horizont
precision_test_horizontal = precision_score(test_results_df['Actual Horizo
recall_test_horizontal = recall_score(test_results_df['Actual Horizontal C
f1_test_horizontal = f1_score(test_results_df['Actual Horizontal Class'],
confusion_matrix_test_horizontal = confusion_matrix(test_results_df['Actua

accuracy_test_vertical = accuracy_score(test_results_df['Actual Vertical C
precision_test_vertical = precision_score(test_results_df['Actual Vertical
recall_test_vertical = recall_score(test_results_df['Actual Vertical Class
f1_test_vertical = f1_score(test_results_df['Actual Vertical Class'], test
confusion_matrix_test_vertical = confusion_matrix(test_results_df['Actual

accuracy_test_box = accuracy_score(test_results_df['Actual Box Class'], te
precision_test_box = precision_score(test_results_df['Actual Box Class'],
recall_test_box = recall_score(test_results_df['Actual Box Class'], test_r
f1_test_box = f1_score(test_results_df['Actual Box Class'], test_results_d
confusion_matrix_test_box = confusion_matrix(test_results_df['Actual Box C

```

In [17]:

```python
# Print results for testing data - Horizontal Class
print("\nMetrics for Testing Data - Horizontal Class:")
print("Accuracy:", accuracy_test_horizontal)
print("Precision:", precision_test_horizontal)
print("Recall:", recall_test_horizontal)
print("F1 Score:", f1_test_horizontal)
print("Confusion Matrix:\n", confusion_matrix_test_horizontal)

# Print results for testing data - Vertical Class
print("\nMetrics for Testing Data - Vertical Class:")
print("Accuracy:", accuracy_test_vertical)
print("Precision:", precision_test_vertical)
print("Recall:", recall_test_vertical)
print("F1 Score:", f1_test_vertical)
print("Confusion Matrix:\n", confusion_matrix_test_vertical)

# Print results for testing data - Box Class
print("\nMetrics for Testing Data - Box Class:")
print("Accuracy:", accuracy_test_box)
print("Precision:", precision_test_box)
print("Recall:", recall_test_box)
print("F1 Score:", f1_test_box)
print("Confusion Matrix:\n", confusion_matrix_test_box)
```

```
Metrics for Testing Data - Horizontal Class:
Accuracy: 0.75
Precision: 0.8155906593406593
Recall: 0.75
F1 Score: 0.7551434990349019
Confusion Matrix:
 [[ 9  0  1  0  0]
 [ 2  8  0  0  0]
 [ 4  2 11  1  0]
 [ 0  0  0  4  0]
 [ 1  0  0  2  7]]

Metrics for Testing Data - Vertical Class:
Accuracy: 0.7307692307692307
Precision: 0.8013157894736841
Recall: 0.7307692307692307
F1 Score: 0.7327421815408086
Confusion Matrix:
 [[10  2  0  0  1]
 [ 0  3  1  0  2]
 [ 0  1  7  0  1]
 [ 0  0  2  7  4]
 [ 0  0  0  0 11]]

Metrics for Testing Data - Box Class:
Accuracy: 0.8269230769230769
Precision: 0.8774038461538461
Recall: 0.8269230769230769
F1 Score: 0.8370898332436794
Confusion Matrix:
 [[4 0 0 3 0 0 0 0 0]
```

```
 [0 5 0 0 0 0 0 0 0]
 [0 0 6 0 0 0 0 0 0]
 [0 0 0 3 0 0 0 0 0]
 [0 0 0 0 6 0 0 0 0]
 [0 0 0 0 0 1 0 0 0]
 [0 0 0 1 0 0 7 0 0]
 [0 0 0 1 0 0 1 6 1]
 [2 0 0 0 0 0 0 0 5]]
```

In [18]:

```
excel_file_path = "D:/MTP/Second part ML project/combined_results_svm.xlsx
combined_results_df.to_excel(excel_file_path, index=False)

box_class_df = combined_results_df[['Actual Box Class','Predicted Box Clas

random_20_rows = box_class_df.sample(n=20)

print("Random 20 rows from the box_class_df DataFrame:")
random_20_rows
```

```
Random 20 rows from the box_class_df DataFrame:
```

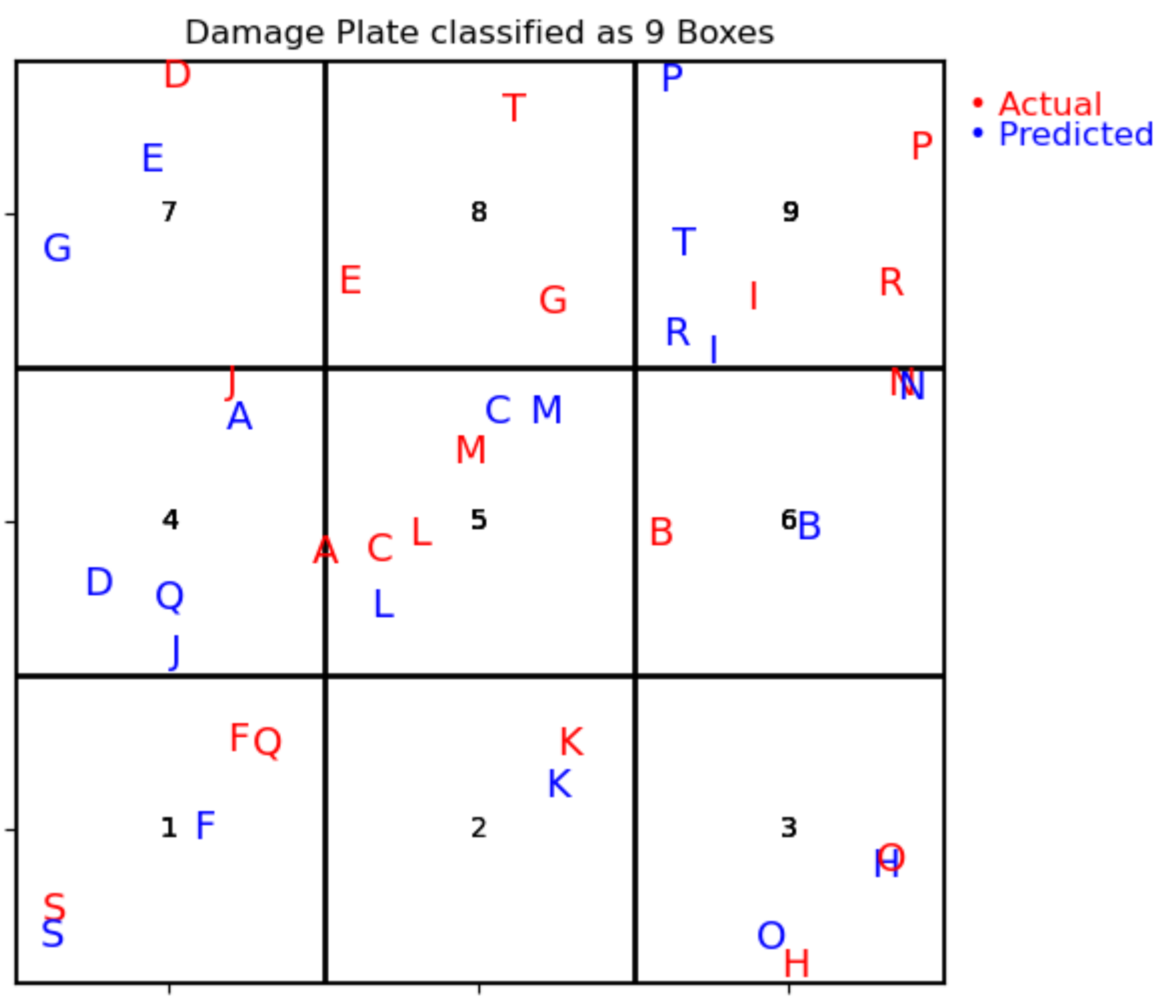
Out[18]:

	Actual Box Class	Predicted Box Class
106	6	6
58	2	2
112	2	2
0	6	6
140	9	9
150	7	7
55	4	7
13	4	7
148	8	8
82	6	6
218	5	5
223	2	2
226	5	5
12	7	7
125	8	8
34	6	6
175	9	9
157	1	1
100	8	8
41	3	3

In [31]:

```python
# Define the Damage_Plate grid size
plate_size = 3  # Adjusted for 20 alphabets

# Create a 3x3 matrix to represent the Damage_Plate grid
Damage_Plate_grid = np.zeros((plate_size, plate_size), dtype=int)

# Assign box numbers to each box in the grid
box_number = 1
for i in range(plate_size):
    for j in range(plate_size):
        Damage_Plate_grid[i, j] = box_number
        box_number += 1

# Function to mark alphabets and write box numbers in the center
def mark_alphabets(box_number_input, alphabet, color):
    # Find the row and column indices of the box
    for i in range(plate_size):
        for j in range(plate_size):
            if Damage_Plate_grid[i, j] == box_number_input:
                row_index = i
                col_index = j

    # Plot the alphabet randomly in the box
    rand_row = row_index + np.random.rand()
    rand_col = col_index + np.random.rand()
    ax.text(rand_col, rand_row, alphabet, fontsize=14, ha='center', va='ce

    # Write the box number in the center
    ax.text(col_index + 0.5, row_index + 0.5, str(box_number_input), fonts

# Create the Damage_Plate grid plot
fig, ax = plt.subplots(figsize=(6, 6))

# Plot the Damage_Plate grid lines
for i in range(plate_size + 1):
    ax.axhline(y=i, color='black', linewidth=2)
    ax.axvline(x=i, color='black', linewidth=2)

# Get box numbers and corresponding values from your DataFrame (replace wi
box1_numbers = box_class_df['Actual Box Class'].sample(n=20, random_state=
box2_numbers = box_class_df['Predicted Box Class'].sample(n=20, random_sta
actual_values = [np.random.randint(1, 10) for _ in range(20)]  # Replace w
predicted_values = [np.random.randint(1, 10) for _ in range(20)]  # Replac

alphabets = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', '
color1 = 'red'
color2 = 'blue'

# Mark alphabets and write box numbers
for i in range(20):
    mark_alphabets(box1_numbers[i], alphabets[i], color1)
    mark_alphabets(box2_numbers[i], alphabets[i], color2)

# Set plot limits and labels
ax.set_xlim(0, plate_size)
```

```python
ax.set_ylim(0, plate_size)
ax.set_xticks(np.arange(0.5, plate_size, 1))
ax.set_yticks(np.arange(0.5, plate_size, 1))
ax.set_xticklabels([])
ax.set_yticklabels([])
ax.grid(False)

# Create text labels outside the plot
offset_x = 3.08  # Adjust x-offset for label placement
offset_y = 2.85  # Adjust y-offset for label placement

label_text1 = plt.text(offset_x, offset_y, "\u2022 Actual", ha='left', va=
label_text2 = plt.text(offset_x, offset_y - 0.1, "\u2022 Predicted", ha='l

# Remove box around the labels (optional)
label_text1.set_bbox(dict(facecolor='none', edgecolor='none', pad=0))
label_text2.set_bbox(dict(facecolor='none', edgecolor='none', pad=0))

# Show the plot
plt.title('Damage Plate classified as 9 Boxes')
plt.show()
```



In [20]:

```python
# Selecting actual and predicted classes for the horizontal class
horizontal_class_df = combined_results_df[['Actual Horizontal Class', 'Pre

random_20_rows_horizontal = horizontal_class_df.sample(n=20)

print("Random 20 rows from the horizontal_class_df DataFrame:")
random_20_rows_horizontal
```

```
Random 20 rows from the horizontal_class_df DataFrame:
```

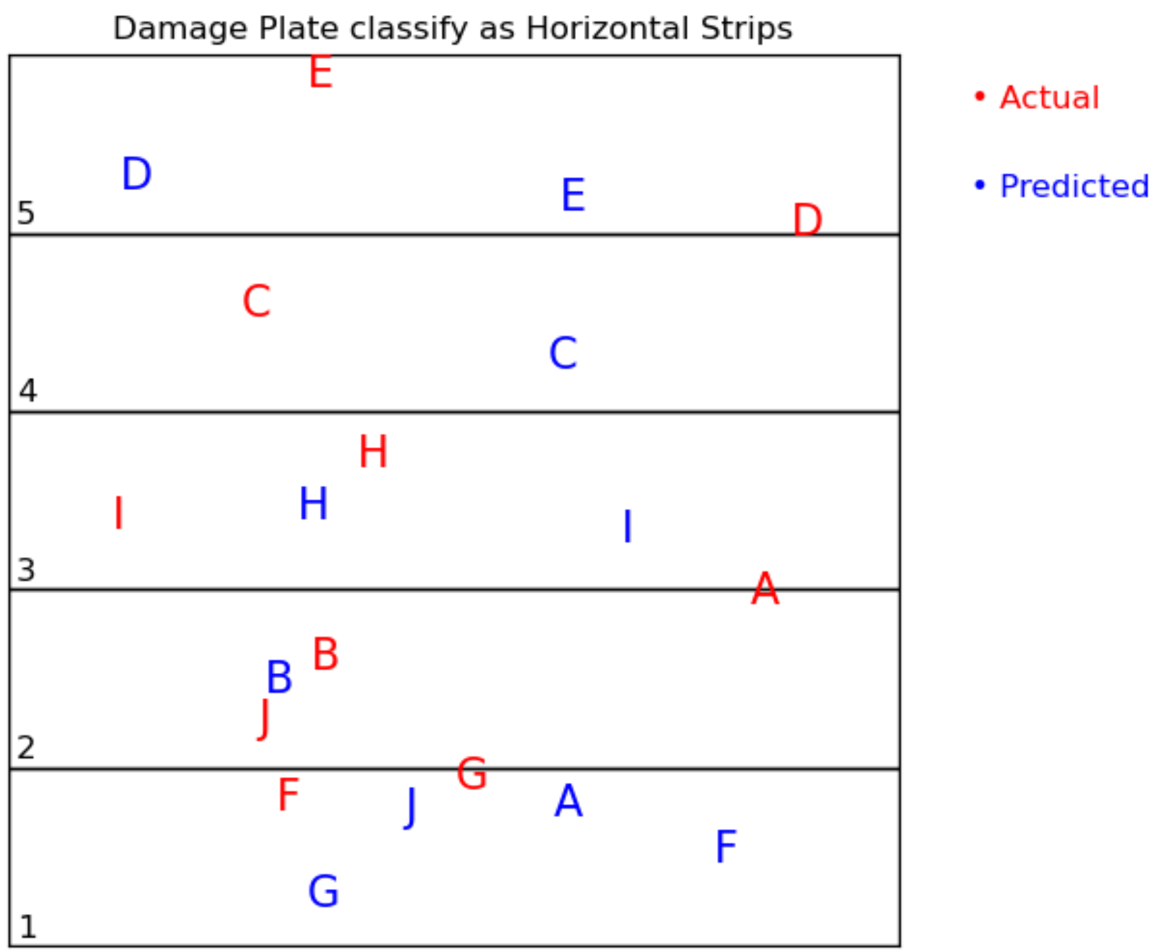
Out[20]:

	Actual Horizontal Class	Predicted Horizontal Class
91	2	1
188	2	2
137	4	4
207	5	5
237	5	5
130	1	1
90	1	1
62	3	3
133	3	3
25	2	1
253	3	2
224	4	4
32	3	3
194	3	3
86	4	4
53	1	1
125	3	1
94	5	5
112	3	3
7	4	4

In [41]:

```
# Extract the actual and predicted horizontal class from the DataFrame
actual_classes = random_20_rows_horizontal['Actual Horizontal Class'].toli
predicted_classes = random_20_rows_horizontal['Predicted Horizontal Class'

# Define the layout of the plate and strip labels
strip_labels = ['5', '4', '3', '2', '1']  # Reverse order to label from bo

# Create a new figure
fig, ax = plt.subplots(figsize=(8, 6))

# Plot horizontal strips and assign numbers
for i, label in enumerate(strip_labels):
    strip_y = (4 - i) / 5  # Adjusting the y-coordinate to represent botto
    rect_strip = plt.Rectangle((0, strip_y), 1, 1 / 5, fill=False, edgecol
    ax.add_patch(rect_strip)
    # Add strip label at the left corner of each strip
    ax.text(0.02, strip_y + 0.02, label, ha='center', va='center', fontsiz

# Set axis limits and labels
ax.set_xlim(0, 1)
ax.set_ylim(0, 1)
ax.set_xticks([])
ax.set_yticks([])
ax.set_aspect('equal')

# Add title
plt.title('Damage Plate classify as Horizontal Strips')

# Initialize lists to store input data for red and blue alphabets
red_alphabets = []
blue_alphabets = []

# Take 5 inputs for red alphabets
for i, strip_num in enumerate(actual_classes):
    letter = chr(ord('A') + i)  # Convert index to corresponding alphabet
    red_alphabets.append((letter, strip_num))

# Take 5 inputs for blue alphabets
for i, strip_num in enumerate(predicted_classes):
    letter = chr(ord('A') + i)  # Convert index to corresponding alphabet
    blue_alphabets.append((letter, strip_num))

# Function to mark alphabets on the plate
def mark_alphabets(alphabets, color):
    for letter, strip_num in alphabets:
        if 1 <= strip_num <= 5:
            strip_y = (strip_num - 1) / 5  # y-coordinate of the selected
            # Generate random x and y coordinates inside the selected stri
            random_x_coordinate = np.random.uniform(0.1, 0.9)  # Adjust th
            random_y_coordinate = np.random.uniform(strip_y, strip_y + 0.2
            # Marking the alphabet with specified color
            ax.text(random_x_coordinate, random_y_coordinate, letter, ha='

# Create text labels outside the plot
offset_x = 1.08  # Adjust x-offset for label placement
```

```
offset_y = .95  # Adjust y-offset for label placement

label_text1 = plt.text(offset_x, offset_y, "\u2022 Actual", ha='left', va=
label_text2 = plt.text(offset_x, offset_y - 0.1, "\u2022 Predicted", ha='l

# Mark red alphabets
mark_alphabets(red_alphabets, 'red')

# Mark blue alphabets
mark_alphabets(blue_alphabets, 'blue')

# Show the plot
plt.show()

```



In [22]:

```python
# Selecting actual and predicted classes for the vertical class
vertical_class_df = combined_results_df[['Actual Vertical Class', 'Predict

# Select 20 random rows for demonstration
random_20_rows_vertical = vertical_class_df.sample(n=20)

print("Random 20 rows from the vertical_class_df DataFrame:")
random_20_rows_vertical
```

```
Random 20 rows from the vertical_class_df DataFrame:
```

Out[22]:

	Actual Vertical Class	Predicted Vertical Class
9	4	4
203	3	3
57	2	2
225	1	2
186	3	3
177	3	3
125	5	5
132	5	5
127	5	5
133	1	1
210	5	5
27	1	1
158	5	5
250	1	1
148	4	4
147	1	1
229	3	3
55	3	5
51	5	5
54	2	2

In [44]:

```
# Extract the actual and predicted vertical class from the DataFrame
actual_classes = random_20_rows_vertical['Actual Vertical Class'].tolist()
predicted_classes = random_20_rows_vertical['Predicted Vertical Class'].to

# Define the layout of the plate and strip labels
strip_labels = ['1', '2', '3', '4', '5']  # Label from bottom to top for v

# Create a new figure
fig, ax = plt.subplots(figsize=(8, 6))

# Plot vertical strips and assign numbers
for i, label in enumerate(strip_labels):
    strip_x = i / 5  # Adjusting the x-coordinate to represent leftmost st
    rect_strip = plt.Rectangle((strip_x, 0), 1 / 5, 1, fill=False, edgecol
    ax.add_patch(rect_strip)
    # Add strip label at the bottom corner of each strip
    ax.text(strip_x + 0.02, 0.02, label, ha='center', va='center', fontsiz

# Set axis limits and labels
ax.set_xlim(0, 1)
ax.set_ylim(0, 1)
ax.set_xticks([])
ax.set_yticks([])
ax.set_aspect('equal')

# Add title
plt.title('Damage Plate classify as  Vertical Strips')

# Initialize lists to store input data for red and blue alphabets
red_alphabets = []
blue_alphabets = []

# Take 5 inputs for red alphabets
for i, strip_num in enumerate(actual_classes):
    letter = chr(ord('A') + i)  # Convert index to corresponding alphabet
    red_alphabets.append((letter, strip_num))

# Take 5 inputs for blue alphabets
for i, strip_num in enumerate(predicted_classes):
    letter = chr(ord('A') + i)  # Convert index to corresponding alphabet
    blue_alphabets.append((letter, strip_num))

# Function to mark alphabets on the plate
def mark_alphabets(alphabets, color):
    for letter, strip_num in alphabets:
        if 1 <= strip_num <= 5:
            strip_x = (strip_num - 1) / 5  # x-coordinate of the selected
            # Generate random x and y coordinates inside the selected stri
            random_x_coordinate = np.random.uniform(strip_x, strip_x + 0.2
            random_y_coordinate = np.random.uniform(0.1, 0.9)  # Adjust th
            # Marking the alphabet with specified color
            ax.text(random_x_coordinate, random_y_coordinate, letter, ha='

# Create text labels outside the plot
offset_x = 1.08  # Adjust x-offset for label placement
```

```python
offset_y = .95  # Adjust y-offset for label placement

label_text1 = plt.text(offset_x, offset_y, "\u2022 Actual", ha='left', va=
label_text2 = plt.text(offset_x, offset_y - 0.1, "\u2022 Predicted", ha='l

# Mark red alphabets
mark_alphabets(red_alphabets, 'red')

# Mark blue alphabets
mark_alphabets(blue_alphabets, 'blue')

# Show the plot
plt.show()

```