MLPC Report - Task 3: Classification Experiments

Team OBSERVE

Johannes Grafinger **Jonas Gantar** **Leonhard Markus Spanring** **Reinhard Josef Pötscher**

Contributions

Johannes, Jonas and Reinhard were responsible for tasks 1.) Labeling Function, 2.) Data Split, 3.) Audio Features and 4.) Evaluation. Leonhard was responsible for task 5.) Experiments. All of us together were responsible for task 6.) Analysing Predictions. We prepared this report in the same constellation. We all worked together on the presentation. We held regular meetings at which each member presented their results and the others critically evaluated the work. Everyone communicated via a dedicated Discord server and the editing was done over a Github repository.

1.) Labeling Function: For your analysis, you may focus on a subset of the 58 classes provided.

For subtasks a.) and b.) we mainly focused on the following, randomly chosen subset of classes of size 7: ['Airplane', 'Bird Chirp', 'Power Drill', 'Cowbell', 'Siren', 'Trumpet', 'Rooster Crow'].

a.) Assess how accurately the applied labeling functions capture the intended classes. Do the mapped classes correspond well to the free-text annotations? Are the labeled events clearly audible within the indicated time regions?

b.) Which audio features appear most useful for distinguishing between the classes of interest?

In order to answer this question, we first of all randomly collected 1000 frames from the dataset (which represents a collection of all frames from all files) per class where it is positive, resulting in a total of around 7000 frames (a bit less, since some frames contain multiple of the classes). For each frame, we additionally include the previous and next two frames as context length by simply concatenating them with the features from the central frame (in temporal order). Each class therefore has 1000 vectors of shape 5 * 942 = 4710, where 942 represents the summed number of dimensions for all features.

For each class, we then estimated the Mutual Information between the label vector and all features dimensions. For each feature, we then calculated the average Mutual Information over all feature dimension corresponding to it. This gives us a measure for how informative the feature generally is for predicting the label correctly, which also captures nonlinear relationships. We then normalized the values per label, resulting in the first plot of (Figure 1). However, since this approach cannot capture effects features only have in combination with one another, we also tried fitting a simple Random Forest Classifier on the data and extracted its feature importances, resulting in the second plot of Figure Figure 1.

The features 'embeddings', 'melspectrogram', 'mfcc' and 'contrast' all seem to be pretty useful according to our Mutual Information graphic, with 'embeddings' being the most important one. However, the fact that the Random Forest almost exclusively relies on the embeddings might suggest heavy correlations between them and the other features. Which does make sense, considering many of the other features are usually used for creating such audio embeddings.

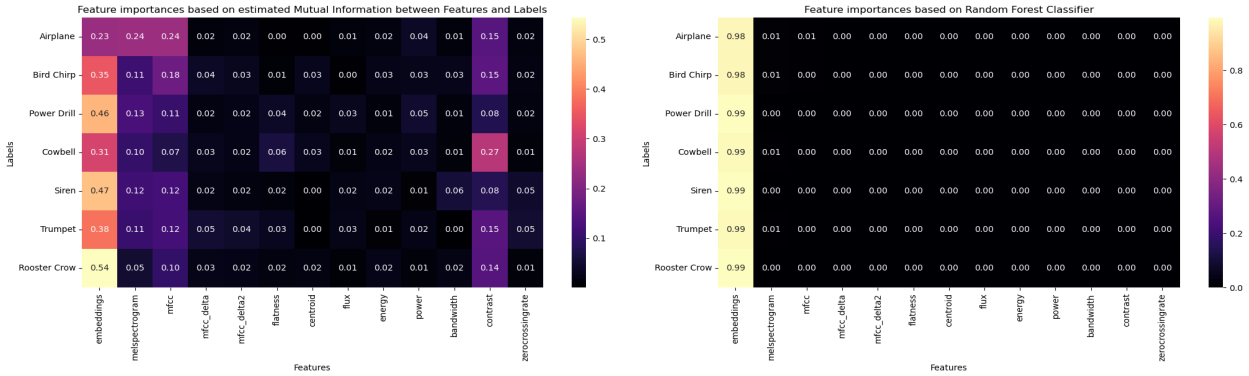

Figure 1: Estimated feature importances as by Mutual Inforamation and Random Forest methods.

c.) How well do the chosen audio features group according to the discretized class labels? Do samples of the same class form tight clusters?

Using the four features mentioned above, we reduced the high-dimensional data to two dimension using T-SNE and plotted the results in (Figure 2). Note that for examples corresponding to more than one of the classes, we randomly chose one for coloring.

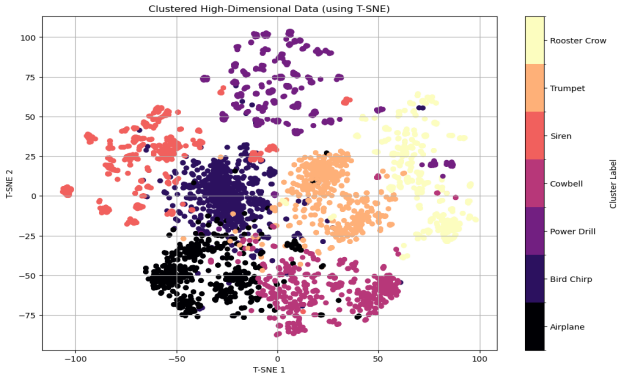

Figure 2: Downprojected examples using selected features and their class correspondences.

Overall, the results look very reasonable. Clusters are identifiable and examples within mostly have the same attributed class. Some overlaps are of course inevitable, considering the relatively high complexity of the dataset.

2.) Data Split

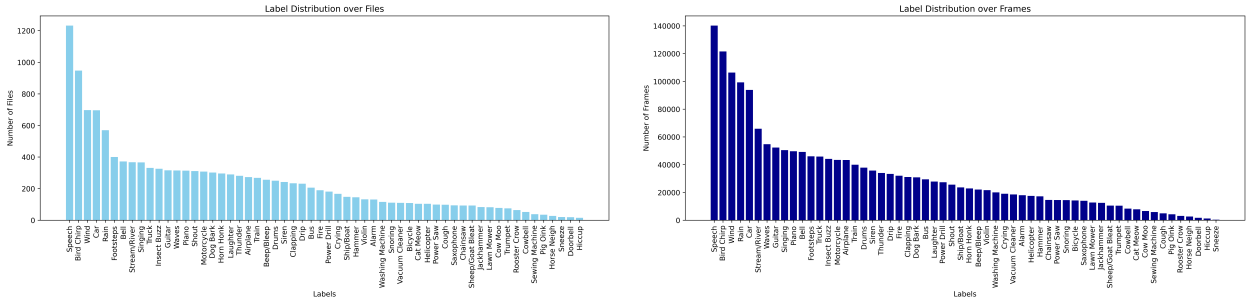

Figure 3: Label distribution

a.) Describe how you split the data for model selection and performance evaluation.

For audio classification tasks, the standard and robust approach is to split the dataset into three distinct subsets to effectively train, tune, and evaluate the model:

1. **Training Set:** Used to train (fit) the model's parameters. Typically constitutes around 60–80
2. **Validation Set:** Used for hyperparameter tuning and model selection. Helps prevent overfitting to the training data. Typically 10–20
3. **Test Set:** Used only for final performance evaluation to estimate how the model will perform on unseen data. Hold back until the very end to provide an *unbiased final estimate of performance*. Typically 10–20

- ALTERNATIVE: **Cross-Validation** If the dataset is small, k-fold cross-validation (often with stratification) on the training+validation split can be used for more robust hyperparameter tuning. The test set remains untouched until final evaluation.

We first loaded all the audio data with the features and the labels and split it into individual frames.

b.) Are there any potential factors that could cause information leakage across the data splits if they are not carefully designed? If yes, how did you address these risks?

Yes, there are **critical risks of information leakage** in audio classification if splits are not carefully designed:

- **Common Sources of Leakage:**
  Overlapping audio segments: If data is split by segment but not by full recording/file/session, temporal leakage can occur.
  Same noise/audio source in multiple splits: If one noise/audio source appears in both training and test sets, the model may learn source-specific features rather than class-specific ones.
  Similar background noise or recording conditions: If recordings from the same session or environment are in multiple splits, the model may overfit to irrelevant cues.
- **Mitigation Strategies:**
  Source-Independent Splits: Ensure that all data from a single noise/audio source is assigned to only one of the training, validation, or test sets.
  Session-Based Splits: If the data includes session or recording metadata (labels), use it to group and split accordingly.
  Careful Preprocessing: Avoid any preprocessing (e.g. feature normalization) that uses statistics from the full dataset — use only training data statistics.

To avoid possible problems when splitting into the three data sets, we only split at file boundaries and tried to have an approximately equal distribution in all parts (stratification) (Figure 3). We worked in frames during pre-processing (normalization).

c.) Describe how you obtained unbiased final performance estimates for your models.

To ensure the final performance estimate is unbiased:

1. **Strict Test Set Separation:** The test set is never used during training or validation. Final model evaluation is performed only once on this set after all model development is complete.
2. **Validation-Driven Tuning:** Hyperparameters and model architecture are selected using only training and validation data.

3.) Audio Features

a.) Which subset of audio features did you select for your final classifier? Describe the selection process and the criteria you used to make your choice.

After reflecting on the results from section 1.), b.), we have decided to run our experiments on the 'embeddings' feature as well as the other three features mentioned in that section. Even though the results imply heavy correlations and possible redundancies to the other three features - especially considering the possible creation process of these audio embeddings - this still offered a slight increase in performance for tests we ran on a small scale.

Unfortunately we were not able to include context frames in our experiments as we did earlier, since the computational complexity was already extremely high.

b.) Did you apply any preprocessing to the audio features? If so, explain which techniques you used and why they were necessary.

Preprocessing is an important step for ensuring consistency in the data and making the lives of models like Support-Vector-Machines much easier. We mainly normalized the data in order to to bring every example to a similar scale magnitude-wise, which generally improves performance and convergence rates.

We tried normalizing the data globally using the global mean and standard deviation from only our training set (to avoid data leakage), normalizing the features frame-wise to have unit mean and variance as well as a combination of these. In the end, doing only the latter proved to be the most performant, so that is what we did.

4.) Evaluation

a.) Which evaluation criterion did you choose to compare hyperparameter settings and algorithms, and why?

In terms of performance metrics we decided to record - per class label - both the Balanced Accuracy as well as the F1-score of our models predictions. As single measures of performance of a model, we then used the macro-averaged Balanced Accuracy / F1-score (over the 58 class labels), like was shown in the tutorial session.

Balanced Accuracy, defined as the average recall per class (always binary in our case) obviously gives a more balanced performance metric, focusing on both the positive and negative classes. This metric was mainly included in order to be able to be able to compare the models to the baseline perfomance (see 4.), b.)).
The F1-score on the other hand is most likely a much better suited performance metric for this task. Representing the harmonic mean of precision and recall, it has a stronger focus on predictions of the positive class. This makes sense for our task, as one classifier essentially consists of 58 binary classifiers - one per class label - which needs to predict in which frames it is present, i.e. positive. Generally this metric will be much lower than the Balanced Accuracy.

b.) What is the baseline performance? What could be the best possible performance?

In our case, the simple Majority-Class baseline classifier would always and for every label predict the negative class, since no single class-label is active in more than half of the frames in our dataset. This in turn would lead to a F1-score of 0 (since there are no True Positives). As already mentioned, in order to still be able to compare our models to the baseline we also recorded the Balanced Accuracy. Obviously, as the recall for the negative class would be 1 and for the positive class 0, our baseline Balanced Accuracy is 0.5.

Considering the unavoidable noise and mislabelings in the data, we suspect the best possible performance one can achieve to be around an F1-score of 0.9 up to 0.95.

5.) Experiments

a.) For at least three different classifiers, systematically vary the most important hyperparameters and answer the following questions for each of them:

i.) How does classification performance change with varying hyperparameter values? Visualize the change in performance.

ii.) (To what extent) Does overfitting or underfitting occur, and what does it depend on?

b.) After selecting appropriate hyperparameters, compare the final performance estimate of the three classifiers.

6.) Analysing Predictions: Find two interesting audio files that have not been used for training and qualitatively evaluate your classifier's predictions.

For this task, we use the predictions from our best performing classifier, namely the Logistic Regression model with 'C=0.1, penalty=l2, solver=lbfgs'.
We analyze predictions for the two files: 584022.wav, where the model does a good job and 74790.wav, where the struggles with its classifications.
For the calculation of prediction quality metrics we skipped class-labels which are not present in the file. Otherwise, some of the misclassifications would actually boost the final Balanced Accuracy.

a.) Use the spectrogram and the sequence of predictions to visualize the classifier output.

We get simply get the model outputs for frames of a single file together with the corresponding 'melspectrogram' feature. We then only plot labels which have been predicted to be present somewhere within the file, see Figure 4.

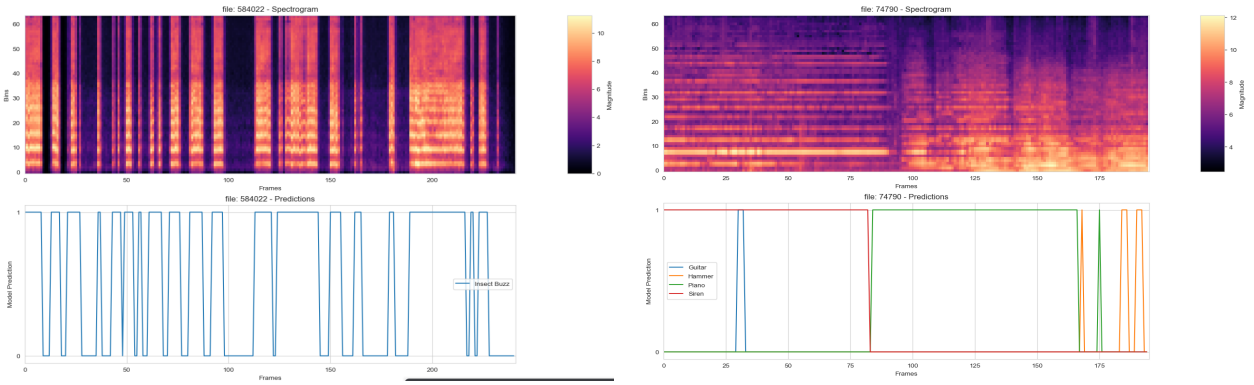

Figure 4: Visualization of predictions from our best model.

b.) Listen to the audios and inspect the corresponding predictions of the classifier. How well does the classifier recognize the classes?

- **584022.wav**: This represents an extremely simple example where the models predictions are really precise and mostly correct. Not only did it correctly predict the (only) class label, but temporally the predictions match very well. Sometimes however, some of the individual buzzes get grouped together, especially when the pauses in between them get really short. The model achieved an F1-score of 0.928 and a Balanced Accuracy of 0.896 for this file.
- **74790.wav**: This represents a more complex example where the models predictions are horribly incorrect. When actually listening to the audio file, one clearly recognizes that it solely consists of string instruments being played and some murmuring (disregarding background noises). Confusing the strings in the beginning for a siren still is understandable, everything else is completely wrong. The model achieved an F1-score of 0 and a Balanced Accuracy of 0.25 for this file.

c.) What are particular problematic conditions that cause the classifier to mispredict classes? Can you think of simple postprocessing steps that might help improve the predictions?

From analyzing several predictions of audio recordings we have identified that the models have a hard time predicting classes in recordings containing lots of noise or where several class labels are active at once. Additionally, predictions for class labels which are underrepresented in the dataset also are lacking.
Often the classifier has little peaks and valleys lasting only a few frames in its predictions, as can be seen in the second plot of Figure 4 with the 'Guitar' or 'Piano' labels. Such spurious (mis-)classifications could easily be removed by setting a lower threshold for how many frames a positive or negative prediction has to last, and filtering the output accordingly. This should overall increase the quality of predictions.