Bird Vocalization Classification Using Machine Learning

111550182

1. Introduction

This project aims to classify bird species based on their vocalizations using machine learning techniques. The dataset was constructed using audio recordings sourced from Xeno-canto, focusing on four bird species: **Anas platyrhynchos**, **Corvus macrorhynchos**, **Larus canus**, and **Passer montanus**. The objective is to evaluate the effectiveness of various classification methods, including supervised and unsupervised learning techniques.

2. Dataset

The dataset consists of bird vocalization recordings obtained from the Xeno-canto database. All recordings are collected by using Xeno-canto API. The recordings were preprocessed and segmented into 10-second WAV files.

The dataset is sharing with the partner, Student ID: 110550205.

- **Total Samples:** 5372 recordings (each divided into 10-second segments)
- **Species Distribution:**
  - *Anas platyrhynchos*: 1342 samples
  - *Corvus macrorhynchos*: 1218 samples
  - *Larus canus*: 1207 samples
  - *Passer montanus*: 1605 samples
- **Feature Extraction:**

  - Mel-Frequency Cepstral Coefficients (MFCCs) were extracted for SVM, Random Forest, and K-Means models. 40 dimensions.
  - Mel-spectrograms were used as input features for EfficientNet.

- **Split:** 80% training, 20%testing

- **Dataset URL:** https://github.com/Kennethii2i/NYCU-AI-Capstone-Spr2025-Project1.git

3. Method

- **Support Vector Machine (SVM)**: Trained on extracted MFCC features.
- **Random Forest**: Utilized for classification based on MFCC features.

- **K-Means Clustering**: Used to group similar bird vocalizations without labels based on MFCC features.

- **EfficientNet**: Applied for feature extraction and fine-tuned (pre-trained model) for classification using mel-spectrogram input.

4. Analysis

Random Forest Classifier:

```
Random Forest Accuracy: 0.9237209302325582
Random Forest Confusion Matrix:
 [[248   6   2   8]
 [  8 206  13  13]
 [  8   1 293   3]
 [ 11   5   4 246]]
```

Accuracy**:** 92.37%

Confusion Matrix: The model correctly classified most species, with minor misclassifications.

SVM Classifier:

```
SVM Accuracy: 0.8213953488372093
SVM Confusion Matrix:
 [[215  16  23  10]
 [ 36 171  20  13]
 [ 28  12 259   6]
 [ 14  10   4 238]]
```

Accuracy: 82.14%

Confusion Matrix: The performance was lower compared to the Random Forest model. Higher misclassification rate in certain species.

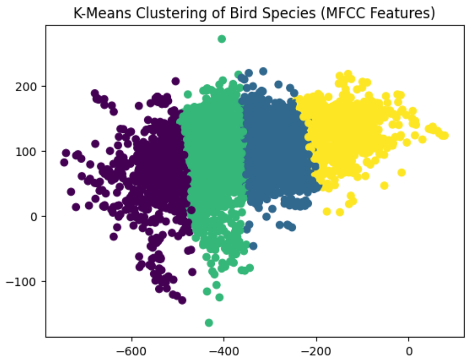
EfficientNet Model:

```
Test Accuracy: 97.12%
Confusion Matrix:
 [[248   1   4   0]
 [  5 227   5   2]
 [  5   4 249   1]
 [  4   0   0 320]]
Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.98      0.96       253
           1       0.98      0.95      0.96       239
           2       0.97      0.96      0.96       259
           3       0.99      0.99      0.99       324

    accuracy                           0.97      1075
   macro avg       0.97      0.97      0.97      1075
weighted avg       0.97      0.97      0.97      1075
```

Accuracy: 97.12%

Confusion Matrix and Classification Report: EfficientNet significantly outperformed Random Forest and SVM models. High precision, recall, and F1-scores across all classes, demonstrating superior classification performance.

K-Means Clustering:




```
Silhouette Score: 0.23670484125614166
Adjusted Rand Index: 0.19648578975772588
```

K-means clustering results indicate limited separability of bird vocalization features. The Silhouette Score (0.237) suggests that clusters are not well-defined, and the Adjusted Rand Index (0.196) shows weak agreement with the ground truth labels. This suggests that bird vocalizations may not be easily separable using simple clustering techniques.

4. Experiments

4.1 Effect of Training Data Size

```
SVM Accuracy with 10.0% of data: 0.6781799379524301
SVM Accuracy with 20.0% of data: 0.7671009771986971
SVM Accuracy with 50.0% of data: 0.8056589724497394
SVM Accuracy with 80.0% of data: 0.8213953488372093
SVM Accuracy with 90.0% of data: 0.8085501858736059
```

Objective: Investigate how training data size impacts classification performance.

- The dataset was split into different training sizes: 10%, 20%, 50%, 80%, and 90%.
- Observation:
  - Increasing the training size improved accuracy across all models, with diminishing returns beyond 80%.
  - The results suggest that after a certain threshold, additional training data does not significantly enhance performance.

4.2 Data Balance and Composition(SMOTE)

```
Support Vector Machine Accuracy with SMOTE: 0.8138629283489096
```

```
Random Forest Accuracy with SMOTE: 0.9610591900311527
```

Objective: Analyze the impact of class balance on model performance.

- Observation:
  - The Random Forest model was more robust to imbalanced data compared to SVM, which suffered significant performance degradation.
  - Applying class weighting in Random Forest improved performance.

4.3 Effect of Data Augmentation

```
SVM Accuracy with Noise Augmentation: 0.7767441860465116
Random Forest Accuracy with Noise Augmentation: 0.9237209302325582
```

```
def add_noise(signal, noise_level=0.005):
    noise = np.random.randn(len(signal)) * noise_level
    return signal + noise
```

Objective: Assess whether augmenting bird vocalization data enhances classification accuracy.

- Data augmentation was applied using noise addition.
- Observation:
  - Both classifiers (SVM and Random Forest) did not improve with augmentation.

➢ This suggests that the introduced noise did not provide meaningful variability or that the models were already robust to minor perturbations.

4.4 Dimensionality Reduction(PCA)

```
SVM Accuracy with PCA: 0.8651162790697674
```

```
Random Forest Accuracy with PCA: 0.9441860465116279
```

Objective: Determine the impact of reducing feature dimensionality on classification.

- PCA was applied to reduce MFCC features from 40 dimensions to 20.
- Observation:
  - ➢ SVM accuracy improved from 82% to 86%.
  - ➢ Random Forest accuracy improved from 92% to 94%.
  - ➢ This indicates that reducing feature dimensions helped remove redundancy and enhanced separability.

4.5 Effect of Data Normalization

```
Random Forest Accuracy with Normalization: 0.9227906976744186
SVM Accuracy with Normalization: 0.932093023255814
```

Research Question: Does normalizing audio features improve classification performance?

- Compare model accuracy with and without feature normalization
- Observation:
  - ➢ Improvement in SVM Accuracy (0.82 → 0.93, +11%), but no improvement in Random Forest Accuracy.
  - ➢ I supposed that SVM heavily relies on feature scaling because it calculates distances in high-dimensional space. After normalization, all features contribute equally, allowing SVM to find better hyperplanes, significantly improving accuracy.
  - ➢ However, Random Forest is not significantly affected by feature scaling since it makes splits based on feature thresholds rather than distance-based calculations.

5. Discussion

Key Observation

- EfficientNet significantly outperformed Random Forest and SVM, achieving 97.12% accuracy. This highlights the effectiveness of Mel-spectrogram features with deep learning.
- SVM had higher misclassification rates, suggesting that:
  - A more refined feature set or hyperparameter tuning is needed
- K-Means clustering performed poorly, indicating that bird vocalization features may not be distinctly separable in an unsupervised setting.
- PCA improved classification accuracy, suggesting that dimensionality reduction helped remove noise and redundant information.
- Normalization is essential for SVM

Challenges and Future Work

- Feature Engineering: Experimenting with alternative audio features like spectrograms.
- Deep Learning Approaches: Exploring convolutional neural networks (CNNs) for spectrogram-based classification.
- Additional Experiments:
  - Hyperparameter tuning for all models.
  - Testing alternative clustering methods such as DBSCAN.
  - Investigating the impact of real-world environmental noise on classification performance.

Remaining Questions

- How can K-Means be further optimized for better clustering?
- How well do these models generalize to different environmental conditions?
- Would alternative deep learning models outperform EfficientNet in this task?

6. Reference

Data Source: https://xeno-canto.org/

Related Works:

https://www.bird-research.jp/1_event/jbraoc2023/poster/p16.pdf

https://www.kaggle.com/code/virajkadam/birdclef-bird-sound-classification

Audio Process:

https://librosa.org/doc/latest/index.html

Machine Learning:

https://scikit-learn.org/stable/

https://pytorch.org/

Data Visualization

https://matplotlib.org/

7. Appendix

Code: https://github.com/Kennethii2i/NYCU-AI-Capstone-Spr2025-Project1.git

Brief Explanation

- Dataset Downloading

  [Reference] https://xeno-canto.org/

  [Reference] data_download.ipynb

➢ Feature Extraction

```
# extract feature(mfcc) from dataset
def extract_features_and_save(data_folder, save_path="mfcc_features.npy"):
    X, y = [], []
    labels = {species: idx for idx, species in enumerate(os.listdir(data_folder))}

    for species, idx in labels.items():
        species_folder = os.path.join(data_folder, species)
        for file in os.listdir(species_folder):
            if file.endswith(".wav"):
                file_path = os.path.join(species_folder, file)
                signal, sr = librosa.load(file_path, sr=None)
                mfcc = librosa.feature.mfcc(y=signal, sr=sr, n_mfcc=40)
                mfcc_mean = np.mean(mfcc, axis=1)
                X.append(mfcc_mean)
                y.append(idx)

    np.save(save_path, {"X": np.array(X), "y": np.array(y), "labels": labels})
    print("Features saved successfully!")

extract_features_and_save("data")
```

➢ Loading data(mfcc)

```
# Loading data from npy file which is precomputed with the dataset.
data = np.load("mfcc_features.npy", allow_pickle=True).item()
X, y, label_map = data["X"], data["y"], data["labels"]
print("Features loaded successfully!")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

➢ Model Training and Evaluation(Random Forest, SVM, K-means)

```
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Random Forest Confusion Matrix:¥n", confusion_matrix(y_test, y_pred_rf))
# Save model
# joblib.dump(rf_model, 'rf_bird_model.pkl')
```

```
[ ]  svm_model = SVC(kernel='rbf', C=1.0)
     svm_model.fit(X_train, y_train)
     y_pred_svm = svm_model.predict(X_test)
     print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
     print("SVM Confusion Matrix:¥n", confusion_matrix(y_test, y_pred_svm))
     # Save model
     # joblib.dump(svm_model, 'svm_bird_model.pkl')
```

```
kmeans = KMeans(n_clusters=len(label_map), random_state=42)
kmeans_labels = kmeans.fit_predict(X)
# External evaluation
silhouette_avg = silhouette_score(X, kmeans_labels)
print(f"Silhouette Score: {silhouette_avg}")
ari = adjusted_rand_score(y, kmeans_labels)
print("Adjusted Rand Index:", ari)
# Graph
plt.scatter(X[:, 0], X[:, 1], c=kmeans_labels, cmap="viridis")
plt.title("K-Means Clustering of Bird Species (MFCC Features)")
plt.show()
```

- Deep learning Model
  - Mel-spectrogram feature extraction

```
# extract mel-spectrum features(execute once),
# to construct mel-spectrogram pictures for each epecies from wav file.
def extract_mel_spectrogram_for_species(species_folder):
    if not os.path.exists(species_folder):
        print(f"Error: {species_folder} does not exist!")
        return

    species_name = os.path.basename(species_folder)
    save_folder = f"{species_folder}_mel_spectrogram"  # New subfolder name
    os.makedirs(save_folder, exist_ok=True)  # Create folder if not exists

    for file in os.listdir(species_folder):
        if file.endswith(".wav"):
            file_path = os.path.join(species_folder, file)
            signal, sr = librosa.load(file_path, sr=None)
            mel_spec = librosa.feature.melspectrogram(y=signal, sr=sr, n_mels=128)
            mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)

            # Save as an image inside the new subfolder
            plt.figure(figsize=(3, 3))
            librosa.display.specshow(mel_spec_db, sr=sr, x_axis='time', y_axis='mel')
            plt.axis("off")

            img_filename = file.replace(".wav", ".png")
            img_path = os.path.join(save_folder, img_filename)
            plt.savefig(img_path, bbox_inches="tight", pad_inches=0)
            plt.close()

    print(f"Mel spectrograms saved in: {save_folder}")

# Example usage (process only one species folder)
extract_mel_spectrogram_for_species("data/Passer_montanus")
```

- Loading data

```python
import torchvision.transforms as transforms
# Define image transformations (resize, normalize)
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# Load dataset directly from labeled subfolders
dataset = datasets.ImageFolder(root="mel_spectrograms", transform=transform)

# Split into training and testing sets
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size, test_size])

# Create DataLoaders
print("create dataloaders")
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=2, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False, num_workers=2, pin_memory=True)

# Print class-to-label mapping
print("Class labels:", dataset.class_to_idx)
```

- Training EfficientNet Model

```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    correct, total = 0, 0

    for images, labels in tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs}", unit="batch"):
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

    accuracy = 100 * correct / total
    print(f"Epoch {epoch+1}/{num_epochs}, Loss: {total_loss:.4f}, Accuracy: {accuracy:.2f}%")

# Save trained model
torch.save(model.state_dict(), "efficientnet_bird_model.pth")
```

- Evaluation

```python
model.eval()
all_preds, all_labels = [], []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)

        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Calculate accuracy
accuracy = accuracy_score(all_labels, all_preds)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Print confusion matrix
conf_matrix = confusion_matrix(all_labels, all_preds)
print("Confusion Matrix:¥n", conf_matrix)

# Print classification report
print("Classification Report:¥n", classification_report(all_labels, all_preds))
```

- Experiments
  - Dataset Size Experiment

```python
for fraction in [0.1, 0.2, 0.5, 0.8, 0.9]:
    # Splitting dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1-fraction, random_state=42)
    # Training the model
    svm_model.fit(X_train, y_train)
    # Making predictions using the trained model
    y_pred_svm = svm_model.predict(X_test)
    print(f"SVM Accuracy with {fraction*100}% of data:", accuracy_score(y_test, y_pred_svm))
```

  - SMOTE Experiment

```python
smote = SMOTE(random_state=42)
# resample with SMOTE
X_resampled, y_resampled = smote.fit_resample(X, y)
# Splitting dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
# Training the model
svm_model.fit(X_train, y_train)
# Making predictions using the trained model
y_pred_svm = svm_model.predict(X_test)
print("Support Vector Machine Accuracy with SMOTE:", accuracy_score(y_test, y_pred_svm))
```

  - Data Augmentation

```python
X_augmented = np.array([np.mean(librosa.feature.mfcc(y=add_noise(librosa.util.normalize(librosa.load(os.path.join("data", species, file), sr=None)[0])),
                                                     sr=22050, n_mfcc=40), axis=1)
                        for species in os.listdir("data") for file in os.listdir(os.path.join("data", species)) if file.endswith(".wav")])
X_train, X_test, y_train, y_test = train_test_split(X_augmented, y, test_size=0.2, random_state=42)
```

- Dimensionality Reduction

```
pca = PCA(n_components=20)
# fit PCA for dataset
X_pca = pca.fit_transform(X)
# Splitting dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)
# Training the model
svm_model.fit(X_train, y_train)
# Making predictions using the trained model
y_pred_svm = svm_model.predict(X_test)
print("SVM Accuracy with PCA:", accuracy_score(y_test, y_pred_svm))
```

- Normalization

```
# Apply normalization
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Traing the model
rf_model.fit(X_train_scaled, y_train)
y_pred_rf = rf_model.predict(X_test_scaled)
print("Random Forest Accuracy with Normalization:", accuracy_score(y_test, y_pred_rf))
# Training the model
svm_model.fit(X_train_scaled, y_train)
y_pred_svm = svm_model.predict(X_test_scaled)
print("SVM Accuracy with Normalization:", accuracy_score(y_test, y_pred_svm))
```