■ SoundPlot: Birdsong Acoustic Analysis System

> *Turning birdsong into geometry to reveal the hidden structure of natural acoustic communication.*

■ Overview

SoundPlot is an audio analysis system that transforms birdsong recordings into measurable acoustic features and maps them into multi-dimensional space. By converting sound into geometry, patterns emerge that no human ear could reliably detect at scale—revealing the internal logic of natural soundscapes.

What This System Does

1. **Extracts Acoustic Features** from audio: pitch contours, rhythm patterns, repetition structures, and spectral textures
2. **Maps Sounds into Multi-dimensional Space** using dimensionality reduction techniques
3. **Discovers Patterns and Clusters** where similar calls group together
4. **Identifies Motifs and Transitions** showing how sounds relate across time, regions, and species

---

■■ Technology Stack

Core Audio Processing

Library	Version	Purpose	Why We Use It
Librosa	0.10+	Audio feature extraction	The gold standard for audio analysis in Python. Provides battle-tested implementations of MFCCs, spectral features, pitch tracking, and beat detection. Built on NumPy for performance.
SoundFile	0.12+	Audio I/O	Robust reading/writing of audio files (WAV, FLAC, OGG). Works seamlessly with librosa and handles edge cases that other libraries miss.
NumPy	1.24+	Numerical computing	Foundation for all array operations. Every audio signal becomes a NumPy array for efficient vectorized processing.

SciPy	1.11+	Signal processing	Advanced signal processing (filtering, FFT, peak detection). Complements librosa for custom audio transformations.

Machine Learning & Pattern Discovery

Library	Version	Purpose	Why We Use It
Scikit-learn	1.3+	Clustering & ML	Industry-standard ML library. Provides K-Means, DBSCAN, and HDBSCAN for clustering, plus PCA for dimensionality reduction. Excellent documentation and reliability.
UMAP-learn	0.5+	Dimensionality reduction	Superior to t-SNE for preserving both local and global structure. Creates more meaningful 2D/3D projections of high-dimensional feature spaces.
HDBSCAN	0.8+	Density-based clustering	Automatically determines the number of clusters and handles noise. Perfect for discovering natural groupings in acoustic data without predefining cluster count.

Data Management

Library	Version	Purpose	Why We Use It
Pandas	2.0+	Data manipulation	Tabular data handling for feature datasets. Easy filtering, grouping, and export to various formats.
PyArrow	14+	Efficient storage	Parquet file format for storing millions of feature vectors efficiently. 10x faster than CSV with better compression.

■ AI/ML Models & Algorithms

1. Feature Extraction Pipeline

Mel-Frequency Cepstral Coefficients (MFCCs)

```
Audio Signal → Pre-emphasis → Windowing → FFT → Mel Filter Bank → Log → DCT → MFCCs
```

Why MFCCs? - Designed to mimic human auditory perception - Capture timbral characteristics that distinguish different bird species/call types - Compact representation (typically 13-20 coefficients per frame) - Proven effectiveness in speech and animal vocalization analysis

Spectral Features

Feature	Description	What It Reveals
Spectral Centroid	"Center of mass" of the spectrum	Brightness/tonal quality of the call
Spectral Bandwidth	Width of the spectral distribution	Richness/complexity of harmonics
Spectral Rolloff	Frequency below which 85% of energy lies	Distinguishes noisy vs tonal sounds
Spectral Contrast	Difference between peaks and valleys	Harmonic structure clarity
Zero Crossing Rate	Rate of sign changes in the signal	Percussive vs sustained sounds

Pitch Tracking (F0 Estimation)

We use **pYIN** (probabilistic YIN) algorithm: - Based on the YIN autocorrelation method but adds probabilistic modeling - Handles the polyphonic nature of birdsong better than simple peak-picking - Provides confidence scores for pitch estimates - Robust to noise and overlapping sounds

Rhythm & Temporal Features

Feature	Algorithm	Purpose
Onset Detection	Spectral flux + peak-picking	Find when notes/calls start
Beat Tracking	Dynamic programming	Detect rhythmic patterns
Tempo Estimation	Autocorrelation of onset envelope	Overall pace of vocalizations
Inter-onset Intervals	Custom analysis	Timing patterns between calls

---

2. Dimensionality Reduction

UMAP (Uniform Manifold Approximation and Projection)

Why UMAP over t-SNE or PCA?

Method	Pros	Cons	Best For
PCA	Fast, deterministic	Only captures linear relationships	Initial exploration, preprocessing
t-SNE	Good local structure	Slow, loses global structure	Small datasets (<10K points)
UMAP	Fast, preserves global AND local structure	Requires tuning	Our use case: large-scale birdsong analysis

How UMAP Works (Simplified): 1. Build a weighted graph of nearest neighbors in high-dimensional space 2. Optimize a low-dimensional representation to preserve these relationships 3. Result: Similar sounds cluster together, dissimilar sounds stay apart

Our UMAP Configuration:

```
umap.UMAP(
   n_neighbors=15,      # Balance local vs global structure
   min_dist=0.1,        # How tightly clusters can pack
   n_components=3,      # 3D for rich visualization
   metric='cosine'      # Angular distance for audio features
)
```

---

3. Clustering Algorithms

HDBSCAN (Hierarchical Density-Based Spatial Clustering)

Why HDBSCAN? - **No need to specify number of clusters**: Discovers natural groupings automatically - **Handles noise**: Labels outliers separately instead of forcing them into clusters - **Varying densities**: Works when some call types are common and others are rare - **Hierarchical**: Can explore clustering at different granularities

How It Works: 1. Build a minimum spanning tree based on mutual reachability distance 2. Construct a hierarchy of connected components 3. Extract stable clusters that persist across different density thresholds

Configuration:

```
hdbscan.HDBSCAN(
   min_cluster_size=10,      # Minimum calls to form a cluster
   min_samples=5,            # Core point density
   cluster_selection_epsilon=0.0,
   metric='euclidean'
)
```

K-Means (Optional, for comparison)

- Used when you want to specify exact number of clusters
- Useful for known taxonomies (e.g., "find 5 distinct call types")
- Faster than HDBSCAN for very large datasets

---

4. Pattern & Motif Detection

Sequence Analysis

We analyze temporal patterns using:

1. **Transition Matrices**: Track which call types follow which others

2. **N-gram Analysis**: Find common sequences (pairs, triplets, etc.)
3. **DTW (Dynamic Time Warping)**: Compare call shapes even at different speeds

Motif Discovery

Using a sliding window approach: 1. Extract short segments from audio 2. Compute similarity between all pairs 3. Group highly similar segments as "motifs" 4. Track motif frequency and context

---

■ Project Structure

```
sound_plot/
|-- README.md                 # This file
|-- requirements.txt          # Python dependencies
|-- setup.py                  # Package installation
|
|-- src/
|   |-- __init__.py
|   |-- audio/
|   |   |-- __init__.py
|   |   |-- loader.py         # Audio file loading & validation
|   |   +-- preprocessor.py   # Noise reduction, normalization
|   |
|   |-- features/
|   |   |-- __init__.py
|   |   |-- spectral.py       # Spectral feature extraction
|   |   |-- temporal.py       # Rhythm, onset, tempo features
|   |   |-- pitch.py          # Pitch/F0 tracking
|   |   +-- mfcc.py           # MFCC extraction
|   |
|   |-- analysis/
|   |   |-- __init__.py
|   |   |-- clustering.py     # HDBSCAN, K-Means
|   |   |-- reduction.py      # UMAP, PCA
|   |   |-- patterns.py       # Motif detection, transitions
|   |   +-- similarity.py     # DTW, distance metrics
|   |
|   +-- utils/
|       |-- __init__.py
|       |-- io.py             # Data export (Parquet, JSON)
|       +-- visualization.py  # Static plots for verification
|
|-- data/
|   |-- raw/                  # Input audio files
|   |-- processed/            # Extracted features
|   +-- results/              # Clustering results, embeddings
|
```

```
|-- tests/
|   |-- test_audio.py
|   |-- test_features.py
|   +-- test_analysis.py
|
+-- examples/
   |-- basic_analysis.py     # Simple usage example
   +-- batch_processing.py   # Process many files
```

■ Quick Start

Installation

```
# Clone the repository
git clone https://github.com/your-username/sound_plot.git
cd sound_plot

# Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt
```

Basic Usage

```
from src.audio import AudioLoader
from src.features import FeatureExtractor
from src.analysis import ClusterAnalyzer

# Load audio
loader = AudioLoader()
audio, sr = loader.load("path/to/birdsong.wav")

# Extract features
extractor = FeatureExtractor(sample_rate=sr)
features = extractor.extract_all(audio)

# Analyze patterns
analyzer = ClusterAnalyzer()
clusters = analyzer.fit_predict(features)
embeddings_3d = analyzer.get_embeddings()

# Export for visualization
```

```
analyzer.export("results/analysis_output.parquet")
```

■ Output Data Format

Feature Vectors (per audio segment)

```
{
 "segment_id": "file001_seg003",
 "source_file": "meadow_recording.wav",
 "start_time": 2.5,
 "end_time": 3.2,
 "features": {
   "mfcc": [13 coefficients],
   "spectral_centroid": 4532.1,
   "spectral_bandwidth": 1234.5,
   "zero_crossing_rate": 0.082,
   "pitch_mean": 2100.3,
   "pitch_std": 156.7,
   "onset_count": 3,
   "tempo": 120.5
 },
 "embedding_3d": [x, y, z],
 "cluster_id": 7,
 "cluster_probability": 0.94
}
```

■ Why These Choices?

On MFCCs

MFCCs were originally designed for human speech but work exceptionally well for birdsong because both involve harmonic structures and frequency modulation. They compress spectral information in a perceptually meaningful way.

On UMAP vs t-SNE

While t-SNE is popular, UMAP is: - **10-100x faster** for large datasets - **Better at preserving global structure** (important for seeing how different species relate) - **More stable** across runs with the same random seed

On HDBSCAN vs K-Means

Birdsong data is messy—some calls are common, others rare. K-Means forces everything into clusters and requires knowing how many clusters exist. HDBSCAN finds natural groupings and correctly identifies

unusual sounds as noise.

On Librosa

Librosa is the most mature audio analysis library in Python. Alternatives like `torchaudio` are faster for deep learning but lack the comprehensive feature extraction we need. Essentia is powerful but has a steeper learning curve.

■ References

- McFee, B., et al. (2015). *librosa: Audio and Music Signal Analysis in Python*
- McInnes, L., Healy, J., & Melville, J. (2018). *UMAP: Uniform Manifold Approximation and Projection*
- Campello, R. J., et al. (2013). *Density-Based Clustering Based on Hierarchical Density Estimates*
- Stowell, D., & Plumbley, M. D. (2014). *Automatic large-scale classification of bird sounds*

■ License