Srashti Singh

241030083

Kaggle ID:25048810

In [55]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import f1_score, classification_report
```

In [57]:
```
import os

# Check the current directory
print("Current working directory:", os.getcwd())

# Check if 'train.txt' exists in the directory
print("Files in current directory:", os.listdir())

# Verify if the file is found
train_path = "upload/train.txt"
test_path = "upload/test.txt"

if not os.path.exists(train_path):
    print("❌ ERROR: The file 'train.txt' was not found! Check the file path.")
else:
    print("✅ 'train.txt' found. Proceeding with loading.")
```

```
Current working directory: C:\Users\Hp\Downloads\ts_classification_pavement_ce784_114fff2c-c186-4332-a225-b273f0cca04b
Files in current directory: ['.ipynb_checkpoints', 'assignment11.ipynb', 'MLASSIGN1.ipynb', 'submission.csv', 'Untitled.ipynb',
'upload', '__MACOSX']
✅ 'train.txt' found. Proceeding with loading.
```

In [59]:

```python
import os

# Check if the file exists
train_path = "upload/train.txt"
test_path = "upload/test.txt"

if not os.path.exists(train_path):
    print(f"❌ ERROR: The file '{train_path}' does not exist!")
else:
    print(f"✅ Found '{train_path}'.")

# Check column inconsistencies
with open(train_path, "r", encoding="utf-8") as file:
    lines = file.readlines()

column_counts = [len(line.split(",")) for line in lines]
print("Unique column counts in train.txt:", set(column_counts))  # Should be one unique value
```

```
✅ Found 'upload/train.txt'.
Unique column counts in train.txt: {98, 99, 100, 101, 103, 105, 107, 108, 110, 111, 112, 113, 114, 115, 116, 118, 122, 124, 12
5, 126, 127, 129, 130, 131, 133, 134, 135, 136, 137, 138, 139, 140, 142, 146, 147, 154, 156, 157, 161, 168, 176, 179, 180, 181,
182, 184, 185, 186, 187, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 207, 210, 215, 21
6, 218, 221, 222, 223, 224, 225, 226, 227, 228, 229, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 245, 246,
247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 264, 265, 266, 267, 268, 270, 271, 272, 273, 27
4, 275, 276, 277, 279, 280, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 296, 297, 299, 300, 301, 302, 303,
304, 305, 306, 307, 309, 312, 313, 314, 315, 318, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 334, 336, 33
7, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 349, 350, 351, 353, 355, 356, 357, 358, 359, 360, 362, 363, 365, 366, 367,
368, 369, 370, 371, 374, 377, 378, 379, 381, 382, 383, 384, 385, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 39
9, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424,
425, 426, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 45
1, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476,
477, 478, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 50
3, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528,
529, 530, 531, 532, 533, 534, 535, 536, 538, 539, 541, 542, 544, 545, 546, 547, 548, 550, 551, 552, 553, 554, 556, 558, 559, 56
1, 563, 564, 565, 567, 568, 569, 570, 572, 573, 575, 577, 578, 579, 580, 581, 582, 583, 584, 586, 587, 588, 590, 591, 592, 594,
596, 598, 599, 600, 602, 604, 607, 609, 610, 611, 613, 614, 616, 618, 620, 621, 622, 623, 624, 625, 627, 628, 634, 635, 636, 64
1, 642, 646, 647, 648, 649, 652, 657, 662, 663, 664, 676, 678, 680, 682, 684, 685, 686, 688, 695, 708, 710, 716, 718, 722, 724,
730, 740, 743, 756, 767, 782, 787, 800, 824, 827, 842, 953, 958, 959, 989, 995, 1130, 1545}
```

In [61]:

```python
# Step 1: Load dataset correctly
def load_data(train_path="upload/train.txt", test_path="upload/test.txt"):
    # Detect correct column count
    with open(train_path, "r", encoding="utf-8") as file:
        lines = file.readlines()
    expected_columns = min([len(line.split(",")) for line in lines])  # Use the most common column count

    # Read file with fixed column count
    train_df = pd.read_csv(train_path, sep=",", header=None, usecols=range(expected_columns))
    test_df = pd.read_csv(test_path, sep=",", header=None, usecols=range(expected_columns))

    print(f"✅ Train and test datasets loaded successfully with {expected_columns} columns.")
    return train_df, test_df
```

In [63]:

```python
# Step 2: Extract Features

def extract_features(data):
    """Extract statistical and frequency-based features from time series data."""
    features = pd.DataFrame()
```

```
    # Statistical Features
    features['mean'] = data.iloc[:, 2:].mean(axis=1)
    features['std'] = data.iloc[:, 2:].std(axis=1)
    features['max'] = data.iloc[:, 2:].max(axis=1)
    features['min'] = data.iloc[:, 2:].min(axis=1)
    features['median'] = data.iloc[:, 2:].median(axis=1)
    features['range'] = features['max'] - features['min']
    features['iqr'] = data.iloc[:, 2:].quantile(0.75, axis=1) - data.iloc[:, 2:].quantile(0.25, axis=1)
    features['variance'] = data.iloc[:, 2:].var(axis=1)

    # Frequency Domain Features (Fourier Transform)
    fft_vals = np.abs(np.fft.fft(data.iloc[:, 2:], axis=1))
    features['fft_mean'] = fft_vals.mean(axis=1)
    features['fft_std'] = fft_vals.std(axis=1)
    features['fft_max'] = fft_vals.max(axis=1)

    return features
```

In [65]:

```
# Step 3: Preprocess Dataset

from sklearn.feature_selection import SelectKBest, f_classif
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler

def preprocess_data(train_path="train.txt", test_path="test.txt"):
    train_df, test_df = load_data(train_path, test_path)

    X_train = extract_features(train_df)
    y_train = train_df.iloc[:, 1]  # Class labels
    X_test = extract_features(test_df)

    # Fill missing values with column means
    X_train.fillna(X_train.mean(), inplace=True)
    X_test.fillna(X_test.mean(), inplace=True)

    # Feature Selection: Keep only top 10 best features
    selector = SelectKBest(f_classif, k=10)  # Selects 10 most relevant features
    X_train_selected = selector.fit_transform(X_train, y_train)
    X_test_selected = selector.transform(X_test)
```

```
    # ✅ Apply SMOTE *AFTER* feature selection
    sm = SMOTE(random_state=42)
    X_train_resampled, y_train_resampled = sm.fit_resample(X_train_selected, y_train)

    # ✅ Standardize only AFTER resampling
    scaler = StandardScaler()
    X_train_final = scaler.fit_transform(X_train_resampled)
    X_test_final = scaler.transform(X_test_selected)

    return X_train_final, y_train_resampled, X_test_final, test_df
```

In [73]:

```
# Step 4: Train and Evaluate Model

from sklearn.model_selection import cross_val_score

def train_and_evaluate(train_path="upload/train.txt", test_path="upload/test.txt"):
    X_train, y_train, X_test, test_df = preprocess_data(train_path, test_path)

    # Logistic Regression
    lr_model = LogisticRegression(C=0.5, solver='liblinear', max_iter=2000)
    lr_model.fit(X_train, y_train)

    # ANN Model
    ann_model = MLPClassifier(hidden_layer_sizes=(256, 128, 64, 32), activation='tanh',
                          solver='adam', max_iter=4000, alpha=0.00001,
                          learning_rate_init=0.0002, batch_size=32)
    ann_model.fit(X_train, y_train)

    return ann_model, X_train, y_train, X_test, test_df

# Train the models first
ann_model, X_train, y_train, X_test, test_df = train_and_evaluate()

# Now, run cross-validation on ANN
cv_scores = cross_val_score(ann_model, X_train, y_train, cv=5, scoring='f1_weighted')
print("Cross-validated ANN F1-score:", np.mean(cv_scores))
```

```
✅ Train and test datasets loaded successfully with 98 columns.
Cross-validated ANN F1-score: 0.7653803176688245
```

In [77]:
```python
# Step 5: Generate predictions and create submission file
def create_submission(model, X_test, test_df, filename="submission.csv"):
    y_pred = model.predict(X_test)
    submission = pd.DataFrame({"index": test_df.iloc[:, 0], "label": y_pred})
    submission.to_csv(filename, index=False)
    print(f"Submission file saved as {filename}")
```

In [ ]:
```python
# Step 6: Run the pipeline
if __name__ == "__main__":
    ann_model, X_train, y_train, X_test, test_df = train_and_evaluate()
    create_submission(ann_model, X_test, test_df)

    # Run cross-validation
    cv_scores = cross_val_score(ann_model, X_train, y_train, cv=5, scoring='f1_weighted')
    print("Cross-validated ANN F1-score:", np.mean(cv_scores))
```

✅ Train and test datasets loaded successfully with 98 columns.
Submission file saved as submission.csv

In [ ]:

In [ ]: