ASAP AND DEEP ASAP: END-TO-END AUDIO SENTIMENT ANALYSIS PIPELINES

A Project

Presented to

The Faculty of the Graduate School

At the University of Missouri

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Avery Wells

Dr. Yi Shang, Advisor

JULY 2017

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

ASAP AND DEEP ASAP: END-TO-END AUDIO SENTIMENT ANALYSIS PIPELINES

Presented by Avery Wells

A candidate for the degree of

Master of Science

And hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Yi Shang

Dr. Dong Xu

Dr. Prasad Calyam

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Dr. Yi Shang for his support throughout my time working on this project. He has offered valuable insights and guidance along the way to help make this project possible.

I would also like to acknowledge the rest of Dr. Shang's lab, especially Nick Wergeles. They always gave honest critiques of my progress and were always available when I needed some assistance. Additionally, I would like to thank Dr. Detelina Marinova and Bitty Balducci from the College of Business. They collaborated closely with me throughout the project and were always pushing me to achieve better results.

Most importantly I would like to thank my family. They paved the way for me to attend the University of Missouri and were always supportive of my decisions. Without them none of this would have been possible.

Lastly, I would like to acknowledge my committee members Dr. Dong Xu and Dr. Prasad Calyam.

Table of Contents

List of Figures

List of Tables

ABSTRACT

Historically sentiment analysis has focused on identifying moods and opinions expressed in texts. However, recently more research has been focused on analyzing audio sentiment. Using audio to identify a speaker's general attitude has several potential use cases. For this project, audio sentiment analysis was applied to call center conversations between a salesperson and a customer to predict if the customer would set up a sales meeting based upon their sentiment during the call. This project contains two main contributions: (1) An end-to-end sentiment analysis pipeline for segmenting, performing feature extraction, and classifying conversational audio files using classical machine learning methods and (2) a second end-to-end sentiment analysis pipeline that utilizes a deep Recurrent Neural Network to predict sentiment. Both pipelines demonstrate performances in line with what is the current state-of-the-art and are freely available to the public.

1. Introduction

Sentiment analysis is the process of identifying the attitude, or private state, of a person towards a particular topic or general context through computational methods. More specifically, sentiment analysis focuses on determining if a private state is positive, negative, or neutral. For example, someone saying, "I like that red car," could be labeled as positive sentiment and someone else saying, "The blue car is ugly," could be a negative sentiment. In practice however, analyzing sentiment is not this straightforward. What if someone said, "I wish the movie would have been more exciting or funnier." The words "exciting" and "funnier" would most likely be considered to have a positive sentiment by a computational model, yet, the sentiment of the sentence as a whole is clearly negative. So how should a model classify the sentiment of this sentence? Additionally, how does the pitch of a person's voice or their rate of speech affect sentiment? It is questions like these that a sentiment analysis model must answer and why sentiment analysis as a whole is not a trivial problem.

Traditionally, researchers have focused on analyzing sentiments expressed through text [1, 2, 3, 4]. Recently though more work has been done to analyze sentiments expressed in audio and visual modalities or even combinations of audio, visual, and textual expressions [5, 6, 7, 8]. This project focuses on performing sentiment analysis using only the audio (acoustic) modality. In particular, the goal of this project is to investigate whether machine learning models can be trained on acoustic features extracted from phone conversations between a salesperson and a customer to accurately predict whether the customer will agree to set up a sales meeting. If successful, researchers would be able to use these models to determine which acoustic features (pitch, rate of speech, number of pauses, etc.) were most indicative of setting up a meeting with a customer.

In order to achieve this goal, two end-to-end sentiment analysis pipelines were developed: ASAP (Audio Sentiment Analysis Pipeline) and Deep ASAP. Both pipelines can take a labeled (positive or negative sentiment) collection of audio files and split them into utterances based on speaker turn, extract acoustic features from each utterance (or the entire call), train a supervised machine learning model from the extracted features, and perform sentiment classification with the trained model. ASAP can train a variety of classical machine learning methods for sentiment analysis including an implemented version of the Hidden Markov Model (HMM) that was described in [5]. On the other hand, at the core of Deep ASAP is a Long Short Term Memory (LSTM) network that utilizes the sequential nature of audio data to perform classification at accuracy rates that are on par with the current state-of-the-art in audio-only sentiment analysis. Both pipelines are implemented in Python so that they may run on any system and are freely available to the public through Github.

2. Related Work

2.1 Audio Sentiment Analysis

Initially, audio sentiment analysis was done by first transcribing the recorded speech into text and then using a classical sentiment analysis tool designed for text as seen in [9]. The problem with this approach as noted in [5] is that it relies on an accurate translation which can be hard to achieve even with newer automatic transcription solutions like IBM Watson and Google Voice [10, 11]. Additionally, the transcription approach relies on large lexicons of words annotated with their sentiment such as those found in [2] and [12] and loses out on valuable information associated with tone or other audible features expressed in spoken dialect. A more streamlined and automatic approach that does not rely on transcribing the audio first and instead uses acoustic features was showcased in [5]. Several other publications adopted this same methodology [6, 7, 13]. The basic idea behind these methods is to split a longer piece of audio into smaller chunks called utterances. Feature extraction is then performed on each utterance and a machine learning model is trained for classification. The acoustic features used in these approaches are usually low-level descriptors such as pitch or Mel-frequency Cepstral Coefficients (MFCC) [5, 6]. In [5] a HMM was used for sentiment classification and achieved about 55% accuracy on their own YouTube dataset containing 280 labeled utterances from 47 individuals. In [7] a support vector machine (SVM) with a linear kernel was used on their MOUD database of 498 utterances from 55 individuals and achieved only 46.75% classification accuracy.

2.2 Audio Sentiment Analysis with Deep Neural Networks

In addition to the approaches discussed above, several researchers have applied deep neural networks (DNNs) to the audio sentiment analysis problem. In [13] a bidirectional LSTM (BLSTM) is trained with a sequence of acoustic features taken at intervals throughout the audio stream (instead of acoustic feature summaries across a whole utterance) to determine sentiment. They report a 54% classification accuracy on the Interactive Emotional Dyadic Motion Capture (IEMOCAP) [14] database. In [6] a convolutional neural network is trained on acoustic features from the MOSI [8] dataset containing 2199 utterances from 93 speakers and achieved 61.8% accuracy. The same CNN trained from MOSI was also tested on YouTube [5] and MOUD [7] databases and achieved 56.4% and 54.9% across-dataset accuracies respectively.

3. Design and Implementation

3.1 Design of ASAP

Currently, non-automated pipelines in audio sentiment analysis follow a similar design:

1.) Divide input audio signals into smaller chunks called utterances

2.) Extract features from each utterance

3.) Select and train a machine learning model for classification

4.) Test and validate the trained model for sentiment analysis

ASAP conforms to this same general methodology but automates the entire process so that researchers can iterate on their models more frequently and more easily. Figure 1 shows a flow chart of how ASAP works from end-to-end. In the case of the call center analysis problem, ASAP will take as inputs audio files containing conversations between two speakers and train a model that's output can be used to predict whether an unlabeled test call resulted in a sales meeting (positive sentiment) or no meeting (negative sentiment). It is a two-class classification problem.

3.2 ASAP Inputs

ASAP takes as input a collection of audio signals stored as WAV files. If the files are in MP3 format a helper script can be used that will convert them to WAV in batch. Because ASAP only works for supervised sentiment analysis, all audio files should be labelled with a 0 for negative sentiment or a 1 for positive sentiment. To process the files with ASAP, a CSV file is given as input in which the first column of the CSV contains the path to a WAV file and the last column contains the label for that particular WAV file.

Each row corresponds to a different WAV file. ASAP will load in all WAV files that are listed in the input CSV.

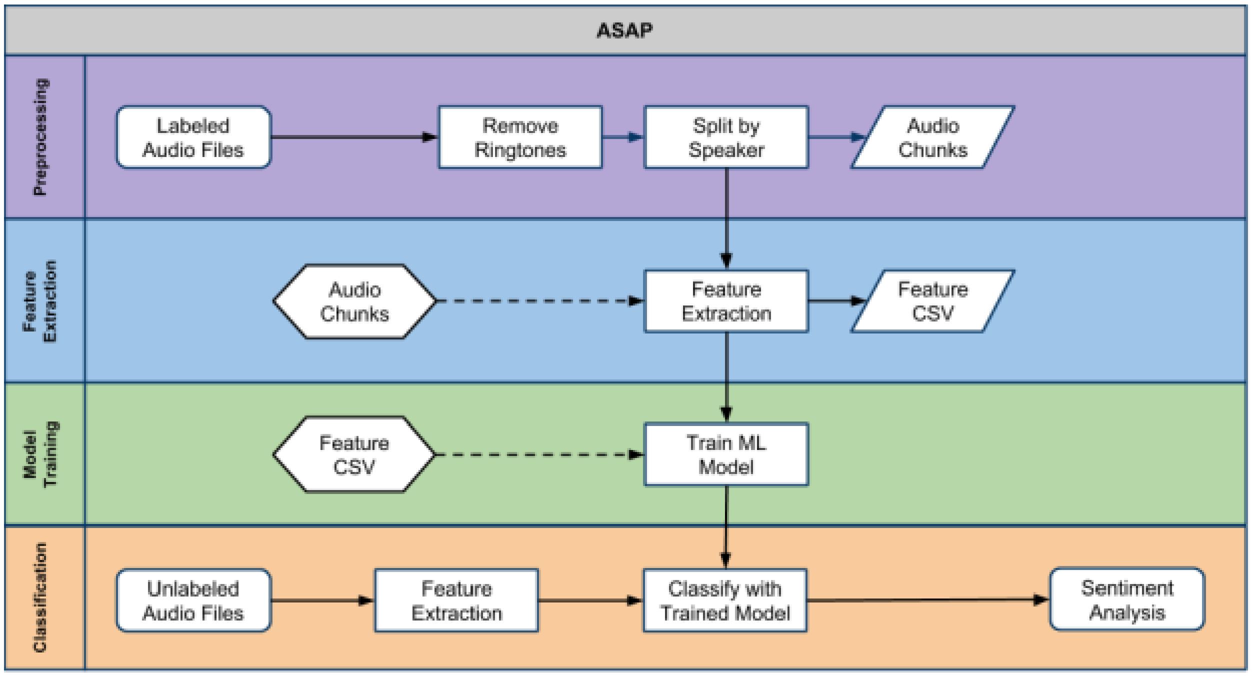

Figure 1: ASAP pipeline flow

3.2.1 Preprocessing

Following the general best practices seen in other audio sentiment analysis papers the first step ASAP does is break down each input audio signal into its component utterances. Since this project is mostly focused on analyzing the sentiment found in call center conversations, ASAP must first remove any receptionist, silence, or ringtones before the sales conversation takes place. Unfortunately, this is a non-trivial task to automate because there is no standard way for these calls to begin. To attempt to remove the first portion of a call before the conversation between the salesperson and customer takes place, a custom HMM was trained to detect ringtones with the *pyAudioAnalysis* software [15].

The ringtones are a very distinct sound within the context of a call so the HMM could be trained with only a handful of samples. Using the pre-trained HMM, each input audio stream is analyzed to determine at which points a ringtone is detected. Once these points are detected everything before the last ringtone is dropped and only the audio signal from after the last ringtone is kept. These clipped audio signals are saved to disk in case the user would like to examine them.

After the rings have been successfully removed the remaining audio should only contain the conversation between the salesperson and the customer. It is at this point that ASAP breaks the audio up into utterances based upon speaker turn. This means that each time the speaker changes ASAP will clip the audio into an utterance. To achieve this a process called speaker diarization is employed. Speaker diarization is an unsupervised, automated process that attempts to answer the question: Who spoke when? There are many varying implementations of speaker diarization but for the purposes of this pipeline the implementation found in *pyAudioAnalysis* was used because it would work nicely with the rest of the Python components in the pipeline. The method of speaker diarization used was published in [16] and uses a combination of Fisher's Linear Discriminant, K-Means, and Viterbi Smoothing. Again, since there are two speakers in each call it is possible to pass this information as a hyperparameter to *pyAudioAnalysis* so that it will look for only two distinct speakers during its clustering routines.

The last step in the preprocessing stage is to assign labels to the generated utterances. For this project labels are given on a per-call basis. That means that the entirety of each audio file was labeled as either a success or a failure. Because it is impossible to know what the sentiment of the produced utterances are they are given the same label as the call from which they came. For example, if a call was labeled as having positive sentiment and the preprocessing stage generated six utterances from that call, then all six utterances would be given the same positive label. Once the preprocessing stage is complete, all utterances are saved to disk so that the user may access them later outside of the pipeline.

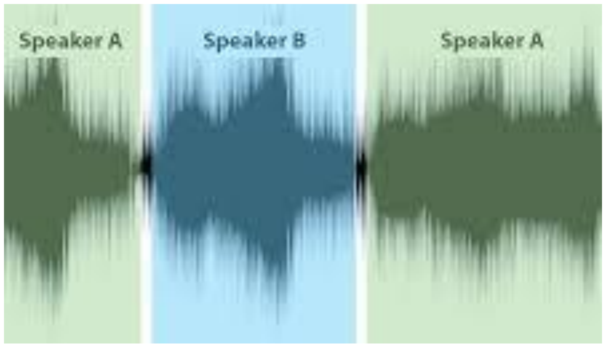

Figure 2: Visualization of the speaker diarization process on an audio signal [17]. Three utterances would be produced in this case.

3.2.2 Feature Extraction

Once the input audio signals have been sliced into utterance chunks, the next step in ASAP is to extract acoustic features from all the utterances. While the feature extraction stage is designed to work with utterances produced from the preprocessing stage of ASAP, any pre-sliced utterances generated from another outside method can be used. To perform the feature extraction a third-party tool called *OpenSMILE* is utilized [18]. *OpenSMILE* is an acoustic feature extraction tool that is commonly used throughout audio sentiment analysis research. It was chosen to be used in this project because it is very flexible and can extract a wide range of acoustic features. The installation of *OpenSMILE* comes with numerous pre-defined configuration files that allow for the extraction of a wide variety of different acoustic feature sets. A number of these configuration files are designed specifically for extracting features from audio for emotion analysis. Additionally, users can define custom configuration files to extract any set of acoustic features that they desire. For this project *OpenSMILE* is used to extract the 385 features defined in the INTERSPEECH 2009 Emotion Challenge [19]. The features selected to be used in this challenge were based and validated off of numerous past research works so they are a good choice to use in ASAP. This means that for every utterance a 385-dimensional feature vector is extracted. Even though the INTERSPEECH 2009 features were chosen for use in this project, because *OpenSMILE* is being used, a researcher could substitute in any different configuration file to extract the acoustic features that they desired.

The last step in the feature extraction stage of ASAP is to save the features in a feature-file for use by the model training or deployment stage. The feature-file is saved as a CSV file where each row represents an utterance. The first column is the name of the original WAV file from which the utterance came, the columns in the middle are the extracted features, and the last column is the utterance's label.

3.2.3 Model Training

Once a feature-file has been created ASAP can begin training a machine learning model. In order to make the pipeline more flexible, ASAP will accept feature-files for this stage of the pipeline that were generated by any other method as long as they are in the proper CSV format.

Since ASAP is designed to work with smaller datasets, training and validation is done using the leave-one-group-out approach. This means that for a dataset of 50 audio files, all of the utterances from one file would be held out for validation and a model would be trained on the utterances of the remaining 49. This process repeats 50 times so that all audio files are used for validation exactly once. The benefit to this method is that a large portion of the dataset doesn’t have to be sacrificed for validation.

As mentioned previously, the audio files in the call center dataset have only a label for the entire call, not the individual utterances. So, each utterance was given the same sentiment label as the call it was clipped from. This means that in order to produce an acceptable classification accuracy on the single call that is held out for validation during training, a single label for the call must be assigned. To achieve this all utterances from the held-out call are classified with a positive or negative sentiment. Then, the whole call is assigned a single label based on what the majority of its utterances were classified as. For example, if more than half of the utterances from a call were deemed to have positive sentiment then the whole call would be labeled positive and vice-versa.

Currently, there two available machine learning methods to train during this stage. The first is a random forest classifier from the *scikit-learn* package [20] and the second is a HMM that was implemented in Python based upon the methods described in [5]. A user can select to train with one or both of the models and all hyperparameters for the models can be set by the user on the command-line at runtime.

The HMM described in [5] was implemented as a Python class that adheres to all of the *scikit-learn* model guidelines. Since ASAP only deals with the two-class classification problem, the HMM works by training a separate HMM with Gaussian mixture emissions for each class. Then, during validation or deployment, a label is assigned to the new audio signal based upon which trained HMM, positive or negative, has a higher score.

The random forest classifier used is from the *scikit-learn* library. In general, it trains by fitting a chosen number of decision tree classifiers to subsets of the data. Then a label is assigned by taking the mode of the prediction from all of the individual decision trees. The random forest algorithm *scikit-learn* uses is found in [21].

The model training stage of ASAP is designed to work with any classifier from the *scikit-learn* library. This means that with some simple additions researchers would have access to easy-to-use support vector machines, multi-layer perceptron, k-nearest neighbors, and a multitude of other classification algorithms.

3.3 Design of Deep ASAP

There are several differences between Deep ASAP and ASAP, namely, the lack of a preprocessing stage, a slight modification to the feature extraction procedure, and of course a different classification model at its core. Figure 3 shows the slightly different pipeline flow of Deep ASAP with the preprocessing stage removed.

3.3.1 (Lack of) Preprocessing and Feature Extraction

As mentioned previously, Deep ASAP does not contain a preprocessing stage. The reasoning behind this is that because Deep ASAP uses a deep recurrent net at the model stage that can handle long sequences of inputs, it would make more sense to use feature sequences extracted at a more fine-grained level like 2-5 seconds as opposed to using features extracted from utterances which could be averaged over a longer time period (upwards of 30 seconds in some cases). This setup also makes more intuitive sense with the call center dataset that only provides labels at a call level instead of utterance level. Before, ASAP had to extrapolate the entire call's labels to its utterances but here in Deep ASAP that doesn't need to happen.

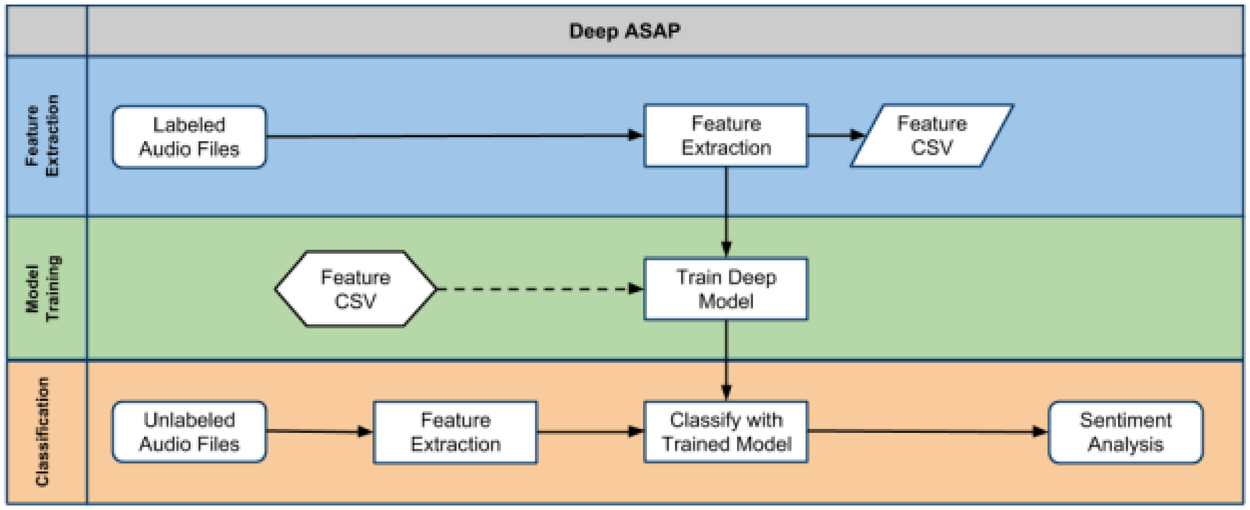

Figure 3: Deep ASAP pipeline flow.

Since Deep ASAP does not use utterances *OpenSMILE* is used to extract features in a slightly different way. The INTERSPEECH feature set is still used in this pipeline but this time features are extracted from a sliding 5-second window over the entire audio signal. This results in a sequence of acoustic features computed for the whole file. Naturally, audio signals with a longer duration will end up with a long feature vector. Again, as with ASAP, different feature sets can be extracted with *OpenSMILE* for use in the deep pipeline. Additionally, the size of the sliding window used during feature extraction and the amount of overlap in the sliding window can easily be changed by adjusting the *OpenSMILE* configuration file. Just like in ASAP, the feature extraction stage of Deep ASAP outputs a feature-file in the same CSV format for use by the deep model.

3.3.2 Model Training

The model trained in Deep ASAP for this stage is a bidirectional long short term memory network (BLSTM). This network was used in [13] for audio sentiment analysis with some success and a similar network was used in a slightly different audio signal classification problem by a group that placed in the top 10 of a TopCoder competition [22]. LSTMs, a subclass of recurrent neural networks (RNN), were originally designed in [23] to utilize sequences of information. This is a big difference from a classical artificial neural network that treats each new input to the network as independent from the last.

LSTMs make a lot of sense to use in language modelling and other fields because they can “remember” what they have previously seen. Additionally, a bidirectional LSTM also simultaneously processes the input in reverse order so it can also see what is coming in the “future”. They achieve this behavior through using what is known as hidden states within the network. These nodes are able to choose to remember or forget certain information on the fly. For example, when translating a sentence an LSTM could remember several words or sentences back to know what the current subject was and use the appropriate pronouns. This approach can be applied to audio sentiment analysis because it is often times how a speaker’s delivery changes over time that can impact sentiment, not necessarily how each part of their speech analyzed is independently. Figure 4 shows a high-level overview of how LSTMs can work for different problems by being able to process a varying sequence of inputs or a fixed number of inputs, and output a sequence of nodes of a fixed number of nodes. For the call center problem addressed in this project the LSTM will process a sequence of inputs (acoustic features) and produce only a single output (sentiment label). This would be a many-to-one architecture.

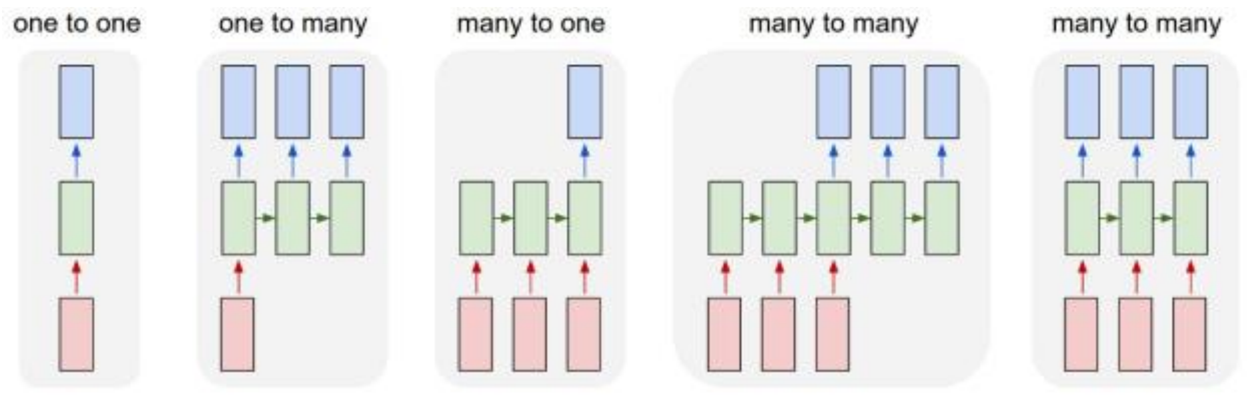

Figure 4: General LSTM architecture. Red boxes represent inputs, green boxes represent hidden states (that remember previous inputs), and blue boxes represent outputs.

The BLSTM used in this project was implemented in a Python deep learning library called *Lasagne* [24]. As stated previously, the inputs to the LSTM in this case are the ordered sequence of features extracted from each time step in the previous stage. However, since the input audio streams are of different lengths and hence have feature vector sequences of different lengths, any input sequences shorter than the longest sequence in a particular training fold must be padded with zeros. This is so that *Lasagne* can properly handle the network and is the best-practice method for most deep learning libraries. While this may seem like it is corrupting the input data, the LSTM will be able to learn that the 0's padded at the beginning of the shorter sequences do not have any meaning.

The training process of Deep ASAP is the same as what is utilized in ASAP. A leave-one-out training and validation approach is used where a single call is held out and the LSTM is trained on the remaining calls. The difference here in validation is that since the LSTM is learning the audio signals as a whole it only outputs a single sentiment label as opposed to having to infer the label of an audio file based upon the majority of its utterances.

4. Experiments and Results

4.1 Dataset Descriptions

The following sections describe the two datasets used in this project. One is a publicly released dataset from some previous audio sentiment analysis works that has been slightly modified to work in these pipelines. The other is the call center dataset that has been extensively referenced throughout this report.

4.1.1 YouTube Dataset

The first dataset used in this report was originally published in [5] and also used in [6]. It is a collection of 47 YouTube videos in which some opinion on a topic was being expressed. Each video contains only one speaker. Utterances from each video were labeled as positive, negative, or neutral. Additionally, each video was given a sentiment label as a whole. Because the original dataset was a three-class problem, all the videos which had a neutral sentiment label were removed for these experiments leaving just a positive-negative two-class problem. After this step, 27 videos were left containing 169 utterances. 12 of the videos were labeled with a positive sentiment and 15 were labeled negative. Of the 169 utterances, 94 were positive and 75 were negative.

4.1.2 Call Center Dataset

The second dataset for this project was obtained from Penske through a research collaboration with partners in the University of Missouri Business School. It contains recorded audio from Penske during their sales representative’s cold calls. These recordings could contain multiple speakers, but the focus of this project is a two-person conversation between the sales representative and the customer. The dataset contains 145 calls, most of which are between 2 and 5 minutes long. When broken down into speaker utterances as described in the ASAP sections there are 1,444 positive utterances and 1,499 negative utterances for a total of 2,943. At the call-level, 57 of them are positive and the remain 88 are negative.

4.2 Experiments

In order to make sure both pipelines were functioning as intending and to compare which of the machine learning methods performed best, ASAP (with the HMM and random forest classifiers) and Deep ASAP were used on both the YouTube and the Penske dataset. The datasets were put through the pipelines from end-to-end as discussed in each section. For ASAP, utterances were used during training and validation and for Deep ASAP the entire audio signal was used. The same 385 INTERSPEECH 2009 features were used in all experiments with features being extracted from utterances in ASAP and from sliding windows over the entire audio signal in Deep ASAP. In all cases training and validation was done in a leave-one-out fashion.

In order to ensure that results across different runs were reproducible all random initializations in the machine learning models were seeded. Additionally, training data was shuffled before passing it on to the models to ensure that they didn't get stuck in a local minimum. As with the model initializations, the random shuffling of the input data was also seeded so that it was the same order across all experimental runs.

Multiple sets of hyperparameters were changed across runs of each machine learning model in an attempt to discover which performed best. To evaluate a model's performance basic classification accuracy was used. To calculate the classification accuracy of a run of a model with particular hyperparameters the accuracy on the withheld validation set is averaged across all folds to give some estimate of overall performance.

4.3 Results

The following sections show the results of each machine learning model on the YouTube and call center datasets. Discussion of these results will be in the section after.

4.3.1 YouTube Results

HMM			
Parameters	*Chunk Accuracy*	*Call Accuracy*	*STD*
mix=1, states=2	62.85	59.26	0.3908
mix=1, states=3	62.11	59.25	0.3865
mix=1, states=4	62.11	59.25	0.3865
mix=1, states=5	61.74	59.25	0.3862
mix=1, states=6	61.74	59.25	0.3862
mix=1, states=7	61.74	59.25	0.3862
mix=2, states=2	58.03	59.25	0.334
mix=2, states=3	56.14	51.85	0.3012
mix=2, states=4	56.33	55.55	0.3216
mix=2, states=5	52.84	51.85	0.3208
mix=2, states=6	**61.86**	**70.37**	**0.3175**
mix=2, states=7	59.57	66.66	0.3143
mix=3, states=2	57.88	55.55	0.3624
mix=3, states=3	59.53	59.25	0.3746
mix=3, states=4	57.88	59.25	0.3868
mix=3, states=5	49.99	48.14	0.3894
mix=3, states=6	61.25	59.25	0.3396
mix=3, states=7	60.67	55.55	0.3485

mix=4, states=2	67.1	66.66	0.3605
mix=4, states=3	67.6	66.66	0.3605
mix=4, states=4	62.83	59.25	0.3727
mix=4, states=5	60.63	59.25	0.3809
mix=4, states=6	60.42	62.96	0.3619
mix=4, states=7	59.89	59.25	0.3634

Table 1: Classification accuracy of the HMM with ASAP. First column is hyperparameters adjusted, number of Gaussian mixtures and number of hidden states.

Random Forest			
Number of Estimators	*Chunk Accuracy*	*Call Accuracy*	*STD*
10	63.39	66.66	0.299
20	58.26	59.25	0.3563
30	59.6	62.96	0.3562
40	61.37	66.66	0.369
50	57.97	59.25	0.3639
60	63.13	62.96	0.3639
70	64.27	66.66	0.385
80	**66.72**	**70.37**	**0.3722**
90	64.67	70.37	0.3807
100	61.95	66.66	0.3846

Table 2: Classification accuracy of the random forest with ASAP. First column is the number of estimators used.

LSTM		
Number of Hidden Units	*Train Accuracy*	*Test Accuracy*
5	49.28	48.14
20	47	29.62
35	**65.52**	**51.85**
50	63.1	48.14
65	62.92	33.33
80	73.5	44.44
95	74.92	37.03
110	75.92	40.74
125	79.91	33.33

Table 3: Classification accuracy of the LSTM with Deep ASAP. First column is number of hidden units in the LSTM layer.

4.3.2 Call Center Results

HMM			
Parameters	*Chunk Accuracy*	*Call Accuracy*	*STD*
mix=1, states=2	58.76	59.72	0.3055
mix=1, states=3	58.54	59.02	0.3052
mix=1, states=4	58.51	59.02	0.3045
mix=1, states=5	58.5	59.02	0.304
mix=1, states=6	58.46	59.02	0.3037
mix=1, states=7	58.45	59.02	0.3036
mix=2, states=2	57.98	59.72	0.2809
mix=2, states=3	58.01	59.72	0.2839
mix=2, states=4	58.07	60.41	0.2838
mix=2, states=5	58.22	60.41	0.2837
mix=2, states=6	58.34	60.41	0.2842
mix=2, states=7	**58.35**	**60.41**	**0.2836**
mix=3, states=2	54.76	56.94	0.2258
mix=3, states=3	55.21	57.63	0.223
mix=3, states=4	55.32	58.33	0.2234
mix=3, states=5	55.22	59.02	0.223
mix=3, states=6	55.41	59.72	0.2217
mix=3, states=7	55.43	60.41	0.2224
mix=4, states=2	52.28	54.16	0.1515
mix=4, states=3	52.18	56.94	0.1666
mix=4, states=4	51.87	54.86	0.1507
mix=4, states=5	51.95	57.63	0.1506
mix=4, states=6	51.91	56.94	0.1541
mix=4, states=7	51.88	56.94	0.1558

Table 4: Classification accuracy of the HMM with ASAP. First column is hyperparameters adjusted, number of Gaussian mixtures and number of hidden states.

Random Forest			
Number of Estimators	*Chunk Accuracy*	*Call Accuracy*	*STD*
10	53.23	59.02	0.2413
20	53.89	58.33	0.2185
30	53.26	56.25	0.2297
40	53.47	60.41	0.2377
50	52.85	55.55	0.2433

60	**53.89**	**61.11**	**0.2422**
70	54.33	58.33	0.2437
80	54.83	60.41	0.2499
90	54.51	59.02	0.2555
100	54.65	58.33	0.2459

Table 5: Classification accuracy of the random forest with ASAP. First column is the number of estimators used.

LSTM		
Number of Hidden Units	*Train Accuracy*	*Test Accuracy*
5	58.65	53.84
20	62.34	55.94
35	69.67	55.94
50	69.57	57.34
65	70.83	61.53
80	70.86	55.94
95	**72.74**	**62.93**
110	72.13	56.64
125	78.02	62.23

Table 6: Classification accuracy of the LSTM with Deep ASAP. First column is number of hidden units in the LSTM layer.

4.3.3 Confusion Matrices of Best Models

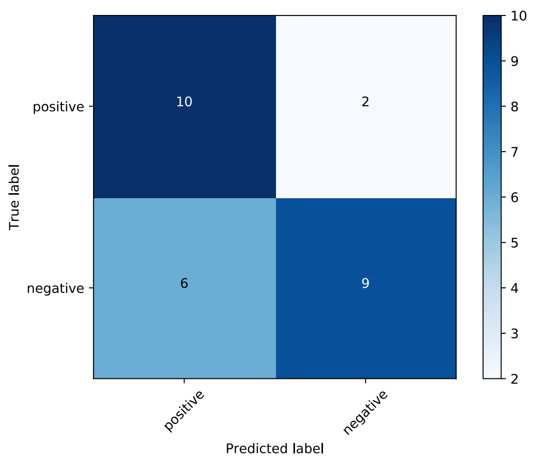

Figure 5: Confusion matrix of best performing HMM (2 Gaussian mixtures, 6 hidden states) on YouTube dataset

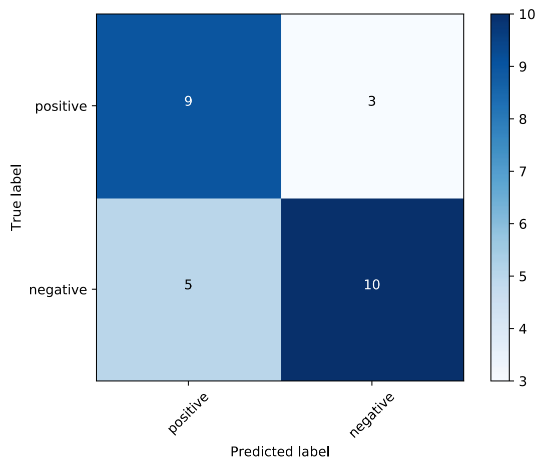

Figure 6: Confusion matrix of best performing random forest (80 estimators) on YouTube dataset

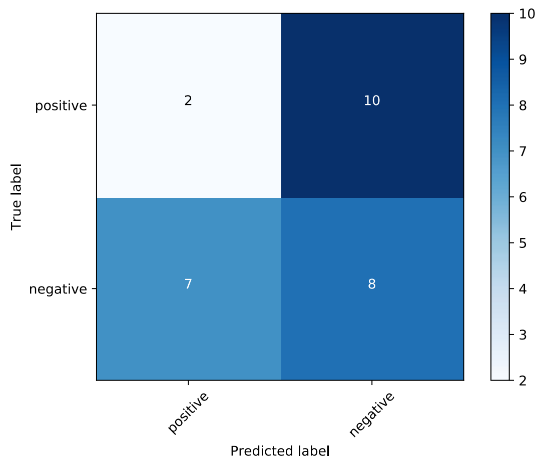

Figure 7: Confusion matrix of best performing LSTM (35 hidden units) on YouTube dataset

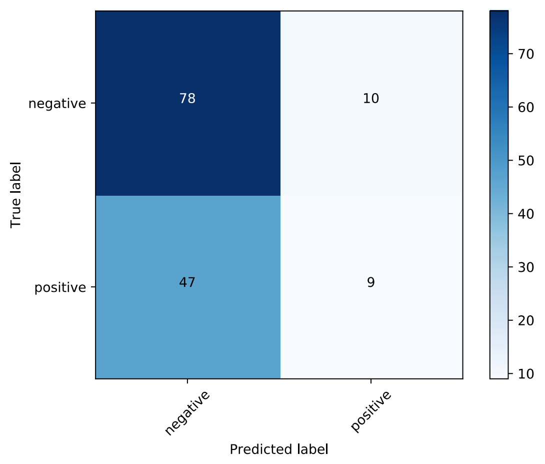

Figure 8: Confusion matrix of best performing HMM (2 Gaussian mixtures, 7 hidden states) on call center dataset

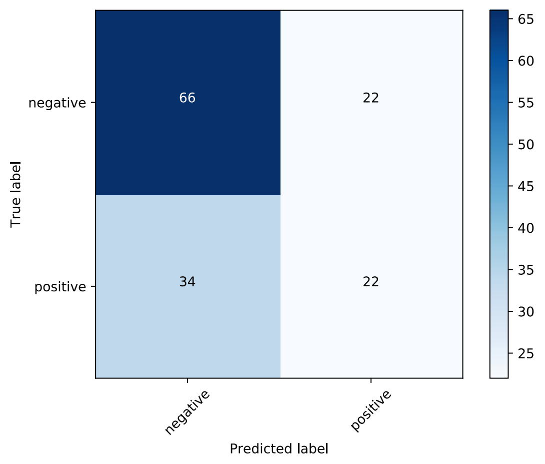

Figure 9: Confusion matrix of best performing random forest (60 estimators) on call center dataset

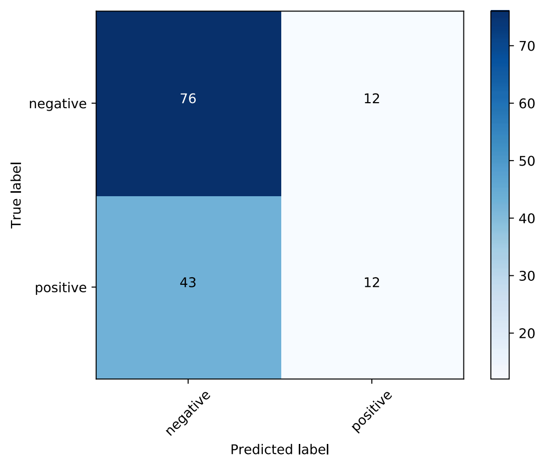

Figure 10: Confusion matrix of best performing LSTM (95 hidden units) on call center dataset

5. Discussion

Overall, all pipelines were able to successfully produce results that are mostly in line with what can be found in the literature. On the YouTube dataset [6] was able to achieve between 56% and 61% accuracies. Here, machine learning models from ASAP are able to achieve those results on the same dataset. When looking at the classification accuracy tables, the random forest probably performed best overall on the YouTube dataset with the HMM close behind it. The LSTM from Deep ASAP performed quite poorly on the YouTube dataset. This is most likely because it is such a small dataset that the LSTM was severely overfitting. In spite of setting the random seeds so that results were reproducible it is tough to see any clear trends within the YouTube results tables. The HMM somewhat performs best with models that have a “medium” complexity but there are also good and bad results intermixed. A similar intermixing of good and bad results can be seen in the random forest and LSTM results from the YouTube experiments.

The confusion matrices for the best models on the YouTube dataset are slightly more telling. From here, it is easier to tell that the random forest did a better job classifying both positive and negative videos correctly. On the other hand, the HMM trended towards classifying too many of the videos as positive and the LSTM trended towards classifying too many videos as negative. Because the dataset is slightly imbalanced, it would make sense that the LSTM favors the negative sentiment class.

Taking a look at the accuracy tables for the call center experiments, it is noticeable that the LSTM had much better performance than it did on the YouTube dataset. In fact, when going by classification accuracies alone it achieved the highest score. The best scores from the HMM and random forest were not far behind on this data though. The trend of the HMM performing best with models of "medium" complexity is much more apparent in the its accuracy table for the call cent experiments. The random forest still did not see much of a trend in the accuracy tables and the LSTM appeared to have better performances as the model became more complex.

When looking at the confusion tables for the call center data the good performances become much less convincing. Because the dataset is so imbalanced all three models misclassified almost all of the positive samples as negative. So, even though the accuracies were getting better, the model performances were not as good. The LSTM was by far the most sensitive to this imbalance, most likely because it is the most complex network of the three and so it was easier to over-train.

Ultimately, the machine learning methods were performing as intended, but because the datasets were so imbalanced the results were not great. Audio sentiment analysis is a hard problem as evidenced by the classification results of recent papers, but to improve the results in the case of the call center data some work should be done to ensure an equal class balance.

6. Conclusion and Future Works

When setting out on this project there were two main goals. First, to create an automated pipeline that researchers could use when doing audio sentiment analysis studies. Second, to apply the pipeline and the machine learning models with it on the call center dataset provided by our partners in the business school. As a whole, both ASAP and Deep ASAP perform the automation of an end-to-end solution very well. They should no doubt be of great help to anyone attempting to perform audio sentiment analysis tasks. Also, while the sentiment analysis results on the call center dataset were slightly underwhelming, the machine learning models were able to perform mostly up to par with what other research papers had achieved on the YouTube dataset.

There are several interesting ways for this project to continue in the future. First, in regard to the call center data, it is possible that better results could be achieved with balancing of the dataset. Second, it would be interesting to try other newer deep learning networks such as convolutional neural networks on this problem. These networks have had great success elsewhere. Another possibility would be to combine a convolution network with the LSTM that was implemented in Deep ASAP.

References

1. B. Pang and L. Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics*, Barcelona, Spain, July 2004.
2. J. Wiebe and E. Riloff. Creating subjective and objective sentence classifiers from unannotated texts. In *CICLing*-2005, 2005.
3. J. Wiebe, T. Wilson, and C. Cardie. Annotating expressions of opinions and emotions in language. *Language Resources and Evaluation*, 39(2-3):165–210, 2005.
4. Bo Pang and Lillian Lee, "Opinion mining and sentiment analysis," *Foundations and trends in information retrieval*, 2008.
5. T Louis-Philippe Morency, Rada Mihalcea, and Payal Doshi. 2011. Towards multimodal sentiment analysis: harvesting opinions from the web. In *Proceedings of the 13th international conference on multimodal interfaces* (ICMI '11). ACM, New York, NY, USA, 169-176.
6. Haohan Wang and (2016). Select-Additive Learning: Improving Cross-individual Generalization. *CoRR*, *abs/1609.05244*.
7. Veronica Rosas, Rada Mihalcea, and Louis-Philippe Morency. 2013. Multimodal Sentiment Analysis of Spanish Online Videos. *IEEE Intelligent Systems* 28, 3 (May 2013), 38-45.
8. Amir Zadeh, "Micro-opinion sentiment intensity analysis and summarization in online videos," in *ICMI*. ACM, 2015
9. Lakshmish Kaushik, Abhijeet Sangwan, and John HL Hansen, "Sentiment extraction from natural audio streams," in *ICASSP*. IEEE, 2013.
10. https://www.ibm.com/watson/developercloud/alchemy-language.html
11. https://cloud.google.com/speech/
12. A. Esuli and F. Sebastiani. SentiWordNet: A publicly available lexical resource for opinion mining. *In LREC*, *Genova*, IT, 2006.
13. Vladimir Chernykh and (2017). Emotion Recognition From Speech With Recurrent Neural Networks. *CoRR, abs/1701.08071*.
14. C. Busso, M. Bulut, C.C. Lee, A. Kazemzadeh, E. Mower, S. Kim, J.N. Chang, S. Lee, and S.S. Narayanan, "IEMOCAP: Interactive emotional dyadic motion capture database," *Journal of Language Resources and Evaluation*, vol. 42, no. 4, pp. 335-359, December 2008.
15. Giannakopoulos T (2015) pyAudioAnalysis: An Open-Source Python Library for Audio Signal Analysis. *PLoS ONE* 10(12): e0144610.
16. Giannakopoulos, Theodoros, and Sergios Petridis. "Fisher linear semi-discriminant analysis for speaker diarization." *Audio, Speech, and Language Processing, IEEE Transactions* on 20.7 (2012): 1913-1922
17. http://multimedia.icsi.berkeley.edu/speaker-diarization/the-meeting-diarist/
18. Florian Eyben, Felix Weninger, Florian Gross, Björn Schuller: "Recent Developments in openSMILE, the Munich Open-Source Multimedia Feature Extractor", *In Proc. ACM Multimedia (MM)*, Barcelona, Spain, ACM, ISBN 978-1-4503-2404-5, pp. 835-838, October 2013.
19. Schuller, B.; Steidl, S.; Batliner, A.: "The Interspeech 2009 Emotion Challenge", Interspeech (2009), ISCA, Brighton, UK, 2009.

20. Pedregosa, F., Varoquaux, G., Gramfort, A. & Michel, V. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research, 12,* 2825-2830.
21. L. Breiman, “Random Forests”, Machine Learning, 45(1), 5-32, 2001.
22. Harutyunyan. “Spoken language identification with deep convolutional networks.” TopCoder Competition.
23. J. Elman, “Finding structure in time,” Cognitive Science, 1990.
24. Sander Dieleman and (2015). Lasagne: First release.