Practical Assignment

Omar Abdel Haq, Omar Siddiqui
omar_abdelhaq@college.harvard.edu, osiddiqui@college.harvard.edu

April 6, 2023

1 Part A: Feature Engineering, Baseline Models

1.1 Approach

We will first explain the primary differences between the amplitude and Mel Spectogram data, then use that to motivate our feature engineering process, and describe how our models predict output probabilities. The primary difference between the amplitude and Mel Spectogram data lies in their dimensionality and the amount of audio feature data encoded in each point. Mel Spectogram data has information on audio-related features at each time window, whereas amplitude data only focuses on amplitudes. Amplitude data is one dimensional; therefore, computation can be performed much quicker on it. Mel data on the other hand is two dimensional. The Mel spectogram data requires rigorous signal prepossessing which also requires significant computational power. The advantage of using spectogram data is that it better captures the smaller nuances within the data, which may not be represented by the amplitude; however, it is very computationally expensive. Amplitude data is nicer to work with computationally without any further feature engineering because of its simplicity, but as a byproduct it fails to capture some nuances in the data and more subtle spectral patterns (also making it less robust to noise).

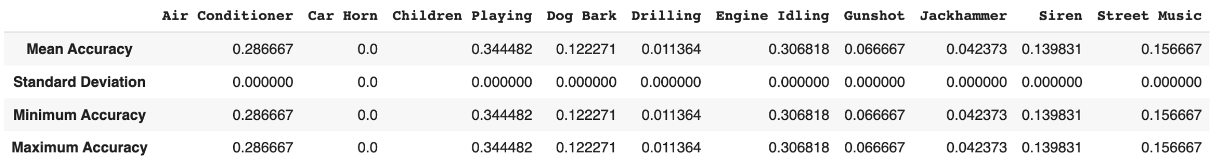
In order to get a better sense of the source of variability within our data, we trained and tested our models on 5 different seeds. What we ended up seeing was no variation in the model predictions, which makes sense because we use standard gradient descent – with no source of randomness included. Due to the complex nature of Mel data, we decided to use PCA in order to reduce our dimensions (and run the code faster). In our implementation of PCA, the first couple lines use the `StandardScalar` class to standardize the input data to have a mean of 0 and a standard deviation of 1. We do this because the underlying assumption of PCA is that the data is normally distributed; additionally, PCA is sensitive to the variance of variables so if we do not standardize, variables with higher variance will have an out-sized effect on our PCA. Then, the remaining lines use the PCA class to perform PCA, where the parameter is set to 0.95 - this means the algorithm will keep enough PCA components to explain 95% of variance in the data (in this case the first 568 components of a total of $11,136$).

The model we trained outputs probabilities for each class using the softmax function for multi-class logistic regression. For each observation, the model will output a probability distribution vector over all the sound types, then select the highest probability as the observations class. We also add a regularization term to penalize large weights, prevent overfitting, and diminish unnecessary features. The LogisticRegression method which we used in sklearn essentially works by transforming input data using a linear model, then passes that result through the softmax activation function to get a probability distribution amongst classes. During training, the model will learn the relationship between features and classes to be used in the linear model, finding the best balance of weights using gradient descent on the negative log likelihood. The linear model essentially tells us the strength of the relationship between a point and a class, and then we use the softmax activation to convert that strength score to probability for each of the classes.

1.2 Results

Multiclass Logistic Regression trained on Amplitude Data:

– Mean Training Set Accuracy: 97.731%
– Mean Testing Set Accuracy: 17.888%%

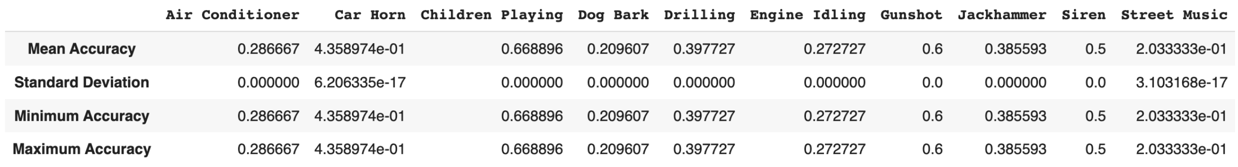


	Air Conditioner	Car Horn	Children Playing	Dog Bark	Drilling	Engine Idling	Gunshot	Jackhammer	Siren	Street Music
Mean Accuracy	0.286667	0.0	0.344482	0.122271	0.011364	0.306818	0.066667	0.042373	0.139831	0.156667
Standard Deviation	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
Minimum Accuracy	0.286667	0.0	0.344482	0.122271	0.011364	0.306818	0.066667	0.042373	0.139831	0.156667
Maximum Accuracy	0.286667	0.0	0.344482	0.122271	0.011364	0.306818	0.066667	0.042373	0.139831	0.156667

Figure 1: Per-Class Accuracy of Amplitude Trained Data

Multiclass Logistic Regression trained on Mel Spectogram Data:

– Mean Training Set Accuracy: 75.923%
– Mean Testing Set Accuracy: 37.142%



	Air Conditioner	Car Horn	Children Playing	Dog Bark	Drilling	Engine Idling	Gunshot	Jackhammer	Siren	Street Music
Mean Accuracy	0.286667	4.358974e-01	0.668896	0.209607	0.397727	0.272727	0.6	0.385593	0.5	2.033333e-01
Standard Deviation	0.000000	6.206335e-17	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.0	3.103168e-17
Minimum Accuracy	0.286667	4.358974e-01	0.668896	0.209607	0.397727	0.272727	0.6	0.385593	0.5	2.033333e-01
Maximum Accuracy	0.286667	4.358974e-01	0.668896	0.209607	0.397727	0.272727	0.6	0.385593	0.5	2.033333e-01

Figure 2: Per-Class Accuracy of Mel Spectrogram Trained Data

1.3 Discussion

The Mel Spectogram performed better on the test set than the amplitude data did. We hypothesize this is the case because as previously explained, the Mel Spectogram data is more rich, and has more information encoded at each time window than just the amplitude. This is likely why we see a marked improvement for the classes: drilling, gunshot, jackhammer and car horn. We hypothesize that these sounds have similar amplitudes, which is why the model trained on amplitude data is so poor on delineating, but the Mel Spectogram trained model is able to capture more nuanced differences rather than just amplitude. Both the Amplitude and Mel Spectogram models have high training accuracies (indicating over-fitting).

These accuracies also lead us to believe that while amplitude is not a discerning feature between some classes, other audio related features at each time window are more indicative of class, which is evidenced further by the overall stronger performance across nearly every class by the Mel Spectogram trained model. Our intuition for why overall these two logistic classification models do not perform super well relative to the test data is because of the limitations of classification techniques that rely heavily on linear relationships. Perhaps the relationships between the audio data features and classes are not linear, which is why a model like a neural network might perform better, due to its ability to approximate non-linear functions according to the universal function approximation theorem.

2 Part B: More Modeling

2.1 First Step

2.1.1 Approach

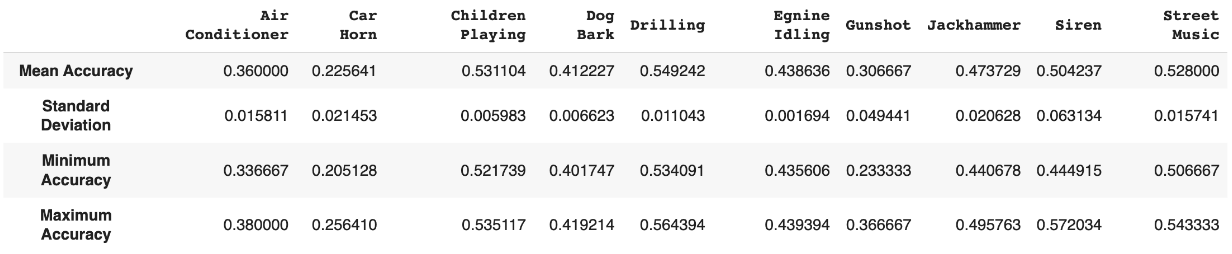
For this section, and all sections hereafter, we chose to use the Mel Spectogram data, since we saw from the previous section that it performed significantly better with the logistic regression model (and since, intuitively, we know it has been feature-engineered for us to better capture the nuances of sound data, and so making it easier to discern different sound-types).

The nonlinear model we chose to implement was a random forest. A random forest creates a pre-specified number of decision trees that try to classify the data at hand. Each decision tree creates cuts in the domain spaces of our features, iteratively creating smaller and smaller groupings of data points, and then labelling each group by its majority classification (a single cut is made such that the resulting data split has the greatest possible homogeneity within each group). In the case of the forest, the final predicted class is the class predicted by the majority of the trees.

For hyperparameter values, we chose 100 as the number of trees in our forest since that seemed like a large number of trees without the tied in expense of increased computational time. Similarly, we chose 10 as the maximum depth of each tree, since that would at most create approximately $2^{10} = 1024$ cuts – allowing for larger leaf nodes while still preserving a good level of granularity.

2.1.2 Results

– Mean Training Set Accuracy: 94.918%
– Mean Testing Set Accuracy: 46.846%



	Air Conditioner	Car Horn	Children Playing	Dog Bark	Drilling	Egnine Idling	Gunshot	Jackhammer	Siren	Street Music
Mean Accuracy	0.360000	0.225641	0.531104	0.412227	0.549242	0.438636	0.306667	0.473729	0.504237	0.528000
Standard Deviation	0.015811	0.021453	0.005983	0.006623	0.011043	0.001694	0.049441	0.020628	0.063134	0.015741
Minimum Accuracy	0.336667	0.205128	0.521739	0.401747	0.534091	0.435606	0.233333	0.440678	0.444915	0.506667
Maximum Accuracy	0.380000	0.256410	0.535117	0.419214	0.564394	0.439394	0.366667	0.495763	0.572034	0.543333

Figure 3: Per-Class Accuracy of Random Forest Model

2.1.3 Discussion

Random forests are inherently random, since if there is more than one way to slice the data at a particular stage (due to both cuts resulting in the same level of homogeneity in the resulting groupings), a tree will chose at random which cut to make. As a result, we see that the classification accuracies for each class have non-zero standard deviations – this is in contrast to our logistic classification models, which implemented non-random, iterative gradient descent (with no main avenue for randomness).

However, despite paying the price of increased model randomness, we see that on average, a random forest performs significantly better than logistic regression, both in terms of general and per-class accuracy. This is a clear indicater that the trend in the data is non-linear, and so a non-linear model is better capable of capturing the nuance in the dataset than simple logistic regression.

2.2 Hyperparameter Tuning and Validation

2.2.1 Approach

kNN-Classifier Hyperparameter Tuning: For our KNN classifier, we tuned one parameter which is the number of neighbors to search over. We implemented hyperparameter tuning using `GridSearchCV`, and searched for the optimal number of neighbors over the range of 1 to 30. `GridSearchCV`, searches over all the parameters in the range, testing each one. We decided upon this range after testing numbers larger than 30 appeared to underfit the data and output small accuracy for both test and train data. We found the best number of neighbors to be 5, after running 10 fold cross-validation.

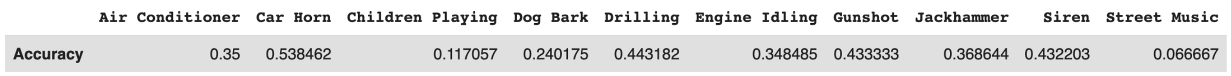
Random Forest Hyperparameter Tuning: Since a random forest involves more hyperparameters we could tune than a kNN Classifier (namely the number of trees in the ensemble and the depth of each tree), we decided to find the best hyperparameters by cross-validation. We implmented hyperparameter tuning using `GridSearchCV`, and although our code ran, it took 2.5 hours to do so and gave us similar results to a `RandomizedSearchCV` implementation we settled upon. The difference between these two methods comes from the fact that `GridSearchCV` tries all hyperparameter combinations it is given (which becomes computational taxing really quickly given two or more hyperparameters to search for), while `RandomizedSearchCV` randomly samples from distributions given for the hyperparameters – not checking every possible combination but trying out quite a few distinct ones.

We supplied the `RandomizedSearchCV` with discrete uniform distributions for both the number of trees in the ensemble and the depth of trees, and ran our code with 5-fold cross-validation. The best combination of estimators and depth turned out to be 185 trees and a maximum depth of 16, both higher than what we initially conjectured would be good values.

2.2.2 Results

kNN-Classifier Hyperparameter Tuning

– Mean Training Set Accuracy: 74.320%
– Mean Testing Set Accuracy: 34.957%

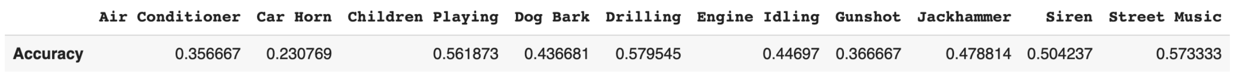


	Air Conditioner	Car Horn	Children Playing	Dog Bark	Drilling	Engine Idling	Gunshot	Jackhammer	Siren	Street Music
Accuracy	0.35	0.538462	0.117057	0.240175	0.443182	0.348485	0.433333	0.368644	0.432203	0.066667

Figure 4: Per-Class Accuracy of Tuned kNN Model

Random Forest Hyperparameter Tuning

– Mean Training Set Accuracy: 99.928%
– Mean Testing Set Accuracy: 47.246%



	Air Conditioner	Car Horn	Children Playing	Dog Bark	Drilling	Engine Idling	Gunshot	Jackhammer	Siren	Street Music
Accuracy	0.356667	0.230769	0.561873	0.436681	0.579545	0.44697	0.366667	0.478814	0.504237	0.573333

Figure 5: Per-Class Accuracy of Tuned Random Forest Model

2.2.3 Discussion

As expected, the two models with tuned hyperparameters performed better than the models with naively-chosen hyperparameter values. This is because although our guesses for what the hyperparameters should be seem well-reasoned, they are conjectures. Cross validation allows us to find the best hyperparameter values by checking what the best hyperparameter values are for an untouched subset of the training data. Without cross-validation, the preferred hyperparameter for any model would be the one that overfits the most, which isn't likely to result in a model that generalizes well. In contrast, doing 5-cross validation allowed us to find hyperparameter values that were likelier to generalize well to the test data, and they did! These tuned models also perform surprisingly well on our under-represented classes, despite obvious class-imbalance: per class accuracies for `Car Horn` and `Gunshot` are on par with the other class accuracies in both models.

3 Final Write-up and Reflections

3.1 Discussion:

Model Name	**Train Acc.**	**Test Acc.**	**Random Variability?**
Logistic Classifier (Amplitude)	97.731%	17.888%	×
Logistic Classifier (Mel Spectogram)	75.923%	37.142%	×
(Naive) Random Forest Classifier	94.918%	46.846%	✓
(Naive) kNN Classifier	71.385%	33.500%	✓
(Tuned) Random Forest Classifier	99.928%	**47.246**%	✓
(Tuned) kNN Classifier	74.320%	34.957%	✓
Convolutional Neural Network	84.837%	**52.799**%	✓

Key Components

- **Data Pipeline:** We conducted a few steps in our data pipeline stage. Firstly we converted our test and train data into dataframes. We also flattened the Mel spectogram data so we could run our logistic regression and our random forest and KNN models. Lastly, we ran principal component analysis (PCA) to reduce the dimesionality of our data, specifically of the Mel data, choosing to select the components that account for 95% of the data. This made our models execute much faster.

- **Model Selection:** For our baseline models, we used two logistic regression models, one trained on the amplitude data and the other trained on the flattened Mel data that on which we ran PCA; for non-linear models, we decided to use KNN and random forests; finally, we then realized that because we were working with 2D data, a CNN would make a good model. We noticed that all the models that were ran on the Mel data performed better than the amplitude models, and also that non-linearized classification models performed better on testing than the logistical baselines.

- **Model Tuning:** Throughout the model training process we made sure to tune our hyperparameters to optimize the performance of our model with respect to accuracy. To do this, we used cross validation on the random forest and KNN, as well as ensemble methods inherently in the random forest. We ran multiple experiments on the CNN, steadily improving the performance by allowing the model to select more optimal weights and modifying architecture.

- **Bias-Variance Trade-off:** For each of our models, we grappled with the bias variance tradeoff: For our logistic baselines, we were sure to include L2 regularization to help with generalization and reduce variance. For KNN and Random forests, we selected hyperparameters that preserved strong train performance, while also generalizing well on the test set, and also verified through K-fold cross validation and ensemble methods that the models generalized well. Finally, to prevent overfitting in the CNN, we made sure to include dropout methods that drop neurons from the net that may make the model prone to over fitting. Our logistic baseline models in particular seemed to generalizae very poorly, and perhaps were overfitting.

- **Evaluation:** We used overall train and test accuracy to evaluate our models – this allowed us to find the best model overall when it came to the total number of correctly classified test points. This however, isn't an ideal way to go about comparing, as some models discard per-class accuracy entirely of low-probability class in favor of overall accuracy (a prime example are our logistic models, where some accuracies are at 0%).

- **Domain-Specific Evaluation:** The data did inform many of the modeling and design decisions we made. In the preprocessing phase, knowing the the Mel data had much more encoded information encouraged us to run PCA. Additionally, because we knew the data was 2D and had a non-linear relationship to the classes, we decided to pursure CNN. Some of our models make intuitive sense, like KNN and random forests, but the CNN is much harder to gain an intuitive interpretability, but given the nature of the data we are working with, using a CNN makes sense intuitively. Lastly, it makes sense that we are optimizing for class accuracy, because it is important that we are identifying the correct sounds, given that some of the noises have serious ramifications like gunshots.
- **Design Review:** We could have done a better job at comparing how our models do on per-class accuracy. Although we looked at/considered per-accuracy in our analysis, we didn't end up making any choices to tailor to the fact that under-represented classes are likelier to be incorrectly labelled. In addition, we could have considered class imbalance further for models like the logistic baselines which performed quite poorly on them - though, we didn't have imbalance for random forest and CNN!
- **Execution & Implementation:** The area to which any of our models is applied to is an important consideration to make when thinking of deploying the models. If, for example, the specific application we are considering involves identifying gunshots across a particular urban center for policing purposes, the fact that we have the fewest samples of gunshots in our data make it an under-represented, and as a result, and often under-performing class when it comes to accuracy. This poses practical, and in the case of issues of over-policing can become an ethical concern as well.

4 Part C: Optional Exploration

4.1 Approach

We decided to implement a Convolutional Neural Network as further exploration. We used Skyler's base code as a guide. Our implementation of CNN does the following:

- Transform the 2-D Mel Spectogram data into the format a CNN takes. This format is that our data set should be of the dimensions (N, C, W, H), where N is the number of data points, C is the number of color-channels (in this case only 1), and W and H are the width and height of the images we have (128×87).
- Apply 2 convolutional layers to the data, which are akin to applying filters to the datapoints to distill the trend separate from the noise. Apply a drop out layer, which randomly disregards some of the nodes used in the network, so as to not to overfit by depending on a particular set of nodes more than all others. The training utilizes Adam, a variation of Stochastic Gradient Descent, which samples points from the domain to find a direction for the descent that is likely to get stuck at a local minimum point.
- We went to office hours about best practices when implementing a CNN, and were told that this specific dataset would likely need a rather large number of epochs during training, mainly due to the high complexity of the data we are dealing with.
- Data cleanup in every epoch. Due to the limited memory size of the GPU we are afforded in Google Colab, we had to implement a step at the end of every epoch that cleared out inter-mediate variables so to not overburden the GPU.
- We tested out different dimensions for the intermediate layers, as well as different learning rates to see what performed best on the training data, and we settled on 1.5×10^{-4} for the learning rate.

4.2 Results

– Mean Training Set Accuracy: 84.837%
– Mean Testing Set Accuracy: 52.799%

Figure 6: CNN Accuracy over the Epochs Used

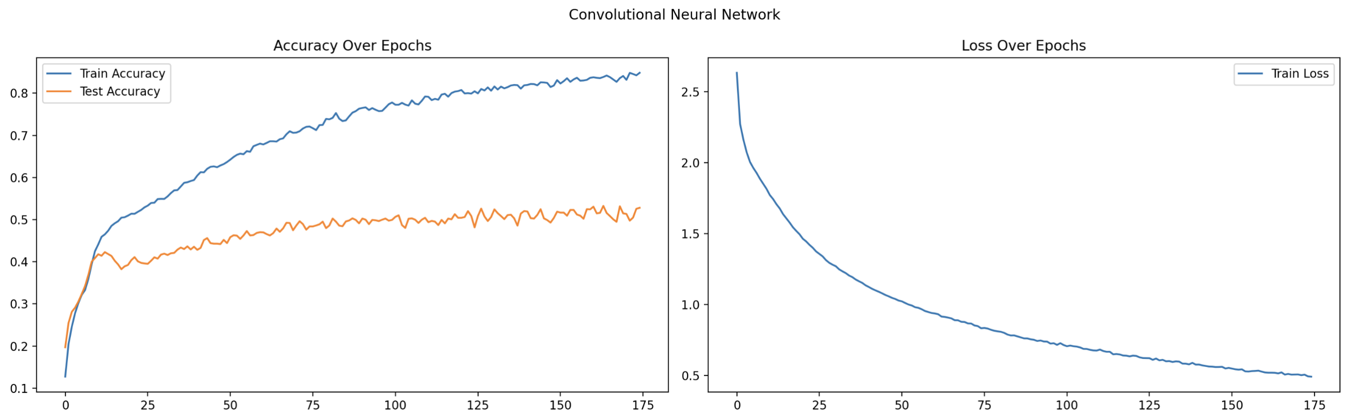
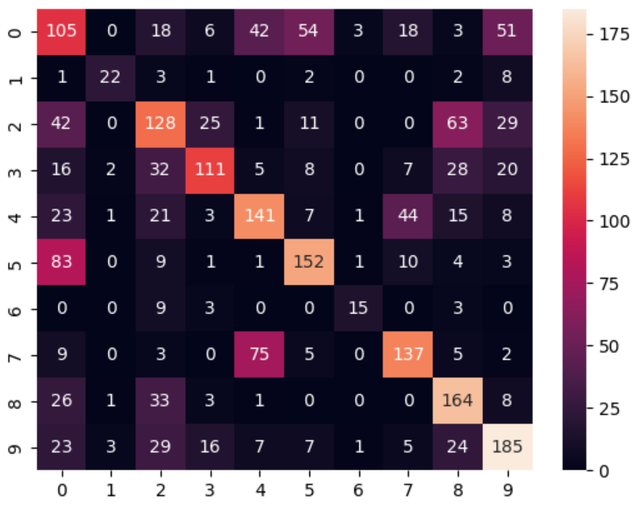


Figure 7: Per-Class Accuracy Confusion Matrix



	0	1	2	3	4	5	6	7	8	9
0	105	0	18	6	42	54	3	18	3	51
1	1	22	3	1	0	2	0	0	2	8
2	42	0	128	25	1	11	0	0	63	29
3	16	2	32	111	5	8	0	7	28	20
4	23	1	21	3	141	7	1	44	15	8
5	83	0	9	1	1	152	1	10	4	3
6	0	0	9	3	0	0	15	0	3	0
7	9	0	3	0	75	5	0	137	5	2
8	26	1	33	3	1	0	0	0	164	8
9	23	3	29	16	7	7	1	5	24	185

4.3 Discussion

We chose to use a CNN model because it would be the first model we explore that is specifically tailored to taking in 2-dimensional data. We also speculate that the trend in the data in non-linear, so a neural network would be a good model to try out given the universal function approximation theorem. After running multiple different combinations of epochs and learning rates, we settled upon $lr = 1.5 \times 10^{-4}$ and 175 epochs, as they maximized the training accuracy while also resulting in a tapered off loss chart. The CNN model got us our highest test accuracy across all models, with 52.799%. We also see that our model performs on par for under-represented classes as for other classes – per class accuracies for `Car Horn` and `Gunshot` (the most under-represented classes) are both around 50%.