DroneDetect V2 - SVM Classification

Train SVM classifier on PSD features:

- RBF kernel (C=1.0, gamma='scale')
- File-level stratified split to prevent data leakage
- Comprehensive performance evaluation

1. Mount Google Drive

In [7]:
```
from google.colab import drive

drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tru
e).
```

2. Imports

In [8]:
```
!pip install -U kaleido==0.2.1
```

```
Requirement already satisfied: kaleido==0.2.1 in /usr/local/lib/python3.12/dist-packages (0.2.1)
```

In [9]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, f1_score, precision_recall_fscore_support
from sklearn.model_selection import StratifiedGroupKFold
import pickle
import os
```

```
import re
from pathlib import Path

plt.style.use('seaborn-v0_8-darkgrid')
%matplotlib inline

# Figure saving configuration
NOTEBOOK_NAME = "training_svm_COLAB"
FIGURES_DIR = Path("figures") / NOTEBOOK_NAME


def save_figure(fig) -> None:
    """Save plotly figure to PNG file using the figure's title as filename."""
    FIGURES_DIR.mkdir(parents=True, exist_ok=True)
    title = fig.layout.title.text if fig.layout.title.text else "untitled"
    filename = re.sub(r'[^\w\s-]', '', title).strip()
    filename = re.sub(r'[\s-]+', '_', filename)
    filepath = FIGURES_DIR / f"{filename}.png"
    try:
        fig.write_image(str(filepath), width=1200, height=800)
        print(f"Saved: {filepath}")
    except Exception as e:
        print(f"Warning: Could not save figure (kaleido required): {e}")
```

3. Configuration

In [10]:

```
CONFIG = {
    'features_path': 'drive/MyDrive/DroneDetect_V2/output/features/psd_features.npz',
    'models_dir': 'drive/MyDrive/DroneDetect_V2/output/models/',
    'test_data_dir': 'drive/MyDrive/DroneDetect_V2/output/sample/test_data/',
    'test_size': 0.2,
    'random_state': 42,
    'C': 1.0,
    'gamma': 'scale',
    'kernel': 'rbf'
}

print("Configuration:")
```

```
for key, value in CONFIG.items():
    print(f"  {key}: {value}")
```

```
Configuration:
  features_path: drive/MyDrive/DroneDetect_V2/output/features/psd_features.npz
  models_dir: drive/MyDrive/DroneDetect_V2/output/models/
  test_data_dir: drive/MyDrive/DroneDetect_V2/output/sample/test_data/
  test_size: 0.2
  random_state: 42
  C: 1.0
  gamma: scale
  kernel: rbf
```

4. File-Level Stratified Split Function

In [11]:

```
def get_stratified_file_split(X, y, file_ids, test_size=0.2, random_state=42):
    """
    Split data at FILE level to prevent data leakage.

    Segments from the same .dat file (~100 segments) will never appear
    in both train and test sets.

    Parameters
    ----------
    X : array-like
        Features (n_samples, ...)
    y : array-like
        Labels for stratification (n_samples,)
    file_ids : array-like
        Source file ID for each sample (n_samples,)
    test_size : float
        Approximate test set proportion (actual may vary due to file grouping)
    random_state : int
        Random seed for reproducibility

    Returns
    -------
    train_idx, test_idx : arrays
        Indices for train/test split
    """
```

```
    n_splits = int(1 / test_size)  # e.g., test_size=0.2 -> 5 splits -> 1 fold = 20%

    sgkf = StratifiedGroupKFold(n_splits=n_splits, shuffle=True, random_state=random_state)

    # Take first fold as train/test split
    train_idx, test_idx = next(sgkf.split(X, y, groups=file_ids))

    # Verify no file leakage
    train_files = set(file_ids[train_idx])
    test_files = set(file_ids[test_idx])
    assert len(train_files & test_files) == 0, "Data leakage detected: files in both splits"

    return train_idx, test_idx

print("Stratified file split function defined")
```

```
Stratified file split function defined
```

5. Load PSD Features

In [12]:

```
data = np.load(CONFIG['features_path'], allow_pickle=True)

X = data['X']
y_drone = data['y_drone']
y_interference = data['y_interference']
y_state = data['y_state']
file_ids = data['file_ids']
drone_classes = data['drone_classes']
interference_classes = data['interference_classes']
state_classes = data['state_classes']

print(f"Features shape: {X.shape}")
print(f"Drone labels shape: {y_drone.shape}")
print(f"File IDs shape: {file_ids.shape} (unique files: {len(np.unique(file_ids))})")
print(f"Drone classes: {drone_classes}")
print(f"Interference classes: {interference_classes}")
print(f"State classes: {state_classes}")
```

```
Features shape: (19478, 1024)
Drone labels shape: (19478,)
File IDs shape: (19478,) (unique files: 195)
Drone classes: ['AIR' 'DIS' 'INS' 'MA1' 'MAV' 'MIN' 'PHA']
Interference classes: ['BOTH' 'CLEAN']
State classes: ['FY' 'HO' 'ON']
```

6. Train/Test Split

In [13]:

```python
# Split for drone classification using file-level stratification
train_idx, test_idx = get_stratified_file_split(
    X, y_drone, file_ids,
    test_size=CONFIG['test_size'],
    random_state=CONFIG['random_state']
)

X_train, X_test = X[train_idx], X[test_idx]
y_train, y_test = y_drone[train_idx], y_drone[test_idx]
y_interference_test = y_interference[test_idx]
y_state_test = y_state[test_idx]

# Verify no file leakage
train_files = set(file_ids[train_idx])
test_files = set(file_ids[test_idx])
print(f"Training files: {len(train_files)}")
print(f"Test files: {len(test_files)}")
print(f"File overlap: {len(train_files & test_files)} (should be 0)")

print(f"\nTraining set: {X_train.shape}")
print(f"Test set: {X_test.shape}")

# Save test data for reuse
test_data_dir = CONFIG['test_data_dir']
os.makedirs(test_data_dir, exist_ok=True)

# Save full test data
test_data_path = os.path.join(test_data_dir, 'svm_test_data.npz')
np.savez(
    test_data_path,
```

```
    X_test=X_test,
    y_test=y_test,
    y_interference_test=y_interference_test,
    y_state_test=y_state_test,
    test_idx=test_idx,
    file_ids_test=file_ids[test_idx],
    drone_classes=drone_classes,
    interference_classes=interference_classes,
    state_classes=state_classes
)
print(f"\nTest data saved to {test_data_path}")

# Save separated files per Drone and Interference (Hierarchical)
print("\nGenerating separated test files (structure: psd/INT/DRONE/)...")

for d_idx, drone_class in enumerate(drone_classes):
    for i_idx, int_class in enumerate(interference_classes):
        # Filter for specific drone and interference
        mask = (y_test == d_idx) & (y_interference_test == i_idx)

        if not np.any(mask):
            continue

        X_sub = X_test[mask]
        y_sub = y_test[mask]
        y_int_sub = y_interference_test[mask]

        # Define components for hierarchy and filename
        data_type = 'psd'
        int_name = str(int_class)
        drone_name = str(drone_class)
        duration = '20' # 20ms fixed duration

        # Create directory structure: output/sample/test_data/{INT}/
        save_dir = os.path.join(test_data_dir, int_name)
        os.makedirs(save_dir, exist_ok=True)

        # Construct filename: psd_{INT}_{DRONE}_20.npz
        filename = f"{data_type}_{int_name}_{drone_name}_{duration}.npz"
        file_path = os.path.join(save_dir, filename)
```

```
        np.savez(
            file_path,
            X=X_sub,
            y=y_sub,
            y_interference=y_int_sub,
            drone_class=drone_class,
            interference_class=int_class
        )
        print(f"  Saved {filename} in {save_dir}")
```

```
Training files: 156
Test files: 39
File overlap: 0 (should be 0)

Training set: (15587, 1024)
Test set: (3891, 1024)

Test data saved to drive/MyDrive/DroneDetect_V2/output/sample/test_data/svm_test_data.npz

Generating separated test files (structure: psd/INT/DRONE/)...
  Saved psd_BOTH_AIR_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/BOTH
  Saved psd_CLEAN_AIR_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/CLEAN
  Saved psd_BOTH_DIS_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/BOTH
  Saved psd_CLEAN_DIS_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/CLEAN
  Saved psd_BOTH_INS_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/BOTH
  Saved psd_CLEAN_INS_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/CLEAN
  Saved psd_BOTH_MA1_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/BOTH
  Saved psd_CLEAN_MA1_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/CLEAN
  Saved psd_BOTH_MAV_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/BOTH
  Saved psd_CLEAN_MAV_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/CLEAN
  Saved psd_BOTH_MIN_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/BOTH
  Saved psd_CLEAN_MIN_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/CLEAN
  Saved psd_BOTH_PHA_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/BOTH
  Saved psd_CLEAN_PHA_20.npz in drive/MyDrive/DroneDetect_V2/output/sample/test_data/CLEAN
```

7. Train SVM Classifier

In [14]:

```
# Initialize and train SVM
svm_model = SVC(C=CONFIG['C'], gamma=CONFIG['gamma'], kernel=CONFIG['kernel'])
```

```
print("Training SVM...")
svm_model.fit(X_train, y_train)
print("Training complete!")
```

```
Training SVM...
Training complete!
```

8. Evaluation

In [15]:
```
# Predictions
y_pred = svm_model.predict(X_test)

# Metrics
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Test Accuracy: {accuracy:.4f}")
print(f"Test F1-Score (weighted): {f1:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=drone_classes))
```

```
Test Accuracy: 0.8293
Test F1-Score (weighted): 0.8277

Classification Report:
              precision    recall  f1-score   support

         AIR       0.67      0.75      0.71       600
         DIS       1.00      0.96      0.98       400
         INS       0.88      0.98      0.92       591
         MA1       0.64      0.60      0.62       600
         MAV       0.78      0.69      0.73       700
         MIN       1.00      1.00      1.00       600
         PHA       0.93      0.94      0.94       400

    accuracy                           0.83      3891
   macro avg       0.84      0.84      0.84      3891
weighted avg       0.83      0.83      0.83      3891
```

9. Confusion Matrix

In [16]:
```
cm = confusion_matrix(y_test, y_pred)

# Create confusion matrix heatmap with plotly
fig = go.Figure(data=go.Heatmap(
    z=cm,
    x=list(drone_classes),
    y=list(drone_classes),
    colorscale='Blues',
    text=cm,
    texttemplate='%{text}',
    textfont={'size': 12},
    hoverongaps=False
))

fig.update_layout(
    title=f'SVM Confusion Matrix - Accuracy: {accuracy:.4f} | F1: {f1:.4f}',
    xaxis_title='Predicted',
    yaxis_title='True',
    xaxis={'side': 'bottom'},
    yaxis={'autorange': 'reversed'},
    width=800,
    height=700
)
fig.show()
save_figure(fig)
```

```
Saved: figures/training_svm_COLAB/SVM_Confusion_Matrix_Accuracy_08293_F1_08277.png
```

10. Per-Class Performance

In [17]:

```python
# Calculate per-class metrics
precision, recall, f1_per_class, support = precision_recall_fscore_support(
    y_test, y_pred, labels=range(len(drone_classes)), zero_division=0
)

# Import pandas for DataFrame
import pandas as pd

# Create a DataFrame for per-class metrics
metrics_df = pd.DataFrame({
    'Class': drone_classes,
    'Precision': precision,
    'Recall': recall,
    'F1-Score': f1_per_class,
    'Support': support
})

print("\nPer-Class Performance:")
print(metrics_df.to_string(index=False))

# Precision plot
fig_precision = px.bar(metrics_df, x='Class', y='Precision', title='SVM Precision per Class',
                       color='Precision', range_y=[0, 1.05])
fig_precision.update_layout(xaxis_title="Class", yaxis_title="Precision",
                            title_font_size=16, height=400)
fig_precision.show()
save_figure(fig_precision)

# Recall plot
fig_recall = px.bar(metrics_df, x='Class', y='Recall', title='SVM Recall per Class',
                    color='Recall', color_continuous_scale=px.colors.sequential.Oranges,
                    range_y=[0, 1.05])
fig_recall.update_layout(xaxis_title="Class", yaxis_title="Recall",
                         title_font_size=16, height=400)
fig_recall.show()
```

```
save_figure(fig_recall)

# F1-Score plot
fig_f1 = px.bar(metrics_df, x='Class', y='F1-Score', title='SVM F1-Score per Class',
                color='F1-Score', color_continuous_scale=px.colors.sequential.Greens,
                range_y=[0, 1.05])
fig_f1.update_layout(xaxis_title="Class", yaxis_title="F1-Score",
                     title_font_size=16, height=400)
fig_f1.show()
save_figure(fig_f1)
```

```
Per-Class Performance:
Class  Precision   Recall  F1-Score  Support
  AIR   0.670641 0.750000  0.708104      600
  DIS   1.000000 0.957500  0.978289      400
  INS   0.875758 0.978003  0.924061      591
  MA1   0.643885 0.596667  0.619377      600
  MAV   0.782820 0.690000  0.733485      700
  MIN   1.000000 0.996667  0.998331      600
  PHA   0.928571 0.942500  0.935484      400
```

Saved: figures/training_svm_COLAB/SVM_Precision_per_Class.png

Saved: figures/training_svm_COLAB/SVM_Recall_per_Class.png

```
Saved: figures/training_svm_COLAB/SVM_F1_Score_per_Class.png
```

11. Save Model

In [18]:
```python
# Ensure the directory exists
os.makedirs(CONFIG['models_dir'], exist_ok=True)

model_path = os.path.join(CONFIG['models_dir'], 'svm_psd_drone.pkl')

with open(model_path, 'wb') as f:
    pickle.dump({
        'model': svm_model,
        'classes': drone_classes,
        'accuracy': accuracy,
        'f1': f1
```

```
    }, f)

print(f"Model saved to {model_path}")
```

```
Model saved to drive/MyDrive/DroneDetect_V2/output/models/svm_psd_drone.pkl
```

12. Summary

In [19]:

```
print("="*60)
print("SVM CLASSIFICATION SUMMARY")
print("="*60)

print(f"\nDataset:")
print(f"  Total samples: {len(X_train) + len(X_test)}")
print(f"  Training samples: {len(X_train)}")
print(f"  Test samples: {len(X_test)}")
print(f"  Number of classes: {len(drone_classes)}")
print(f"  Classes: {', '.join(drone_classes)}")

print(f"\nModel Configuration:")
print(f"  Algorithm: Support Vector Machine (SVM)")
print(f"  Kernel: {CONFIG['kernel']}")
print(f"  C: {CONFIG['C']}")
print(f"  Gamma: {CONFIG['gamma']}")
print(f"  Features: PSD (Power Spectral Density)")

print(f"\nPerformance:")
print(f"  Test Accuracy: {accuracy:.4f}")
print(f"  Test F1-Score: {f1:.4f}")

print(f"\nModel saved to: {model_path}")

print("\n" + "="*60)
```

```
============================================================
SVM CLASSIFICATION SUMMARY
============================================================

Dataset:
  Total samples: 19478
  Training samples: 15587
  Test samples: 3891
  Number of classes: 7
  Classes: AIR, DIS, INS, MA1, MAV, MIN, PHA

Model Configuration:
  Algorithm: Support Vector Machine (SVM)
  Kernel: rbf
  C: 1.0
  Gamma: scale
  Features: PSD (Power Spectral Density)

Performance:
  Test Accuracy: 0.8293
  Test F1-Score: 0.8277

Model saved to: drive/MyDrive/DroneDetect_V2/output/models/svm_psd_drone.pkl

============================================================
```