Analyzing Voice Samples to Predict Age

Final Report

Raj Singh

Problem Statement

The project is for a company called Neurolex Labs. Neurolex analyzes voice samples to help diagnose diseases such as schizophrenia, Parkinson's disease, Alzheimer's disease, and depression. In this case, we are analyzing human voice samples to determine the age of the speaker. We may want to know the age range of the individual so we can get specialized care for people in that age range. Moreover, knowing the age of the individual can give us a better context for the nature of their disease, and how likely they are to have a particular disease.

The Data

The name of the dataset is called the Common Voice Dataset by Mozilla. It contains about 200,000 voice samples. It contains the feature we are interested in, which is the age of the speaker. In order to featurize this audio data, we will be using two Python libraries: pyAudioAnalysis and Librosa.

Data Wrangling

The data can be downloaded as a .csv file from the internet, so we can load in the data as a Pandas Dataframe. Not all entries of the dataset were labeled with age categories, so those entries must be deleted. We are using two different models and we need to do two separate data wrangling/featurization processes for each of these models (featurization refers to the process of turning the audio file into a set of numerical features).

Multilayer Perceptron Model Data Wrangling

In order to featurize the data for our multilayer perceptron model, we use a Python library called pyAudioAnalysis. pyAudioAnalysis extracts information about the audio wave at a particular time. Examples of common audio features include mel spectrogram feature coefficients (MFCC's), spectral centroid, spectral contrast, and spectral rolloff. A full reference for audio features that are extracted, along with descriptions of each, can be found here. Since each audio feature is extracted at a particular time, we compute them in intervals and then take the mean, median, maximum, minimum, and standard deviation of all the extracted values. The script extracts 35 audio features, and because we are taking 5 statistics, we have 170 total features. The full code can be found here. 170 is a lot of features, so we use a Support Vector Classifier with L1 regularization to reduce the number of features to approximately 80. This is a common technique in dimensional reduction, which essentially means that we are eliminating features that do not have much predictive value.

Convolutional Neural Network Model Data Wrangling

Convolutional neural networks are most commonly used to classify images. We will discuss these in details later, but first we must manipulate the audio data to produce something that is analogous to the format that image data is represented in. Typically, an image is represented with a 3-dimensional array, with the dimensions being the height, width, and number of channels (color images have 3 channels: red, green, and blue). For example, the value at height 0, width 0, and the green channel would represent "how green" the image is at the bottom left pixel of the image. In this case, we use a Python library called Librosa to convert the audio files into log-mel-spectrograms, which is essentially image data with two channels. The difference is that our dimensions represent time, frequency, and the number of channels (the first channel is the "normal value" and the second channel represents the "delta channel", which takes the derivative of the value along a particular axis). The value represents the amplitude of the wave at a particular time and frequency. Any information representation of this type is called a spectrogram. A common technique in signal processing is to convert spectrograms into log-mel spectrograms, which can be described as performing an initial Fourier transform, mapping the powers of the spectrum onto the mel scale, taking the logs of the powers, then taking the discrete cosine transform of the list of mel log powers. A more thorough description of this can be found here. The full details for the Python featurization code can be found here.

Modeling

We will try two approaches: a Multilayer Perceptron Model as well as a Convolutional Neural Network.

Multilayer Perceptron Model

Our basic architecture for the model was an input layer with nodes equal to the number of features (~80) and an output layer equal to the number of classes (8). There are four hidden layers. The first one has 256 nodes, the second has 1024, the third has 1024, and the fourth has 256. Because there are is a relatively high number of layers, as well as nodes, the neural network can make more complex and accurate predictions. Of course, this comes at the expense of training time and overfitting. This is why we use a GPU provided by Google Colaboratory and dropouts after the first and third layers (to deal with high training times and overfitting, respectively).

We tried two models that were slightly different in nature. Both had dropouts of 0.2 after the first layer, but the second only had a dropout of 0.15 after the third layer. To compensate for the higher dropout in the first model, there were 50 training epochs in the first model and 40 training epochs in the second model. Generally speaking, as we increased the training epochs after this point, the accuracy started to increase by marginal amounts, and the model tended to overfit more. Thus, this MLP architecture optimized a model with a relatively high accuracy and low tendency to overfit.

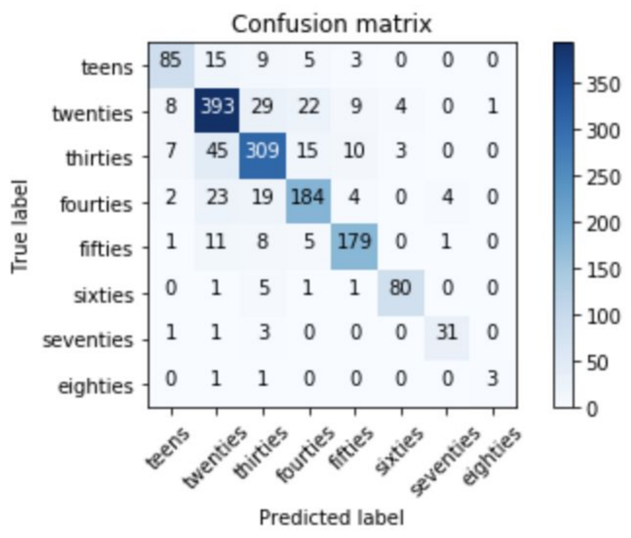
Since there were an uneven number of samples in each age category, we used balanced class weights in our model. That is, we weighted samples that appeared less often higher than samples that appeared more often. The results can be found on the next page.

MLP Model 1 Results

Training Accuracy: 84.5%

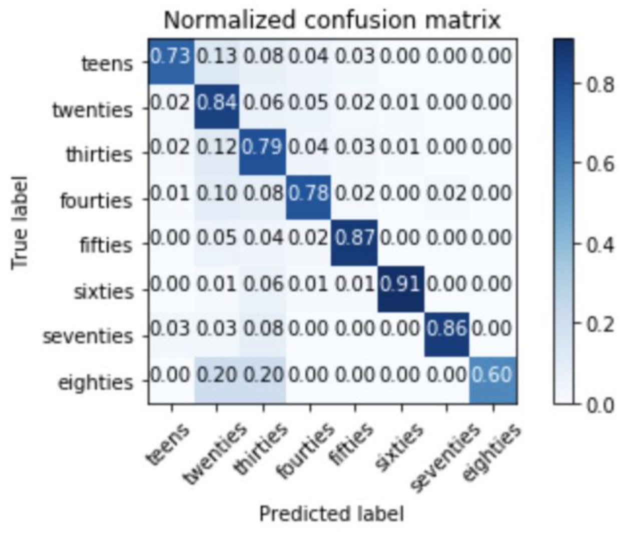
Testing Accuracy: 82.0%

Confusion Matrices:



Confusion matrix

True label \ Predicted label	teens	twenties	thirties	fourties	fifties	sixties	seventies	eighties
teens	85	15	9	5	3	0	0	0
twenties	8	393	29	22	9	4	0	1
thirties	7	45	309	15	10	3	0	0
fourties	2	23	19	184	4	0	4	0
fifties	1	11	8	5	179	0	1	0
sixties	0	1	5	1	1	80	0	0
seventies	1	1	3	0	0	0	31	0
eighties	0	1	1	0	0	0	0	3



Normalized confusion matrix

True label \ Predicted label	teens	twenties	thirties	fourties	fifties	sixties	seventies	eighties
teens	0.73	0.13	0.08	0.04	0.03	0.00	0.00	0.00
twenties	0.02	0.84	0.06	0.05	0.02	0.01	0.00	0.00
thirties	0.02	0.12	0.79	0.04	0.03	0.01	0.00	0.00
fourties	0.01	0.10	0.08	0.78	0.02	0.00	0.02	0.00
fifties	0.00	0.05	0.04	0.02	0.87	0.00	0.00	0.00
sixties	0.00	0.01	0.06	0.01	0.01	0.91	0.00	0.00
seventies	0.03	0.03	0.08	0.00	0.00	0.00	0.86	0.00
eighties	0.00	0.20	0.20	0.00	0.00	0.00	0.00	0.60

Classification Report:

	Precision	**Recall**	**F1-Score**	**# Samples**
Teens	0.82	0.73	0.77	117
Twenties	0.80	0.84	0.82	466
Thirties	0.81	0.79	0.80	389
Fourties	0.79	0.78	0.79	236
Fifties	0.87	0.87	0.87	205
Sixties	0.92	0.91	0.91	88
Seventies	0.86	0.86	0.86	36
Eighties	0.75	0.60	0.67	5
Micro Avg.	0.82	0.82	0.82	1542
Macro Avg.	0.83	0.80	0.81	1542
Weighted Avg.	0.82	0.82	0.82	1542

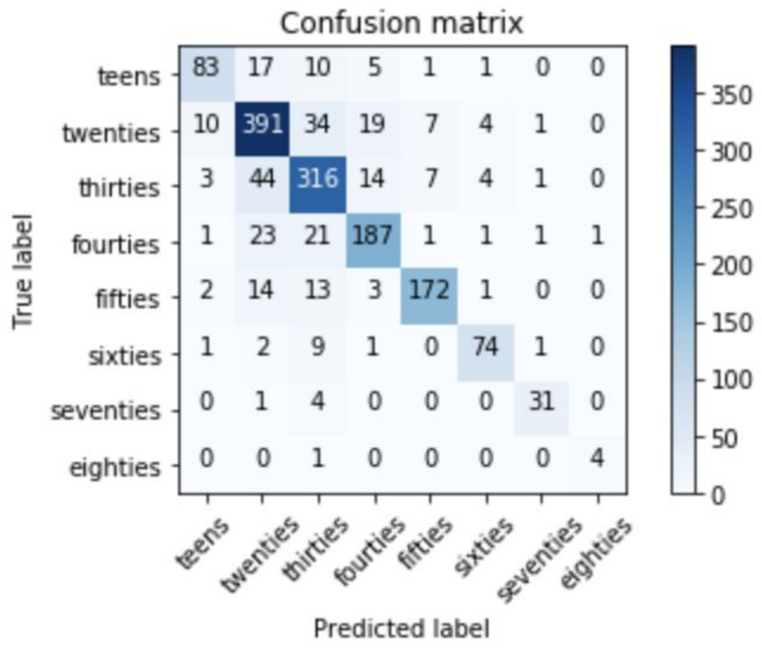
The code for this model can be found here.

MLP Model 2 Results

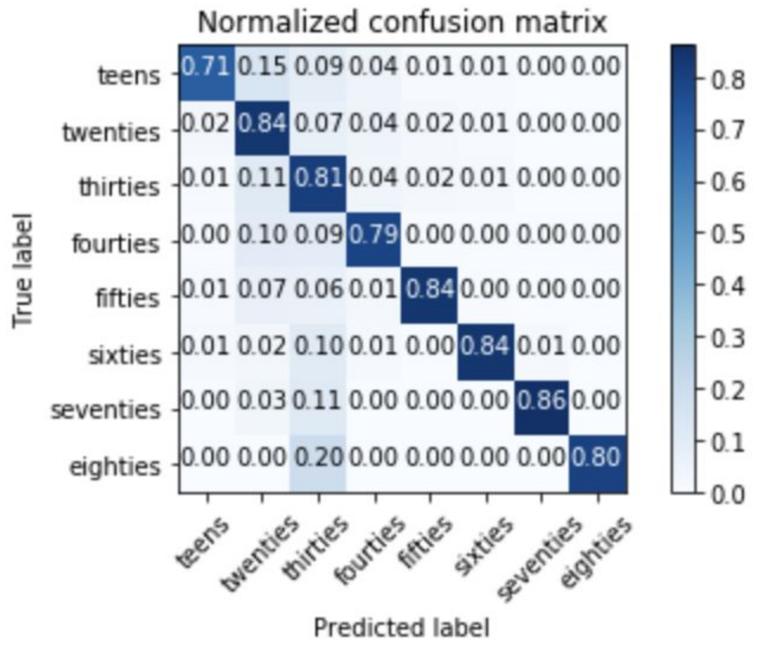
Training Accuracy: 85.1%

Testing Accuracy: 81.6%

Confusion Matrices:



	teens	twenties	thirties	fourties	fifties	sixties	seventies	eighties
teens	83	17	10	5	1	1	0	0
twenties	10	391	34	19	7	4	1	0
thirties	3	44	316	14	7	4	1	0
fourties	1	23	21	187	1	1	1	1
fifties	2	14	13	3	172	1	0	0
sixties	1	2	9	1	0	74	1	0
seventies	0	1	4	0	0	0	31	0
eighties	0	0	1	0	0	0	0	4



	teens	twenties	thirties	fourties	fifties	sixties	seventies	eighties
teens	0.71	0.15	0.09	0.04	0.01	0.01	0.00	0.00
twenties	0.02	0.84	0.07	0.04	0.02	0.01	0.00	0.00
thirties	0.01	0.11	0.81	0.04	0.02	0.01	0.00	0.00
fourties	0.00	0.10	0.09	0.79	0.00	0.00	0.00	0.00
fifties	0.01	0.07	0.06	0.01	0.84	0.00	0.00	0.00
sixties	0.01	0.02	0.10	0.01	0.00	0.84	0.01	0.00
seventies	0.00	0.03	0.11	0.00	0.00	0.00	0.86	0.00
eighties	0.00	0.00	0.20	0.00	0.00	0.00	0.00	0.80

Classification Report:

	Precision	**Recall**	**F1-Score**	**# Samples**
Teens	0.83	0.71	0.76	117
Twenties	0.79	0.84	0.82	466
Thirties	0.77	0.81	0.79	389
Fourties	0.82	0.79	0.80	236
Fifties	0.91	0.84	0.88	205
Sixties	0.87	0.84	0.86	88
Seventies	0.89	0.86	0.87	36
Eighties	0.80	0.80	0.80	5
Micro Avg.	0.82	0.82	0.82	1542
Macro Avg.	0.84	0.81	0.82	1542
Weighted Avg.	0.82	0.82	0.82	1542

We can see that the average precisions, recalls, and f1-scores are very similar between the two models. The second model has a slightly higher accuracy on the test set. We would prefer the second model because it overfit less. Thus, perhaps having more dropout and more epochs works better for this particular problem. There seems to be some variability between the results for those in their eighties, and this is likely because there are so few samples that have those labels. We would likely benefit from more training data in this particular category. The code for this model can be found here.

Convolutional Neural Network Model

Featurization of the audio files into log-mel spectrograms resulted in huge files which were very hard to process with Keras. Moreover, even when the data was able to be successfully loaded, the CNN models had extremely low accuracies and took significantly longer than the MLP model. The architecture of the CNN can be found here.

It is not completely clear why the CNN model did not work, but there are a few possible explanations. The files were featurized into 60x41 spectrograms, and this may have been too small of an image for the CNN to process effectively. Furthermore, the exact architecture of the CNN may not be optimal for the problem we are dealing with. It may be helpful to learn more about how many bands and frames were used (what are the optimal dimensions for the spectrogram?) with typical audio classification models that use the log-mel spectrogram. Thus, reading a research paper on the topic could be beneficial.

Conclusion

In this deep learning project, we featurized audio samples from the Common Voice Dataset to predict the age category of the speaker. Because there were so many categories, and we don't have intuition about the features that the Python libraries are capturing, using a deep learning model is preferable over a typical machine learning approach.

Generally speaking, we got very good results by doing some basic LASSO feature selection and using a normal artificial neural network. When the models trained on a GPU, training was fast. Additionally, the model did not overfit. Precision, recall, and f1-scores were relatively high.

Another common approach for audio classification problems is using a log-mel spectrogram to turn the problem into an image classification problem, and accordingly using a CNN. However, it was very difficult to work with the 4D Numpy array (N x 61 x 40 x 2) and so running the models became very tedious. The accuracies were also very low (around 30%), so using this approach for this particular problem may not be useful. As a result, it is not recommended to use a CNN for this problem.

Though we got adequate results from our MLP model, our performance metrics could always be better. Perhaps, we should see what happens when continuing to add more epochs, and balancing any overfitting by adding more dropout. We also did not try regularization as a way of dealing with overfitting. This could be a more effective way than using dropout. We could also experiment with deeper networks and/or more nodes. Finally, using batch normalization might be an effective way of stabilizing the neural network.

Appendix

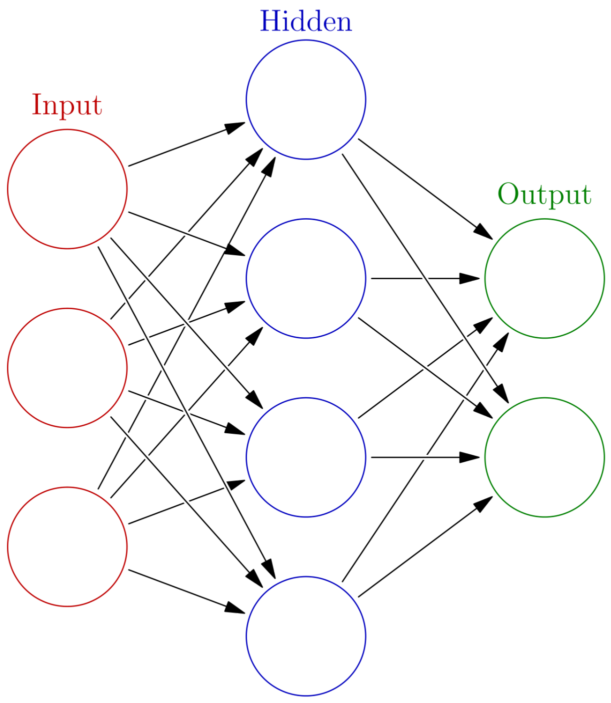
Explanation of Multilayer Perceptron Model

A multilayer perceptron model refers to a typical artificial neural network. A neural network is best described as a network of logistic regression units. A logistic regression unit can be defined as follows: each feature is assigned a random weight, and each unit is assigned a random bias term. The output can be described by the following equation:

$$P = \frac{1}{1 + e^{-(a+bX)}}$$

We can see that 'a' represents the bias term, 'b' represents a weight vector (with dimensions 1XD, where 'D' is the number of features), 'X' represents the vector of samples (with dimensions DxN, where 'N' is the number of samples in the training set). This equation is typically described as a sigmoid function, and we do this so we can ensure that the output is between 0 and 1. In the simple example of binary classification, an output greater than 0.5 would classify a sample as a '1' or a 'Yes', and accordingly if the output was less than or equal to 0.5, the sample would be classified as a '0' or 'No'.

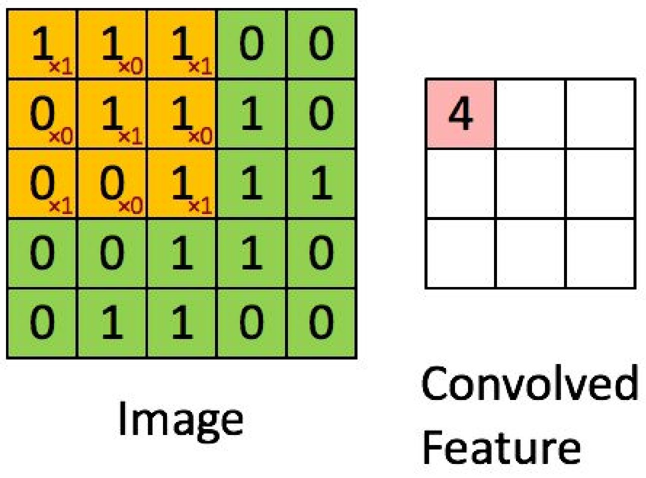
At first the weights are random, but we use an error function to compute the error of the prediction, and use a technique called gradient descent to optimize the weights by minimizing the error. By creating a network of these units, we can create a complex decision making process:



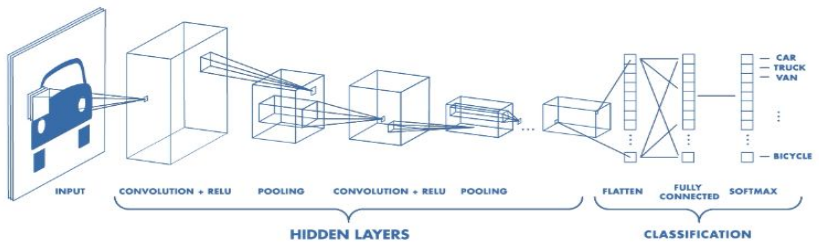
As we can see in the diagram, one node of the input layer (keep in mind that there is an input node for every feature) connects to every node of the hidden layer, and every node of the hidden layer is connected to every node of the output layer. Because there are multiple layers, the error is backpropagated and used to optimize the model. More information on neural networks can be found at various resources online.

Explanation of Convolutional Neural Network

Convolutional neural networks are a very common modeling approach when dealing with image data. Instead of every node in one layer being mapped to every node in the next, nodes are only mapped to particular nodes, based on their proximity to each other in the image. The operation of feeding to the next layer in the network is an operation called convolution, which can be described by the following diagram:



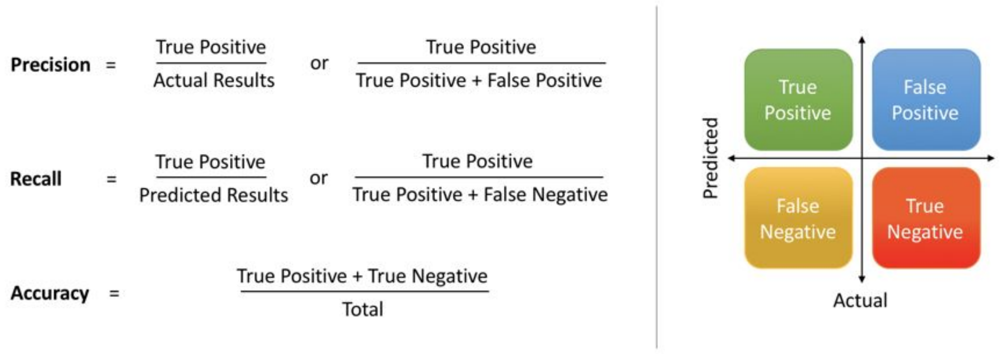
As we can see, a 3x3 subgrid of the image is multiplied by a weight matrix called a ‘filter.’ Then, the results are added together to create the output for the next node in the layer. The weight matrices are similarly optimized with backpropagation. Here is a diagram of a typical convolutional neural network architecture:



The image data is passed through multiple convolution operations, along with multiple other operations called pooling. It is eventually flattened into a layer for a conventional neural network, and classified accordingly. As with artificial neural networks, there are a lot of resources that explain CNN's in depth.

Info on Performance Metrics

The following diagram gives a good summary of performance measures for a classification model:


Precision = True Positive / Actual Results or True Positive / True Positive + False Positive
Recall = True Positive / Predicted Results or True Positive / True Positive + False Negative
Accuracy = True Positive + True Negative / Total
True Positive
False Positive
False Negative
True Negative
Predicted
Actual

Of course, the above applies for a binary classification model only. This is why in our classification report, there is a calculated precision and recall for every age category. For example, in our first MLP model, the precision in the "Teens" row means:

(# of voice samples that were correctly classified as teens) / ((# of voice samplesthat were correctly classified as teens) + (# of voice samples that were incorrectly classified as teens)) = 0.82.

The "Weighted Average" gives an average of all the metrics, and accordingly weights labels that show up more frequently higher than labels that show up less frequently. For example, because there are 466 voice samples that have a speaker in their twenties, the precision would be weighted higher (when computing the average) than the precision for voice samples with speakers in their thirties (because there are only 389).

The f1 score can be described as a harmonic mean of the precision and recall:

$$F_1 = \left(\frac{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}}{2} \right)^{-1} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}.$$

End of Appendix.

Works Cited

Schwoebel, J. (2018). *An Introduction to Voice Computing in Python.* Boston; Seattle, Atlanta: NeuroLex Laboratories. https://github.com/jim-schwoebel/voicebook

Saxena, Shruti. “Precision vs Recall – Towards Data Science.” *Towards Data Science*, Towards Data Science, 11 May 2018, towardsdatascience.com/precision-vs-recall-386cf9f89488.

Van Rijsbergen, C. J. (1979). *Information Retrieval* (2nd ed.). Butterworth-Heinemann.

Regression Basics, faculty.cas.usf.edu/mbrannick/regression/Logistic.html.

Santos, Leonardo Araujo dos. “Convolution.” *Deep Q Learning · Artificial Inteligence*, leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/convolution.html.

Cornelisse, Daphne. “An Intuitive Guide to Convolutional Neural Networks.” *FreeCodeCamp.org*, FreeCodeCamp.org, 24 Apr. 2018, medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050.