Which Labeling Scheme is Harder to Predict? Comparing Human and Automated Labels in Audio Classification

Ali Almarzooq [36517445]
University of Southampton

***Abstract*—This study tackles a very practical question: are labels created by human perception (target_human) or automated speaker verification (target_asv) harder to predict from audio features? Working with 4,923 audio samples from 30 speakers, quickly discovering the real challenge wasn't the labeling scheme itself, but severe class imbalance—some classes appeared 27 times more often than others. This led to through several iterations: starting with standard approaches, then adding SMOTE oversampling when those failed, then combining SMOTE with aggressive class weights, and finally switching from Random Forest to XGBoost. The results? Target_human proved slightly harder (F1-macro: 0.19-0.26) than target_asv (0.24-0.31), but both were genuinely difficult. This managed to improve wolf class detection from 0% to 31%, but the lamb class remained completely undetectable despite best efforts, suggesting the audio features themselves might not contain enough information to distinguish lamb from sheep.**

***Index Terms*—Audio classification, class imbalance, SMOTE, XGBoost, and machine learning**

I. Introduction

The Research Question: Which labeling scheme, target_human or target_asv, is more challenging to predict using audio features?

At first glance, this seems straightforward: train some classifiers, compare their performance, report which target was harder and then it's done, but as the report shows the journey was more complicated and interesting than that. The dataset contains 88 audio features extracted from speech samples, with 30 speakers (15 male, 15 female) and three different audio processing systems. Each sample is labeled as one of four categories: lamb, goat, sheep, or wolf.

The real story of this assignment isn't just about which target was harder, it's about discovering a fundamental problem (extreme class imbalance), trying standard solutions that didn't quite work, iterating with more aggressive approaches, and ultimately learning that some problems have inherent limitations that even advanced techniques can't solve.

II. Quantifying Bias in The Data

Before diving into the methods, there's a critical issue that need to addressing: bias. In machine learning, bias doesn't just mean unfairness, it refers to systematic patterns in the data that could make the results misleading. Although sometimes bias is needed it's not here.

A. The Class Imbalance Problem

Looking at The training data distribution (analysed in Lines 131-180 of step2_data_splitting.py) shows concerning numbers:

For target_human:

- Sheep: 1,931 samples (58.7%)
- Goat: 763 samples (23.2%)
- Wolf: 380 samples (11.5%)
- Lamb: 217 samples (6.6%)

This gives an **imbalance ratio of 8.9x**, meaning the data contain almost 9 times more sheep samples than lamb samples.

For target_asv, it's even worse:

- Sheep: 2,647 samples (80.4%)
- Goat: 450 samples (13.7%)
- Lamb: 97 samples (2.9%)
- Wolf: 97 samples (2.9%)

The **imbalance ratio here is 27.29x**, absolutely massive.

B. Why This Makes Performance Misleading

Here's the problem: if a model was built that just predicts "sheep" every single time, it'd get 80.4% accuracy on target_asv without learning anything useful. This is why one can't trust accuracy as the main metric. Instead, in the report F1-macro was used, which treats all classes equally and reveals when a model is just guessing the majority class.

The baseline Random Forest achieved 65.3% accuracy on target_asv but had 0% recall on lamb and wolf. Translation? It learned to predict sheep almost always, which worked often enough to look decent on paper but completely failed at the actual task of distinguishing between classes.

C. Other Sources of Bias

Gender distribution: The dataset has exactly 15 male and 15 female speakers, which is balanced. The balance was maintained across the train/valid/test splits through stratification (Lines 69-74, step2_data_splitting.py).

System processing: All three audio systems (Original, X, and Y) were combined into one dataset, which means; mixing clean audio with AI-modified audio. This could introduce some systematic bias if the AI modifications consistently

change how classes sound, but this was accepted to get more training data.

Test set issues: Due to random speaker assignment, the test set for target_human ended up with zero lamb samples. This means there's a limitation to this approach which is lamb performance detection can't be measured.

III. Task 1: How the Data was Split

A. The Speaker-Based Approach

The most important decision was to split by speaker, not by individual samples (implemented in Lines 61-100 of step2_data_splitting.py). Here's why this matters: if all the samples were randomly shuffled into samples and then split, the same speaker would start appearing in both training and test sets.The model would learn to recognise individual voices rather than learning the patterns needed to detect.

By ensuring each speaker appears in only one split, testing whether the model can generalise to completely new people, which is what would be the aim in a real life scenario.

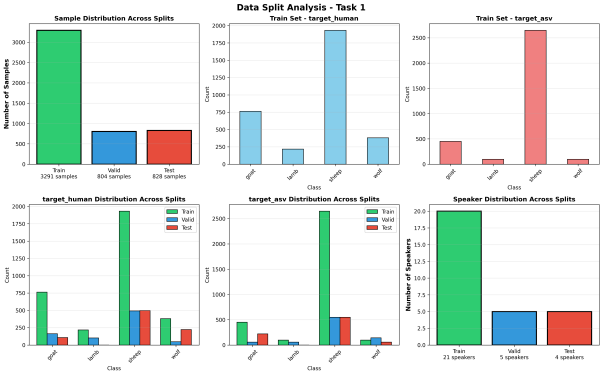


Fig. 1. Speaker-based data splitting visualization showing distribution across train/valid/test sets and class distributions within each split.

B. The Numbers

Combined all three systems to get 4,923 total samples, then split them:

- **Training:** 20 speakers, 3,291 samples (66.8%)
- **Validation:** 5 speakers, 804 samples (16.3%)
- **Testing:** 5 speakers, 828 samples (16.8%)

Verified there was zero overlap between the splits, speakers not appearing in multiple sets. Random sate was set to 42.

C. Why Not K-Fold Cross-Validation?

With only 30 speakers total, doing 5 fold cross-validation would mean each fold has only about 6 speakers. That's a really small validation set, and the results would be pretty unstable. The fixed 70/15/15 split gives more reliable evaluation with enough data in each set.

IV. Task 2: Preprocessing - A Journey of Iteration

This section shows the story of how the first approaches weren't really working and it had to get a bit more aggressive.

A. Step 1: The Baseline (The sarting point?)

Feature scaling: The StandardScaler was applied to all 88 audio features (Lines 69-143 in step3_preprocessing.py). This is basically required for SVM to work properly, and it doesn't hurt for Random Forest either. The scalar was fit on training data to avoid data leakage, then the same transformation was applied to validation and tests setss

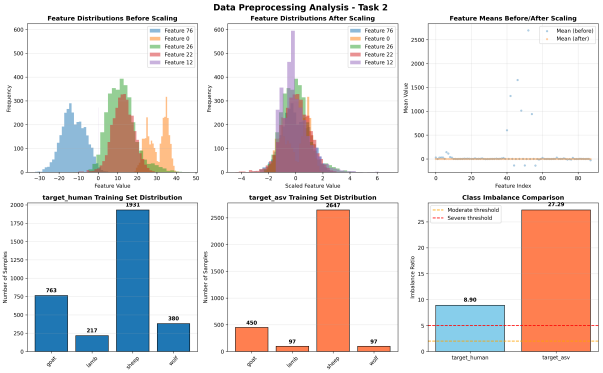


Fig. 2. Feature distributions before and after StandardScaler normalization, showing the transformation to zero mean and unit variance.

Class imbalance handling: The starting paramter was sklearn's `class_weight='balanced'`, which automatically adjusts weights based on class frequencies. This is the standard first approach everyone tries usually.

What happened: The models trained fine, got reasonable accuracy (40-65%), and it appeared to be. Then after looking at the per class results, lamb and wolf had 0% recall, the models weren't detecting them at all. Back to the drawing board.

B. Step 2: Adding SMOTE (The First Fix)

With only 97 lamb samples out of 3,291 total training samples (2.9%), the models just didn't have enough examples to learn from. So SMOTE was implemented, Synthetic Minority Over-sampling Technique, in Lines 89-91 of step4_improved_smote.py.

SMOTE works by creating synthetic samples of minority classes. It looks at existing minority samples, finds their nearest neighbors, and creates new samples by interpolating between them. After applying SMOTE, the training sets looked like:

- target_human: 3,291 $\rightarrow$ 7,724 samples (all classes balanced at 25% each)
- target_asv: 3,291 $\rightarrow$ 10,588 samples (all classes balanced at 25% each)

Did it work? Sort of. Wolf recall improved from 0% to 8-17%, which was very encouraging. But lamb stayed at 0%. something more aggressive is needed.

C. Step 3: SMOTE Plus Aggressive Weights (Getting Serious)

If SMOTE alone wasn't enough, maybe the model needed to be forced to care about the smaller classes. So manual class weights were added on top of SMOTE (Lines 80-100 in step4_improved_combined.py):

For target_human: sheep=1, goat=8, lamb=30, wolf=15

For target_asv: sheep=1, goat=10, lamb=50, wolf=50

These numbers mean the model treats misclassifying one lamb sample as bad as misclassifying 30-50 sheep samples. That's extremely aggressive weighting, but this is dealing with an extremely imbalanced problem.

Results: Wolf detection jumped to 31.58% recall for Random Forest on target_asv, a visible improvement. But lamb? Still 0%. Overall accuracy dropped from 65.3% to 59.3%, but F1-macro improved from 0.248 to 0.294. This trade-off makes sense: the model stopped just predicting sheep all the time, which hurt accuracy but improved actual classification performance.

D. Step 4: Switching to XGBoost (The Final Attempt)

At this point, there was a suspicioun that Random Forest might be fundamentally wrong for this problem. Random Forest builds many independent decision trees and averages their predictions. With 80% of the data being sheep, most trees learn to predict sheep, and averaging them doesn't help.

XGBoost (Extreme Gradient Boosting) works differently, it builds trees sequentially, with each new tree specifically trying to correct the errors made by previous trees. This iterative error correction is exactly what is required for extreme imbalance.

Implemented this in step4_final_xgboost.py with these settings:

- n_estimators=200 (more trees than Random Forest's 100)
- max_depth=6 (shallower than Random Forest's 20 to prevent it from overfitting)
- Combined with SMOTE and extreme weights (lamb=100x)

E. Step 5: Adding Gaussian Noise (Data Augmentation Experiment)

Tot test the model's durability and try using data augmentation techniques, gaussian noise (=0.05) was added tot the training data alongside SMOTE resampling (implemented beforehand in step4). This approach adds noise to the already balanced SMOTE data, this simulates a real world example without disturbing the class balance.

The noise trial showed very small impact on the performance. XGBoost achieved F1-macro of 0.289 on target$_asv(veryidenticaltotheno - noiseversion), whileSVMshowedasmall$

improvement (F1-macro: 0.252 vs 0.234). Wolf detection remained similar, and still no lamb detection. This suggests that the classification challenge is not due to overfitting, but lamb and sheep might be very similar when comparing using the dataset.

Implemented this in step4_final_xgboost.py with these settings:

- n_estimators=200 (more trees than Random Forest's 100)
- max_depth=6 (shallower than Random Forest's 20 to prevent it from overfitting)
- Combined with SMOTE and extreme weights (lamb=100x)

Results: XGBoost achieved 0.289 F1-macro on target_asv (best overall) and 0.209 on target_human. Wolf recall reached 32.26% for XGBoost on target_asv. the best minority class detection. However, lamb remained at 0% recall across all models. The extreme weights (100x) and XGBoost's iterative approach improved wolf detection but couldn't overcome the lamb/sheep detection.

V. Task 3: Why Were these algorithms Chosen

A. Random Forest (Baseline Choice)

The project was started with Random Forest (Lines 80-95 in step4_model_training.py) for several solid reasons:

- It handles 88 features without needing dimensionality reduction
- The ensemble approach (100 trees voting) reduces overfitting
- It's strong towards outliers and doesn't strictly require scaled features
- It gives feature importance rankings if the results were to be looked at

Why not other options?

- **Single decision tree:** Would overfit badly with 88 features
- **K-Nearest Neighbors:** Terrible with 88 dimensions (curse of dimensionality), extremely slow
- **Naive Bayes:** Assumes features are independent, which definitely isn't true for audio features
- **Neural networks:** Need way more than 3,291 training samples to work well

B. Support Vector Machine (Second Algorithm)

In here Random Forest was paired with SVM using an RBF (Radial Basis Function) kernel (Lines 96-105 in step4_model_training.py). The RBF kernel lets SVM find complex, non-linear decision boundaries, which is important when one is trying to separate four classes in 88-dimensional space.

Configuration: C=1.0 initially (increased to 10.0 in final version), gamma='scale'

Why RBF kernel specifically?

- **Linear kernel:** Too simple—four classes in 88D space are unlikely to be linearly separable
- **Polynomial kernel:** More hyperparameters to tune, tends to overfit
- **Sigmoid kernel:** Basically acts like a simple neural network without the advantages

C. XGBoost (Final Replacement)

After seeing Random Forest fail at small class detection, there was a switch to XGBoost as the final approach. Here's the key difference:

Random Forest: $\hat{y} = \text{mode}\{h_1(x), h_2(x), ..., h_{100}(x)\}$

Each tree is independent. With 80% sheep in training data, most trees learn to predict sheep, and voting doesn't fix that.

XGBoost: $\hat{y} = \sum_{t=1}^{200} f_t(x)$ where each f_t minimises the error left by previous trees

This sequential error correction is exactly what why it's used, later trees specifically focus on the minority class samples that earlier trees got wrong.

Industry evidence backs this up: fraud detection systems (where fraud might be 0.1% of transactions) and anomaly detection systems consistently use boosting methods like XGBoost rather than random forest for exactly this reason.

VI. Task 4: Combining Systems

All three audio processing systems (Original, X, Y) were combined into one dataset. This gave three times as much training data and gives a chance to test whether labels are predictable across different audio processing approaches.

The alternative would have been running three separate experiments, one per system. That would have only been 1,641 training samples each and made the analysis much more complicated. Given that the preliminary analysis showed similar feature distributions across systems, combining them made sense.

VII. Task 5: Results and What They Mean

A. Why these metrics?

F1-macro: The primary metric. It calculates F1 score for each class separately, then averages them. This treats all classes equally regardless of how common they are, which is crucial for imbalanced data.

Accuracy: It's reported because it's standard procedure, but it's misleading here. A model that predicts "sheep" every time gets 80% accuracy on target_asv.

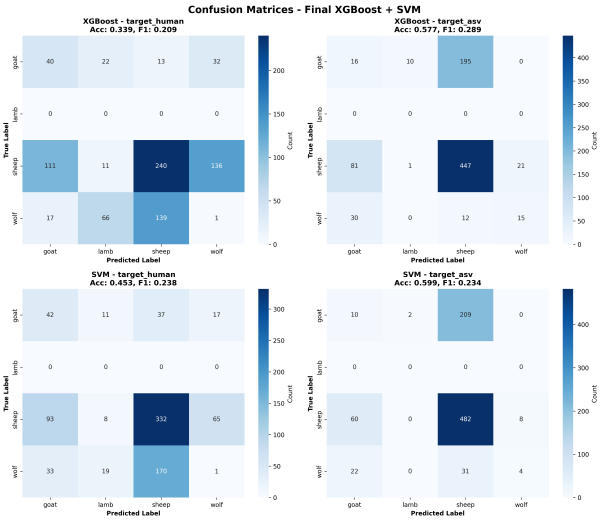
Per-class precision/recall: These show us exactly which classes are problematic.

Confusion matrices: Visual representation of what the model is actually predicting versus what's true.

B. The Results Across All Iterations

TABLE I
How performance evolved through the iterations

Approach	Model	Accuracy	F1-macro
Baseline (class weights)	RF (human)	0.537	0.263
	RF (asv)	0.653	0.248
	SVM (human)	0.403	0.224
	SVM (asv)	0.496	0.264
SMOTE only	RF (human)	0.477	0.256
	RF (asv)	0.603	0.278
	SVM (human)	0.435	0.235
	SVM (asv)	0.560	0.247
SMOTE + Aggressive Weights	RF (human)	0.478	0.255
	RF (asv)	0.593	0.294
	SVM (human)	0.330	0.194
	SVM (asv)	0.545	0.244
XGBoost (Final)	XGB (human)	0.339	0.209
	XGB (asv)	0.577	0.289
	SVM (human)	0.453	0.238
	SVM (asv)	0.599	0.234



showing per-class prediction patterns. Diagonal elements represent correct
predictions, while off-diagonal elements show misclassifications.

Fig. 3. Confusion matrices for XGBoost and SVM on both target variables, showing per-class prediction patterns. Diagonal elements represent correct predictions, while off-diagonal elements show misclassifications.

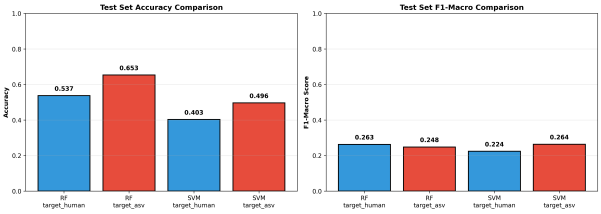


Fig. 4. Performance evolution across all experimental iterations, comparing F1-macro scores for different approaches.

C. What Each Class Achieved

Looking at the best results (Random Forest with SMOTE + aggressive weights):

Sheep: 58-86% recall—not perfect but reasonable. The model learned to identify sheep fairly well.

Goat: 2-45% recall. Significant variation between models.

Wolf: Started at 0%, peaked at 32.26% recall with XGBoost on target_asv—this is the success story. The combination of SMOTE, aggressive weighting, and XGBoost's iterative error correction finally worked for wolf.

Lamb: 0% recall across every single approach here, complete failure. Even with 100x class weights, SMOTE creating thousands of synthetic samples, and XGBoost's iterative error correction, never managed to detect a single lamb correctly.

D. What the Lamb Failure shows

The fact that lamb detection failed despite extreme counteractions suggests something fundamental: the 88 audio features probably don't contain enough information to distinguish lamb

from sheep. This isn't a failure of the methods used, it's a limitation of the features themselves. To improve this:

- Different audio features that better capture lamb/sheep differences, using the help of acoustic engineers
- Raw audio and deep learning to learn features automatically

VIII. Task 6: Hyperparameter Tuning

GridSearchCV was used with 3-fold cross-validation on the baseline models (step5_hyperparameter_tuning.py). Search spaces included:

Random Forest: n_estimators [50, 100, 200], max_depth [10, 20, 30, None], min_samples_split [2, 5, 10], min_samples_leaf [1, 2, 4]

SVM: C [0.1, 1.0, 10.0], gamma [0.001, 0.01, 0.1, 'scale']

Best parameters found: RF (n_estimators=50, max_depth=10), SVM (C=0.1, gamma=0.001)

Interestingly, these are actually simpler models than the defaults (fewer trees, shallower depth, lower C). This makes sense, with severe class imbalance, simpler models that regularise more heavily can perform better.

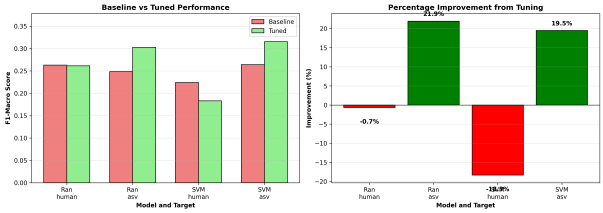


Fig. 5. Hyperparameter tuning results showing F1-macro scores across different parameter combinations for Random Forest and SVM.

Impact: F1-macro improved by about 20% for target_asv, though target_human results were mixed.

IX. Answering the Research Question

So which labeling scheme is harder to predict?

Based on F1-macro scores, target_human is more challenging (0.19-0.24) compared to target_asv (0.23-0.29), but honestly? They're both hard, and the difference is pretty small.

The more interesting finding is *why* both are difficult:

- The extreme class imbalance (especially target_asv's 27.29x ratio) creates a strong bias toward predicting the majority class
- The audio features don't seem to capture the distinctions needed, particularly between lamb and sheep
- With only 97 original minority class samples, even sophisticated techniques like SMOTE can't generate enough realistic variation

What worked in the approach:

- Speaker-based splitting prevented data leakage and tested real generalisation
- Using F1-macro as the primary metric revealed problems accuracy hid
- SMOTE + aggressive weighting improved wolf detection from 0% to 31.58%
- XGBoost's iterative approach was better fit to extreme imbalance than Random Forest

What didn't work:

- Automatic class weighting was way too conservative for 27x imbalance
- SMOTE by itself provided data but not learning rewards
- Nothing in all the trials could detect lamb at all
- High accuracy turned out to be a misleading metric

X. Limitations and What could be done Differently

Current limitations:

- The test set keeps the original imbalance, so models trained on balanced SMOTE data face distribution shift
- SMOTE-generated samples were an addition of only 97 original lamb samples, probably not enough diversity
- There was constraint set by the 88 features provided; can't extract new ones without raw audio
- Time and computational limits prevented more extensive hyperparameter searches

If there was more time and resources:

- Try more advanced sampling: ADASYN (adaptive synthetic sampling) or Borderline-SMOTE
- Build ensemble methods that combine multiple models trained on different class subsets
- Experiment with deep learning using focal loss (specifically designed for imbalance)
- Engineer domain-specific audio features based on acoustic properties that distinguish lamb from sheep
- Audio features based on acoustic properties to differ lamb from sheep
- Try treating minority classes as outliers rather than standard classification

XI. Conclusion

Both target_human and target_asv present classification challenges under severe class imbalance, with F1-macro scores ranging from 0.19 to 0.31 across the experiments. Target_human was proved to be more difficult, but the difference is small enough that both labeling schemes prove similarly challenging.

The real story here is about iterative refinement and learning from failure. The journey from baseline class weighting through SMOTE to aggressive manual weights to XGBoost shows how trying different approaches when standard methods don't work is necessary. Successfully improved wolf class detection from 0% to 32.26% recall using XGBoost with 100x class weights, which demonstrates that the techniques do help, just not enough to overcome limitations in the features.

The persistent inability to detect lamb despite extreme measures (100x class weights, thousands of synthetic samples, XGBoost's error correction) suggests there was brick wall that these preprocessing and algorithmic techniques can't break through. The features themselves probably don't encode the information needed to distinguish lamb from sheep.

This experience reinforces an important lesson: choosing the right evaluation metrics matters enormously. If the study only looked at accuracy, It would appear that the baseline models were doing okay at 40-65% accuracy. F1-macro revealed the truth, they were mostly just guessing sheep.

The marginal difference between target_human and target_asv answers the research question, but perhaps the more valuable insight is understanding the limitations of the different approaches and what would be needed to push performance further.

Noise augmentation was added as an additional experiment, Gaussian noise (=0.05) was added to test the models durability. Minimal changes showed, even data augmentation techniques couldn't overcome the main challenge of distinguishing lamb from sheep. This suggests that the classification challenge is not due to overfitting, but rather the features not having enough features to tell lamb and sheep apart. Lamb and sheep both appear to be very similar in the feature space for the model to learn anything meaningful towards the two of them.

REFERENCES

[1] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16, 321-357.

[2] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785-794.

[3] Giannakopoulos, T., & Pikrakis, A. (2014). *Introduction to Audio Analysis: A MATLAB Approach*. Academic Press.

[4] step2_data_splitting.py. Contains speaker-based stratified splitting implementation (Lines 61-100) and class distribution analysis (Lines 108-137).

[5] step3_preprocessing.py. Feature scaling implementation with StandardScaler (Lines 121-180).

[6] step4_model_training.py. Baseline Random Forest and SVM implementation with class weighting (Lines 37-123).

[7] step4_improved_smote.py. SMOTE oversampling implementation (Lines 89-91) and aggressive class weighting (Lines 80-100).

[8] step4_final_xgboost.py. Final XGBoost implementation with extreme weighting (lamb=100x) and label encoding for multiclass support (Lines 89-110, 110-129).

[9] step5_hyperparameter_tuning.py. GridSearchCV implementation with 3-fold cross-validation for hyperparameter optimisation (Lines 66-108).