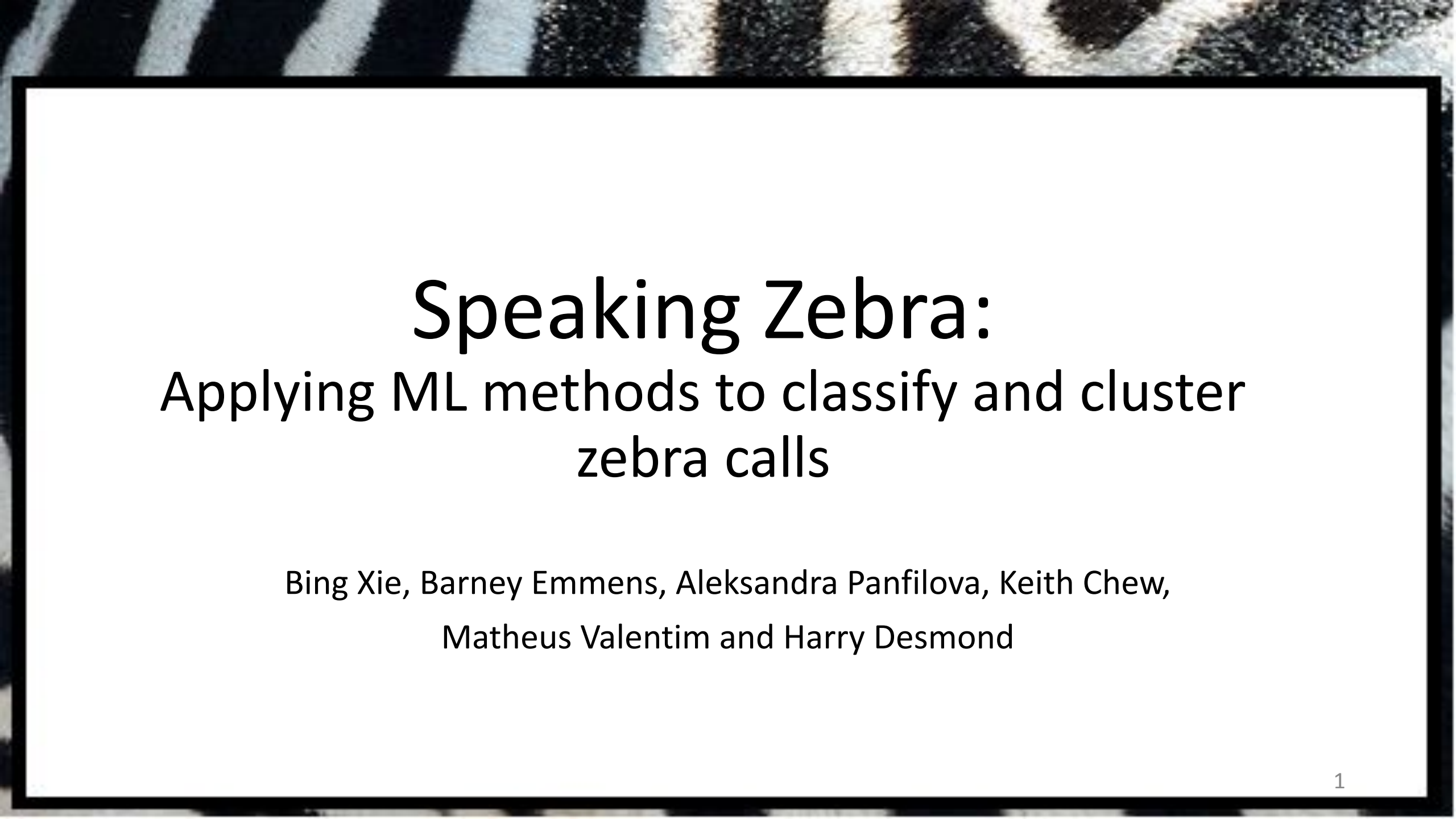


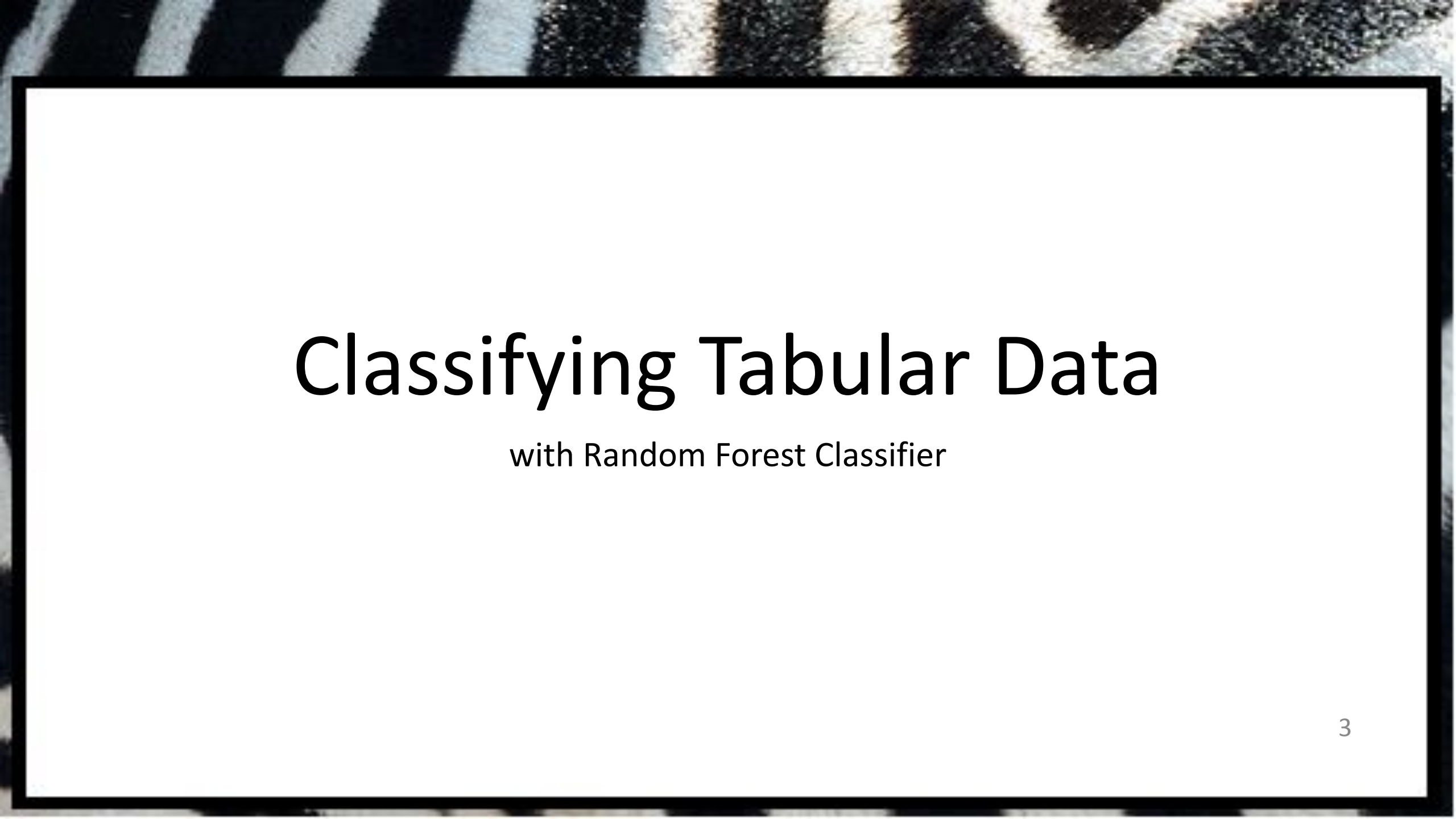
Speaking Zebra:
Applying ML methods to classify and cluster zebra calls

Bing Xie, Barney Emmens, Aleksandra Panfilova, Keith Chew,
Matheus Valentim and Harry Desmond

Our project

- Zebras produce 4 different types of calls
- We gathered zebra sounds data via field trip
- We applied different ML techniques on that data

	Supervised	Unsupervised
Tabular data	Tabular data classification	Clustering
Audio data	Audio data classification	Enconding and decoding audio



Classifying Tabular Data

with Random Forest Classifier

Data, issues and how we handled them

Easy to use and test small dataset with **413** obs and **9** features

Highly **unbalanced** data (snorts and whinnies) → Oversampling

Different scales → Tree based

Some quite **skewed** features → Tree based

Choosing among the 9 variables to avoid **overfitting** → SHAP values and RFE

Modelling tabular data challenges

- The best model (Random Forest)
- The best hyperparameters (n of vars, n of trees and n to split leaf)
- Best features (4/9 or 6/9)
- Extra techniques like oversampling

Model selection

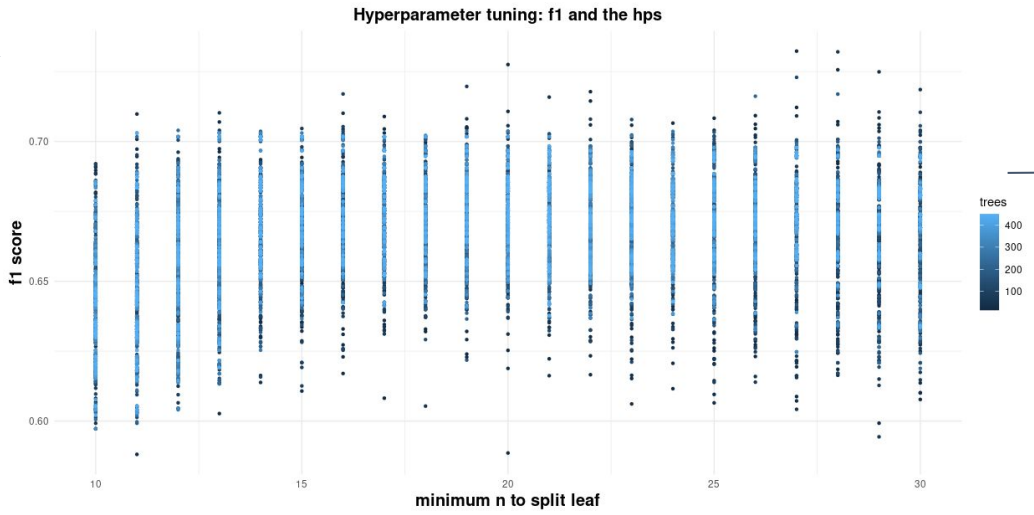
RF

Dec tree

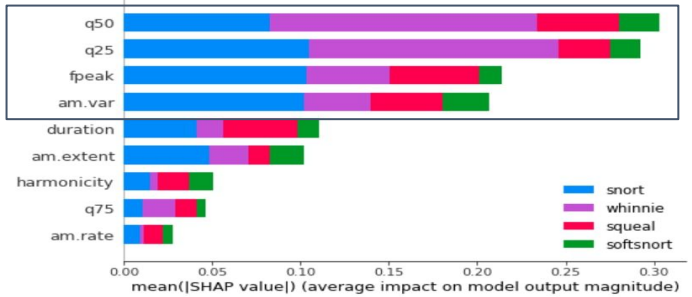
NN

XGB

Hyperparameter tuning



Feature selection

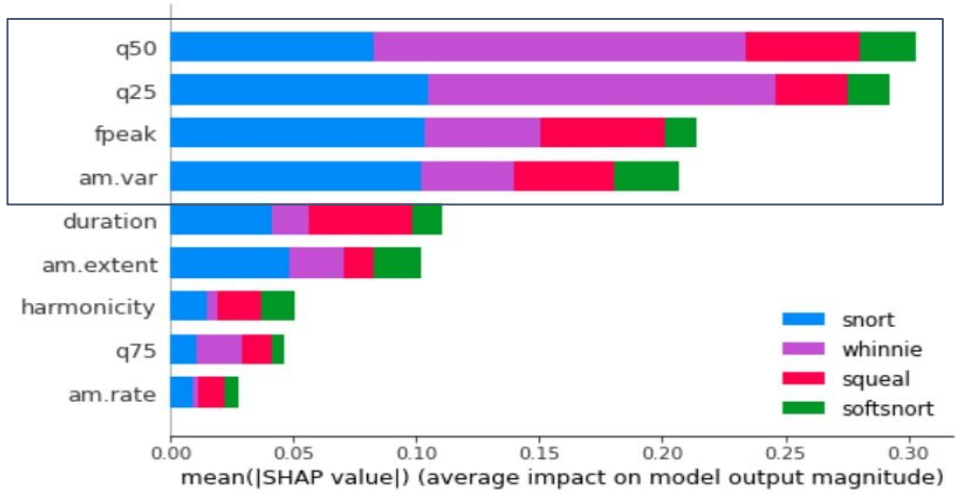


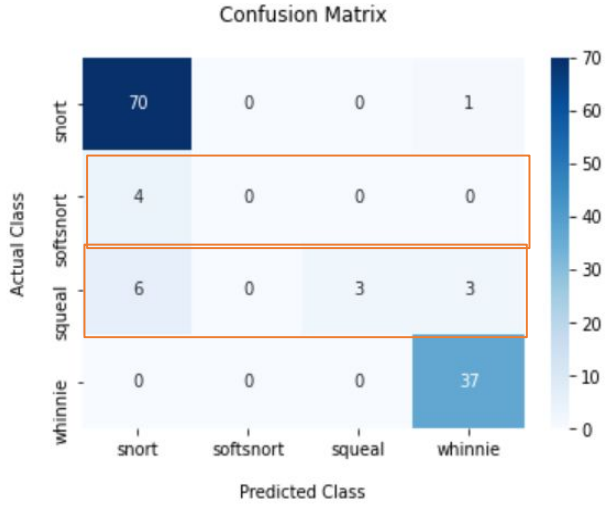
SHAP values and RFE agreed on: q25, q50,fpeak, am.var

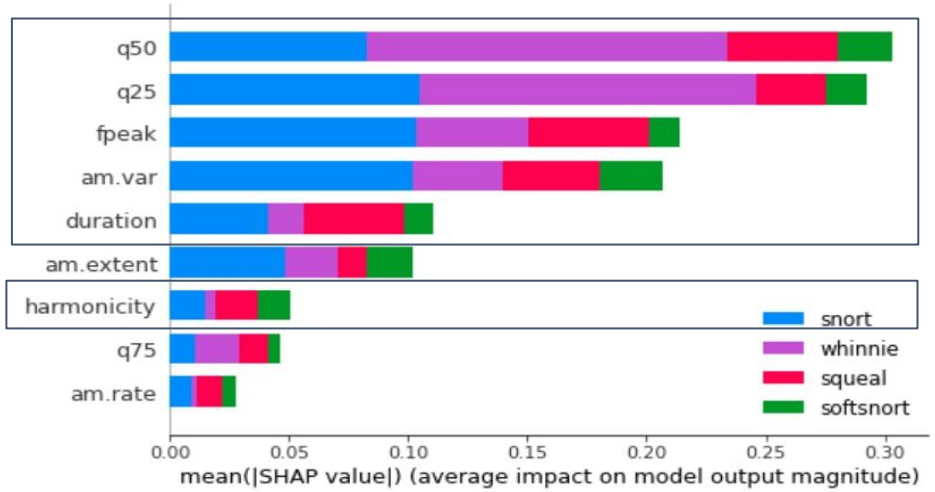
Models attempts and fails timeline:

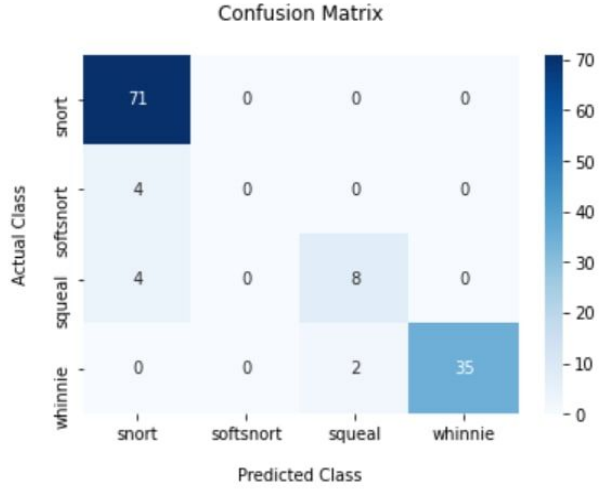
	Accuracy	Precision (whinnie, squeal, snort, softsnort)	Recall (whinnie, squeal, snort, softsnort)	Observations
Random Forest, HP tuned	91-93	(85,67,88,0)	(100,3,97,0)	Initial model
RF, HP tuned + less vars	91-95	(97,60,90,0)	(95,50,99,0)	2nd Best accuracy, no oversampling
RF, HP tuned less vars + us	98-99	(97,100,99,100)	(100,100,99,75)	Best accuracy
Decision Tree, HP tuned	89-91	(96,50,88,100)	(96,43,94,33)	Highly interpretable
Neural Networks, HP tuned	89	(95,100,85,0)	(100,33,97,0)	Initial model, Discontinued

Feature importances using SHAP values



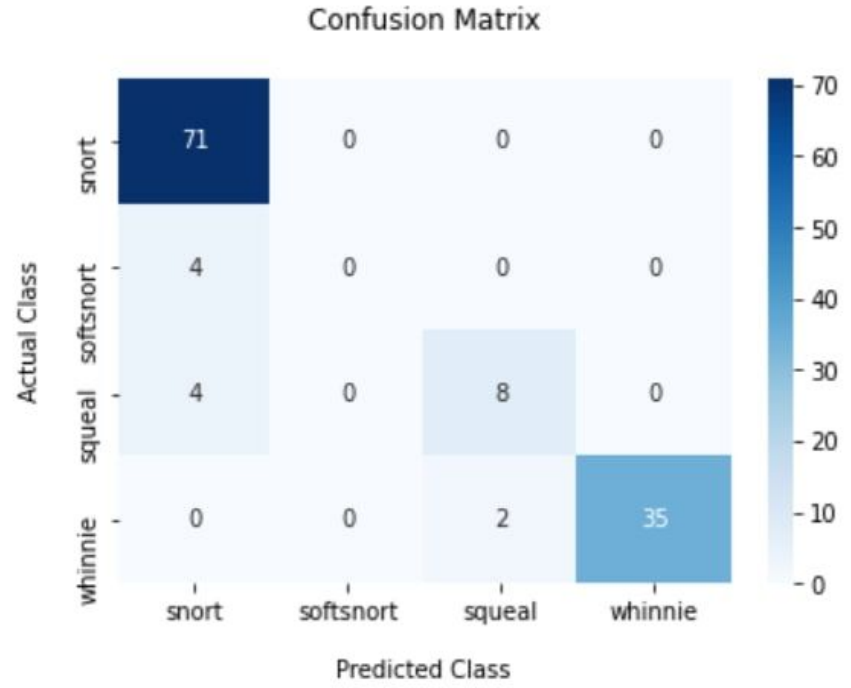




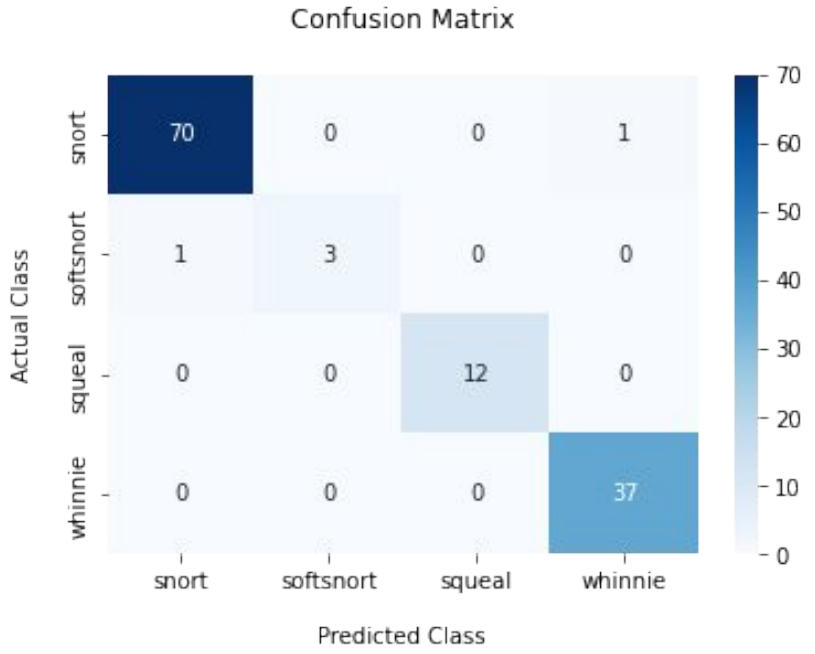


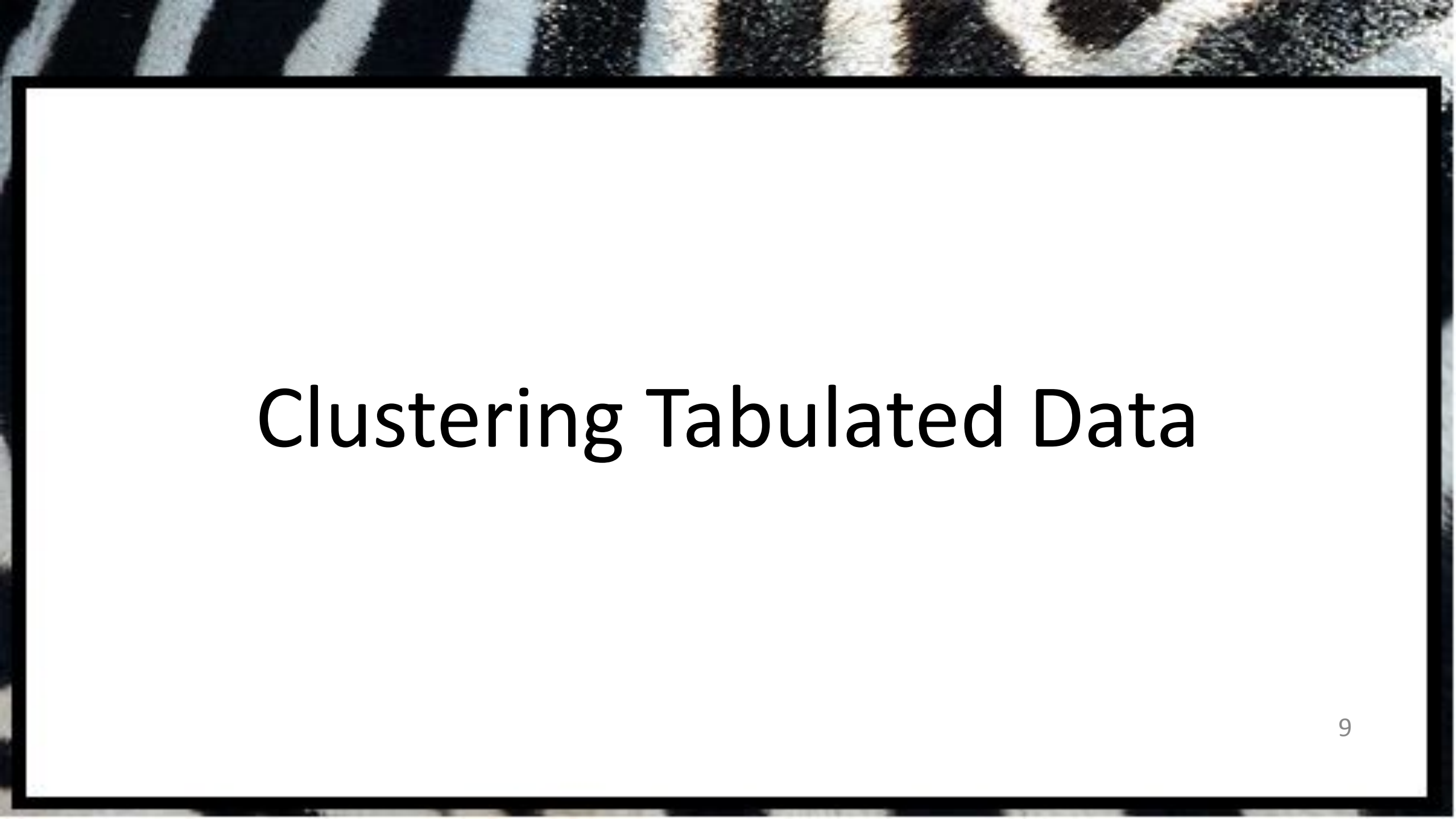
Confusion matrices from different methods

RF model with 6 features (SHAP values)



RF with 4 features & upsampling





Clustering Tabulated Data

Why?

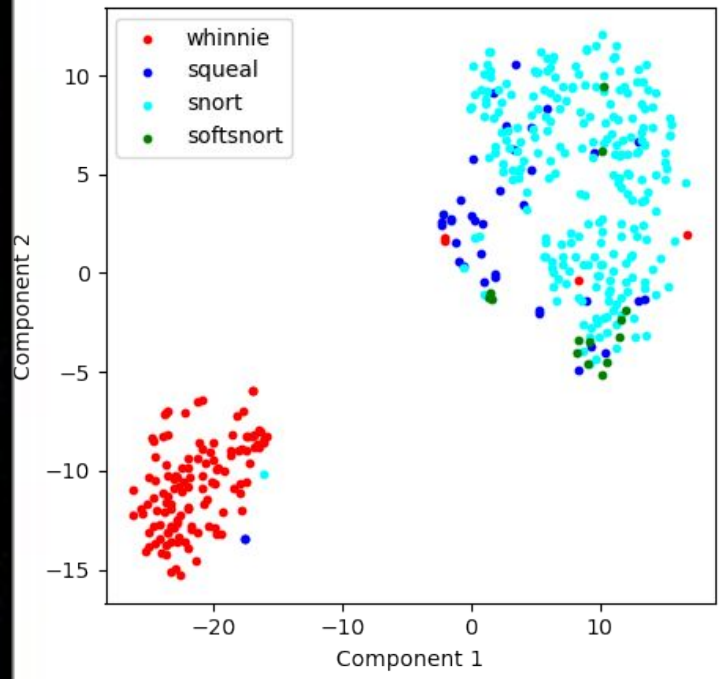
- To find new classes for the calls
- To divide classes into sub-classes
- To visualise how the calls relate to each other

Preprocessing:

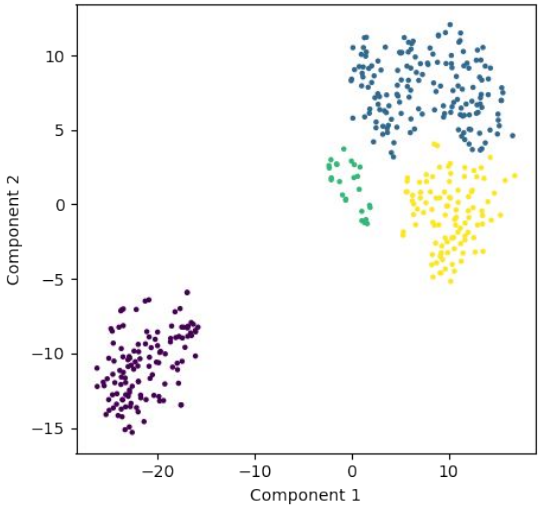
- Using features taken from SHAP values
- MinMax Scaling
- TSNE for dimensionality reduction

Plots

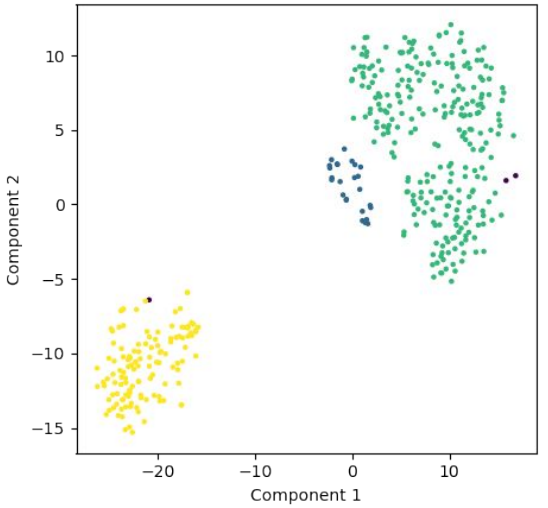
Labelled TSNE



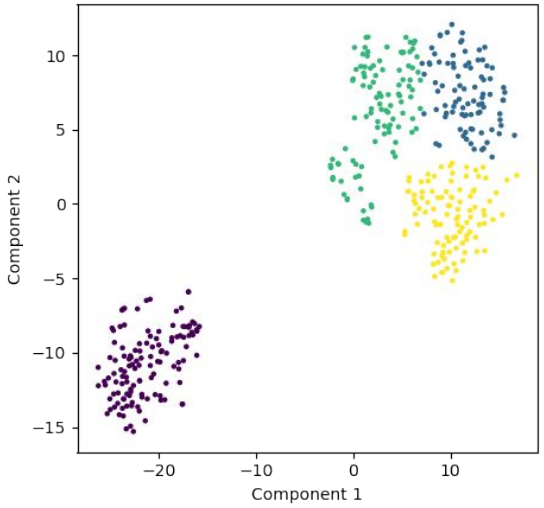
Spectral Clustering (4)



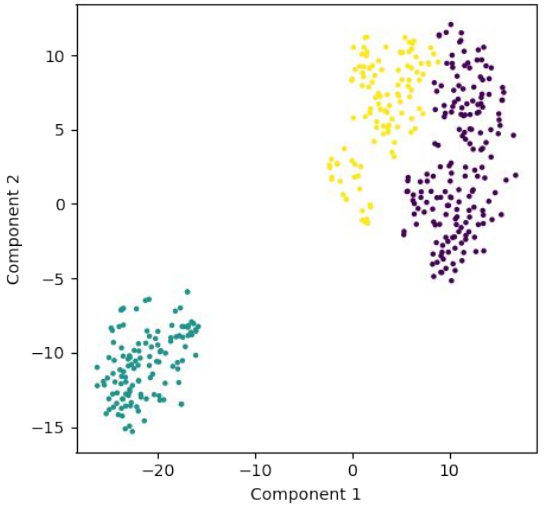
DBSCAN (3)

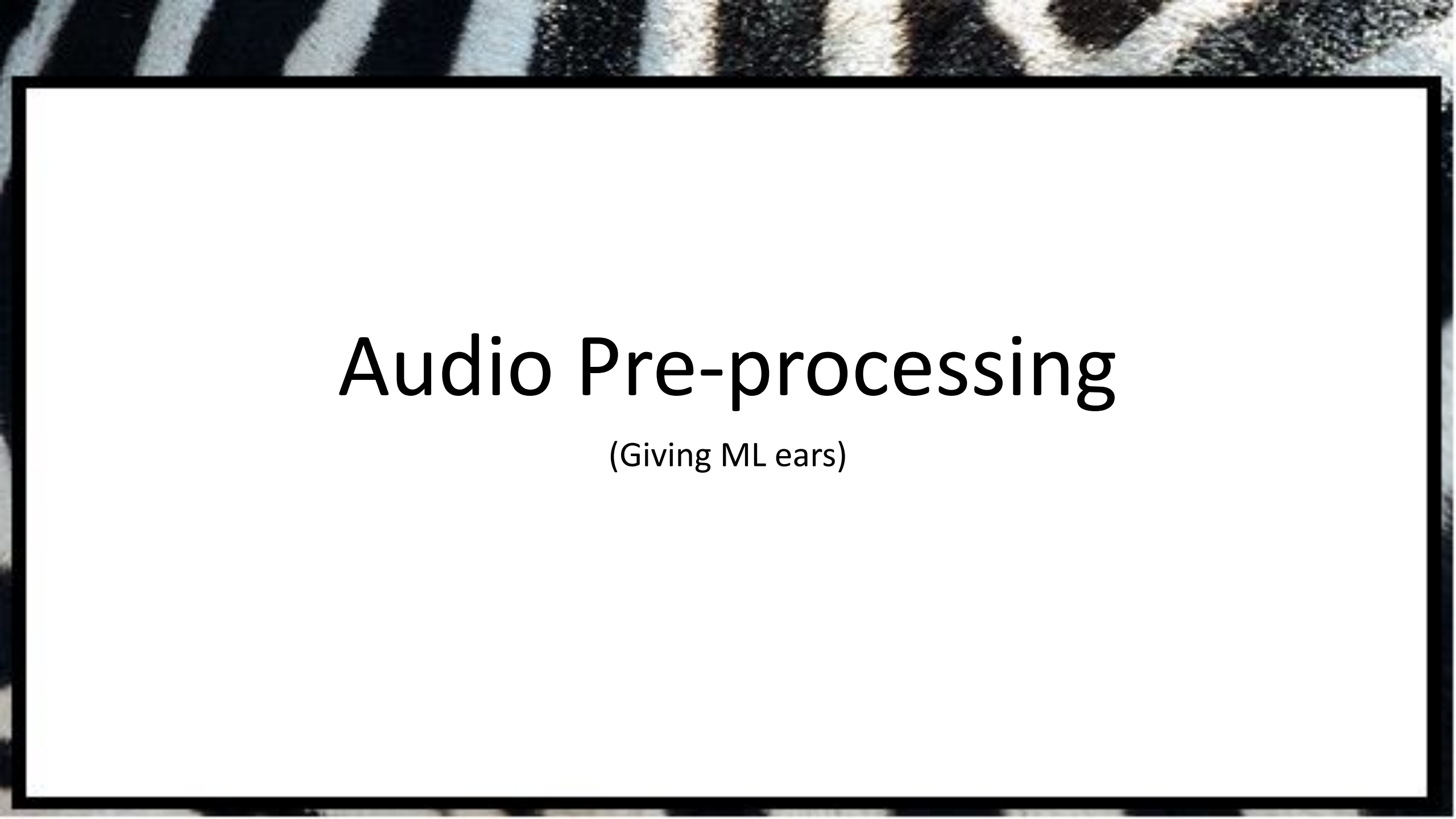


KMeans (4)



Agglomerative clustering (3)





Audio Pre-processing

(Giving ML ears)

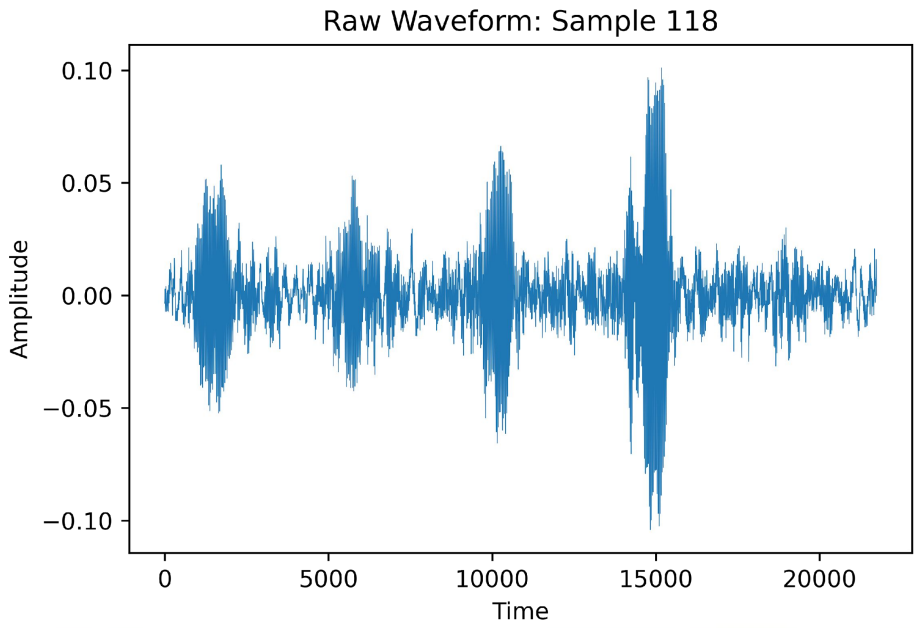
Choice of dataset

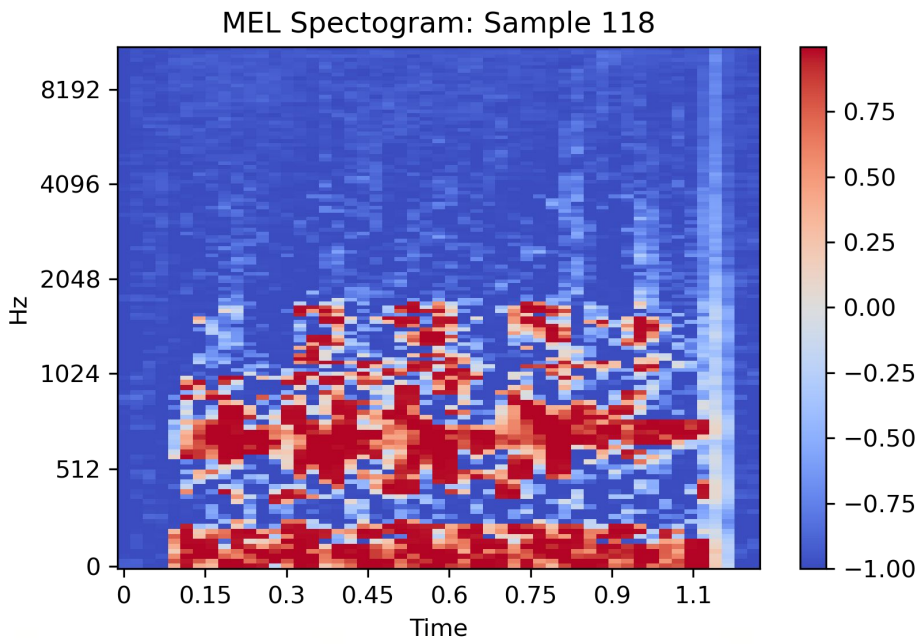
‘Good’ Labels	‘Bad’ Labels*	Not-Zebra**
• More classes (snort, softsnort, squeal, whinny) • Less background noise • Audio trimmed to call precisely • Less Samples (~400)	• Only two classes (snort, squeal) • Lots of background noise • Roughly trimmed audio • Lots of samples (>1000)	• Used to detect zebras from background noise. • Could generate data automatically.

* ‘Bad’ Labels contains the ‘Good’ Labels audio.
** Program written to detect this but data unavailable.

Challenge 1: Transform waveform into CNN friendly image.

1. Pad samples to same length of longest sample with zeros. Shape: (21739)
2. MEL Spectrogram. Shape: (128, 51)
3. dB
4. Normalise
5. One hot encoding. Shape: (4)





Challenge 2: (Try to) Ensure there is enough data

Balance

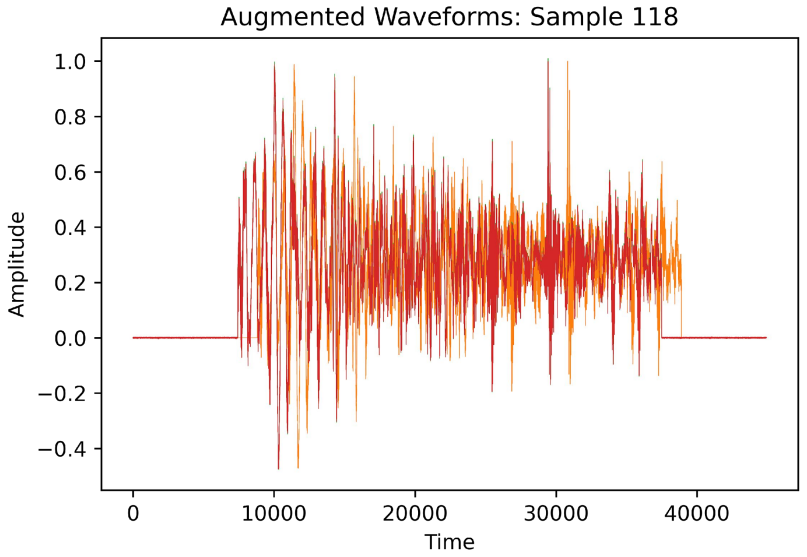
```
Initial class proportions:
[0.592 0.035 0.095 0.278]
----------------------
Balancing: snort
Multiplier: 1.0
----------------------
Balancing: softsnort
Multiplier: 10.0
----------------------
Balancing: squeal
Multiplier: 2.0
----------------------
Balancing: whinnie
Multiplier: 1.0
----------------------
Final class proportions:
[0.42  0.248 0.135 0.197]
```

Augment

- (Normalise & Buffer)
- Time shift: <10%
- Volume shift: <5%
- Noise: < 0.1%

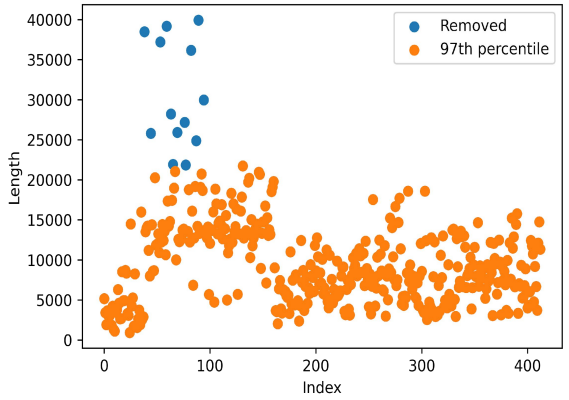
Data shape: (4, 564 ,26085)

Waveforms for MEL spectrogram

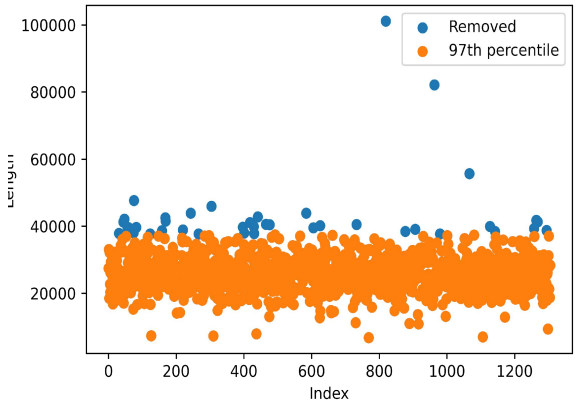


Challenge 3: Keep it manageable for a CPU.

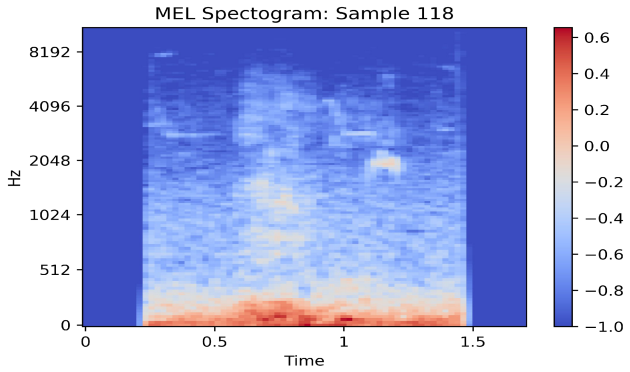
Good Labels



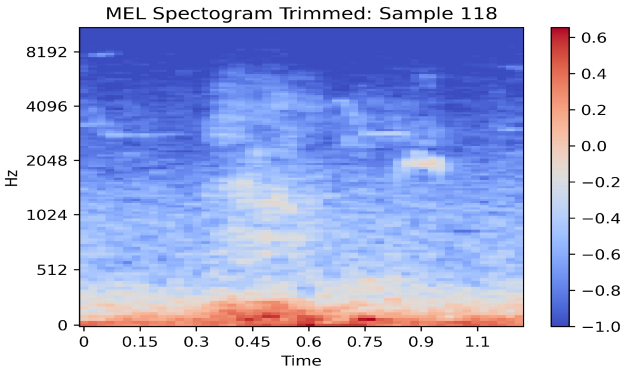
Bad Labels

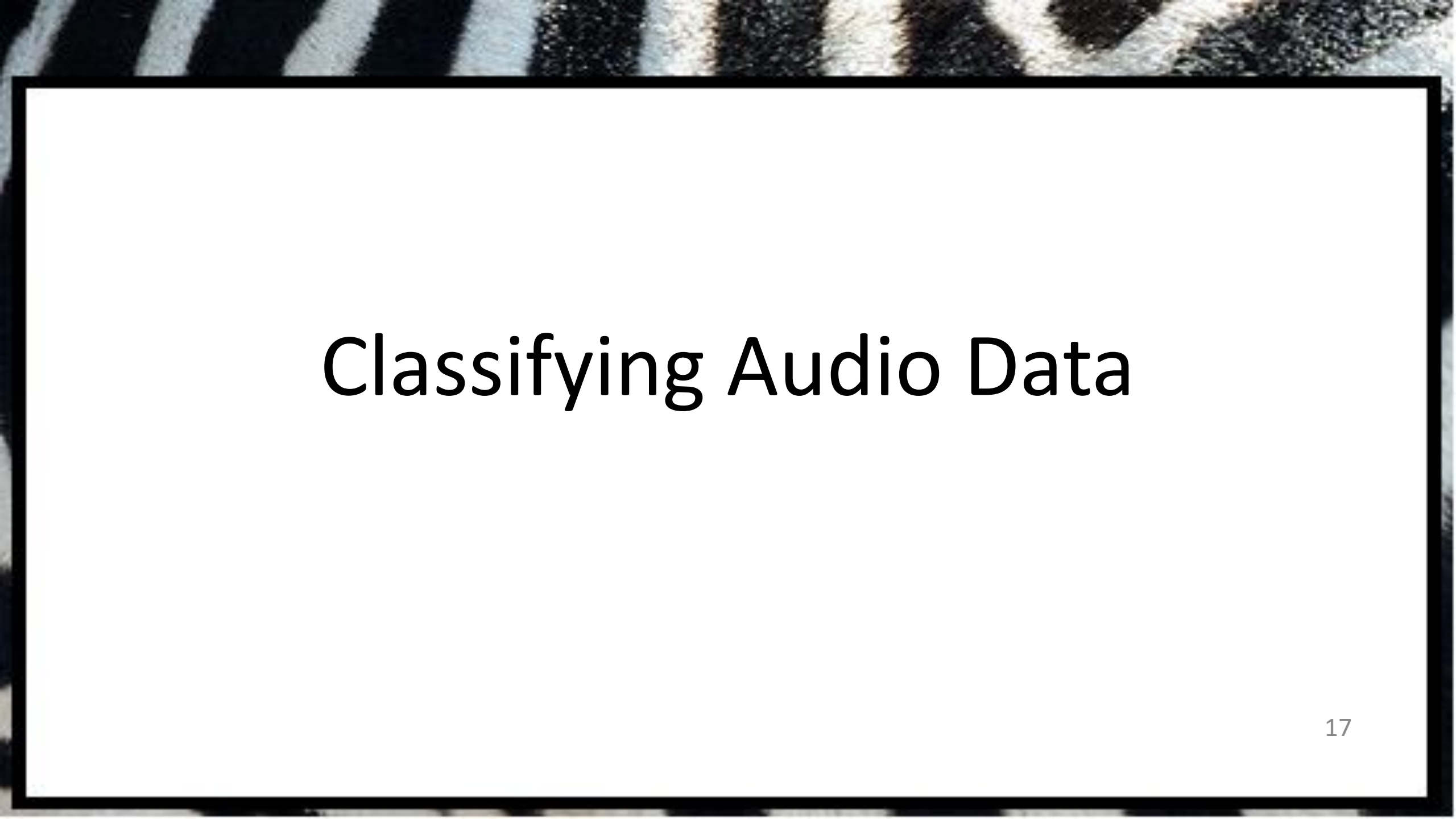


Initial shape: (128,74)



Final shape: (128,51)




Classifying Audio Data

CNN Structure

Input (n_samples, 128, 51, 1)

25 epochs

Cov2D (n_filters = 30, kernel_size = 8, activation = ‘relu’)

MaxPooling2D (2x2)

Conv2D (n_filters = 26, kernel_size = 4, activation = ‘selu’)

MaxPooling2D (3x3)

Dropout (rate = 0.404)

Flatten ()

Dense (units = 94)

Dense (units = num_class)

1st convolutional layer

2nd convolutional layer

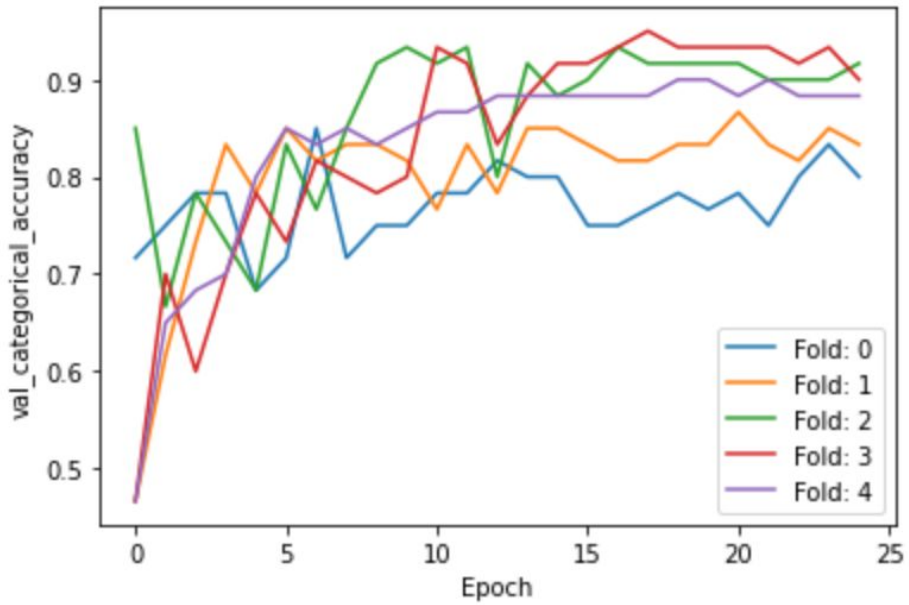
Space for further optimisation: architecture

Dataset split and processing

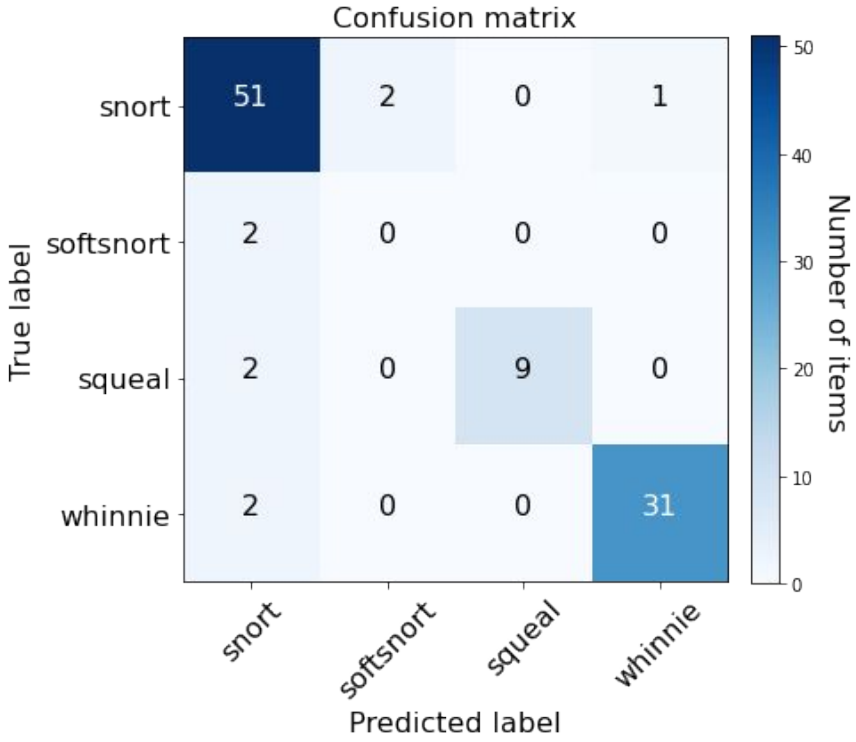
	Training	**Validation**	**Test**
Balanced	+	-	-
Augumented	+	-	-
Implementation	Optimisation, K-fold		Final prediction

Training time: ~10 minutes, CPU

Results: 4 classes, ‘Good’ labels

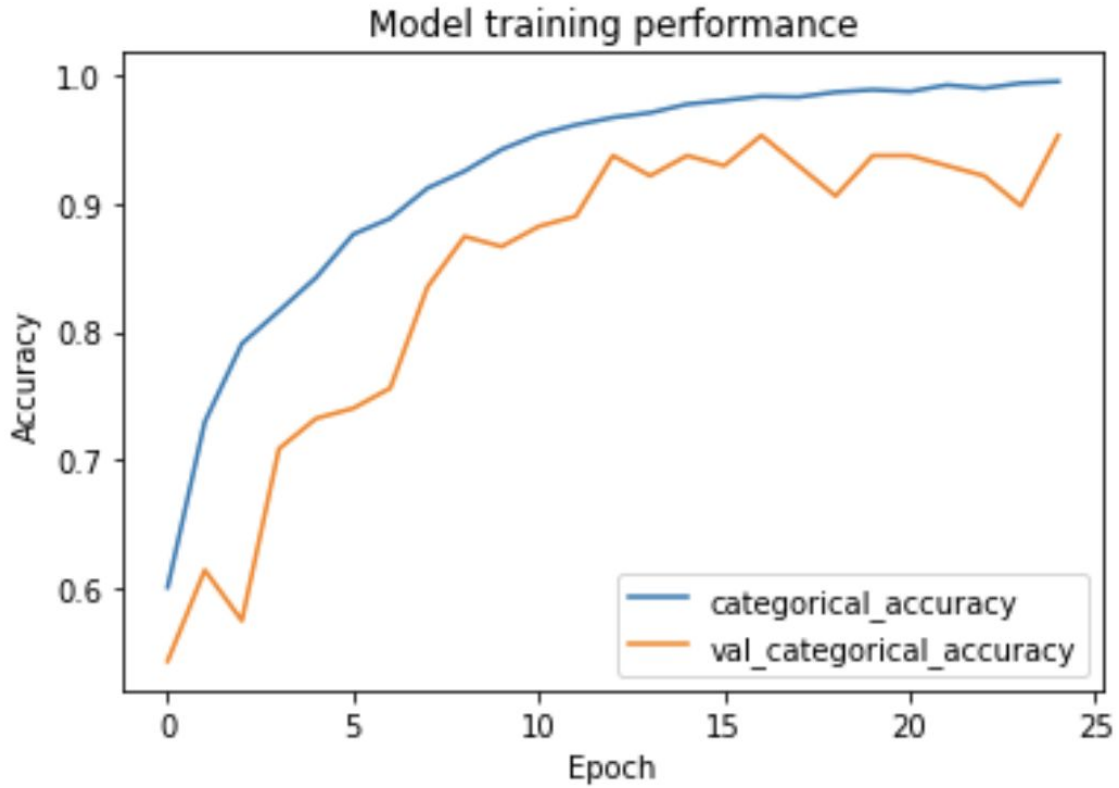


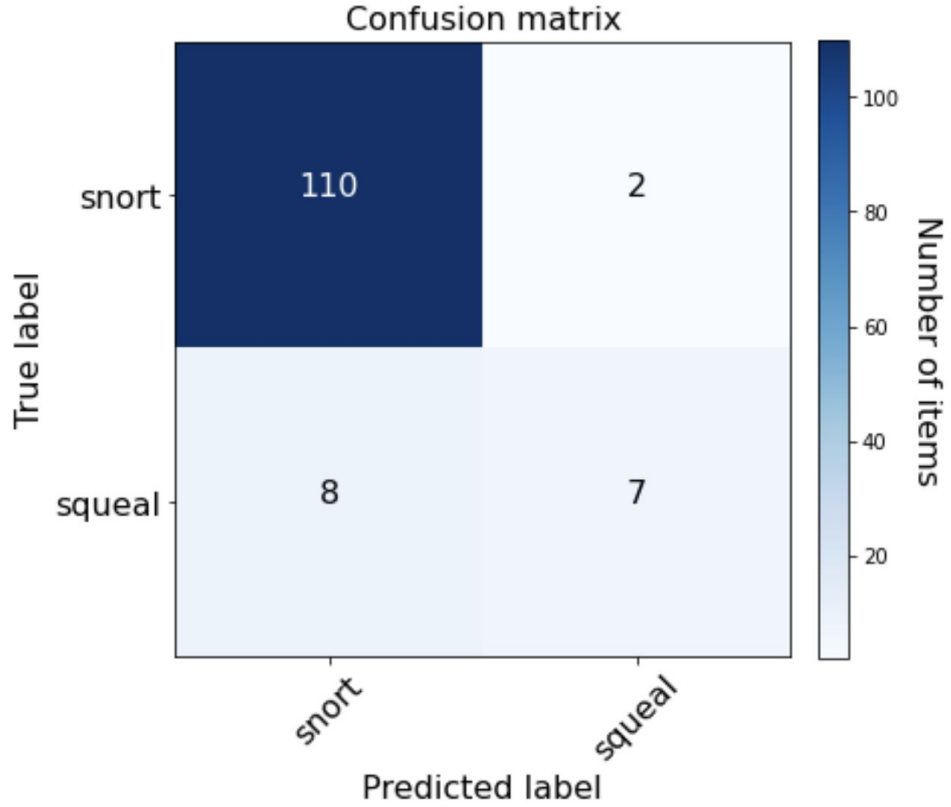
Mean val_categorical_accuracy: 0.86667



accuracy=0.910; misclass=0.090

Results: 2 classes, ‘Bad’ labels





`val_categorical_accuracy: 0.9527559280395508`

accuracy=0.921; misclass=0.079

Autoencoders

Aim and model

A1: Reconstruct calls

A2: Clustering

A3: Look into cNN:

- Model structure was set to the same with cNN
-

Hyperparameter optimization:

- Optimizer (Adam & SGD), Learning rate and weight_decay were tuned by *optuna*

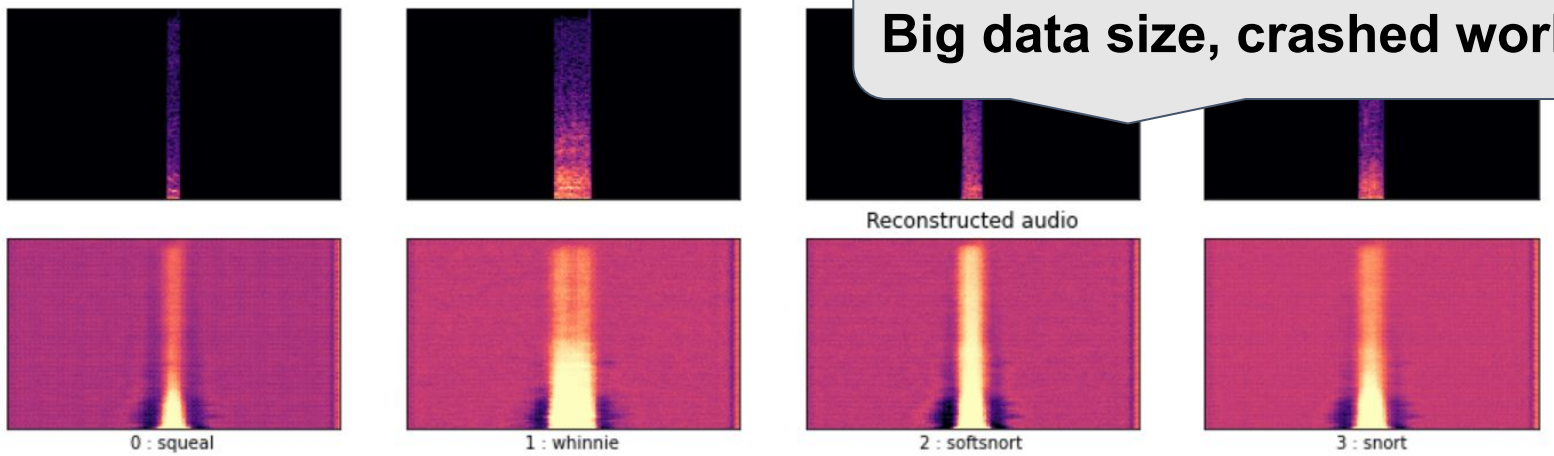
Workplace: On google colab

Results

Round 1:

Input

(413, 1025, 250)

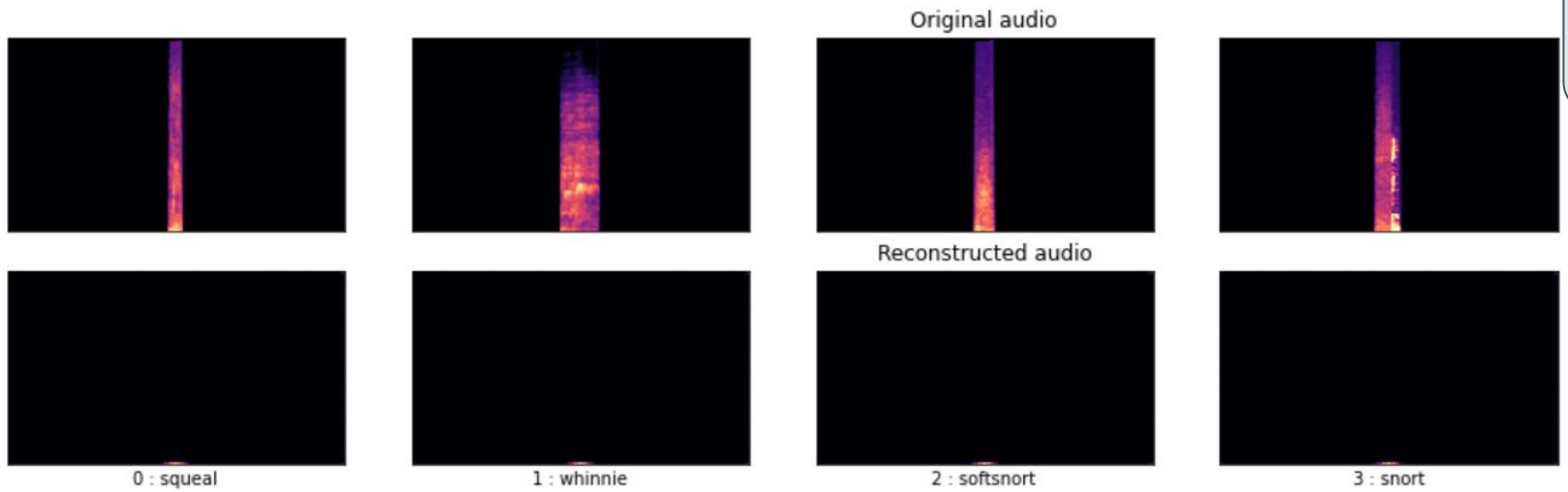


Too long paddings, unscaled
Big data size, crashed workplace

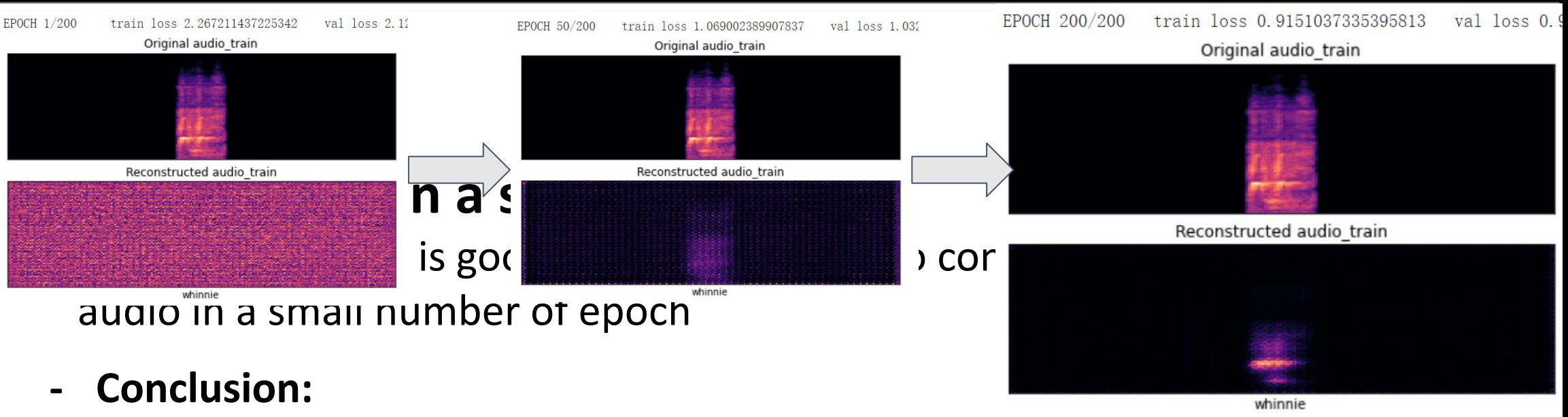
Round 2:

Input

(400, 128, 234)



???



R

n a s

is goc › cor

audio in a small number of epoch

- **Conclusion:**
  1. the structure of the model (cNN) learns in a way to **capture the most outstanding patterns**, which is enough to classify different call types.
  2. But it **loses details**, which makes it failed to reconstruct a call.

Upcoming:

- **Tune the model structure.**

Results

Upcoming:

- ~~**Tune the model structure.**~~

! Train on one sample may lead to false minimum loss.

Explore larger data size

- Augmented zebra dataset (1600, 128, 234)

- Try MNIST Dataset on our model structure.
  60,000 training images and 10,000 testing images

Summary

Results summary

! Considering the small dataset, it's hard to give any convincing conclusion

Answers to key questions:

Number of call types: **4**

The best model for classification: **HP tuned RF, important variables, upsampling tabular data (99%)**

But maybe

Number of call types: **3 (soft snort is snort)**

All other models say no to soft snort + clustering say no to soft snort

Results summary

Answers to key questions:

How clusterings help: **Argue for soft snort; sub clusters for the snort**

Autoencoders: **A tough task. Failed so far, but we know where to go in the next step.**

Next step

! Larger and more balanced dataset

Tabular data:

- Classifications: extract more features
- Clusterings: look into sub clusters of snort

Audio data:

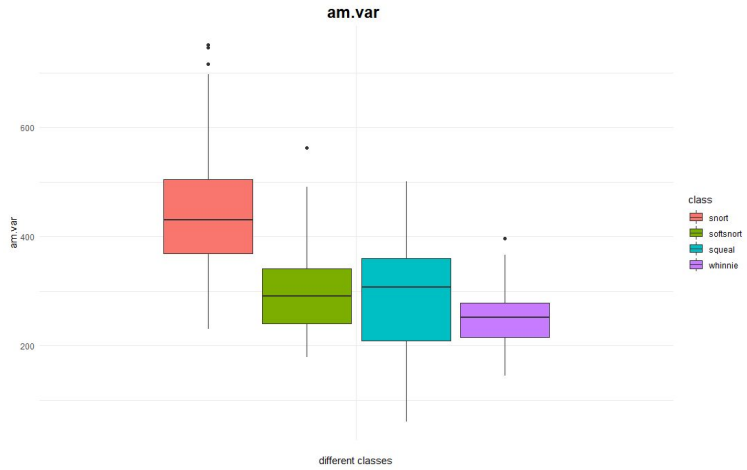
- Classifications: Increase data resolution, RNN
- Autoencoders: many things to do
- Clusterings: other methods

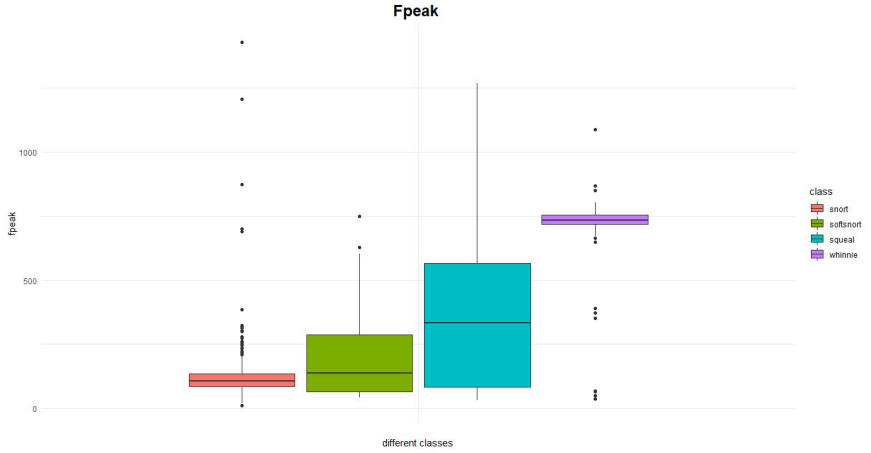
Contributions

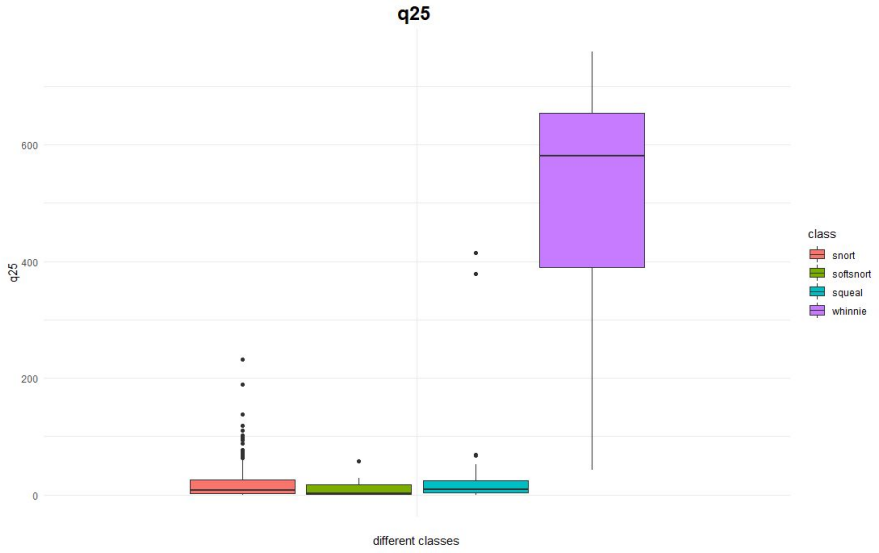
- Matheus worked on tabular data classification
- Keith worked on tabular data classification
- Harry worked on tabular data clustering
- Barnaby worked on audio data preprocessing and cNN
- Aleksandra worked on cNN
- Bing provided audio and tabular data, and worked on Autoencoders.

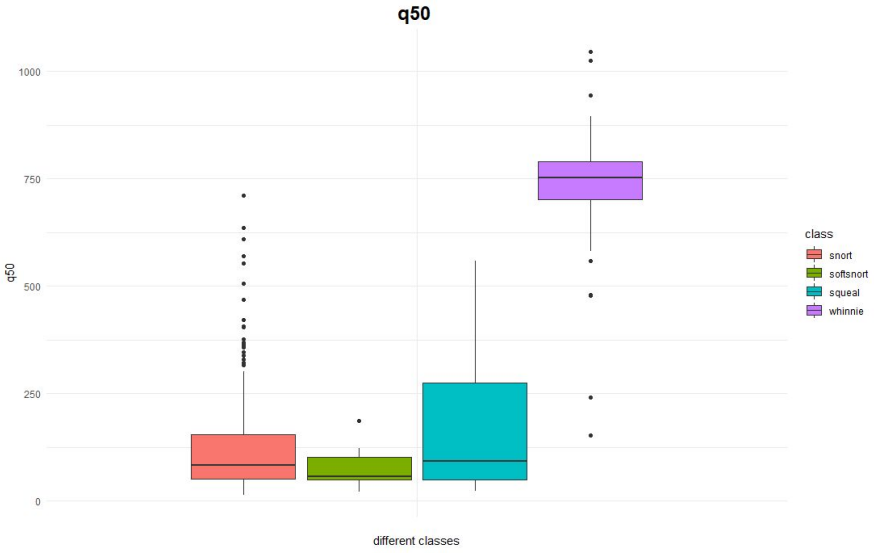
Appendix

Exploring the data: relevant variables

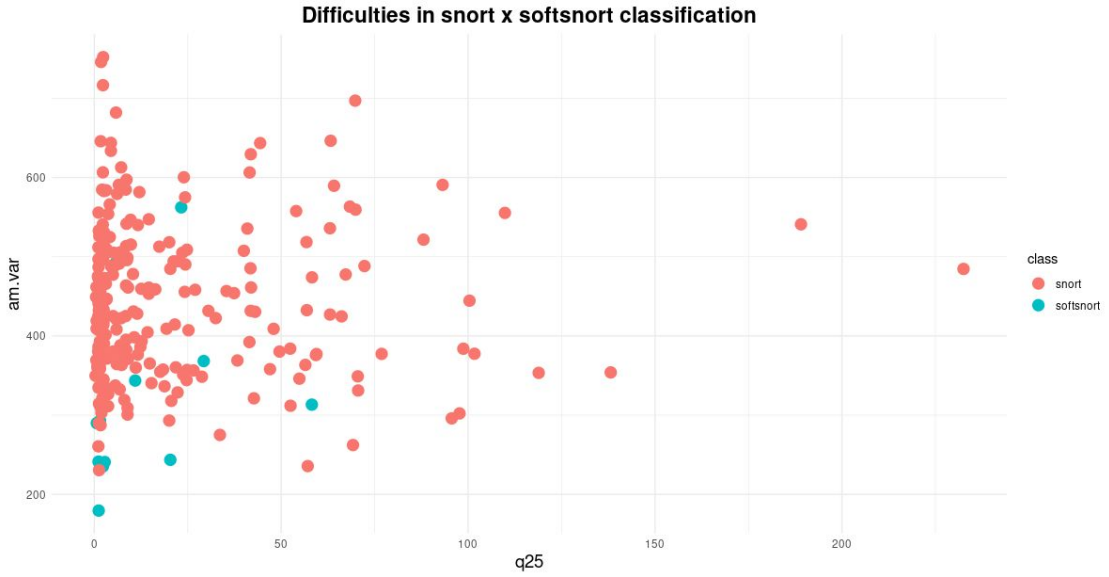


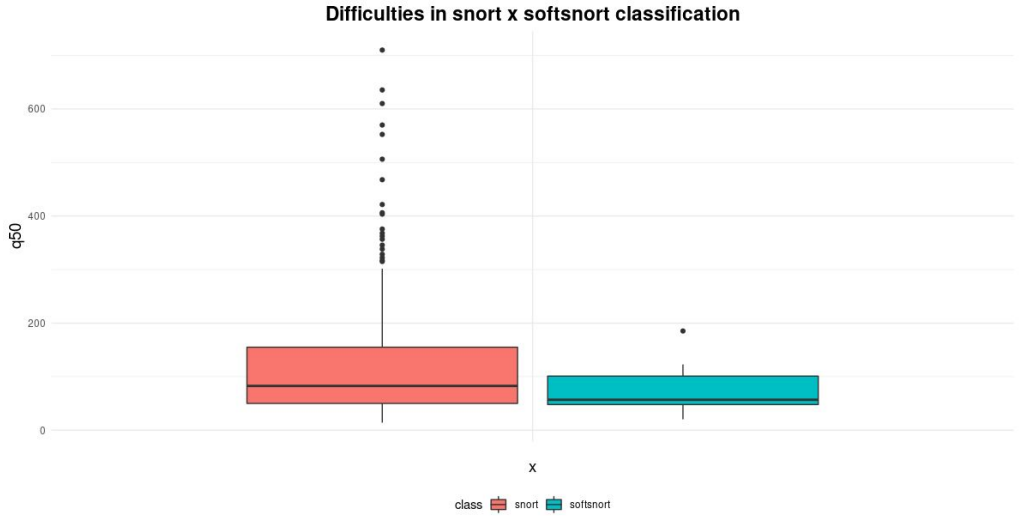




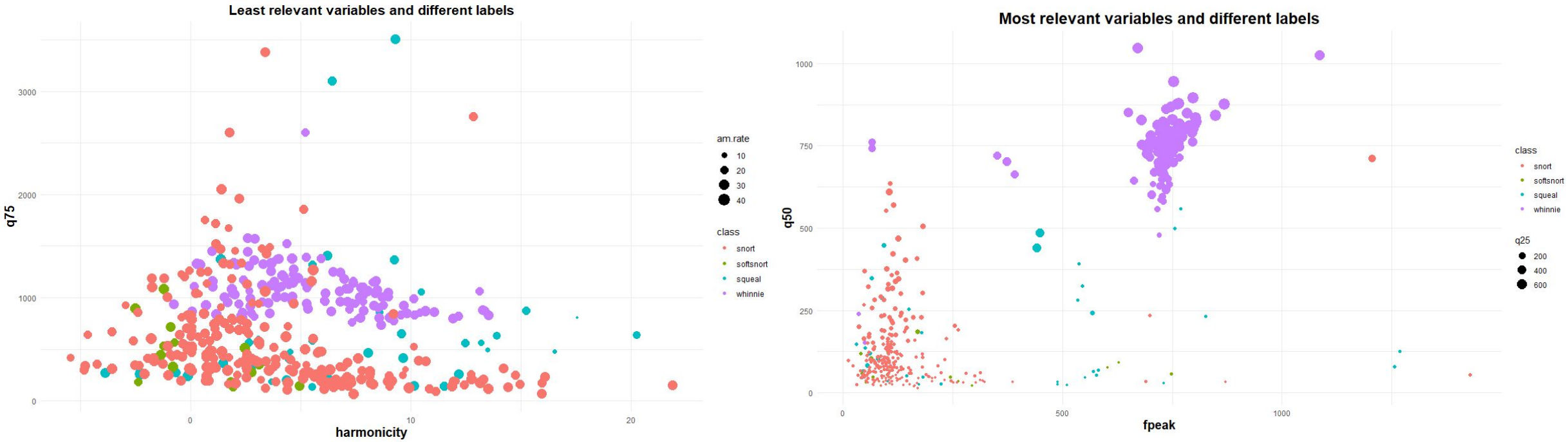


Exploring the data: difficulties on snort classification





Exploring zebra sounds: most relevant x least relevant vars



Grid Search

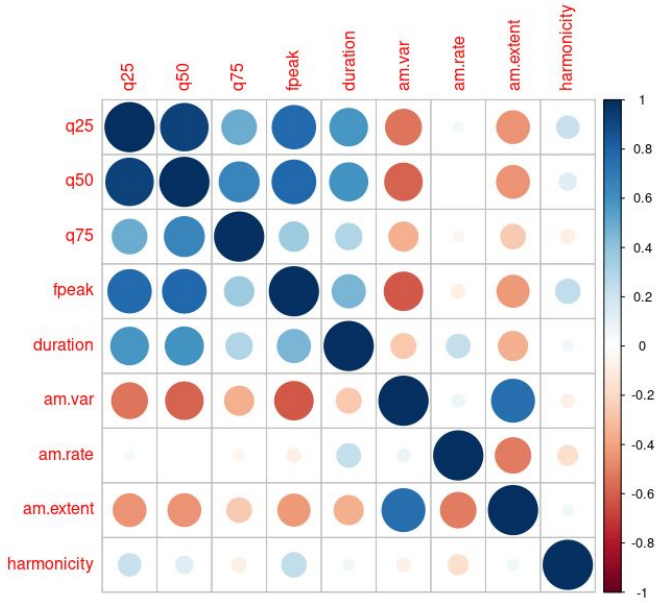
```
n_estimators = list(range(1,51))
min_samples_split =  list(range(15,25))
max_depth = list(range(2, 6))

params_grid = {
    'n_estimators' : n_estimators,
    'min_samples_split': min_samples_split,
    'max_depth': max_depth
}

grid_model = RandomForestClassifier(random_state=42)

GridSearch = GridSearchCV(grid_model,
                          params_grid,
                          cv=5,
                          return_train_score=True,
                          refit=True,
                          verbose=5
                         )
GridSearch.fit(X_train, y_train)
GridSearch_results = pd.DataFrame(GridSearch.cv_results_)
```

```
print(f"The optimal hyperparameters are: {GridSearch.best_estimator_}")
```

```
The optimal hyperparameters are: RandomForestClassifier(max_depth=4, min_samples_split=17, n_estimators=28,
                       random_state=42)
```

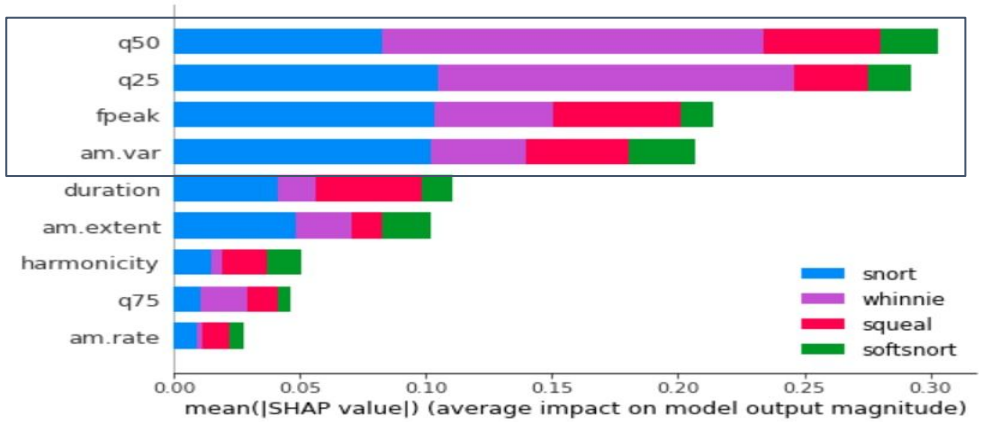
Some important decisions: variable selection

Why? Avoid overfitting and highly correlated features



How:

SHAP values



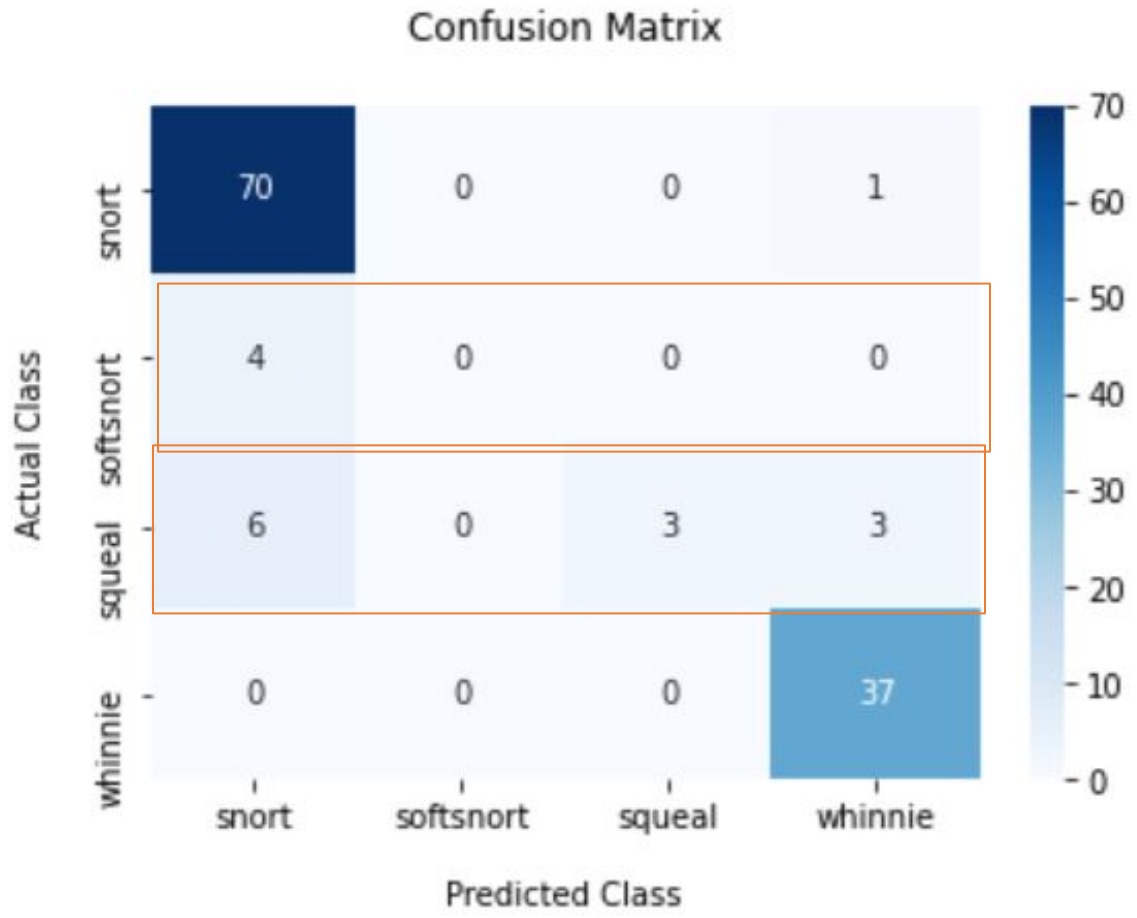
RFE

RandomForest

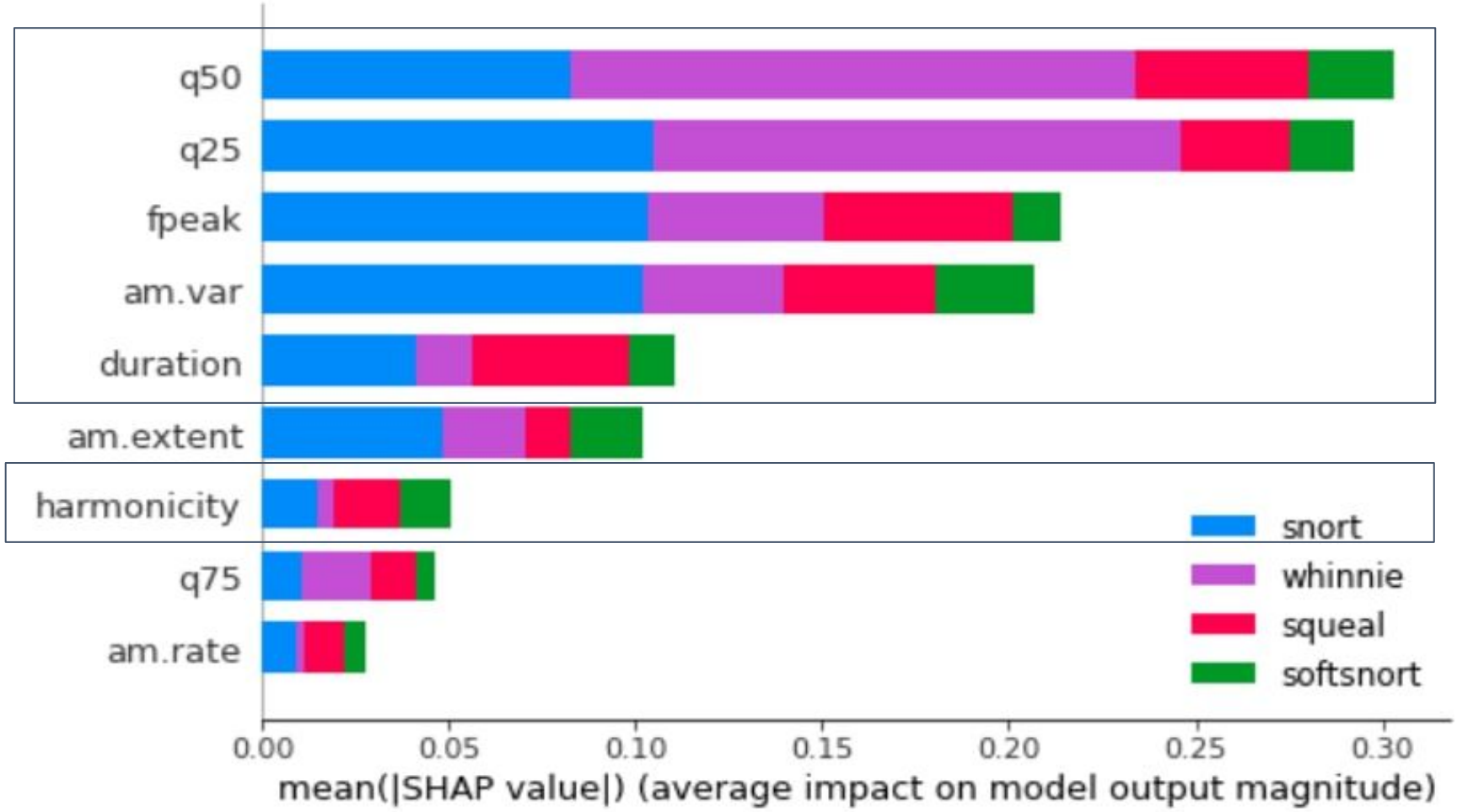
Gradient Boosting

Relevant ones:
q25, q50, fpeak

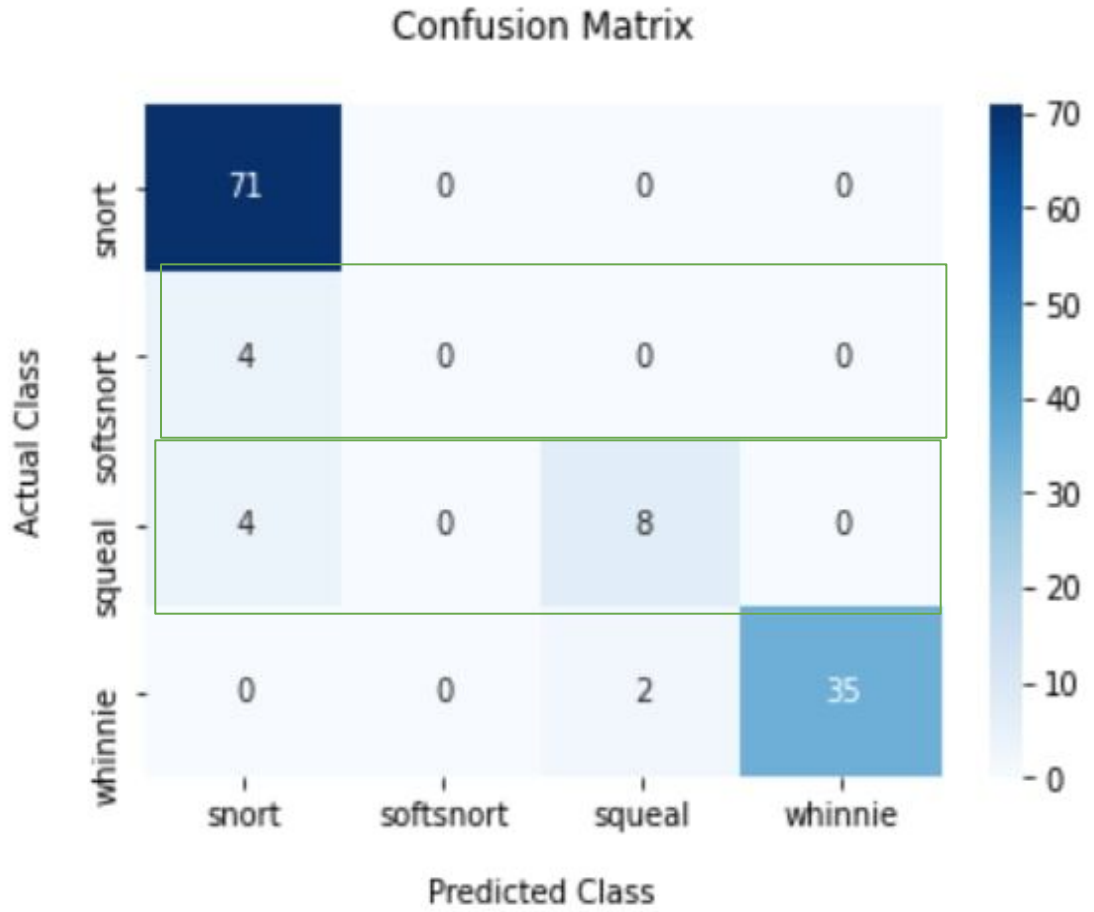
Confusion matrix



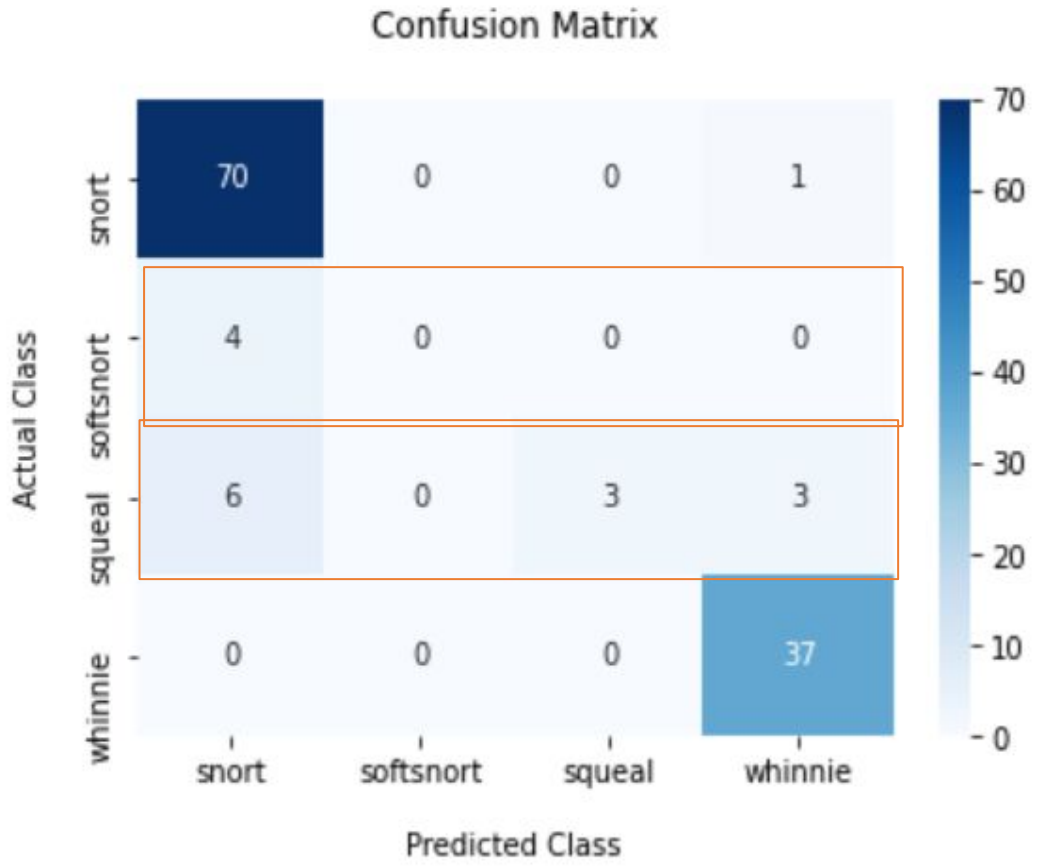
SHAP values


q50
q25
fpeak
am.var
duration
am.extent
harmonicity
q75
am.rate
snort
whinnie
squeal
softsnort
0.00
0.05
0.10
0.15
0.20
0.25
0.30
mean(|SHAP value|) (average impact on model output magnitude)

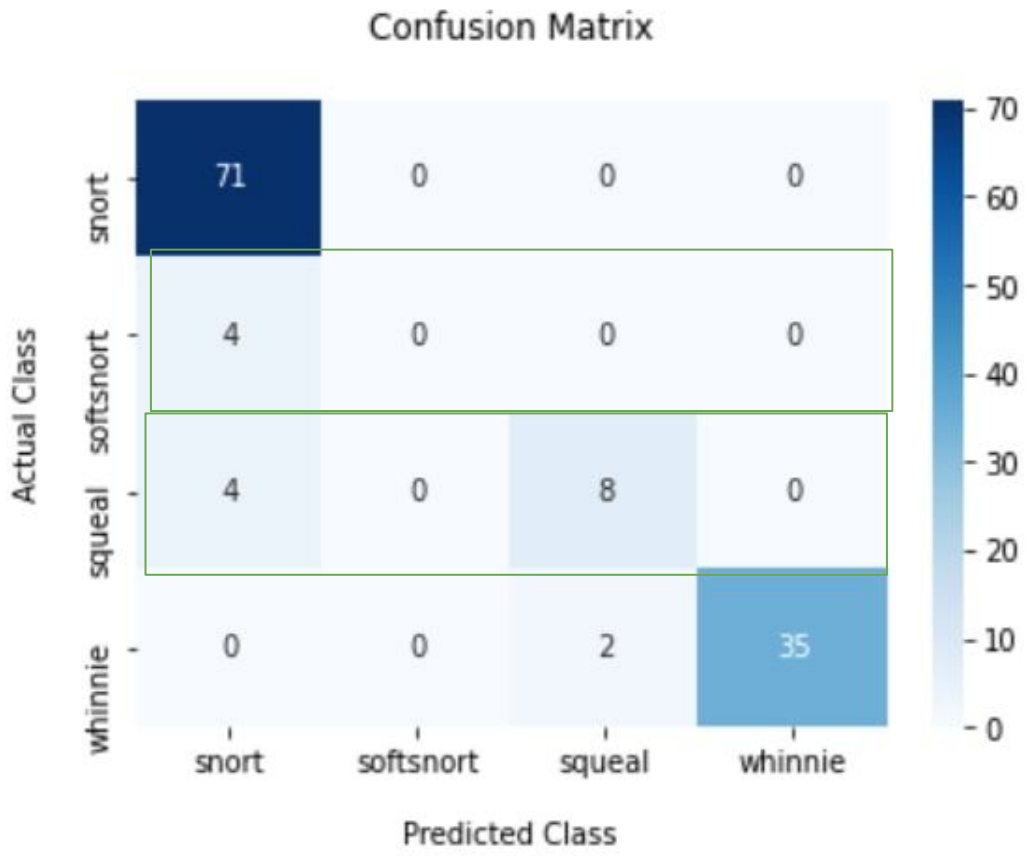
Confusion matrix



Confusion matrix comparison



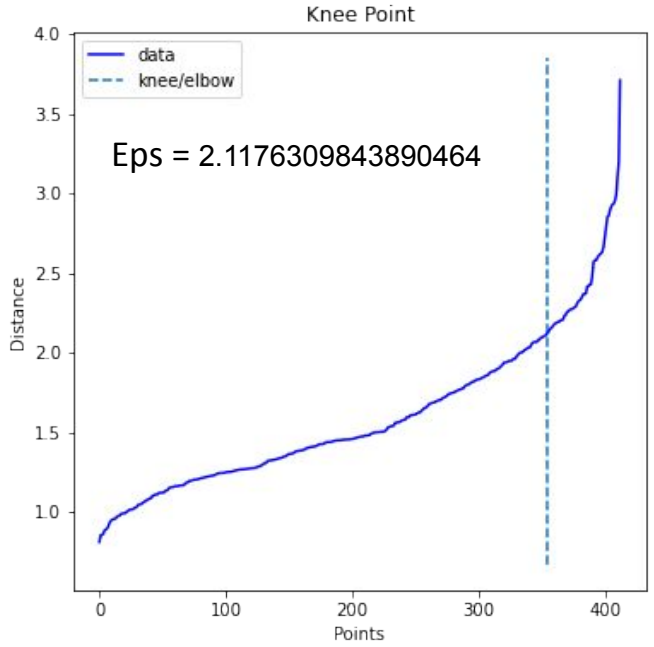
Before

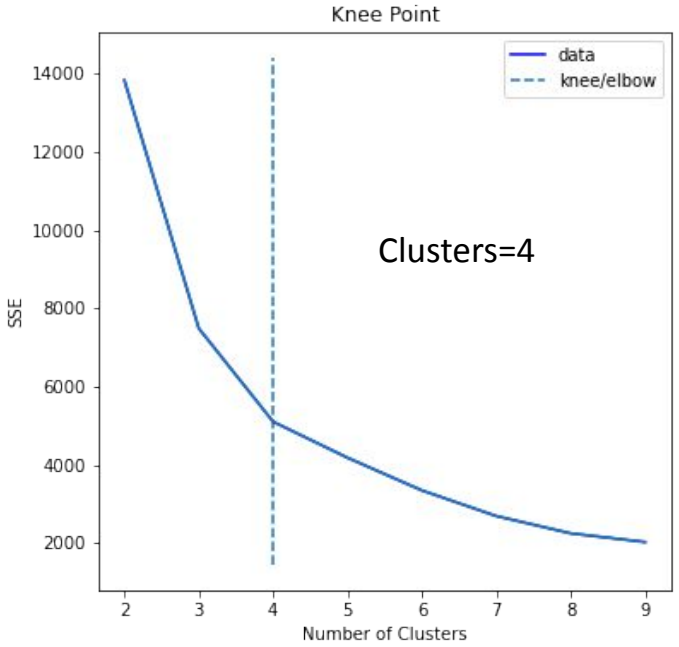


After

Clustering Tabulated Data

- Cleaning: Features from SHAP values selected and MinMax Scaling used
- TSNE, PCA and UMAP were all tried and tested, PCA was the clearest
- Many clustering methods tried and ignored such as Affinity Clustering





Optuna hyperparameters search space (4 classes)

	N epoch	Learning rate	Filters 1&2		Kernels 1&2		Activation 1&2	Pool size 1&2	Dropout rate	1st Dense layer units
Low	15	1e-4	8	16	6	4	1) Rectified Linear Unit, 2) Hyperbolic Tangent, 3) Scaled Exponential Linear Unit, 4) Exponential Linear Units	2	0.3	70
Hight	40	1e-3	30	60	12	12		4	0.6	126
Step	5	Random float	2	2	2	2		1	Random float	4
Best trial number	**25**	**0.000835**	**30**	**26**	**8**	**4**	**1st: ReLU 2nd: SeLU**	**1st: 2 2nd: 3**	**0.404**	**94**

Audio Pre-processing Appendix

EBI (Even Better If)

- Use a GPU to do more optimising/training
- Increase resolution of spectrograms
- Have more data
- Normalise as final step only
- Don’t use volume shift augmentation
- When recording, use multiple microphones (real life augmentation)

Lessons Learnt

- Start with low resolution then increase IF NEEDED.
- No point augmenting with volume if data is normalised.
- Don’t use compound scaling (numerical errors may develop).
- Label files more consistently.
- Starting again is sometimes faster than adapting pre-existing code.
- Don’t use augmentations of train data as validation/test data (just in case).

Dud Methods Pre-processing

Adding noise to the whole sample is much faster than adding it to just the section of the sample with audio so we swapped to that.

We didn’t use dB originally

Streamline the loading method so that it can be done easily and stored for later

The labels were originally assigned from the tabulated data, bu then we swapped to using the filenames and generating a set of labels from them.

We originally had multiples of the augmentations, but reduced them to one of each type to save space.

Some augmentations may actually be zero because they are assigned randomly (so that the NN doesn’t start detecting augmentations)

The non zebra classification would have run the same way as the binary classification but with different labels. We would have randomly chopped up the long audiofile (from which the samples are taken) and used that as non-zebra samples. The chance of a zebra being wrongly labeled as not azebra was probably sufficiently low for this to be a quick and easy method.

It would have been better to not use notebooks for formatting data as the cells (or me) often got confused.

Many of the processing functions were written by me but may exist already (for example i found one for padding), using these would probably be better optimised and easier to keep track of. (although i learned a lot about python writing them) (the audio loading functions are in barney_functions).

It would be much better if all the audio files were the same length and had background noise rather than padding. (a la bad labels)

CNN thoughts

We could have built the optimiser to test different architectures (number convs or dense layers for example).

It might have been a good idea to build two networks. One for longer calls and one for shorter calls, this would mean we wouldn't have to waste time convolving mostly empty images.

A function that that split and augmented only the train data would have been very convenient.

We could have built in a voting system that could use the tabulated data as well, as the two different methods probably pick up on different and hopefully complementary features.

The validation accuracy curves seemed to be not very smooth compared to the training data. This might have been because the validation sets were too small… more data required or augmentation of the validation set? (For cross validation we could have used less folds to increased the size of the validation set, at the cost of training data)

Our metric for validation might not have been ideal for the unbalanced data set, maybe f1 would have been more appropriate as cases where the less popular class were ignored weren't necessarily fairly penalized.

Without a more powerful computer it is hard to know if the reduction in resolution of the data is the cause of the lower accuracy than tabulated data or wether the model itself is worse.

We never tried using pretrained model like resnet, maybe this would have helped.

Other ML libraries like pytorch might run faster than tensorflow?

When optimising, the optimiser often pruned the trial within the first couple of epochs, maybe giving it more patience (somehow?) would have allowed us to find a better set of hyperparameters. (At the cost of time).

The final model was taken from the last fold of the cross validation. It might have been more efficient to retrain the model using the train and validation set before testing it on the test set. (To squeak out just a little more data in the evaluation)

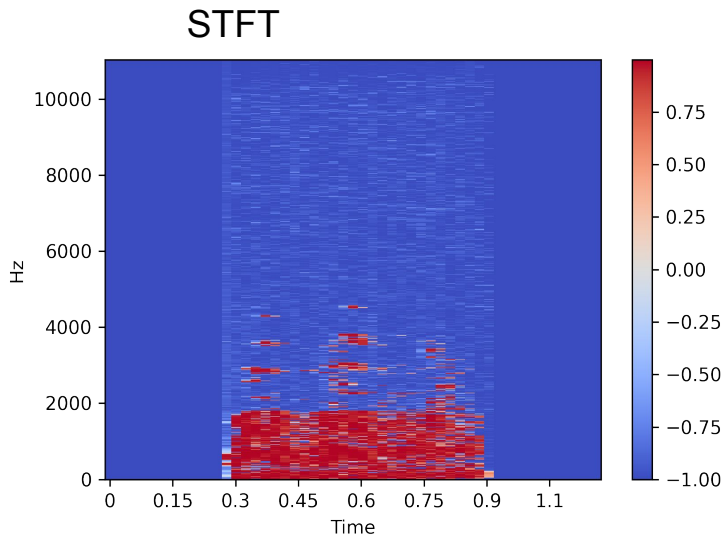
The 'bad' labels model did better than the 'good' labels indication that the quality of the data was relatively sufficient. QUANTITY IS KING.

Example dataset reduction methods

- Halve the sample rate (10kHz max frequency)
- Remove longest files
- Reduce time shift range/padding
- Add noise to padding and the sound (speeds up preprocessing)
- Use compressed files to send dataset to CNN
- Use MEL spectrogram (1/10th Hz resolution compared to STFT)

Appendix

- We halved the original sample rate to reduce dataset size
- We had to be check test,train,val contained all classes
- STFT has higher default resolution.



CNN Notes

- tf.keras. model
- Optuna optimisation
- Kfolding (sklearn)
- Confusion matrix

Binary Model Structure

- **Input** (n_samples, n_rows = 128, n_cols = 51, n_channels = 1)
- **Cov2D** (n_filters = 18, kernel_size = 16)
- **MaxPooling2D** (pool_size = 2)
- **Conv2D** (42,8)
- **MaxPooling2D** (2)
- **Dropout** (rate = 0.15)
- **Flatten**
- **Dense** (Activation = 'relu', n_neurons = 31)
- **Dense** (Output) ('softmax', 2)

n_epochs = 25

Batch_size = 384

Adam(learning_rate = 0.000350)

Loss = categorical_crossentropy

Metric = categorical_accuracy

Train data shape:
(7016, 128, 51, 1)
Validation data shape:
(127, 128, 51, 1)
Test data shape:
(127, 128, 51, 1)

Train Accuracy:
0.9947
Validation Accuracy:
0.9528
Test Accuracy
0.921
(overfitting?)

Autoencoders on augmented dataset

Explore larger data size

- Augmented dataset (1600, 128, 234)
- No improvement so far