

Phoneme Classification: A Comparative Study of Machine Learning Approaches

Francesco Cardia *

Marco Niccolò Loddo †

Abstract

This report presents a comprehensive study on phoneme classification using the OpenML Phoneme dataset. The objective is to distinguish between nasal and oral vowels through various machine learning and deep learning approaches. We evaluate multiple classifiers including Decision Trees, Random Forest, XGBoost, Support Vector Machines, K-Nearest Neighbors, Naive Bayes, Multi-Layer Perceptron, a custom Ensemble Classifier and a Deep Convolutional Neural Network. The study investigates the impact of different sampling strategies (ADASYN oversampling and undersampling) and dimensionality reduction techniques (PCA and UMAP) on classification performance. Model hyperparameters are optimized through systematic tuning using stratified cross-validation. Performance is assessed using accuracy, precision, recall, F1-score, and ROC-AUC metrics, with results validated through comprehensive model comparison. Source code is at the link https://github.com/fcardia/phoneme_classification

1 Introduction

The purpose of this project is to address the problem of phoneme classification as proposed in the well-known OpenML Phoneme dataset [3]. Phonemes are the smallest units of sound in a language that can distinguish words from one another. The aim of this dataset is to distinguish between nasal and oral vowels, which represent two distinct phonetic classes.

The remainder of this report is organized as follows: Section 2 describes the dataset, including its structure and characteristics. Section 3 details the preprocessing pipeline, covering data sampling strategies and dimensionality reduction techniques. Section 4 presents the classification models, hyperparameter tuning, and experimental results. Finally, Section 5 provides a discussion of the findings and summarizes the key conclusions.

2 Dataset

The dataset contains vowels extracted from 1,809 isolated syllables (e.g., pa, ta, pan). The objective is to distinguish between nasal and oral vowels, representing a binary classification task.

2.1 Feature Description

Each vowel is characterized by five acoustic features (V1–V5), corresponding to the amplitudes of the first five harmonics (AHi) normalized by the total energy (Ene), expressed as AHi/Ene. Each harmonic is assigned a sign: positive when it corresponds to a local maximum in the spectrum and negative otherwise. For each vowel, measurements were extracted at three observation moments: at the time of maximum total energy, 8 milliseconds before this maximum, and 8 milliseconds after it. From the initial set of 5,427 instances, 23 samples exhibiting zero amplitude across all five harmonics were excluded, yielding a final dataset of 5,404 instances presented in random order.

*Student ID: 60/99/00017

†Student ID: 60/99/00022

2.2 Exploratory Data Analysis

To investigate the inner structure of the dataset, we performed several exploratory analyses. Tab 1 shows the main statistics of the features in our dataset.

	V1	V2	V3	V4	V5
count	5404	5404	5404	5404	5404
mean	0.000	0.000	0.000	0.000	0.000
std	1.000	1.000	1.000	1.000	1.000
min	-2.934	-3.039	-2.796	-2.485	-2.367
25%	-0.670	-0.779	-0.951	-0.758	-0.541
50%	-0.380	-0.215	-0.039	-0.142	-0.213
75%	0.315	0.714	0.777	0.676	0.205
max	3.829	3.665	2.631	3.047	4.587

Table 1: Descriptive statistics of the features

2.2.1 Feature and Class Distribution

Figure 1 presents three visualizations providing an initial overview of the dataset. The boxplots (left) show the distribution of each feature; since the data is standard scaled, no significant differences in position or variability are expected, though some minor variations can be observed. The barplot (center) illustrates the class distribution, highlighting the moderate imbalance: the majority class (nasal vowels, Class 0) comprises approximately 70% of the samples, while the minority class (oral vowels, Class 1) represents the remaining 30%. The parallel coordinates plot (right) provides insight into the feature structure across classes; the class imbalance is also visible from this perspective, as the majority class lines (blue) almost cover the minority class lines (orange).

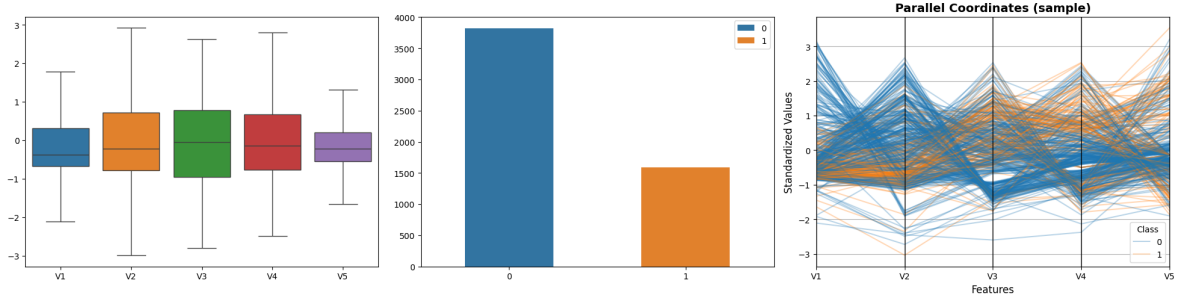


Figure 1: Exploratory data analysis overview: feature distributions (left), class distribution (center), and parallel coordinates plot (right).

2.2.2 Feature Correlation

To investigate feature correlation, we computed a correlation matrix and visualized it as a heatmap (Figure 2). The results reveal that each feature exhibits very weak pairwise correlation with the others, with values close to zero. This suggests that the data may have been preprocessed in an earlier stage.

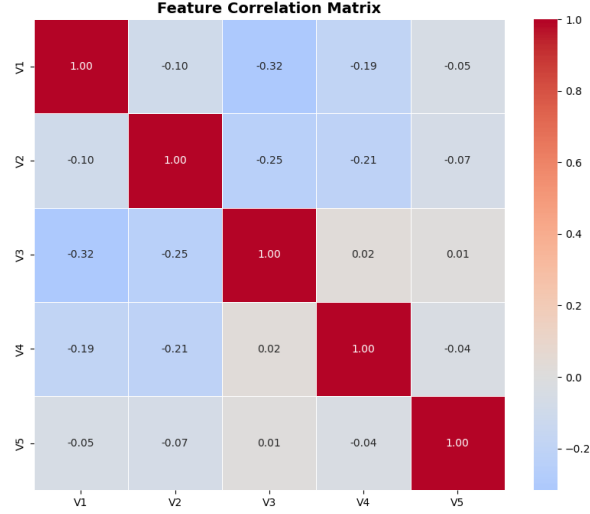


Figure 2: Correlation matrix heatmap showing weak pairwise correlations between features.

2.3 Class-wise Feature Distribution

2.3.1 Violin plots

Figure 3 presents violin plots for each feature, illustrating how the distributions differ between the two classes. A violin plot combines a box plot with a kernel density estimation, where the width of the "violin" represents the frequency of data points at each value level—wider sections indicate higher concentration of observations.

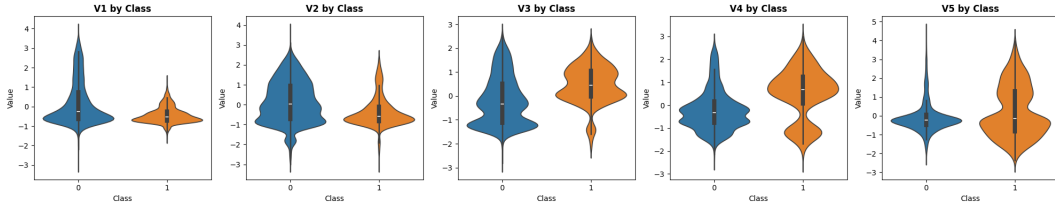


Figure 3: Violin plots showing feature distributions for each class. The width represents data density at each value level.

V3 and **V4** are the most effective features for classification, exhibiting nearly mirror-image distributions with minimal IQR overlap: Class 0 concentrates on negative values while Class 1 favors positive values for V3, and vice versa for V4.

2.3.2 Scatter Matrix

Figure 4 presents a scatter matrix showing pairwise relationships between all features. This visualization combines two types of information: univariate distributions along the diagonal and bivariate relationships in the off-diagonal plots.

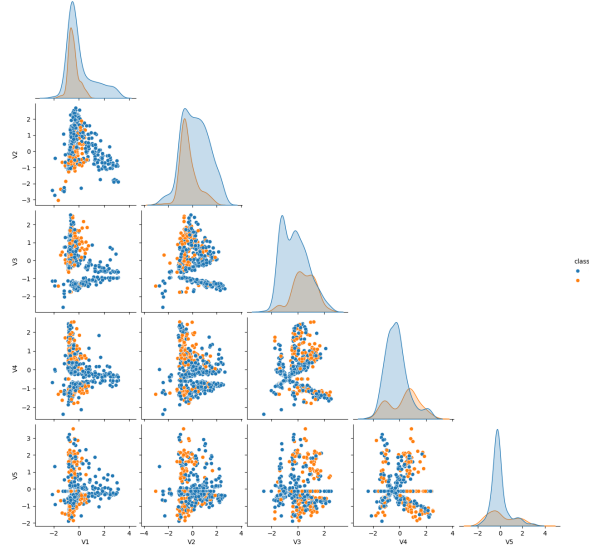


Figure 4: Scatter matrix showing pairwise feature relationships. Diagonal elements display KDE plots for univariate distributions; off-diagonal elements show bivariate scatter plots colored by class membership.

The diagonal displays Kernel Density Estimation (KDE) plots, confirming that V3 and V4 exhibit clearly separated peaks between classes, while other features show greater overlap.

3 Preprocessing

This section describes the preprocessing pipeline applied to the dataset prior to model training, encompassing missing value analysis and class balancing strategies.

3.1 Missing Values Analysis

The dataset was examined for missing values. The analysis confirmed that all 5,404 instances contain complete observations across all five features, with no missing or null values detected. Consequently, no imputation or removal strategies for missing value handling were required.

3.2 Feature Standardization

As observed during the exploratory data analysis (see Table 1), all features exhibit mean values approximately equal to zero and standard deviations equal to one. This characteristic indicates that the dataset was standardized during a previous processing stage; therefore, no additional scaling transformation was applied.

3.3 Train-Test Split

The dataset was partitioned into training and test sets using a 75/25 ratio and stratified sampling to preserve class proportions across both subsets. Labels originally encoded as “1” and “2” were remapped to 0 and 1, respectively, as certain algorithms (e.g., XGBoost) require numeric labels with zero-based indexing.

3.4 Class Balancing

Given the moderate class imbalance present in the dataset, two rebalancing strategies were investigated: undersampling and oversampling. Both techniques were evaluated to assess their impact on model performance.

It is important to note that balancing was performed exclusively on the training set *after* the train-test split. Applying these techniques to the entire dataset prior to partitioning would introduce data leakage, as the resampling process would incorporate information from instances subsequently allocated to the test set. To ensure unbiased evaluation, the test set remained untouched throughout the preprocessing stage.

3.4.1 Undersampling

The undersampling approach employed the `RandomUnderSampler` scikit-learn class. This technique randomly removes instances from the majority class until class parity is achieved. The resulting training set comprises 2,380 samples, with 1,190 instances per class.

3.4.2 Oversampling

The oversampling approach utilized ADASYN (Adaptive Synthetic Sampling) [1]. Unlike naive random oversampling, ADASYN generates synthetic minority class samples by focusing on boundary regions where classification is more challenging. This adaptive mechanism produces samples in areas of lower density, potentially improving the classifier’s ability to learn complex decision boundaries. The resulting training set comprises 5,771 samples, with 2,863 instances for Class 0 and 2,908 for Class 1.

Configuration	Class 0	Class 1	Total
Original	2,863	1,190	4,053
Undersampling (Random)	1,190	1,190	2,380
Oversampling (ADASYN)	2,863	2,908	5,771

Table 2: Training set composition for each sampling configuration.

3.5 Feature Extraction

Three dimensionality reduction techniques are employed for feature extraction and visualization. Principal Component Analysis (PCA) is first applied as an unsupervised method, reducing the data to two components for visualization purposes. To prevent data leakage, the PCA model is fitted exclusively on the training set and subsequently applied to the test set. Linear Discriminant Analysis (LDA) is then used as a supervised alternative, requiring class labels during fitting and following the same train–test separation strategy. Due to theoretical constraints, the number of LDA components is limited to one (equal to number of classes - 1), resulting in a one-dimensional projection. Finally, t-distributed Stochastic Neighbor Embedding (t-SNE) is adopted solely for visualization; since it is not used for model training, it is applied to the entire dataset without concern for data leakage, using the default parameter settings.

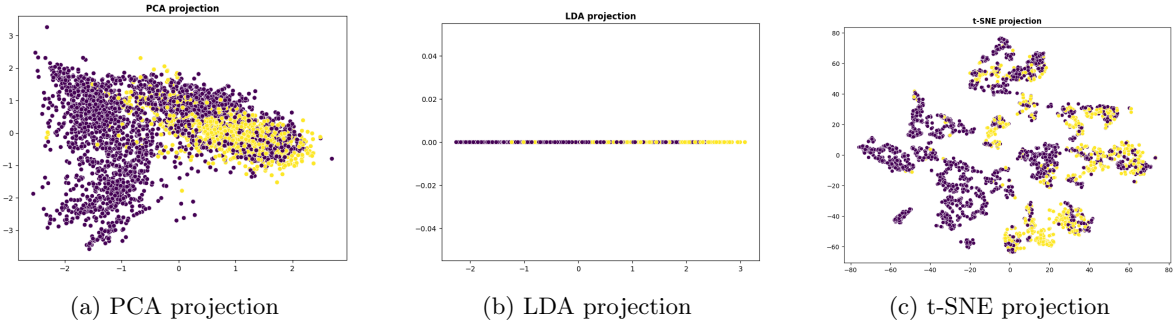


Figure 5: Feature reduction techniques we used to visualize data

4 Classification

This section presents the classification methodologies employed in this study. For each classifier, we describe the underlying algorithm, the hyperparameters selected through tuning, and the experimental results obtained across the three sampling strategies.

4.1 Decision Tree

A Decision Tree is a hierarchical model composed of decision nodes and leaf nodes. Each decision node represents a feature and contains a splitting rule, while leaf nodes represent output classes. During training, the tree is constructed recursively by partitioning the dataset based on feature values that maximize class separation.

4.1.1 Hyperparameters

Hyperparameters were optimized using RandomizedSearchCV with 5-fold cross-validation. Table 3 reports the selected parameters for each sampling strategy.

Strategy	max_depth	criterion
No Sampling	16	gini
Oversampling	20	entropy
Undersampling	13	entropy

Table 3: Optimized hyperparameters for Decision Tree.

4.1.2 Results

presents the confusion matrices, ROC curves, and feature importance plots for each sampling strategy.

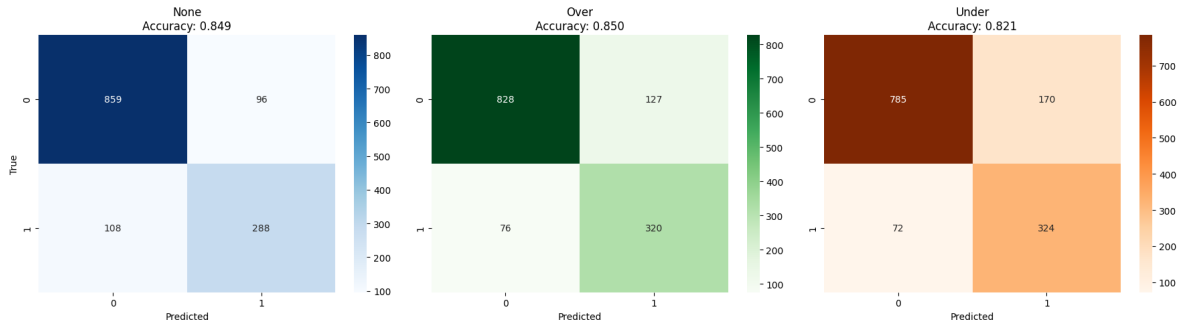


Figure 6: Decision Tree confusion matrices for each sampling strategy.

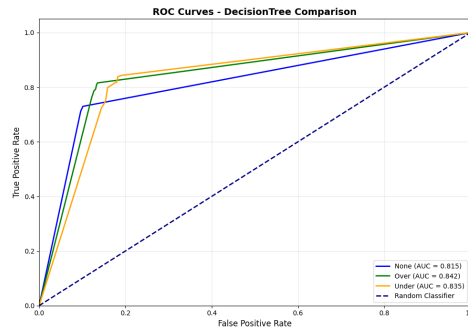


Figure 7: Decision Tree ROC curves comparison.

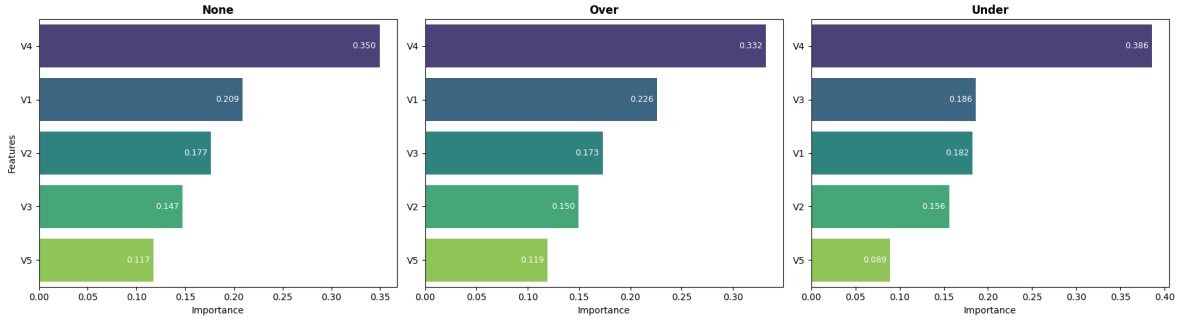


Figure 8: Decision Tree feature importance for each sampling strategy.

4.2 Naive Bayes

Naive Bayes is a probabilistic classifier based on Bayes' theorem with the assumption of conditional independence between features. For continuous features, the Gaussian Naive Bayes variant assumes that feature values follow a normal distribution within each class.

4.2.1 Hyperparameters

Gaussian Naive Bayes has no significant hyperparameters requiring optimization. The default configuration was employed, with the only parameter being `var_smoothing` (10^{-9}), which adds a portion of the largest variance to all variances for numerical stability.

4.2.2 Results

Figure 9 presents the confusion matrices and Figure 10 shows the ROC curves for each sampling strategy.

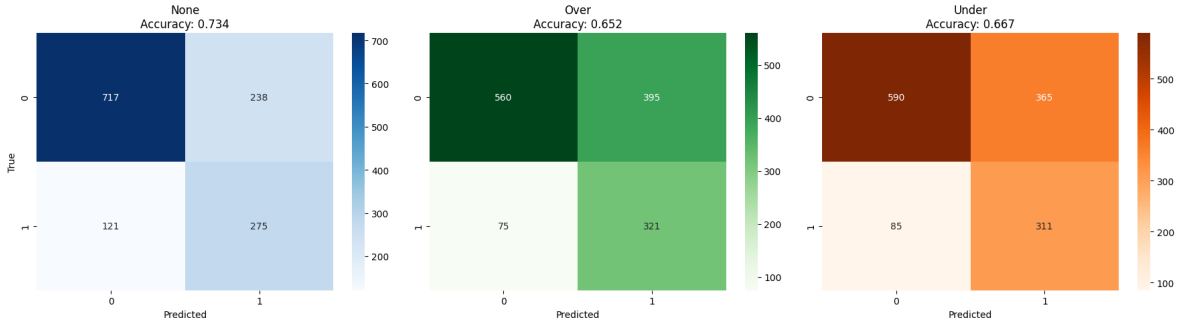


Figure 9: Naive Bayes confusion matrices for each sampling strategy.

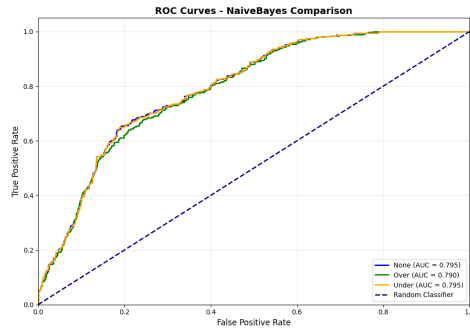


Figure 10: Naive Bayes ROC curves comparison.

4.3 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a non-parametric, instance-based learning algorithm that classifies new instances based on the majority class among their k nearest neighbors in the feature space. The algorithm does not build an explicit model during training; instead, it stores the entire training dataset and performs computation at prediction time. The choice of distance metric is fundamental in defining neighborhood membership, with **Euclidean distance** and **Cosine similarity** representing the two most widely used approaches.

4.3.1 Hyperparameters

Hyperparameters optimized in Table 4.

Strategy	n_neighbors	metric
No Sampling	1	euclidean
Oversampling	1	euclidean
Undersampling	1	cosine

Table 4: Optimized hyperparameters for KNN.

4.3.2 Results

Figure 11 presents the confusion matrices and Figure 12 shows the ROC curves for each sampling strategy.

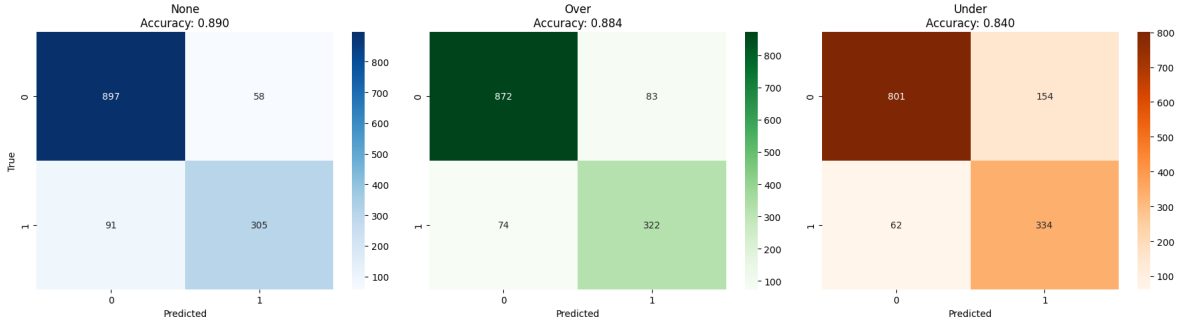


Figure 11: KNN confusion matrices for each sampling strategy.

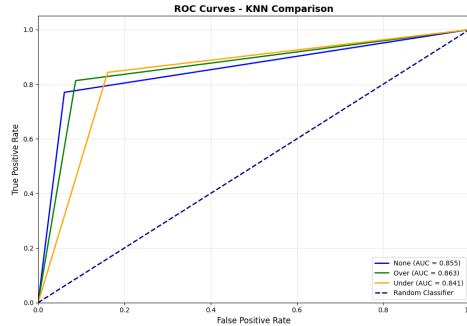


Figure 12: KNN ROC curves comparison.

4.4 Support Vector Machine

Support Vector Machine (SVM) is a supervised learning algorithm that seeks the optimal hyperplane separating classes in a high-dimensional feature space. The objective is to maximize the margin

between the hyperplane and the nearest data points (support vectors) from each class. For non-linearly separable data, SVM employs the kernel trick to map features into a higher-dimensional space where linear separation becomes possible.

4.4.1 Hyperparameters

Hyperparameters were optimized using RandomizedSearchCV with 5-fold cross-validation. Table 5 reports the selected parameters for each sampling strategy.

Strategy	kernel	gamma	C
No Sampling	rbf	1	100
Oversampling	rbf	1	100
Undersampling	rbf	1	10

Table 5: Optimized hyperparameters for SVM.

4.4.2 Results

Figure 13 presents the confusion matrices and Figure 14 shows the ROC curves for each sampling strategy.

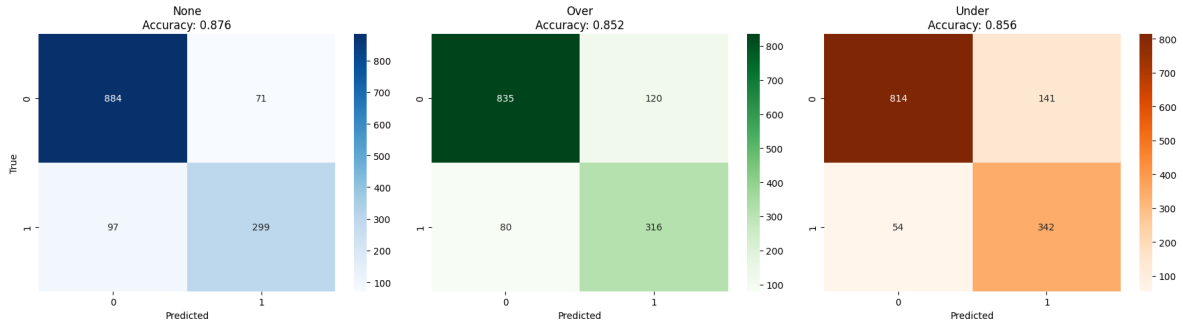


Figure 13: SVM confusion matrices for each sampling strategy.

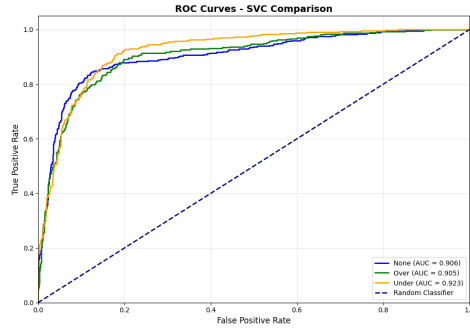


Figure 14: SVM ROC curves comparison.

4.4.3 Dimensionality Reduction Experiments

To evaluate the impact of feature extraction on classification performance, we applied two dimensionality reduction techniques to project the data into a 2-dimensional space: Principal Component Analysis (PCA) and Uniform Manifold Approximation and Projection (UMAP) [2]. The SVM classifier was then trained on the reduced feature space and compared against the original 5-dimensional approach.

PCA Approach: PCA performs linear dimensionality reduction by projecting data onto the principal components that capture maximum variance.

UMAP Approach: UMAP is a non-linear dimensionality reduction technique that preserves both local and global data structure through manifold learning.

Figure 15 and Figure 16 display the training set distribution in the projected 2D space along with the learned decision boundaries for each sampling strategy.

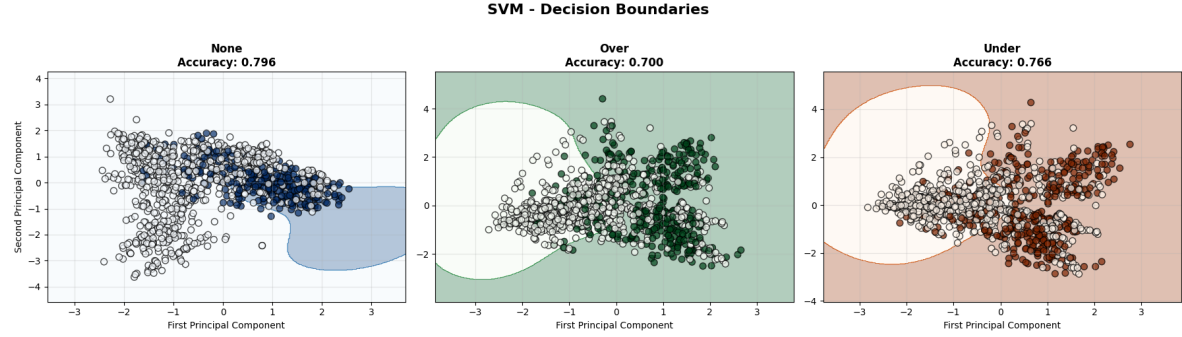


Figure 15: SVM trained on PCA-reduced features: training set distribution and decision boundaries.

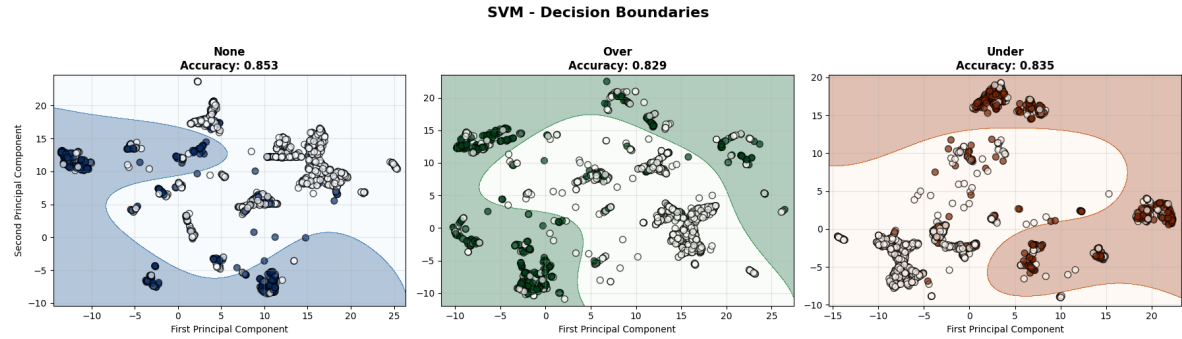


Figure 16: SVM trained on UMAP-reduced features: training set distribution and decision boundaries.

Table 6 compares the classification performance across the three approaches: original 5D features, PCA-reduced 2D features, and UMAP-reduced 2D features.

Approach	Accuracy (None)	Accuracy (Over)	Accuracy (Under)
SVM (5D)	0.87	0.85	0.85
SVM + PCA (2D)	0.79	0.70	0.76
SVM + UMAP (2D)	0.85	0.82	0.83

Table 6: SVM performance comparison: original features vs. dimensionality reduction.

4.5 Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of their predictions. Each tree is trained on a bootstrap sample of the data, and at each split, only a random subset of features is considered. This dual randomization reduces overfitting and variance compared to individual decision trees the final prediction is determined by majority voting. The ensemble approach provides robustness to noise and typically achieves higher accuracy than individual trees.

4.5.1 Hyperparameters

Hyperparameters were optimized using RandomizedSearchCV with 5-fold cross-validation. Table 7 reports the selected parameters for each sampling strategy.

Strategy	n_estimators	max_depth	criterion
No Sampling	190	39	entropy
Oversampling	113	26	gini
Undersampling	199	19	gini

Table 7: Optimized hyperparameters for Random Forest.

4.5.2 Results

Figure 17 presents the confusion matrices, Figure 18 shows the ROC curves, and Figure 19 displays the feature importance for each sampling strategy.

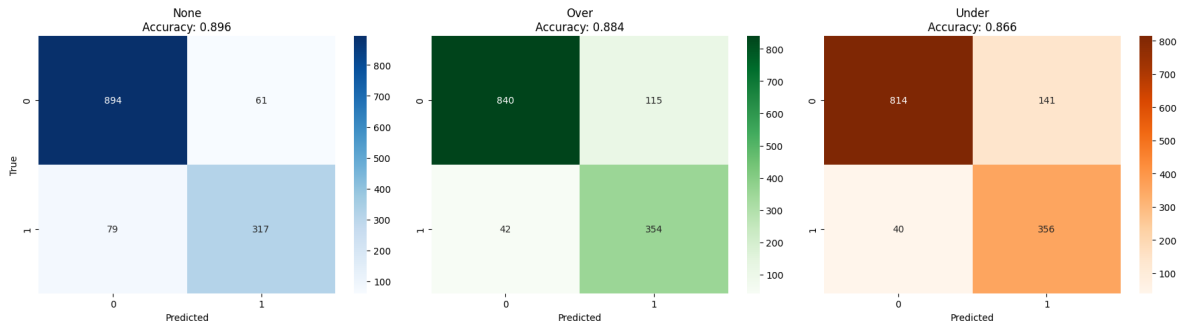


Figure 17: Random Forest confusion matrices for each sampling strategy.

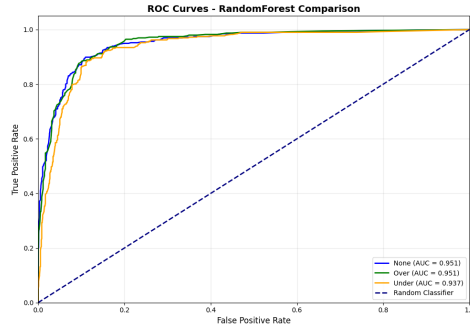


Figure 18: Random Forest ROC curves comparison.

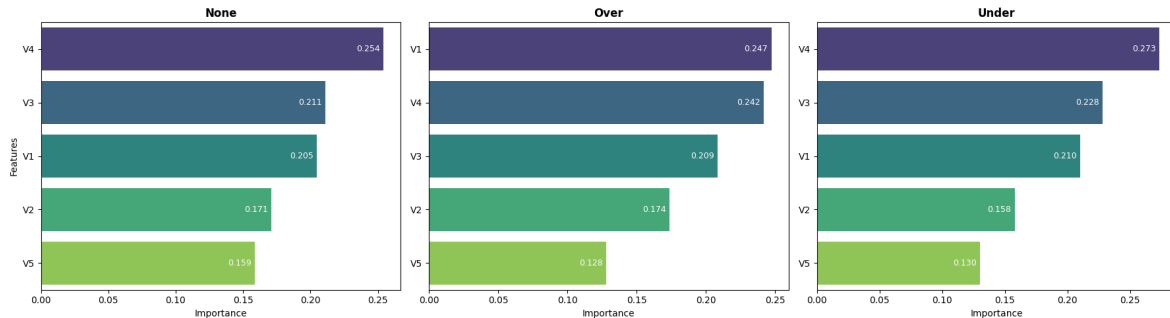


Figure 19: Random Forest feature importance for each sampling strategy.

4.6 AdaBoost

AdaBoost (Adaptive Boosting) is an ensemble method that combines multiple weak learners sequentially to create a strong classifier. Unlike Random Forest’s parallel approach, AdaBoost trains each subsequent learner on a reweighted version of the dataset, assigning higher weights to previously misclassified instances. This adaptive reweighting mechanism forces the ensemble to focus on difficult-to-classify instances.

4.6.1 Hyperparameters

Hyperparameters were optimized using RandomizedSearchCV with 5-fold cross-validation. Table 8 reports the selected parameters for each sampling strategy.

Strategy	n_estimators
No Sampling	196
Oversampling	131
Undersampling	156

Table 8: Optimized hyperparameters for AdaBoost.

4.6.2 Results

Figure 20 presents the confusion matrices, Figure 21 shows the ROC curves, and Figure 22 displays the feature importance for each sampling strategy.

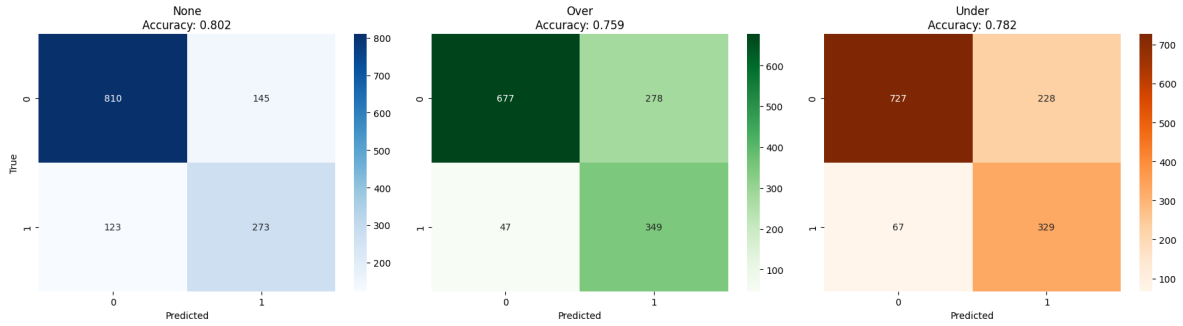


Figure 20: AdaBoost confusion matrices for each sampling strategy.

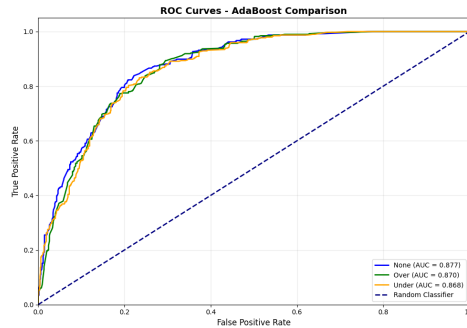


Figure 21: AdaBoost ROC curves comparison.

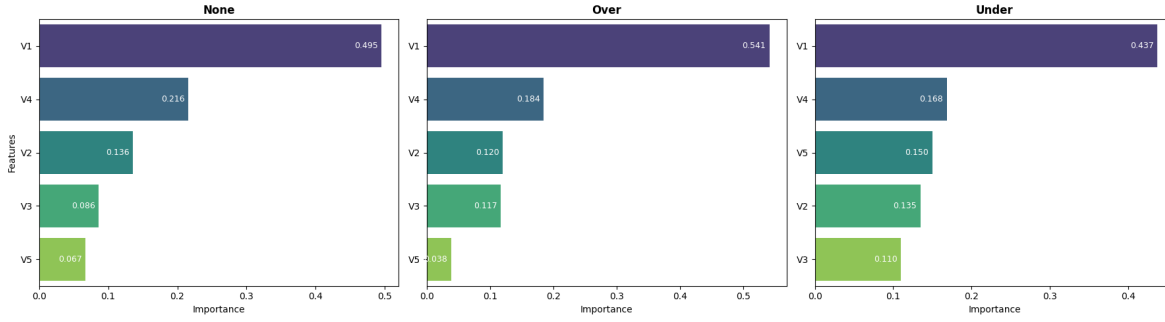


Figure 22: AdaBoost feature importance for each sampling strategy.

4.7 Extreme Gradient Boosting

XGBoost (Extreme Gradient Boosting) is an optimized implementation of gradient boosting that employs regularization techniques to prevent overfitting. The algorithm builds trees sequentially, with each new tree correcting the residual errors of the previous ensemble. XGBoost incorporates both L1 and L2 regularization, along with tree pruning and built-in handling of missing values, making it highly effective for structured data.

4.7.1 Hyperparameters

Hyperparameters were optimized using RandomizedSearchCV with 5-fold cross-validation. Table 9 reports the selected parameters for each sampling strategy.

Strategy	n_estimators	max_depth
No Sampling	56	13
Oversampling	190	36
Undersampling	82	8

Table 9: Optimized hyperparameters for XGBoost.

4.7.2 Results

Figure 23 presents the confusion matrices, Figure 24 shows the ROC curves, and Figure 25 displays the feature importance for each sampling strategy.

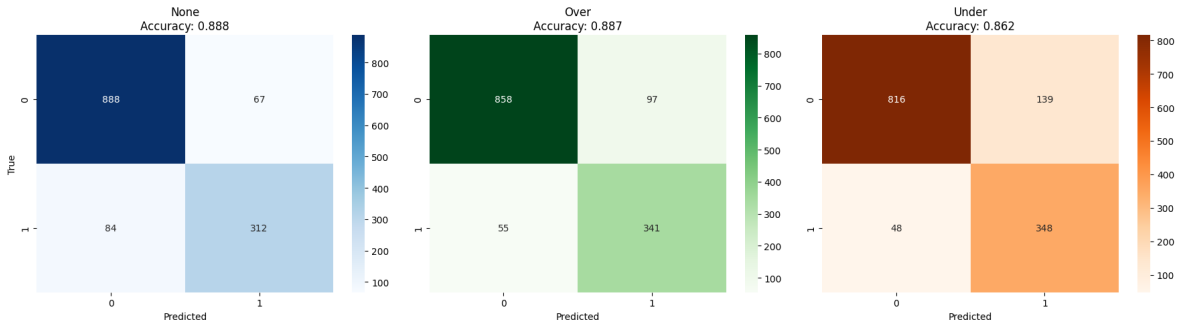


Figure 23: XGBoost confusion matrices for each sampling strategy.

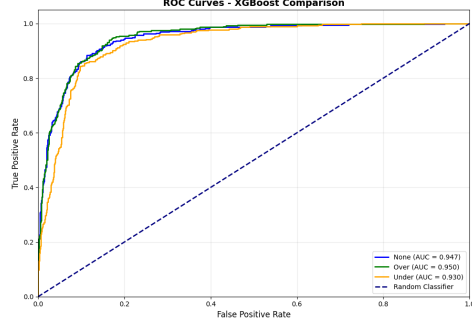


Figure 24: XGBoost ROC curves comparison.

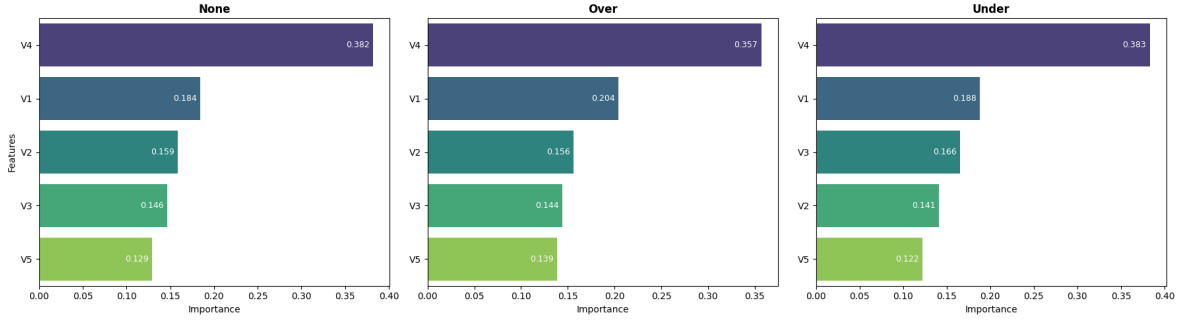


Figure 25: XGBoost feature importance for each sampling strategy.

4.8 Custom Ensemble

To investigate whether aggregating predictions from multiple classifiers can improve robustness and accuracy, we designed a custom ensemble model based on soft voting. The ensemble combines the predicted probabilities from multiple base learners by averaging, rather than relying on a single model's output.

Given m base models, each producing probability predictions $P_i = [p_{i1}, p_{i2}, \dots, p_{in}]$ for n instances, the ensemble probability is computed as:

$$P_{ensemble} = \frac{1}{m} \sum_{i=1}^m P_i \quad (1)$$

The final class prediction is obtained by rounding the aggregated probability to the nearest integer. This soft voting approach leverages the strengths of diverse classifiers while mitigating individual weaknesses through averaging.

4.8.1 Ensemble Composition

The custom ensemble combines three classifiers: **SVC**, **XGBoost**, and **AdaBoost**. Each base model uses its individually tuned hyperparameters as reported in Sections 4.4, 4.7, and 4.6, respectively. This configuration was chosen to combine a kernel-based method (SVC) with two boosting approaches (XGBoost, AdaBoost), providing diversity in learning strategies.

4.8.2 Results

Figure 26 presents the confusion matrices and Figure 27 shows the ROC curves for each sampling strategy.

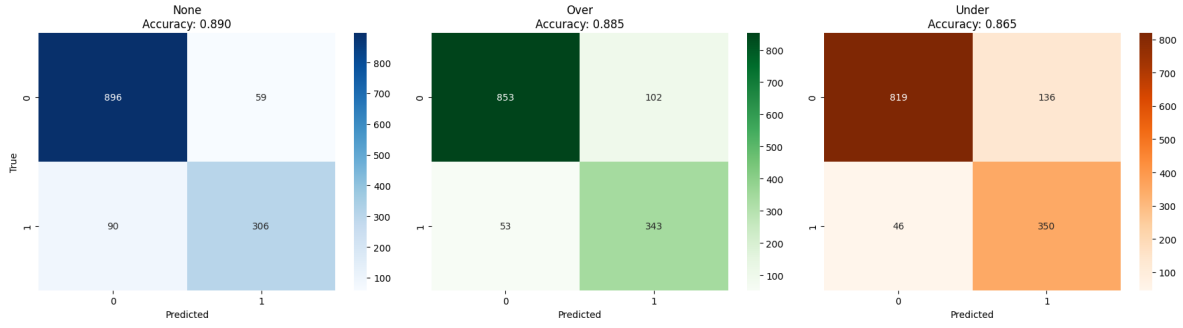


Figure 26: Custom Ensemble confusion matrices for each sampling strategy.

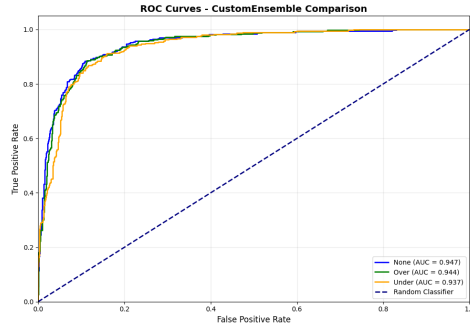


Figure 27: Custom Ensemble ROC curves comparison.

4.9 Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP) is a class of feedforward artificial neural networks. It consists of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer. Except for the input nodes, each node is a neuron that uses a non-linear activation function. MLP utilizes backpropagation for training, a supervised learning technique that adjusts the weights of the connections based on the error of the output compared to the target values.

4.9.1 Hyperparameters

Strategy	hidden_layer_sizes	learning_rate
No Sampling	37	constant
Oversampling	47	constant
Undersampling	43	invscaling

Table 10: Optimized hyperparameters for Multi-Layer Perceptron.

4.9.2 Results

Figure 28 presents the confusion matrices and Figure 29 displays the ROC curves for the MLP classifier across the different sampling strategies.

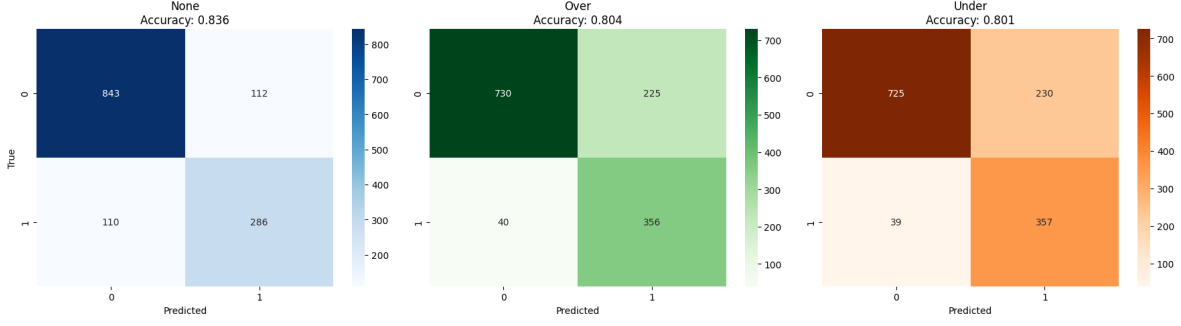


Figure 28: MLP confusion matrices for each sampling strategy.

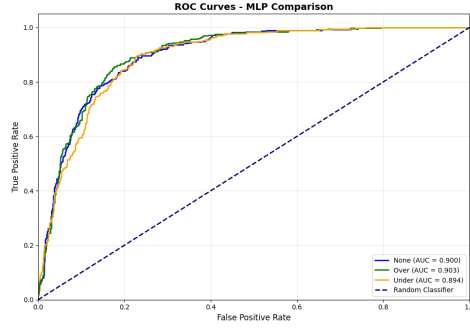


Figure 29: MLP ROC curves comparison.

4.10 Convolutional Neural Network

To further explore deep learning architectures, we implemented a 1D Convolutional Neural Network (Conv1D). 1D CNNs are particularly effective at detecting local patterns and spatial correlations in sequential or linear data. In this study, the convolutional layers are used to extract hierarchical features from the 5-dimensional acoustic feature vector.

4.10.1 Model Implementation

To maintain consistency with the experimental pipeline, the CNN was implemented using a custom wrapper class. This wrapper encapsulates the `TensorFlow/Keras` model and exposes `fit`, `predict`, and `predict_proba` methods, making the neural network fully compatible with the Scikit-Learn API used for the other classifiers.

4.10.2 Architecture and Hyperparameters

Unlike the previous models, the 1D CNN was evaluated using a fixed architecture without extensive hyperparameter tuning. The network consists of two convolutional layers followed by a Global Average Pooling layer to reduce dimensionality while preserving global information. The final stage comprises a dense fully-connected layer leading to a sigmoid output node. Table 11 details the specific configuration used.

Parameter	Value
Number of Conv1D Layers	2
Filters per Layer	[32, 16]
Kernel Size	2
Dense Layer Units	[10]
Activation (Hidden)	ReLU
Activation (Output)	Sigmoid
Optimizer	RMSprop
Loss Function	Binary Cross-Entropy

Table 11: Fixed architecture configuration for the 1D CNN.

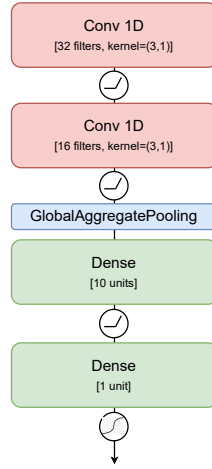


Figure 30: 1D CNN architecture schema sampling strategy.

4.10.3 Results

Figure 31 presents the confusion matrices and Figure 32 shows the ROC curves for the 1D CNN across the sampling strategies.

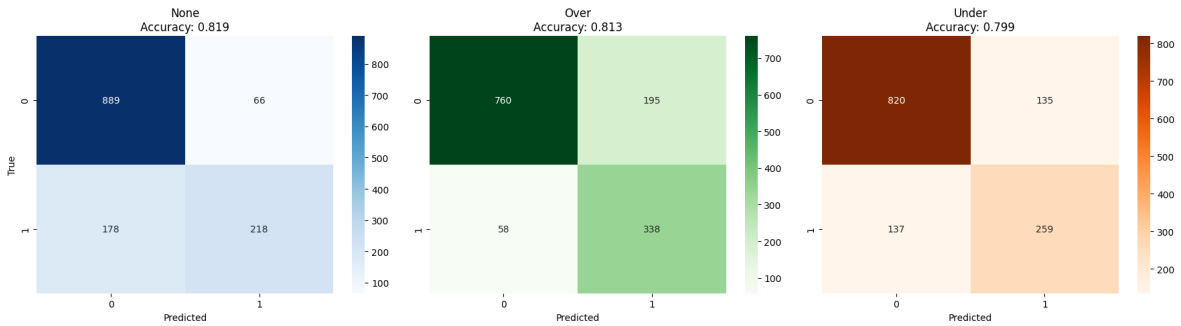


Figure 31: 1D CNN confusion matrices for each sampling strategy.

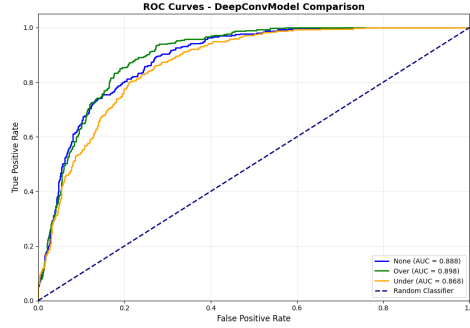


Figure 32: 1D CNN ROC curves comparison.

5 Discussion and Conclusions

This section addresses the four research questions that guided our experimental evaluation, followed by a summary of the key findings and recommendations.

5.1 Q1: Which classification models performed best?

RandomForest, **CustomEnsemble**, and **XGBoost** consistently outperformed other models across all sampling strategies. Table 12 presents the complete model ranking by weighted F1-score averaged across the three sampling configurations.

Model	Weighted F1-score
RandomForest	0.8821
CustomEnsemble	0.8801
XGBoost	0.8791
KNN	0.8712
SVC	0.8611
DecisionTree	0.8399
SVC_UMAP	0.8389
MLP	0.8135
DeepConvModel	0.8103
AdaBoost	0.7809
SVC_PCA	0.7540
NaiveBayes	0.6844

Table 12: Model ranking by weighted F1-score averaged across sampling strategies.

The superior performance of ensemble tree-based methods can be attributed to two factors: (1) ensemble averaging reduces variance and mitigates the impact of noisy samples; (2) tree-based methods are inherently less sensitive to class imbalance due to their recursive partitioning approach.

KNN also demonstrated strong performance, which can be explained by the data density characteristics. Distance-based models perform well when datapoints belonging to the same class are spatially clustered, as observed in the dimensionality reduction visualizations.

Notably, deep learning models (MLP, DeepConvModel) underperformed compared to both ensemble and shallow models. This can be attributed to: (1) the data is tabular and well-engineered, favoring traditional models; (2) the dataset size ($\approx 4,000$ samples) is insufficient relative to deep model parameters, limiting generalization; (3) deep architectures were not exhaustively tuned, unlike shallow models.

Q1 Summary

- **Best models:** RandomForest, CustomEnsemble, XGBoost ($F1 > 0.87$)
- **Why:** Ensemble averaging and robustness to class imbalance
- **Deep models underperform:** Tabular data, small dataset, limited tuning

5.2 Q2: Are there significant differences between the results obtained on the original data and those obtained after dimensionality reduction?

The results demonstrate that applying dimensionality reduction leads to performance degradation compared to the original 5-dimensional feature space. Table 13 presents the comparison between SVC trained on original data and on reduced representations.

Model	Weighted F1-score	vs SVC Baseline
SVC (5D)	0.8611	—
SVC_PCA (2D)	0.7540	-12.44%
SVC_UMAP (2D)	0.8389	-2.58%

Table 13: SVM performance comparison across dimensionality reduction approaches.

Since the dataset already has low dimensionality (only five features), models can efficiently process the original data without reduction. Both PCA and UMAP necessarily discard information, leading to degraded performance. PCA causes the largest drop due to its linear nature, which fails to preserve non-linear relationships in the data. UMAP performs substantially better, nearly matching the baseline, demonstrating its superior ability to retain both local and global data structure through non-linear manifold learning.

Q2 Summary

- **Original data performs best:** Low dimensionality (5 features) does not require reduction
- **PCA:** Significant performance loss (-12.44%) due to linear projections
- **UMAP:** Minimal degradation (-2.58%), preserves non-linear structure

5.3 Q3: Which algorithms are most sensitive to data dimensionality?

To comprehensively evaluate algorithm sensitivity to dimensionality, we collected performance metrics from all models across the three sampling strategies. Figure 33 presents grouped bar plots comparing the F1-scores for each model, displaying the overall weighted F1-score and class-specific F1-scores for Class 0 (nasal sounds, 70% of data) and Class 1 (oral sounds, 30% of data).

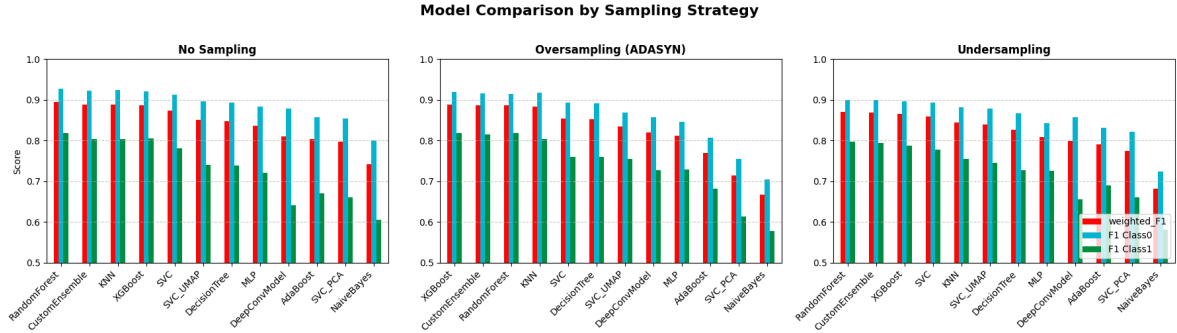


Figure 33: F1-score comparison across models and sampling strategies: overall weighted (red), Class 0 (cyan), and Class 1 (green).

Figure 34 presents Kernel Density Estimation (KDE) plots showing the distribution of model F1-scores for each sampling strategy. Table 14 provides the quantitative performance differential (Δ) to support the visual analysis.

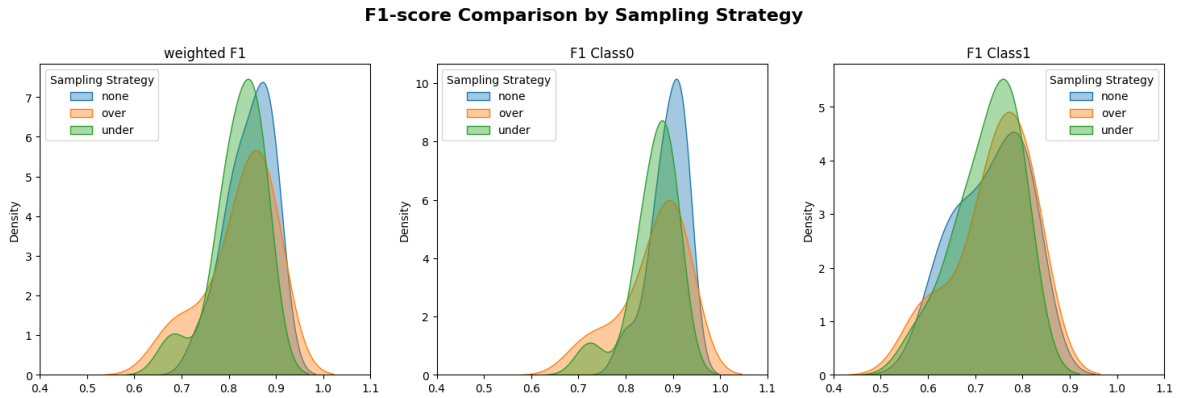


Figure 34: KDE plots showing F1-score distributions: overall weighted (left), Class 0 (center), and Class 1 (right) across sampling strategies. Each observation of these distribution is the performance of a model

Strategy	Weighted F1	F1 (Class 0)	F1 (Class 1)
Oversampling	$\Delta = -2.55\%$, $p = 0.41$	$\Delta = -3.75\%$, $p = 0.17$	$\Delta = +0.80\%$, $p = 0.85$
Undersampling	$\Delta = -3.01\%$, $p = 0.25$	$\Delta = -3.70\%$, $p = 0.09$	$\Delta = -1.02\%$, $p = 0.80$

Table 14: Performance differential (Δ) compared to no sampling strategy.

The KDE plots reveal no substantial difference between sampling strategies, with no sampling consistently performing better on average. The performance differentials confirm this observation: both oversampling and undersampling lead to degradation, with no statistically significant improvements (all p -values > 0.05).

Our results indicate that **kernel-based and probabilistic algorithms** are the most sensitive to data dimensionality. This is clearly visible for SVC, whose performance drops significantly when dimensionality reduction is applied: the baseline SVC outperforms both SVC-UMAP and, more markedly, SVC_PCA, demonstrating that reducing an already low-dimensional feature space leads to information loss that directly impacts these models.

Similarly, **Naive Bayes** underperformance can be attributed to multiple factors beyond dimensionality sensitivity: the features are continuous rather than categorical (the optimal data type for Naive Bayes), and while feature correlation is low, low correlation does not imply independence—the fundamental assumption of Bayesian classifiers may not be satisfied.

In contrast, **tree-based and ensemble methods** (Random Forest, XGBoost, Custom Ensemble) consistently achieve the highest F1-scores, demonstrating low sensitivity to dimensionality. These models operate via feature-wise splits and are robust to both irrelevant features and moderate changes in feature space representation.

Neural models (MLP, DeepConvModel) show lower performance overall, suggesting that in low-dimensional tabular settings they do not benefit from increased model complexity and are indirectly affected by dimensionality through underfitting or inefficient representation learning.

Q3 Summary

- **High sensitivity:** Kernel-based (SVC) and probabilistic models (Naive Bayes)
- **Moderate sensitivity:** Neural networks (MLP, DeepConvModel) and distance-based (KNN)
- **Low sensitivity:** Tree-based and ensemble methods (RandomForest, XGBoost)
- **Sampling impact:** No significant benefit; no sampling strategy performs best

5.4 Q4: Strengths, limitations, and possible extensions

This section discusses the strengths and limitations of our approach, along with possible extensions, from both preprocessing and classification perspectives.

5.4.1 Preprocessing and Feature Extraction

Strengths: The low-dimensional tabular data benefits from minimal preprocessing. With only five features, most models can learn efficiently without heavy transformations.

Limitations: Dimensionality reduction can harm performance, as demonstrated by the SVC experiments with PCA and UMAP. Class balancing, while expected to improve performance, did not have a significant impact on generalization capability in the test set. Underfitting affects models with a large number of parameters and not enough data to learn, as observed with our deep neural models.

Possible Extensions: Data augmentation could be explored to generate synthetic samples and observe whether deep models manage to improve and scale their performance. Model-based feature selection, such as tree-based importance ranking, could provide an alternative approach. Given the results, avoiding feature extraction and dimensionality reduction is recommended for this dataset.

5.4.2 Classification Models

Strengths: Tree-based ensembles (Random Forest, XGBoost, Custom Ensemble) show performances above 87% F1-score, demonstrating consistent robustness across all experimental conditions.

Limitations: Neural networks underperform with small datasets; our data size ($\approx 4,000$ samples) is insufficient for deep learning models to generalize effectively. Naive Bayes performs poorly, probably because the independence assumption fails for the correlated acoustic features.

Possible Extensions: Ensemble-based models can be improved using more sophisticated voting, stacking, or blending techniques to combine the best-performing classifiers. Model tuning and architecture search can be enhanced by performing grid search or Bayesian optimization for hyperparameters.

Q4 Summary

Preprocessing: Minimal transformation is sufficient; dimensionality reduction and class balancing do not improve performance. Data augmentation and model-based feature selection are potential extensions.

Classification: Tree-based ensembles achieve the best results ($F1 > 0.87$), while neural networks and Naive Bayes underperform. Advanced ensemble techniques and Bayesian optimization could further improve results.

References

- [1] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *IEEE International Joint Conference on Neural Networks*, pages 1322–1328, 2008.
- [2] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger. Umap: Uniform manifold approximation and projection. *The Journal of Open Source Software*, 3(29):861, 2018.
- [3] OpenML. Phoneme dataset. <https://www.openml.org/d/1489>, 2014.