Abstract

Sentiment classification for natural language is a developing area of research with a variety of real-world applications, but sentiment classification on audio streams has received relatively little attention. For our project, we built a model to identify sentiment of an audio stream, using unsupervised learning. Our work is split into two tasks: first, we trained a model that predicts the next timestep of the speech data given a stream of speech data, ignoring the training labels. This essentially learns the representation of emotion in the data. Second, we froze the network, and further trained our model with a linear classifier to predict the sentiment. To see how our model fared in a variety of situations, we implemented many different models, including recurrent neural networks, convolutional neural networks, and multiplicative LSTMs, to see which conditions achieved the highest accuracy. Ultimately, our best model was a multiplicative LSTM, using MFCC features. In the future, since our group is interested in biosciences, we would like to see this framework extended to learn important predictors of disease in speech.

1 Introduction

The field of unsupervised sentiment classification has developed in recent years especially within the field of natural language processing. Our project is directly inspired by OpenAI’s recent work in Unsupervised Sentiment Classification (Radford et al., 2017). The OpenAI group took a new approach to sentiment classification on an Amazon review dataset. They trained a language model on the dataset, only predicting the next character and ignoring the labels, and showed that a single neuron in their character predictor was able to predict sentiment with an accuracy on par with the state of the art. This is extremely useful, because creating labeled datasets for classification in NLP is extremely costly. Using this new method, large models can be trained using cheaper, raw data, and classifiers can be built off of these models using relatively small labeled datasets. For our project, we wanted to extend unsupervised sentiment classification to spoken language processing, and hopefully find a similar neuron or other method of strong classification.

One of the most pressing problems of audio sentiment classification is a lack of labeled data: it both takes a grueling amount of time and is expensive to produce it in large quantities. Our goal was: given an audio file, guess the sentiment (anger, disgust, fear, happiness, sadness, surprise, or neutral) conveyed by the speaker. A quick and cheap way to label unlabeled audio data would not only be a massive help to research, but also holds great promise for other fields.

The fundamental architecture of our project is tripartite. First, we extract the requisite features of each time step of the audio files in our training set. Second, we feed the input into a long short-term memory neural network (LSTM), a structure adept at classification and predicting time series, with a hundred hidden layers. The output of the first LSTM feeds into a second LSTM, which predicts the next time steps features. The output of the final hidden state predicts the class of the next time step given its features. This architecture is then trained, and the unsupervised learning is complete.

To optimize our model, we experimented with a vast number of factors. Not only did we alter the size, learning rates, iterations of training, and other properties of our neural networks, we experimented thoroughly with the features and models we used: we tried different combinations of fea-
tured available from the pyAudioAnalysis library (Giannakopoulos, 2015), and different types of models, including multiplicative LSTMs and convolutional neural networks.

2 Background

The main inspiration for our project is Radford et al’s work for OpenAI (Radford et al., 2017), which determines the sentiment for Amazon text reviews using multiplicative LSTM neural networks and unsupervised learning. The authors train a character level language model on the Amazon reviews, completely ignoring the label. The key discovery of this paper is the existence of a single sentiment neuron within their network that predicts future reviews accurately. We hoped to use their framework and ideas of unsupervised learning for spoken language processing.

Other studies have attempted to classify sentiment, or even diagnose disease, from spoken language. Perez-Rosas and Mihalcea classified the sentiment of spoken reviews (Prez-Rosas and Mihalcea, 2013), though through transcriptions, comparing the success of both manual and automatic transcriptions. They found that as long as the transcriptions were high-quality, the accuracy rate of the classification was quite high.

In a similar project, Kaushik et al used a combination of part-of-speech tagging, maximum entropy modeling, and automatic speech recognition transcripts to extract sentiment from YouTube videos (Kaushik et al., 2013). The results were also encouraging, indicating fairly accurate classification of sentiment.

Bhaskar et al (Bhaskar et al., 2015) combined both text and speech analysis to classify emotion. Using a variety of lexicons for speech analysis and a multiclass support vector machine to learn the audio input, the authors show that their hybrid approachs accuracy is higher than either that of solely audio or text data.

In addition to researching related work in sentiment classification, we looked into work dealing with audio autoencoding. An interesting article that attempted unsupervised learning on audio data is Audio Word2Vec (Chung et al., 2016). This paper does not attempt to solve the same problem that we do, but it is interesting for the approach it takes in using unlabeled audio data. Specifically, this paper emulates the Natural Language Processing Word2Vec idea. It is able to create fixed length vector representations of variable length audio sequences. The authors train an architecture that consists of two RNNs using LSTM cells, an encoder and a decoder. The encoder maps the input audio onto a fixed length representation, and the decoder maps the vector back to the original input. The architecture is trained on exclusively unlabeled data and provides a semantic representation of variable length audio.

Although our project directly involves sentiment, we hope that it can be extended in the future to diagnose disease in the biomedical fields. Already, useful methods of psychosis (Mota et al., 2012), post-traumatic stress disorder (Vergyri et al., 2015), and Parkinson’s disease (Das, 2010) have been created. If these projects become widespread and successful, medical practitioners may be able to diagnose and monitor treatment and recovery of common diseases non-invasively, on par with other state-of-the-art procedures.

3 Approach

Using unlabeled data to train our model necessitates an autoencoding of time series data. This problem is extremely difficult, and remains a largely unexplored area of research. Because of this lack of direction we were forced to experiment on our own, with the goal of trying as many different conceivable models as possible in order to find at least one that shows promise. Our best chance of success was to run as many experiments as possible, tuning our model architecture and hyperparameters to find a solution to this unique problem.

Framework

Without an intuition for the strongest architecture, we decided to build a framework that would allow us to easily and quickly develop and iterate on models. The framework reflected the problem we were trying to solve, and included two main parts: the autoencoder and the classifier.

The autoencoder was constructed to find the best way to take advantage of large amounts of unlabeled data. The stronger the autoencoder, the more efficiently we would be able to take advantage of our data and the more successful we would be at classification. This part of the architecture was the core idea we were attempting to develop. The classifier was largely used as a measure of how well our autoencoder had done. Because we
constrained the classifier to be a simple dot product, it was a fair way to measure the quality of our lower dimensional embedding.

The figures give a better idea of the architecture. Figure 1 shows an example of how we could train an autoencoder. In this case, our source is a number of timesteps (1 ... N), and our target is time step N+1. Essentially, given N time steps of features, we use an RNN in an attempt to predict the next one. After training on a large amount of unlabeled data, we freeze the weights of our network, and train a linear classifier from the final hidden state of the RNN to our small set of labeled data. The resulting accuracy on the test set gives us an idea of how well our autoencoder performed.

Our framework allowed us to use any architecture of our choosing for the autoencoder, giving us flexibility in training a network. However, we decided to further limit our assumptions in order to give ourselves the best chance of success. In doing so, we added two additional options to our framework.

First, there was still room for experimentation in the classifier. Namely, where to connect the classifier to the autoencoder became an important decision. The optimal attachment of our linear classifier was not always obvious, and became an important hyperparameter for which we optimized. Because of this, we not only allowed our models to have any architecture for the autoencoder, but also gave ourselves the option of adding the linear classifier to any tensor in that architecture. Figure 3 shows a two layer RNN network that, given N time steps, has been trained to predict the next time step. It gives two examples of where the linear classifier could be attached. The green classifier is attached to all outputs of the top layer RNN, while the red classifier is only attached to the final state of the top layer RNN.

Second, we noted that the best target for our autoencoder could be a number of different things. In the examples shown thus far, given N featurized time steps, the target has been the N+1th featurized time step. However, we allowed ourselves three distinct options. Given featurized time steps (1 ... N), we added the option to predict 1) featurized time step N+1, 2) featurized time steps (2 ... N+1), or 3) featurized time steps (1 ... N). These options are shown in Figure 4.

Using this framework, we enjoyed the ability to train many different models, with various hyperparameter combinations. This gave us the best possible chance of finding a working model.

Models
Once we completed the framework, we faced the question of the optimal model to use for our autoencoder. We attempted four main classes of architecture.

Vanilla Autoencoder
The first and most obvious choice for the autoencoder step is the vanilla autoencoder. The classic autoencoder architecture involves multiple fully
connected layers that start from the input, decrease in size down to a bottleneck, and eventually work their way back up to the input size. The linear classifier could then be connected from the bottleneck to the classes. Using featurized time-series data, there were two different methods that we tried, which had approximately the same performance. The first involved flattening the feature vector of each time step, combining the results, and running the autoencoder on the resulting vector. The second involved autoencoding each feature individually, and then combining the results. Ultimately, the vanilla autoencoder disregards the time-series nature of the data, and is therefore a relatively naive model to try. As is shown in the results section, this model did not give a significant improvement over random guessing.

**RNNs**

Our main approach and experimentation involved various types of RNNs. Because of the importance of the time series nature of the problem, using an RNN makes intuitive sense. We believed, and eventually showed, that using this powerful tool we would be able to gain significant improvements over a supervised learning method on the small dataset, proving the validity of what we are trying to do.

Our original RNN model involved a single vanilla RNN layer using 15 time steps to predict the next single time step. However, this model proved to be insufficient, and was not able to improve significantly on the vanilla autoencoder. We attempted many iterations on this model, varying the number of hidden units in each cell, the number of hidden layers, and many hyperparameters such as learning rate and optimizer type. Over time, we were able to gain improvements in our classification score, achieving our best score with two RNN layers of size 200 and the number of features, respectively. We found that making this model smaller led to weak performance due to an insufficient number of parameters. However, making our model much larger than this started causing overfitting, which also led to decreased performance.

Although the RNN model worked well, we wanted to try something more complex. Our next step was to try using the same structure but with a different cell type. First, we tried the LSTM cell. The LSTM cell is powerful in that it adds a forget gate, which allows the model to learn when to forget information and when to keep it in memory. The LSTM cell, as expected, gave us improvements in performance as compared to the vanilla RNN cell. However, as we were met with greatest success using this structure, we decided to continue onto a cell type that might give us an even larger boost in performance.

For our final RNN model, we used a Multiplicative LSTM cell type. We were motivated to try the Multiplicative LSTM cell type (mLSTM) because it had good performance in recent literature for autoencoding tasks. The Multiplicative LSTM cell is a variation on the LSTM cell which allows each input to have a different recurrent transition function as described by Krause et al. (2016). The Multiplicative LSTM cell combines the advantages of the input-specific transition weight from multiplicative RNN with the LSTMs forget-gate. The mLSTMs input-specific transitions are flexibly applied while retaining memory. As we will discuss later, we found that the mLSTM resulted in improved performance over the LSTM.

**Convolutional Networks**

The third major addition to our model was the convolutional layer. We had two main motivations for trying this. Our first motivation was the thought that there may be some localized features that we could take advantage of in learning. In order to test this hypothesis, we used a one dimensional convolution over each feature, using varying filter size (from 3 to 8), and a small stride (either 1 or 2). Using this architecture, we expected the layer to pick up on important short-term features, which we would then feed into our original RNN structure to capture the changes in these features over time. Our inspiration for this experiment came from computational genomics, where one dimensional convolutions have been used successfully (Mota et al., 2012).

Our second motivation for the convolutional layer was to combine groups of timesteps to allow the RNN to focus on longer term dependencies. In order to test this hypothesis, we used the same one dimensional convolutional layer, but with a larger stride. Specifically, we used a stride as long as the filter size, which would combine distinct groups of timesteps into single observations. These observations would then be fed into our RNN structure to capture the time series nature of the structure.

As will be shown in the results section, the added convolutional layers either didn’t help or
hurt performance. There are many possible reasons for this failure, but we would hypothesize that the changes in features from one timestep to the next are much more important than any features that are generalizable over multiple timesteps.

**Seq2Seq**

Lastly, we attempted to train a Seq2seq model. The motivation for this is that Seq2seq autoencoding has been shown to work well in many capacities (Chung et al., 2016). Aside from this, Seq2seq models have an encoded vector that is a natural choice for the linear classifier. We built our architecture as in (Chung et al., 2016), with an encoder RNN and a decoder RNN, and trained our classifier from the encoded vector.

We experimented with various hyperparameters including number of layers, number of hidden units, learning rate, and number of timesteps. We used our best feature set for all experiments. Unfortunately, none of our experiments was successful, or able to predict with reliability significantly better than random. Our strongest hypothesis for this failure is that Seq2seq models take a notoriously large amount of time and data to train, and we did not have sufficient resources to attain a strong model.

**4 Experiments**

We got our data from two sources: the first, the Surrey Audio-Visual Expressed Emotion (SAVEE) Database, had four individuals read a combined 485 utterances, and the second, the Ryerson Multimedia Laboratory Emotion Database, had eight individuals read a combined 735 utterances. For the first database, there were seven classes of emotions (Anger, Disgust, Fear, Happiness, Sadness, Surprise, and Neutral), and for the second database, there were six (all of the above except for Neutral). We used seven classes for all of our experiments. The audio files were generally about five seconds in length. As in most spoken language processing endeavors, our project could have benefited from more labeled data and more variety of data. Below, we report a representative sample of our results.

We subdivided the data randomly into 80% train, 10% validation, and 10% test sets. The autoencoder was trained on the full training set, and the classifier was trained on a subset of the training set equivalent in size to 10% of the total dataset. To evaluate the performance of our models, we implemented a baseline classifier which was trained with only the classifier training data and did not use the autoencoder architecture. This classifier achieved an accuracy of 0.16690341.

From the features available from the pyAudioAnalysis library, three sets of features immediately stood out as strong options: one that dealt with energy and its entropy, one that dealt with the spectrum and its various properties, and one that dealt with the audio’s Mel Frequency Cepstral Coefficients (MFCCs).

We trained RNN LSTM models with various feature combinations to determine which features had the best performance so that we could continue to use them as we refined our model. Given constant conditions with an RNN model, the MFCC features perform the best. This makes sense, as MFCC are the gold standard features for almost all speech recognition systems. The spectral analysis, which examines the center, spread, and entropy of the spectrum, does fairly well as well. We used features from the spectral analysis and MFCC going forward. We tuned our models on the validation set.

<table>
<thead>
<tr>
<th>Features</th>
<th>Description</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>Energy</td>
<td>0.128551</td>
</tr>
<tr>
<td>3-7</td>
<td>Spectrum</td>
<td>0.160511</td>
</tr>
<tr>
<td>8-20</td>
<td>MFCC</td>
<td><strong>0.222301</strong></td>
</tr>
<tr>
<td>21-32</td>
<td>Chroma</td>
<td>0.133523</td>
</tr>
</tbody>
</table>

Table 1: Features.

As shown below, a learning rate of .001 seems to be best for our architecture, using an RNN and keeping all other hyperparameters constant.

<table>
<thead>
<tr>
<th>Learning Rate</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000001</td>
<td>0.0703125</td>
</tr>
<tr>
<td>0.0001</td>
<td>0.171165</td>
</tr>
<tr>
<td>0.001</td>
<td>0.198153</td>
</tr>
<tr>
<td><strong>0.001</strong></td>
<td><strong>0.245739</strong></td>
</tr>
<tr>
<td>0.01</td>
<td>0.129972</td>
</tr>
</tbody>
</table>

Table 2: Learning Rate.

It seems that the autoencoder does best when trained for 10,000 iterations, while the number of iterations to train the classifier does not greatly affect the results. All trials were run using an RNN and keeping all other hyperparameters constant.
We tested multiple combinations of the models discussed in the previous section. Below, we report the best performance for each architecture using the most promising features (a combination of MFCC features and spectral features), and 10000 iterations for both the autoencoder and the classifier. We report the results on the held-out test set below. The results indicate that the multiplicative LSTM architecture is the most promising.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>mLSTM</td>
<td>0.288352</td>
</tr>
<tr>
<td>Large RNN</td>
<td>0.245028</td>
</tr>
<tr>
<td>Convolutional mLSTM</td>
<td>0.240057</td>
</tr>
<tr>
<td>Large mLSTM</td>
<td>0.232955</td>
</tr>
<tr>
<td>Large Convolutional mLSTM</td>
<td>0.232955</td>
</tr>
<tr>
<td>RNN</td>
<td>0.191761</td>
</tr>
<tr>
<td>Simple Classifier</td>
<td>0.166903</td>
</tr>
<tr>
<td>Autoencoder</td>
<td>0.166193</td>
</tr>
<tr>
<td>mLSTM State Classifier</td>
<td>0.150568</td>
</tr>
</tbody>
</table>

Table 4: Results of Various Models.

Many of our models improved upon the performance of the simple classifier, and the smaller multiplicative LSTM model achieved the greatest accuracy. The larger architectures using the mLSTM or RNN did not perform as well. This may be because the longer-term effects or features are less relevant to the sentiment classification task.

Conclusions

Ultimately, our results were encouraging in our pursuit of sentiment classification using unsupervised learning. It was interesting to compare all of our different methods and models: with our multiplicative LSTM, one of our simpler models, we were able to obtain our strongest accuracy rates. We are encouraged by the success of the multiplicative LSTM architecture in handling the spectral and MFCC features for autoencoding. Further, our results show that training an autoencoder with a large set of unlabeled data does improve sentiment classification accuracy and suggest that new, flexible autoencoder architectures will support such work. Of course, while further work must be done to improve the accuracy of the model to make it feasible for real-world applications, the fact that our model was to perform quite a bit above our simple classifier (and that both outperformed random chance) is auspicious given the difficulty of the problem.

There are many areas for future work for this problem. First, the amount of data we had was limited, as are many other projects dealing with spoken language processing. Also, our audio files had moments of silence and other irregularities that can negatively affect the training of the model. Having more data that is regularized would be a great benefit. Second, it would be great to have audio streams that exhibit other characteristics other than sentiment; for example, predicting disease may be possible using this model, as similar projects have been proven successful. There is considerable potential for powerful models in this area to be useful in a range of practical applications.

Acknowledgements

We would like to thank Professor Andrew Maas and the course staff of CS 224s for discussing and supporting our work. We would also like to thank Professor Dan Jurafsky for his textbook which has been invaluable over the quarter.

References


