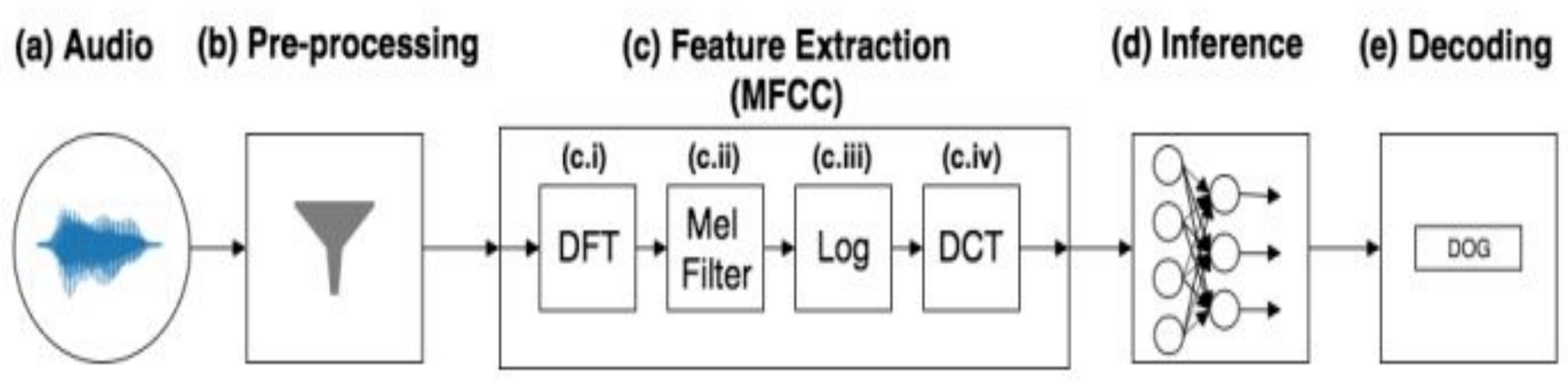
Understanding the “Reasoning” of Automated Voice Processing Systems

André Shannon[1], Nikita Morozov[1], Angela Stefanovska[1], Christ Athiogbey[1], Vi Pham[1], Grant Szajda[1], Oheneba Berko[1]
Dr. Szajda[1], Daniel Capecci[2], Hadi Abdullah[2], Patrick Traynor[2]
[1]University of Richmond, [2]University of Florida

Abstract

Automated Voice Processing Systems (VPSes) are becoming increasingly widespread. We see them not only in voice assistants such as Amazon Alexa, Google Assistant, and Apple Siri, but also in devices that are more user friendly to the visually impared or elderly and in other devices, such as required by the “Internet of Things” where traditional interfaces are inadequate or impractical. These systems are capable of initiating virtually any command that a mobile phone can generate, which makes their security especially important. Recent work has demonstrated a number of attacks against VPSes, most exploiting differences between how humans and deep learning models process speech. Our understanding of these differences, however, is limited, so that we do not understand why some attacks succeed when similar attacks fail. Our research seeks to shed light on how VPSes “reason”, in order to gain a better understanding of the factors they use during classification, thus understanding potential new attacks and defenses.

Voice Processing Pipeline



An audio sample is collected by periodically measuring the amplitude of sound waves. Thus an audio sample is just a sequence of amplitudes. These amplitudes are pre-processed, during which noise is filtered out along with frequencies outside the range of human perception. The resulting sequence values are broken into overlapping time frames (e.g., of duration 20ms), which are then fed, in order, into the feature extraction phase. During the feature extraction phase, spectral (frequency) characteristics of each frame are captured and processed further, in order to better mimic the way in which humans perceive audio (e.g., we distinguish between low frequencies much better than between high). The resulting feature vectors are then fed to a deep recurrent neural network, which outputs a single character for each input frame. The sequence of characters is then decoded and fed to a language model which generates the final word sequence.

Explanation Methods

The behavior of deep neural networks is based on the values of (sometimes billions of) parameters. *Training* a network determines the values of these parameters. Training data (known input-output pairs) is fed through the network, and parameters adjusted until the difference between the resulting network outputs and the true outputs are minimized. Though the mathematics of this optimization process is well understood, the extremely large number of parameters means that, once trained, the “reasoning” behind why the network decides to classify a specific input as it does is relatively opaque. Explanation methods seek to better understand model “reasoning”.

LIME

LIME is an explanation method that, given a specific input, seeks to explain the classification of that input by using a linear approximation of the model behavior near the input. For input **x** with feature vector (x_1, x_2, … , x_n), and model f, such a linear approximation would have the form:

$$f(\mathbf{x}) \approx \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

where the beta coefficients are fixed and determine the linear model. The value of approximating f in this way is that the relative magnitudes of the beta coefficients determine which features of x drive its classification.

Challenges

Applying LIME to VPSes comes with many difficulties. Grouping frequencies into bins makes generating approximations more computationally feasible. There is a tradeoff between having a more accurate model with more bins and time spent computing the model. We tried many different binning methods: fixed numbers of evenly spaced bins, overlapping bins, logarithmically spaced bins, and even basing the bins on peaks in the intensities of the frequencies.

Once approximating linear models were generated, we used R squared (R2) and Mean Standard Error (MSE) to evaluate how well the linear models locally approximated the actual model. R2 compares variance between predicted and actual values to total variance in outputs and MSE is the average of the squared differences between the actual and predicted values. For both of these scoring metrics, our explanations were consistent, but performed less well than we expected. In order to explain this, we ran experiments to determine whether our methods for evaluating the results were sound, and whether the use of linear models was sufficient to capture the behavior of the original classifier, even in small regions of the input space.

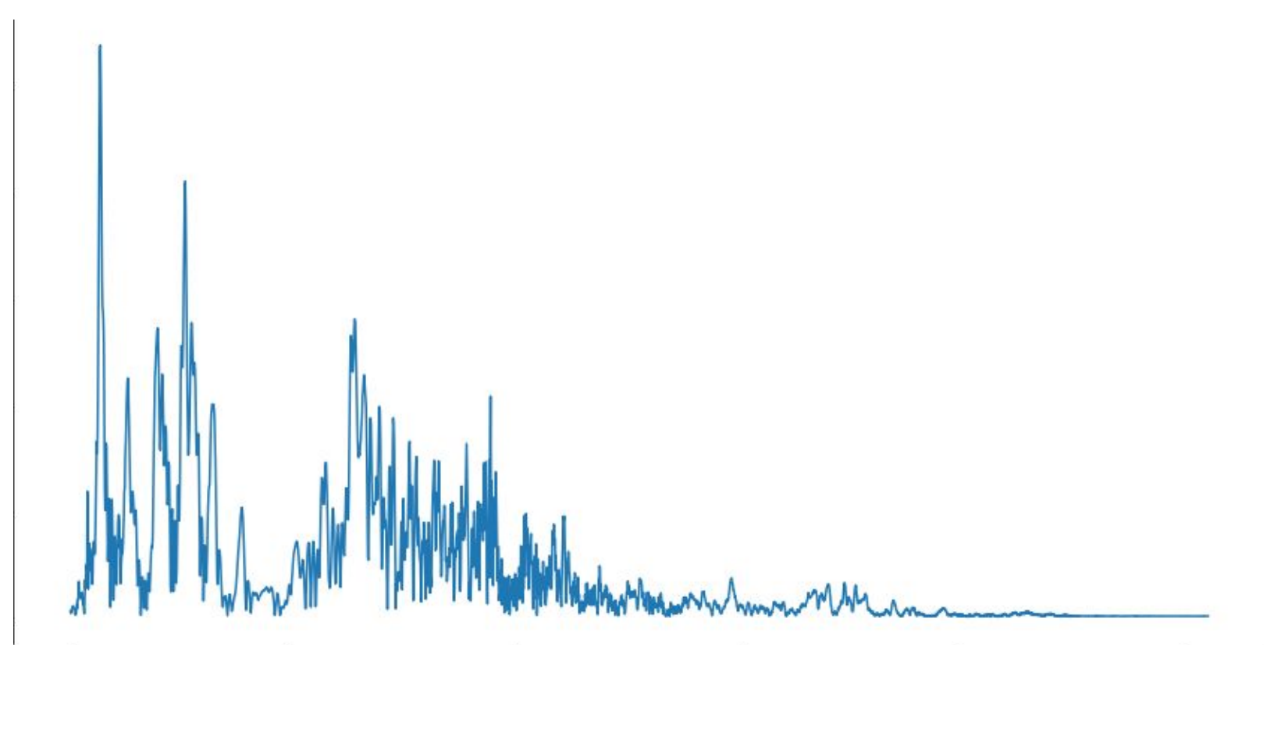
To test the latter, we began implementation of a method that employs a non-linear approximation of the classifier. This new explanation method, LEMNA, uses a Gaussian mixture model to locally approximate the original classifier. LEMNA also better captures feature dependencies by effectively grouping similar coefficients. Unfortunately, we did not finish our LEMNA experiments before summer research ended.

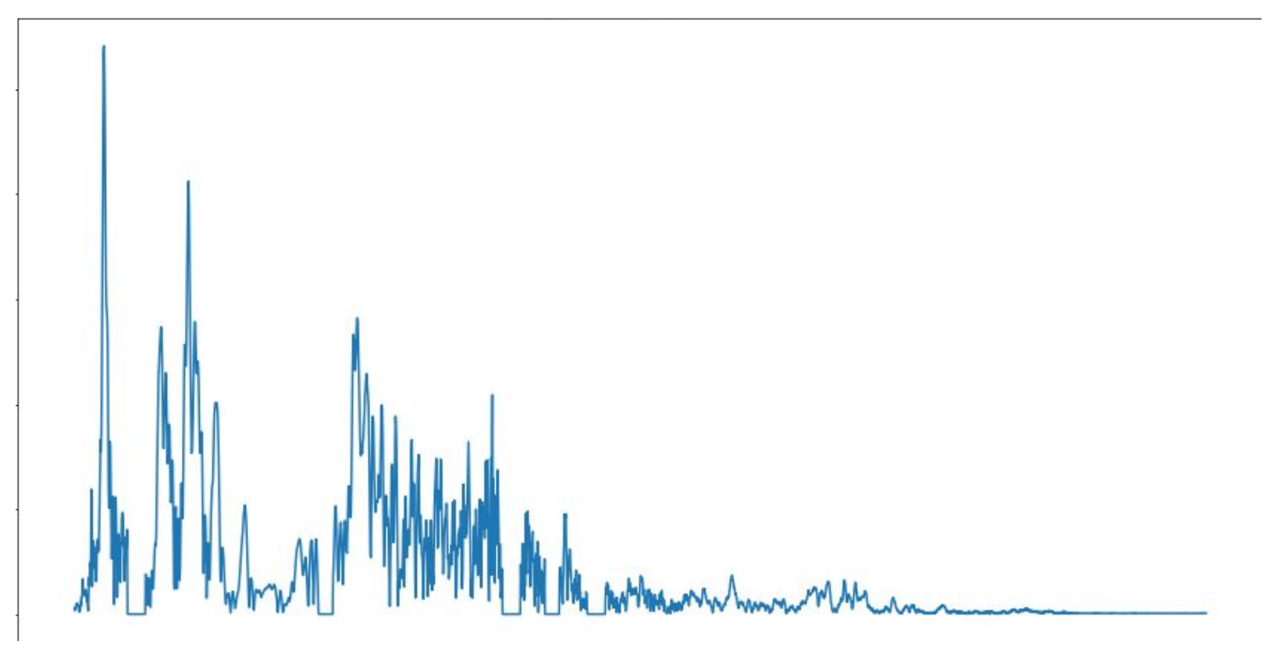
Applying LIME

Though straightforward in theory, the practical application of LIME can be tricky. Several details appropriate to the application domain must be chosen. This can by itself take several months, as potential choices are implemented and tested for efficacy. Once application details are set, LIME generates linear approximations by sampling inputs near the input under consideration. These samples are fed through the original classifier. The samples, along with the outputs generated by the classifier, are used as training data for the approximating linear model.

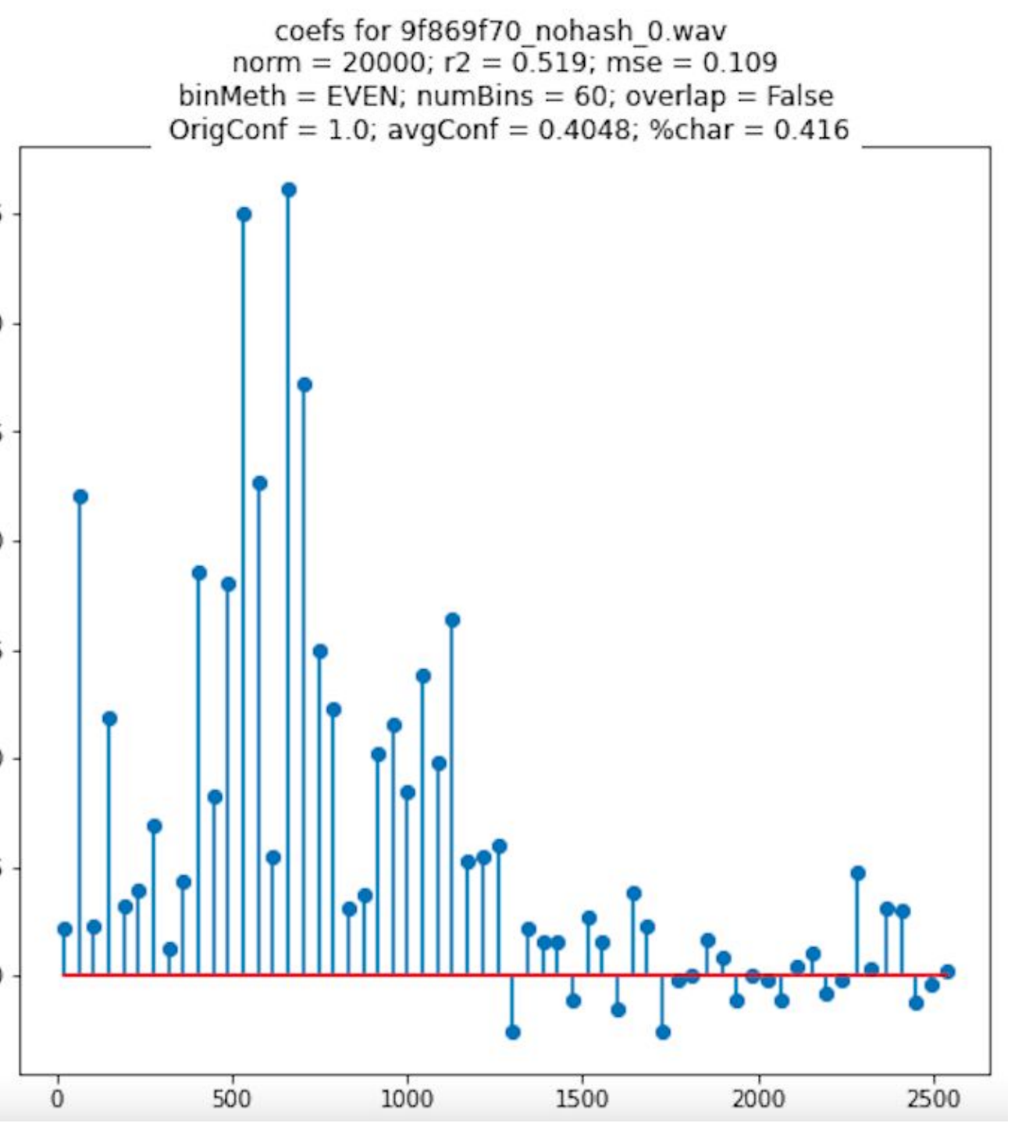
Generating sufficient samples to achieve a reasonable approximation can be difficult, as the input space can have high dimension. To avoid this, we group frequencies together in “bins”.

The intensities of frequencies will look something like the figure at right (top). To create a similar sample slightly different than the original, we set a small, random number of bins to zero. We call this “perturbing” the input. We make around 2000 perturbations, classify them all, and use that data to create the linear model.

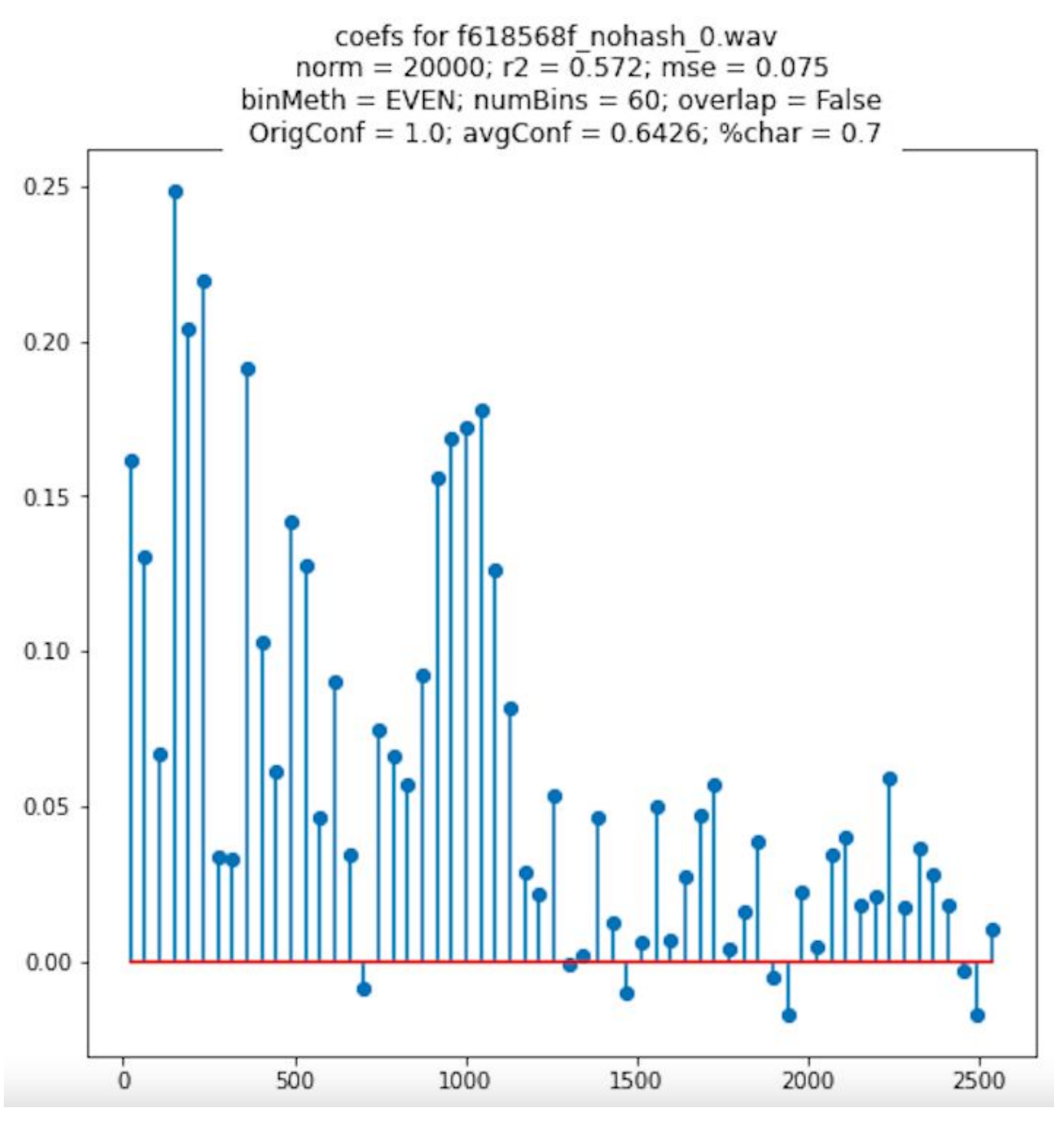




What we Found



For the explanations we did generate, we were surprised at how different they were, even when explaining the same character in the same word but with different people saying the word. We were looking for a single “catalog” entry of the features that cause the classification to be a specific character, but later realized that the characteristics of the feature space are such that there are a multitude of “catalog” entries at the least.



References

Hadi Abdullah, Kevin Warren, Vincent Bindschaedler, Nicolas Papernot, and Patrick Traynor, *SoK: The Faults in our ASRs: An Overview of Attacks against Automatic Speech Recognition and Speaker Identification Systems.* In Proceedings of the 42nd IEEE Symposium on Security and Privacy, May 2021, 730-747.

Wenbo Guo, Dongliang Mu, JunXu, Purui Su, Gang Wang, and Xinyu Xing. *LEMNA: Explaining Deep Learning Based Security Applications.* In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), 364–379.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. *“Why Should I Trust You?”: Explaining the Predictions of Any Classifier.* In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Aug. 2016,1135–1144.