Mental Health Sensing Using Machine Learning

A Major Qualifying Project submitted to the faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science in Computer Science

Joseph Caltabiano, Nicolas Pingal, Myo Min Thant, Yared Taye, Adam Sargent, Yosias Seifu

Advised by Professor Elke A. Rundensteiner
and
Professor Randy C. Paffenroth

April 1, 2020

IRB-18-0031

Abstract

Using audio and text data from multiple sources, we evaluated the viability of using machine and deep learning to identify depression and anxiety. Machine learning methods using sub-clip boosting achieved an F1 score of 0.81 for depression and 0.83 for anxiety. Our convolutional neural networks and long-term short term memory models achieved F1 scores of 0.55 and 0.68 respectively for depression. As feature engineering, we used topological data analysis to create Betti curves in our machine learning pipeline. Furthermore, we developed a pipeline to generate text messages with deep learning models, for data augmentation purposes.

Acknowledgements

There are a number of others involved with this project that we would like to thank for making the whole thing possible. First, we would like to thank Professor Elke Rundensteiner, our project advisor, and our two graduate students, ML Tlachac and Ermal Toto, for their continual support and help during our project. Ermal in particular, was a huge help with explaining deep learning and machine learning techniques, managing git repositories and Ace cluster and setting up pipelines for us to continue working on each sub-project systematically, while ML helped us out with the TDA and text generation facets of our project, providing some code to help run machine learning experiments with TDA data in particular. For the math portion of the MQP, we would like to thank Professor Randy Paffenroth for advising and providing support with the finer details of the mathematical writing process.

Next, we would like to thank the computer science department of WPI as a whole, for allowing us to use their resources for our experiments. James Kingsley must be thanked, for helping us to figure out how to configure the necessary resources to run experiments on the ACE and Turing clusters. We wouldn't be able to complete our experiments without being able to run on the ACE and Turing clusters.

Finally, we would like to thank a few specific individuals. Weili Nie, one of the head developers of RelGAN, was incredibly helpful while we were trying to figure out how to run RelGAN with our data set, and we would like to thank him for that. We would also like to thank Meryll Dindin for devloping the TdaToolbox on github, which served as a basis for our Betti curve generation[1].

[1] https://github.com/Coricos/TdaToolbox

Contents

List of Figures

List of Tables

1 Introduction

Depression is one of the most common mental disorders in the world, effecting over 300 million people worldwide (Marcus et al., 2012). Beyond just shorter-term emotional effects, long term moderate or severe depression can cause decreased function and impairment. At its very worst, depression can lead to suicide, one of the leading causes of death among 15-29 year-olds (WHO, 2018). There exists a range of effective treatments for depression, such as therapy and antidepressants, but due to stigmatization of mental disorders and a lack of resources, these are underutilized. Inaccurate assessment is another barrier to treatment, as people who are depressed are not always correctly diagnosed, and people who are not depressed are frequently misdiagnosed (WHO, 2018).

According to a 2019 survey by the Pew Research Center, 81% of adult Americans own a smartphone (Anderson, 2019). Modern smartphones carry a wide range of sensors that can be used for gathering data, which could aid in the diagnosis of depression. Furthermore, the enormous amounts of data gathered by social media platforms could be used to assist the diagnosis of depression. We used both data collected from smartphones from the EMU and Moodable data sets for our experimentation, as well as data from the Distress Analysis Interview Corpus (DAIC) for our research.

Diagnosis of depression is a very challenging problem, and we strove to solve this using both well established and novel machine learning methods. Our machine learning methods included Support Vector Machine, Random Forest, XGBoost, AdaptiveBoosting, k-Nearest Neighbours, and Multilayer Perceptron. The novel methods included Convolutional Neural Networks (CNN), and Long Short-Term Memory (LSTM). We also experimented with novel feature extraction techniques like topological data analysis (TDA) to see if the shape of data may provide indicators for depression. We hypothesises that we will be able to use these methods to predict mental health issues such as depression or anxiety.

1.1 Related Works

There exists a range of studies before us that use machine learning or deep learning to identify depression. A review (Guntuku et al., 2017) concisely

shows the various methods used to identify depression using data from various social media platforms. Figure 1 shows the findings of this review, including the sources of the data, the types of models used, and the performance of those models. One very important thing to note about this study is that it

Ref.	Year	Dataset			Section	Mental Illness Criteria	Features (predictors)						Outcome Type	Model	Metric	Performance
		Platform	N (users)	Cases (conditions; base rate [BR])			n-grams	LIWC	Sentiment	Topics	Metadata	Others				
[8]	2013	Twitter	476	Depression = 171 (BR = 36%)	A	survey (CESD + BDI)		Y	Y		Y	Social Network	Binary	PCA, SVM w/ RBF Kernel	Accuracy	.72
[13]	2014	Facebook	165	Post-partum Depression = 28 (BR = 17%)	A	survey (PHQ-9)		Y	Y		Y	User Activity, Social Capital	Binary	Logistic Regression	pseudo-R2[b]	.36
[14]	2014	Facebook	28,749	(continous Depression score)	A	survey (Personality)	Y	Y		Y			Continuous	Ridge Regression	Correlation	.38
[12]	2015	Twitter	209	Depression = 81 (BR = 39%)	A	survey (CESD)	Y	Y	Y	Y	Y	User Activity	Binary	SVM	Accuracy	.69
[11]	2016	Twitter	378	Depression = 105 (BR = 28%) PTSD = 63 (BR = 17%)	A	survey (CESD)		Y	Y		Y	Time-Series, LabMT	Binary	Random Forests	AUC	Depression = .87 PTSD = .89
[40]	2014	Twitter	5,972	PTSD = 244 (BR = 4%)	B	self-declared	Y	Y					Binary	(not reported)	ROC	(AUC not reported)
[42]	2014	Twitter	21,866	11,866 (across 4 Conditions, BR = 54%)	B	self-declared	Y	Y	Y		Y	User Activity	Binary	Log linear classifier	Precision[a]	Depression = .48 Bipolar = .64 PTSD = .67 SAD = .42
[17]	2015	Twitter	1,957	Depression = 483 (BR = 25%) PTSD = 370 (BR = 19%)	B	self-declared	Y	Y	Y	Y		Age, Gender, Personality	Binary	Logistic Regression	AUC	Depression = .85 PTSD = .91
[21]	2015	Twitter	4,026	2,013 (across 10 Conditions, BR = 50%)	B	self-declared	Y	Y					Binary	(not reported)	Precision[a]	Depression = .48 Bipolar = .63 Anxiety = .85 Eating Dis. = .76
[41]	2016	Twitter	250	Suicide Attempt = 125 (BR = 50%)	B	self-declared	Y		Y		Y	User Activity	Binary	(not reported)	Precision[a]	.70
[43]	2016	Twitter	900	Depression = 326 (BR = 36%)	B	self-declared	Y						Binary	Naive Bayes	AUC	.70
[19]	2017	Twitter	9,611	4820 (across 8 Conditions, BR = 50%)	B	self-declared	Y					Gender	Multi-Task	Neural Network	AUC	Depression = .76 Bipolar = .75 Depression = .76 Suicide Attempt = .83

Figure 1: Results of the 2017 Review (Guntuku et al., 2017)

only uses text data from larger social networks. Another important finding corroborated by this study is the distinction between self-declared depression versus diagnosing with a survey: the machine learning models were able to more accurately identify depression when it was self-identified by each poster, as opposed to diagnosed with a survey. This study concludes by saying that the most potential in this field could be in diagnosing depression, which is where our particular research fits into the big picture.

Another study (Al Hanai et al., 2018) used voice and text data from a database related to the one we are using: DAIC. This study performed three different experiments, using both the text data and voice data independently, as well as together. The LSTM model that used both the text and audio performed much better than the other approaches, suggesting a potential way forward for our model, which only uses audio. This study is also one of

very few which actually use voice data, as text-based data sets from social media platforms are far more accessible.

Also in the area of Deep Learning, one study by (Ma et al., 2016) used DepAudioNet with its main components as CNN and LSTM to detect depression on DAIC-WOZ data set. This study showed that DepAudioNet could perform better than the previous averaging baseline in terms of F1 score. This study used both Mel-scale filter bank and spectrograms but not in conjunction.

There exists a much wider field of studies that attempted to diagnose depression from text-based posts on social media platforms. One particularly interesting Japanese study (Tsugawa et al., 2015) gathered data using the CES-D (Center for Epidemiological Studies Depression Scale) survey, answered by Japanese-speaking Twitter users. Furthermore, other posts by those users were also used, and had features extracted such as words, post topics, post frequency, post length. All of this data together were used to train a support-vector machine, which was able to get an accuracy of up to 69%. Furthermore, this study also found that it took approximately two months of observation to make an accurate prediction of depression, and that collecting data for longer than that did not improve the accuracy at all.

Other studies approach text-based depression diagnosis from other angles. One study (Resnik et al., 2015) experimented with new modeling techniques on both stream-of-consciousness essays and twitter posts, identifying individual words which were more or less likely to indicate depression. This study found that the accuracy of their models depended more on how data was aggregated, taking in post data from each user chronologically, to track emotional trends over time.

1.2 Our Approach

In this study, we attempted to improve on the detection of depression and anxiety using audio. We also explored how text messages and transcript data can be paired with audio data to improve detection. Machine learning was implemented for regression and classification, for both depression and anxiety. Convolutional and sequential deep learning models were both utilized

for depression classification.

1.2.1 Machine Learning

Much research involved with depression and emotional detection have used Machine Learning methods with relatively positive outcomes (Valstar et al., 2016). As a result, we opted to experiment with different machine learning models for classifying audio features and transcript features for our research. Specifically, we experimented with ADABoost, XGBoost, Random Forest, K-Nearest Neighbors, Multilayer Perceptron, and Support Vector Machine models. In addition to classification analysis, we used these models for regression analysis, tuning model parameters to maximize performance. We set up a pipeline to run our experiments throughout the research.

1.2.2 Convolutional Neural Network

Convolutional Neural Network (CNN) has been widely used as a deep learning method for image classification. Since the data set we are going to use consists of large number of audio clips, CNN is one of the crucial methods for classifying spectrograms derived from those audio clips. Previous studies such as (Ma et al., 2016) involve using CNN in comparison to other methods such as LSTM and Machine Learning techniques to predict depression from audio sources. Therefore, in this research we will set up a baseline CNN model and continually improve our model to find efficient parameters which will help us predict depression.

1.2.3 Sequential Deep Learning Model

Sequence modelling, and long short-term memory (LSTM) models specifically, have shown promising results for diagnosing depression. Voice recordings can also be easily treated as sequential data in a variety of ways. We created an LSTM-based model to further explore optimal architectures and attempt to improve on previous LSTMs used to predict depression from voice audio. We compare our results with other deep learning and machine learning techniques to find the best solution for depression classification. We also researched and outlined possible avenues of future improvement for our sequence models.

1.2.4 GANs

SMS text messages have a very promising modality for detecting depression, but previous studies have shown that only 33% of participants are willing to have their SMS data collected (Dogrucu et al., 2018). This makes it difficult for enough data to be collected for use in deep learning. Fortunately, generative adversarial networks (GANs) look like a very promising way to artificially expand sets of text data (Yu et al., 2016). Thus, we set up a framework to efficiently run GAN experiments to test the feasibility of using GANs to expand existing text message data sets.

1.2.5 Topological Data Analysis

Our main motivation for using topological data analysis is to see how it may compare to other forms of feature extraction on audio waves. TDA has not been used much for analysing audio data, and very few studies in recent memory has used TDA to analyze time series. One study used TDA with Betti curves (see section 2.10.5) to categorize and find heart arrhythmias based upon heart beats (Dindin et al., 2019). As seen by Figure 2, this study has relatively good results compared to not using TDA. However, this study also used TDA as part of a multi-modal approach which used TDA as another feature set to consider. Because this study has good results, and because both sound and heart beats are represented by waves, this lead us to see if this method would also be applicable to classifying audio data.

ID	Arrhythmia Detection		Arrhythmia Classification	
	With TDA	*Without TDA*	*With TDA*	*Without TDA*
0	**0.99**	0.98	**0.73**	0.68
1	**0.96**	0.90	**0.75**	0.69
2	0.85	**0.86**	**0.68**	0.65
3	0.94	**0.95**	0.95	**0.96**
4	**0.85**	0.80	**0.97**	0.97
5	**0.87**	0.77	**0.96**	0.93
6	0.78	**0.80**	**0.94**	0.93
7	**0.81**	0.63	**0.90**	0.80
8	**0.79**	0.65	**0.85**	0.78
9	0.84	**0.86**	**0.68**	0.47

Figure 2: Results of Arrhythmia Study (Dindin et al., 2019)

2 Background

2.1 Previous MQPs

2.1.1 2018 MQP summary

The 2018 MQP started out by doing a study on how willing users would be to share their data. The research showed that people would be most willing to share audio and camera data.

Using the study, the team proceeded to build and deploy an application on Amazon's Mechanical Turk (MTurk), called Moodable. The purpose of Moodable was to collect user data two weeks prior to when the application was downloaded. The application collected the users' texts, social media data, geospatial data, and voice samples. The users also filled out a Patient Health Questionnaire-9 (PHQ-9), a standard 9 question survey used to assist medical professionals in depression diagnosis. The team used 85 percent of their data to train the models and the remaining 15 percent to test them. They were able to achieve "an average test set root mean square error of 5.67 across all modalities in the task of PHQ-9 score predictions" (Dogrucu et al., 2018).

2.1.2 2019 MQP summary

The 2019 MQP deployed an application on AmazonTurk called EMU, which: "... collect[ed] demographic information, PHQ-9, GAD-7, basic phone data (which includes text messages, call logs, calendar, and contacts), Google Maps location data, voice recordings, Instagram posts and tweets from Twitter" (Resom et al., 2019). From these, the team focused on the "audio, text messages, social media data, as well as GPS modalities" (Resom et al., 2019).

For their text data, they extracted features using Empath, Linguistic Inquiry and Word Count, Textblob, and using manual extraction methods. The highest F1 score they attained using these features was 0.83. For their GPS data, they got features based on activity data and raw GPS data and were able to get an F1 score of 0.63. For their audio data, they extracted pause time features using a Pratt script and the signal based features using

openSMILE. They found that they got the best results by using the XGBoost algorithms, "50 features for feature selection, and using the data set from openSMILE"(Resom et al., 2019).

2.2 Depression Statistics

Mental health is an ever evolving field, where methods for defining depression are frequently refined. Because of this, older studies tend to not have information that is as reliable and up-to-date as more recent ones. Self identified depression versus tested depression also tends to lead to wildly different results. For example, in 2001 a study was performed where 1433 students were asked if they had depression, and the severity of the depression was not measured (Furr et al., 2001). In contrast, a 2010 study found that 17% of students were depressed and 9% were depressed via using the Patient Health Questionnaire with 9 questions, or PHQ-9 (Hunt and Eisenberg, 2010). This was up from 10% depressed in a previous study from 2000, which also used the PHQ-9 test. Notably, only 24% of the students diagnosed with depression in the 2010 study were receiving any treatment. From this data, it is easy to see that depression is a major problem in colleges that seems to only be getting more prevalent. Since self diagnosis seems to be unreliable compared to actual tests, it would be useful to have a way to automatically determine if someone was depressed using a framework like PHQ-9.

2.2.1 Patient Health Questionnaire

The Patient Health Questionnaire, or the PHQ, is a test to determine is somebody is depressed based on a series of questions, ranked 0-3. The answers are then added up to give a PHQ score. The most common test had nine questions, and is thus known as the PHQ-9. From these questions, a level of depression can be determined. This test has been shown to be relatively accurate in categorizing if people have depression or not, as shown in Figure 3. For some of the data sets used in our project, the PHQ-8 is used instead of the PHQ-9. The PHQ-8 is identical to the PHQ-9, with the same ranges of score, but with the question about suicidal ideation omitted. This questionnaire is sometimes used instead of the PHQ-9, as if a participant in a study says that they have suicidal thoughts, one is often required to report it, so the question is thus omitted for legal and ethical reasons.

Table 2. Distribution of PHQ-9 Scores According to Depression Diagnostic Status*

Level of Depression Severity, PHQ-9 Score	Major Depressive Disorder (*N* = 41) *n* (%)	Other Depressive Disorder (*N* = 65) *n* (%)	No Depressive Disorder (*N* = 474) *n* (%)
Minimal, 0–4	1 (2.4)	8 (12.3)	348 (73.4)
Mild, 5–9	4 (9.8)	23 (35.4)	93 (19.6)
Moderate, 10–14	8 (19.5)	17 (26.1)	23 (4.9)
Moderately severe, 15–19	14 (34.1)	14 (21.5)	8 (1.7)
Severe, 20–27	14 (34.1)	3 (4.6)	2 (0.4)

* *Depression diagnostic status was determined in 580 primary care patients by having a mental health professional who was blinded to the PHQ-9 score administer a structured psychiatric interview.*

Figure 3: PHQ-9 Accuracy from 2009 study (Kroenke et al., 2009)

2.2.2 General Anxiety Disorder

Like the PHQ, the General Anxiety Disorder (GAD) is a questionnaire based rating system; it ranks every response from 0 to 3. However, it is used as a screening tool and severity measure for generalised anxiety disorder rather than depression. The response ratings are summed to determine a GAD-7 score which ranges from 0 to 21. Scores of 5, 10, and 15 represent cut-points for mild, moderate, and severe anxiety, respectively (Spitzer et al., 2006). When used as a screening tool, further evaluation is recommended when the score is 10 or greater (Spitzer et al., 2006).

2.3 Data Sources

For our project, we are looking at using multiple data sets for training different models. These include DAIC-WOZ, EMU and Moodable. As part of our research, we looked at what is contained in these data sets, and how they have been used in the past.

2.3.1 DAIC-WOZ

DAIC-WOZ stands for Distress Analysis Interview Corpus, Wizard of Oz. DAIC is a series of clinical interviews which served to diagnose depression, anxiety, PTSD, and other disorders. DAIC-WOZ is a subset of this data set in which a virtual interviewer named Ellie is used to ask questions, hence the name Wizard of Oz. These interviews each are given a PHQ-8 score based on answers to the interviewer's questions. The publicly available portion of the data set contains 189 interviews. This contains audio of the interviews, associated transcripts, and facial data.

A recent use for the DAIC-WOZ data set was with the Audio/Visual Emotion Challenge and Workshop (AVEC). Specifcally, AVEC had a challenge for 2016 and 2017 to detect depression via training on the DAIC-WOZ data set (Ringeval et al., 2017). For teams participating in this challenge, COVAREP, a speech analysis program, was used to extract audio features for input into a model. AVEC also provided a histogram of the DAIC-WOZ data, showing the average scores across the data set (Figure 4). The average PHQ-8 score was 6.67 for this data set. DAIC also provides baseline performance metrics for classifying depression using their data. Using a Support Vector Machine model for binary classification, they list an F1 score of 0.46, a precision of 0.32 and a recall of 0.86 (Valstar et al., 2016).

Many studies using DAIC-WOZ are part of the AVEC challenge, either 2016 or 2017. One such study, DepAudioNet, used an approach with Deep Convolutional Neural Networks (DCNN), and Long Short-Term Memory (LSTM) (These are explained in more detail in Section 2.8). This was done with the goal of producing a more comprehensive audio representation within the model (Ma et al., 2016). For this study, only the audio was used, using a mel-spectrogram input for the model. The model used in this study had a binary output, with an F1 score of 0.52, a precision of 0.35 and a recall of 1. In 2017, a team also used a DCNN in a multimodal approach which took into account both audio, video, and text data sets in separate models, before combining the results into a central model (Yang et al., 2017). This model outputted a PHQ-8 score, instead of a binary output, and had a RMSE of 5.974 on the testing set. Another 2017 study used a support vector machine for AVEC, using audio, video, and text data (Dham et al., 2017). Also outputting a PHQ-8 score, this model had a RMSE of 5.3586.

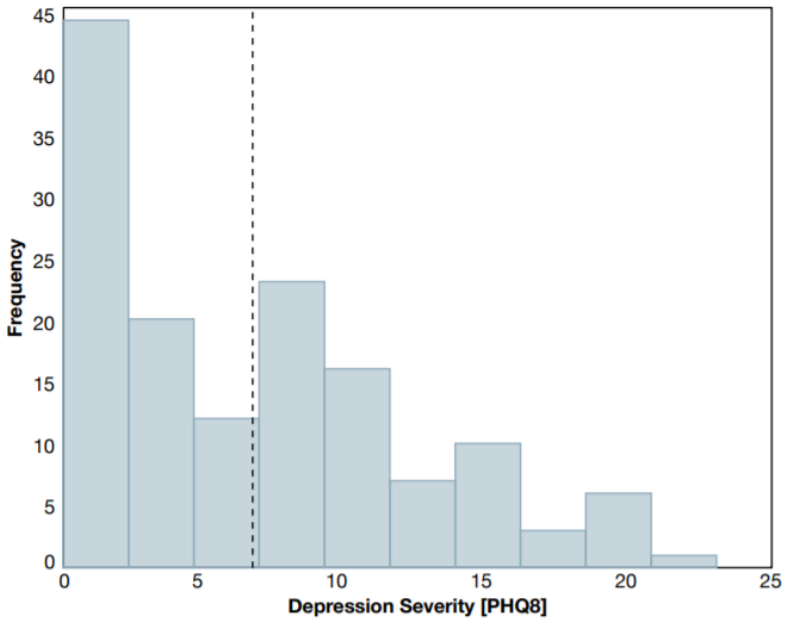


Figure 4: Histogram of PHQ-8 scores from AVEC data set (Ringeval et al., 2017)

Lastly, the paper by Al Hanai et al. (2018) in Interspeech has shown what appear to be the best results so far for sequence modelling on DAIC audio features. Their audio-based LSTM model showed an F1 of 0.63, a precision of 0.71, and a recall of 0.56. We use these numbers as the main comparison for our sequence model. With multi-modal input of both audio features and transcripts, their sequence model acheived an F1 of 0.77, a 0.71 precision, 0.83 recall, an MAE score of 5.10 and an RMSE score of 6.37.

2.3.2 Moodable

The 2018 Major Qualifying Project team, Sensing Depression, deployed the application Moodable on Amazon Turk. Moodable collected "user data (texts, social media content, geospatial data, and voice samples that are two weeks prior to the point when they give consent to the application) on the spot"(Dogrucu et al., 2018).

For audio data gathering, the team had users read out the phrase "the quick brown fox jumps over the lazy dog." After the responses were filtered, and those deemed as unusable were removed, there remained 230 recordings.

They collected the PHQ-9 scores of the users, so the data set has the corresponding PHQ-9 score of the users in the recordings.

2.3.3 EMU

The EMU data set refers to the data set collected by the 2019 Major Qualifying Project: Machine Learning for Mental Health Detection. They deployed the application EMU on Amazon Mechanical Turk and collected audio, text messages, social media and GPS data from the user.

For the audio data, the team had users read out the phrase "that which we call a rose by any other word would smell as sweet" (Resom et al., 2019). They also had users respond to the open-ended question "Describe a good friend" and restricted the response to 15-30 seconds. In total, after removing the unusable responses, the team has 60 responses for the EMU audio data set.

Like the 2018 team, they collected the PHQ-9 scores of the users, so the data set has the corresponding PHQ-9 score of the users in the recordings.

2.4 Dimensionality Reduction Techniques

Dimensionality reduction techniques provide ways to reduce the size and complexity of data while retaining the most important parts of them. This is especially important with machine learning or deep learning, as having a feature set that is too large may lead to a model running for too long to get useful results. The primary methods we have looked at are Principle Component Analysis (PCA) and Non-negative Matrix Factorization. Both these methods are based in mathematics and provide different ways to reduce the dimension of data for use.

2.4.1 Principle Component Analysis (PCA)

Principle Component Analysis (PCA) is a dimensionality reduction technique based around performing an orthogonal transformation to align data with a covariance matrix. An example of this can be seen with Figure 5. The covariance matrix shows which parts of the data have the most variance, and from this the principle components of the data can be found. To then reduce the dimension the data, the components with the least variance are dropped. One place where PCA is used in our project is with reducing the dimension of the openSMILE feature set (Section 3.2.2). Since this data set is very large, it is necessary to use this technique to be able to use the data efficiently.

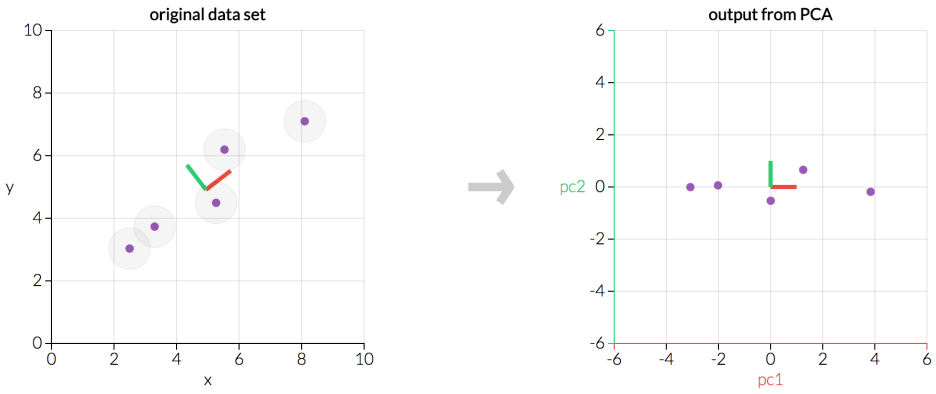


Figure 5: PCA on a small data set (Brems, 2019)

2.4.2 Non-Negative Matrix Factorization (NMF)

Non-Negative Matrix Factorization is a dimensionality reduction technique based around data which can be represented as a matrix. A matrix of data is factored into two discrete matrices, which isolate characteristics of the data into two disjoint sets. These two matrices are known as a features (or components) matrix and a coefficients (or activations) matrix, and when combined provide a very good approximation for the original data. Because these matrices are factors of the original data, the dimension of each is much smaller than the original data. Usually, when using NMF, only the features matrix is kept.

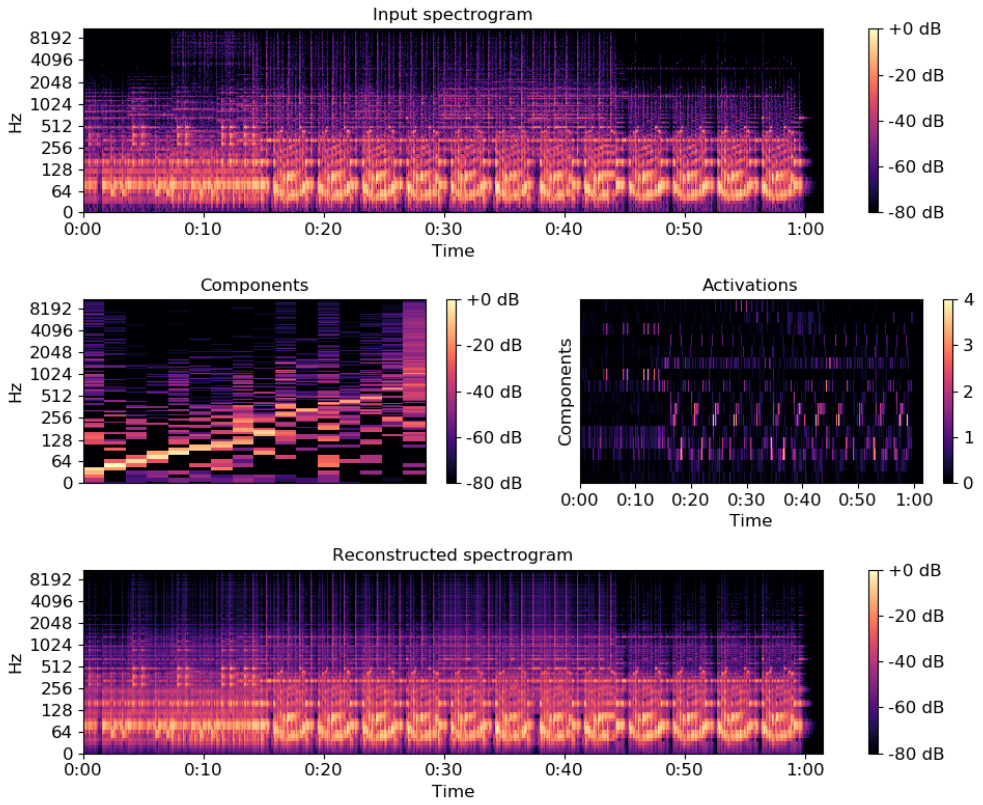


Figure 6: NMF on an audio spectrogram (Librosa Development Team, 2019)

One good use for NMF is dimensional reduction on audio spectrograms. This would be very useful to use in our project in the future to reduce the size of the spectrograms we use, while retaining the majority of the information. This can be done easily through using librosa.decompose.decompose (Librosa Development Team, 2019).

2.5 Data Balancing

Most real world data are rarely perfectly balanced. In fact, usually when collecting data, there should be a certain level of class imbalance expected. These inconsistencies with the number of instances available for classes are the result of the difficulties posed for gathering data for events that have rare occurrence patterns. For example, in data sets which are used to characterize

fraudulent transactions, there will likely be imbalanced as most transactions will naturally swing towards being non-fraudulent transactions. This leads to class imbalance problem (Batista et al., 2004), which is the challenge of learning from a class which has fewer instances. Imbalanced data compromise most machine learning algorithm's effectiveness as these models generally expect to inherit balanced class distribution (Guillaume Lemaˆıtre, 2017). Some common approaches to balancing our data include under sampling, oversampling, and synthetic sampling.

2.5.1 Under sampling

Under sampling is a type of data re-sampling technique where the instances of majority classes are reduced through random selection. Under sampling has shown to be a powerful performance booster if there the class imbalance within the data set. Compared with over-sampling, one advantage is that under-sampling generates a smaller balanced training sample thereby reducing the training time (ZhiYong Lin, 2009).

Drawbacks to this approach are the possibility that the reduced data set could be a biased one. This approach is also susceptible to loosing relevant data. However, through combining undersampling with methods like ensemble learning, the degradation of the classifier's performance can be mitigated (ZhiYong Lin, 2009).

2.5.2 Oversampling

Oversampling is an approach used to expand data set available to our minority class through random duplication. Contrary to undersampling, all original data set is retained which prevents important data from being lost. Though the retention of all original data set, when oversampling the danger arises with the possibility that the data could be overfit (Martínez-Trinidad, 2013).

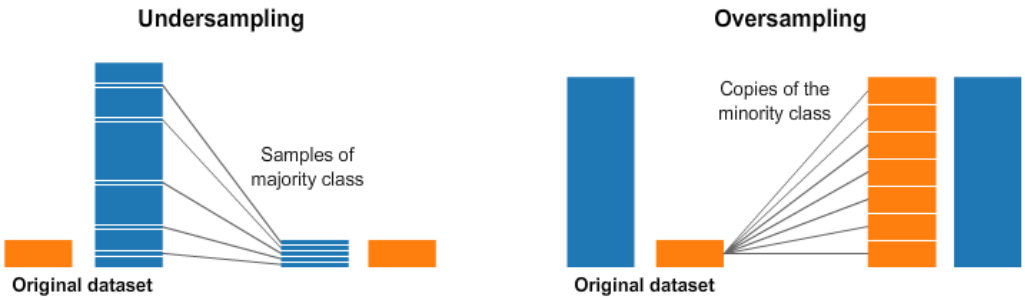


Figure 7: Resampling of data set
(Darji, 2019)

2.5.3 Synthetic Minority Oversampling Technique

Synthetic Minority Oversampling Technique, commonly known as SMOTE, is a technique based on nearest neighbors judged by Euclidean Distance between data points in feature space. The minority class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors.

Synthetic samples are generated in the following way: Take the difference between the feature vector and its nearest neighbor. Multiply this difference by a random number between 0 and 1, and add it to the feature vector. This causes the selection of a random point along the line segment between two specific features. This approach effectively forces the decision region of the minority class to become more general (Chawla, 2002).

2.6 Evaluation Metrics

We can classify data into multiple types of classification: binary, multi-class (categorical), and numerical. To evaluate the effectiveness of our models' predictions, we use evaluation metrics. Evaluation metrics are measurement tools that gauge the performance of our models. These metrics are used both in the training phase and testing phase.

2.6.1 Confusion Matrix

Confusion Matrices are tables constructed with the intersection of the predicted and actual values, used to visualize the performance of a model. When

the model predicts an instance accurately we nominate that prediction as true. Similarly, we nominate incorrect predictions as false.

Depending on the actual value of the instance that was predicted by models, we label the predicted values as positives and negatives. In most cases, the positive class is the area of interest in our problem. For instance, in this research project we would label actual depressed participants as positive, and non depressed participants as negative. Accordingly, the True Positives (TP) and False Positives (FP) would represent the accurate and incorrect predictions of a depressed (positive class) instances, respectively. While the True Negative (TN) and False Negative (FN) are the accurate and incorrect predictions of a non-depressed (negative class) instances, respectively. The counts of TP, TN, FP and FN within our confusion matrix are used to derive many important evaluation metrics (Hossin and Sulaiman, 2015).

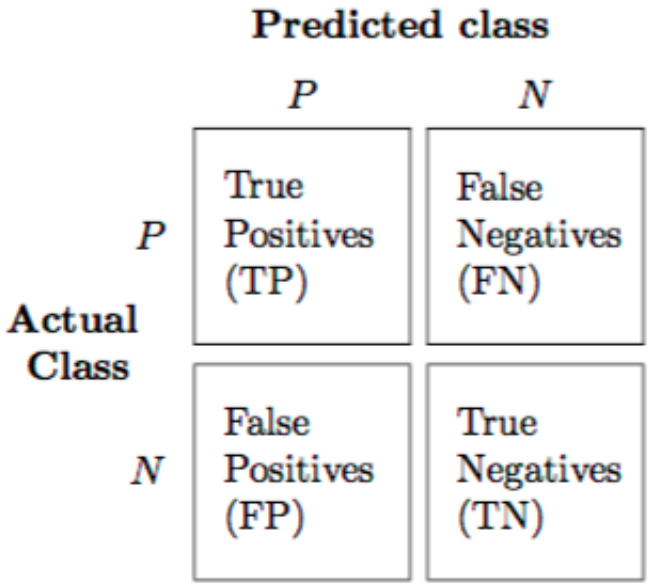


Figure 8: Confusion Matrix

2.6.2 Accuracy

Accuracy is the ratio correct predictions to the total predictions made. It is the sum of the TP and TN divided by the sum of the TP, TN, FP, and

FN. Although accuracy can often hide details of our models' performance and might not be a good indicator of efficient model. Accuracy becomes insufficient when dealing with imbalanced classes. For instance, You may achieve accuracy of 0.9 or more, but this is not a good score if 90 records for every 100 belong to one class and you can achieve this score by always predicting the most common class value (Hossin and Sulaiman, 2015).

2.6.3 Precision

Precision is a measurement that indicates how many of the predictions for the positive class instances were correct. In Mathematical terms, it is the division of TP by the sum of TP and FP (Hossin and Sulaiman, 2015).

2.6.4 Sensitivity

Sensitivity is the positive rate. Also referred to as recall, sensitivity represents the measure of the proportion of actual positives that are correctly identified as such. For instance, in this case study, sensitivity would show the percentage of depressed participants from the population that are correctly predicted. It is the division of TP by the sum of TP and FN (Hossin and Sulaiman, 2015).

2.6.5 Specificity

Specificity is the negative rate. It is the complementary metric to sensitivity. Specificity measures the proportion of actual negatives that are correctly identified as such. In this case study, it would represent the percentage of non depressed participants from the population that are correctly predicted as non depressed. It is the division of TN by the sum of TN and FP (Hossin and Sulaiman, 2015).

2.6.6 F1-Score

F1-Score represents the harmonic mean between recall and precision values. It gives a better measure of the incorrectly classified cases when compared to Accuracy values. They are better evaluation metrics in cases where the data set has imbalanced classes. In addition, F1-score outperforms accuracy metric when the False Negatives and False Positives are costly (Hossin and Sulaiman, 2015).

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

Figure 9: F1-Score

2.6.7 Area under the ROC Curve (AUC)

AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. The closer AUC for a model comes to 1, the better it is. So models with higher AUCs are preferred over those with lower AUCs (Flach, 2003).

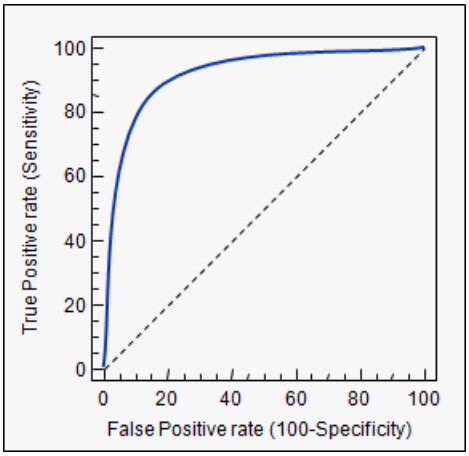


Figure 10: AUC-ROC graph

As shown in the figure above, we use Sensitivity as our Y-axis, and Specificity as our X-axis.

2.6.8 Mean Squared Error (MSE)

MSE measures average squared error of predictions. For each point, it calculates square difference between the predictions and the target and then average those values. MSE are useful in evaluating regression models, as the higher this value, the worse the model is. It is never negative, and would be zero for a perfect model (Drakos, 2018).

MSE generally should be supplemented with other metrics. MSE tends to penalize the model harshly for incorrect classification of noisy data, and leads to high overemphasis of the model's mistake. Likewise, it can make a model underestimate a model's mistake if all the errors are small (Drakos, 2018).

$$\text{MSE} = \frac{1}{N}\sum_{i=0}^{N}(y_i - \hat{y}_i)^2$$

Figure 11: Mean Square Error Formula

In the figure above, the y is the actual value of our target while $\hat{y}$ represents the predicted values.

2.6.9 Root Mean Squared Error (RMSE)

RMSE is simply the square root of MSE. The square root is introduced to make scale of the errors to be the same as the scale of targets (Drakos, 2018).

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=0}^{N}(y_i - \hat{y}_i)^2}$$

Figure 12: Root Mean Square Error Formula

In the figure above, the y is the actual value of our target while $\hat{y}$ represents the predicted values.

2.6.10 Mean Absolute Error (MAE)

Mean Absolute Error or MAE is the average of the absolute value difference between the actual target values and the predicted values. This metric is helpful in evaluating regression models. It is a linear score, since all the individual differences are weighted equally. Compared to MSE, MAE penalizes huge errors less significantly as a result it's less sensitive to outliers. In general, we should favor MAE instead of MSE for data sets with outliers, however, if those are actually unexpected values that are relevant it is better to use MSE (Drakos, 2018).

$$\text{MAE} = \frac{1}{N}\sum_{i=0}^{N}|y_i - \hat{y}_i|$$

Figure 13: Root Mean Absolute Error Formula

In the figure above, the y is the actual value of our target while $\hat{y}$ represents the predicted values.

2.6.11 R-Squared

The coefficient of determination, or R^2, is another metric used to evaluate regression models, It is built on the MSE metric. It's has the advantage of being scale free, as R^2 is always going to be between negative infinity and a

negative R^2 score signifies that our model is worse than predicting the mean (Drakos, 2018).

$$R^2 = 1 - \frac{\text{MSE(model)}}{\text{MSE(baseline)}}$$

Figure 14: R^2

In the figure, we have two variations of the MSE metric. The MSE model version is the MSE metric as described in the previous section. However, the MSE baseline differs as it uses the mean of the observed targets instead of the prediction from a model. In other words, $\hat{y}$ would represent the observed mean of the data.

2.7 Machine Learning

In this project, numerous machine learning models are utilized. These are kNN (K-nearest neighbors, XGB (XGBoost), RF (Random Forest), ADA (ADABoost), MLP (Multilayer Perceptron), and SVM (Support Vector Machines).

2.7.1 K-Nearest Neighbors

K-nearest neighbors is a supervised machine learning algorithms that works for both classification and regression problems. The principle of this algorithm is that it classifies data based on the values of the closest k-neighbors. For classification, it uses the most frequent value from the k nearest neighbors. For regression, it averages the values from the k nearest neighbors (Harrison, 2018).

The value of k is determined by the user. For example, in the image below, we are trying to determine whether the new piece is a square or triangle . If K was set to 1, the program would see that the closest neighbor is a square and the new example would set as such. If K was 3, then it would be set to triangle since from the three closest neighbors, two are triangles.

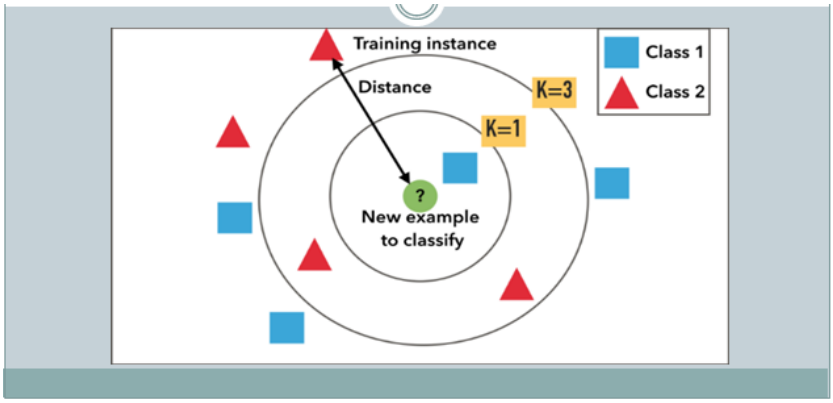


Figure 15: K Nearest Neighbors Diagram (Ali, 2018)

2.7.2 Support Vector Machines

Support Vector Machine is a algorithm that finds a hyper plane in an "N-dimensional space (N — the number of features) that distinctly classifies the data points" (Gandhi, 2018). In two dimensions, this is done using a line, while higher dimensions require hyper planes (Gandhi, 2018).

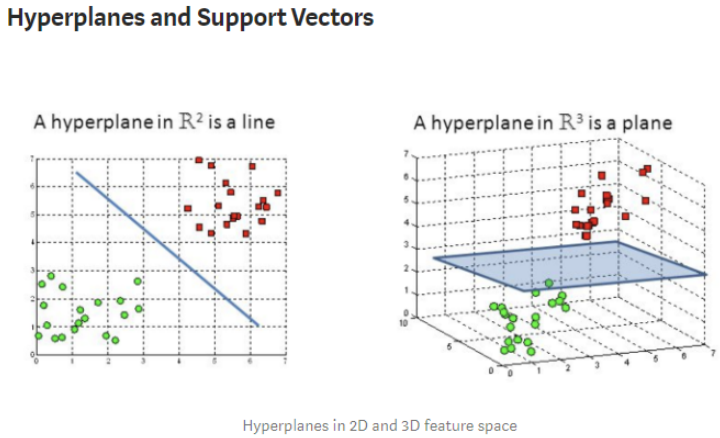


Figure 16: SVC Diagram (Gandhi, 2018)

2.7.3 Logistic Regression

Logistic Regression is an algorithm used for classification. It is used to frame a binary output (Varghese, 2018). There are three types of logistic regression: binary, multi and ordinal. Binary logistic regression refers to having two possible outputs, such as pass or fail. Multi logistic regression refers to having multiple outputs such as cats, dogs, sheep. Ordinal logistic regression refers to having multiple outputs with order like low,medium, and high (Fortuner, 2017).

2.7.4 Artificial Neural Network

Artificial Neural Networks are, in their simplest form, imitations of the human brain's processing mechanisms. Like the human brain, these models are adaptive learners that provide the framework for different algorithms to collaborate and process complex data. These systems learn from the examples they process in order to perform in their required tasks instead of being programmed with specific instructions (Ahire, 2018).

ANNs consist of neurons, which are slightly based on neurons within the human brain, that can transmit signals amongst each other. In most implementations, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs. The link between these neurons are called edges. These edges have scores assigned to them which are dynamically adjusted as the system keeps learning (Ahire, 2018).

ANNs work in parallel, unlike most computer programs that typically run serially. These complex systems can be used for pattern recognition, data classification and even for applications where data is unclear. In addition, they can process fuzzy, noisy and incomplete data. ANN can create its own organization while learning. A normal program is fixed for its task and will not do anything other than what it is intended to do (Shiruru, 2016).

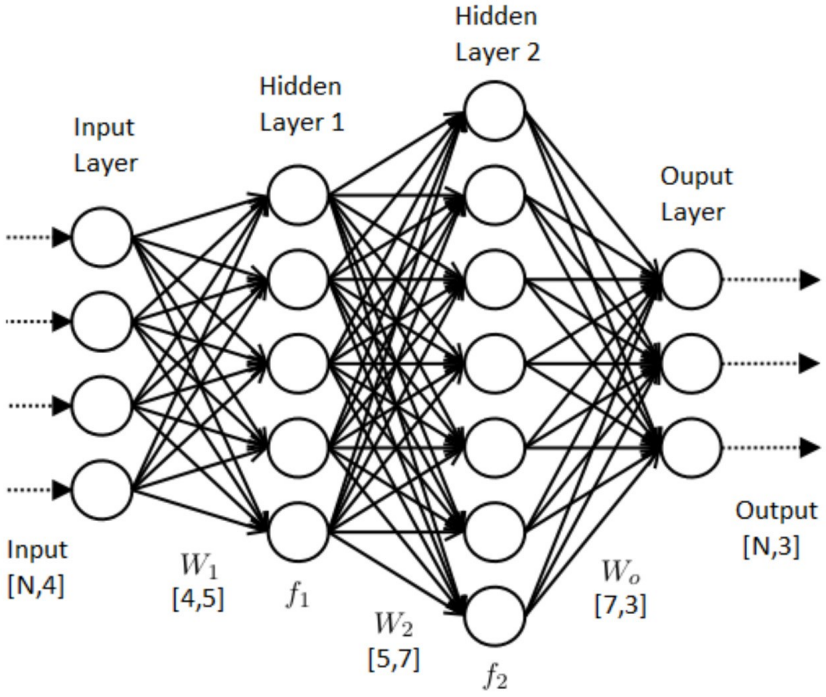


Figure 17: Artificial Neural Network Diagram

2.7.5 Random Forest and Decision Trees

Random Forests and Decision are similar algorithms, where the former builds upon the latter. As the name would suggest, Random Forests are congregations of Decision Trees. Decision Trees are tree-like graph structures, where each internal node represents a "test" on a attribute, each branch represents the outcomes of the test, and each leaf node represents a class label (Bhumika Gupta, 2017). Random Forests are efficient at solving regression or classification problems. The figure below shows a basic Decision Tree model.

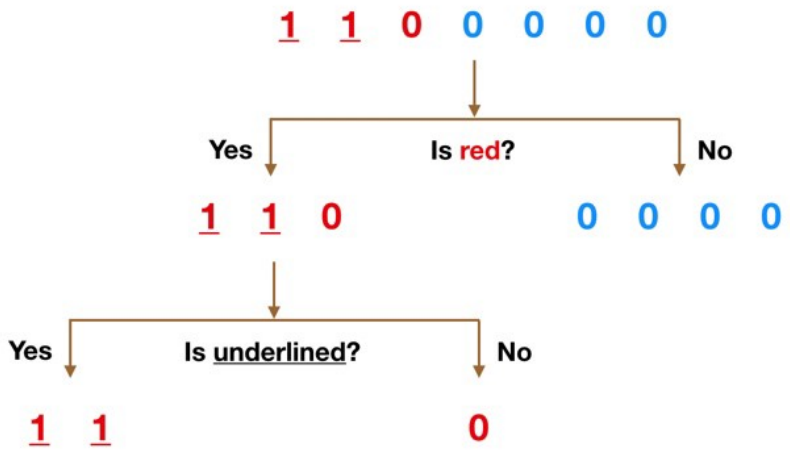


Figure 18: Simple Decision Tree

Common terms used with decision trees: **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets. **Splitting**: It is a process of dividing a node into two or more sub-nodes. **Decision Node**: When a sub-node splits into further sub-nodes, then it is called decision node. **Leaf/ Terminal Node**: Nodes do not split is called Leaf or Terminal node. **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting. **Branch / Sub-Tree:** A sub section of entire tree is called branch or sub-tree. **Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes whereas sub-nodes are the child of parent node. (Brid, 2018)

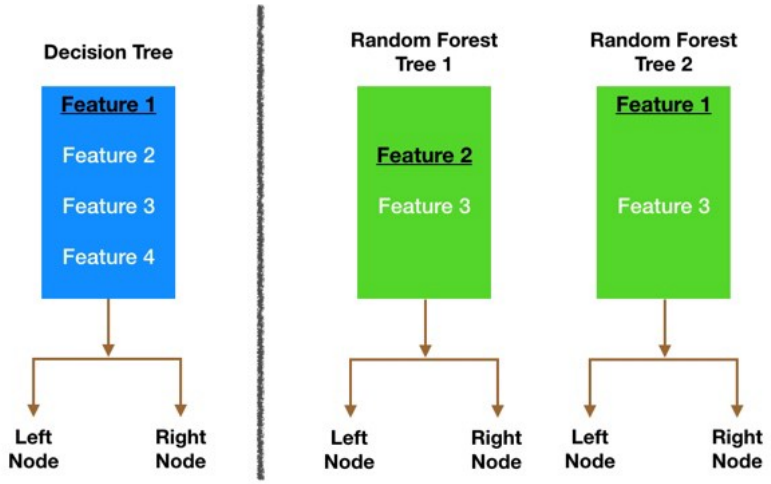


Figure 19: Random Forest vs Decision Tree

Random Forests, are congregations of decision trees. Each tree within the Random Forest(RF) outputs a class prediction, and the class with most predictions will be the model's prediction. The belief with RF is that the collection of trees will outperform an individual one. However, ensuring low correlation between aggregated models is vital. This aids in minimizing errors as all trees won't vote in very similar outcomes. To implement RF, there needs to be indicators in our features so that models built using those features outperform random guessing. In addition, like mentioned earlier, the predictions made by the individual trees need to have low correlations. In general, Random Forests are algorithms that use bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree (Yiu, 2018).

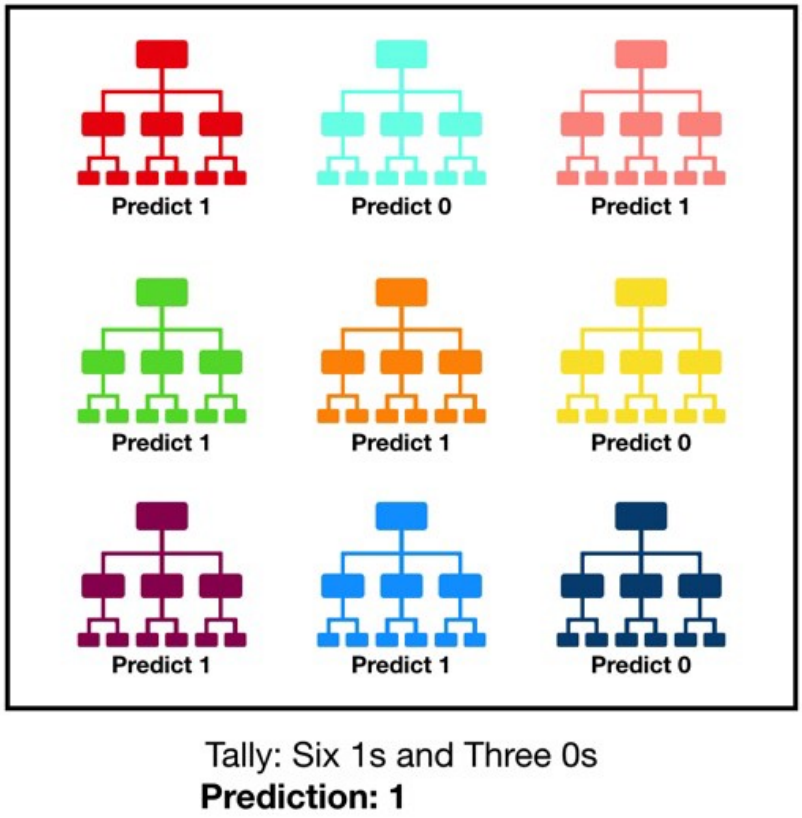


Figure 20: Random Forest Classifier

2.7.6 XGBoost

XGBoost is an ensemble learning technique like Random Forest. However, it is a boosting method unlike Random Forest - which is a bagging approach. In contrast to bagging techniques like Random Forest, in which trees are grown to their maximum extent, boosting makes use of trees with fewer splits (Abu-Rmileh, 2019).

XGBoost has three options for measuring features importance: Weight, Coverage, Gain. Coverage represents the amount of times a feature is used to split all the data across the set of tree weighted by amount of training data points that pass through those splits. Next, weight represents how many times a specific feature was used to split data across all trees. Lastly, gain represents the average training loss reduction gained when using a feature

for splitting. XGBoost is a system that is favorable for system optimization, as it helps alleviate computational limits to provide a scalable mechanism (Abu-Rmileh, 2019).

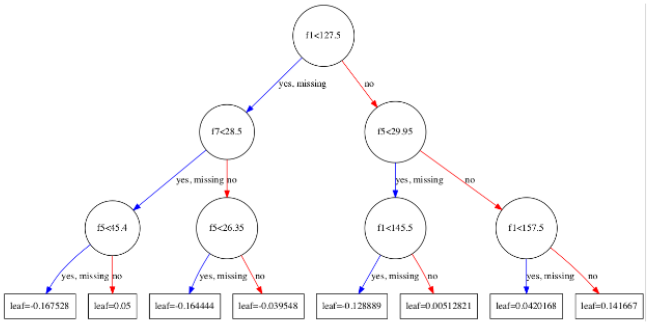


Figure 21: XGBoost Plot of Single Decision Tree

2.7.7 Voting

Voting is another type of ensemble learning method, used primarily for regression, which is are further split into types of voting techniques: majority voting and weighted voting.

Majority voting is an approach where every model makes a prediction for each test instance and the final prediction will be determined based on which instance receives more than half the votes. In cases where model can't have a prediction that gets more than half the votes, we conclude that the ensemble method wasn't able to make a stable prediction (Demir, 2015). Weighted voting differs from majority voting in that all models don't get same voting privileges. Some models will have a vote that weighs more, which is implemented by counting the predictions of the better models numerous times. Determining a reasonable set of weights is usually the challenge with this approach (Demir, 2015).

2.8 Deep Learning

We explored an array of different types of neural networks to find the ones best suited to test our hypothesis. Convolutional neural networks (CNNs), recurrent neural networks (RNNs), and long short term memory models (LSTMs) all showed the most promise. CNNs are best at assigning learnable weights and biases to various aspects of the input, and being able to differentiate one from another. RNNs excel when understanding the context of the input is critical. Taking a sequence of inputs, each computation is dependent on the computations that came before it. RNNs have frequently been used before for analyzing sound. As a special type of RNN, LSTMs can process data when there is a significant time gap in the data. Using recurrent gates, LSTMs manage their remembered information by determining which data should be remembered or forgotten at each step.

2.8.1 Convolutional Neural Network (CNN)

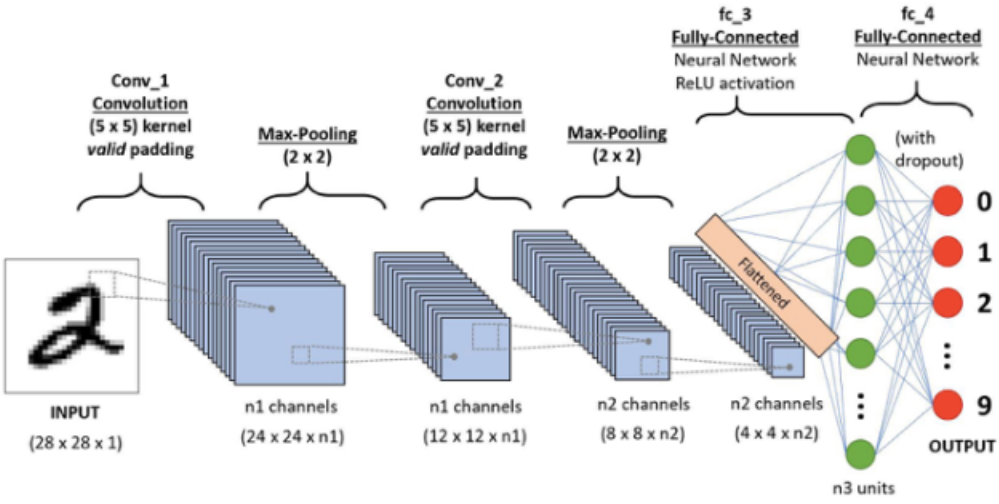


Figure 22: Convolutional Neural Network (Saha, 2018)

A Convolutional Neural Network or CNN is a deep learning algorithm which processes on an input image, assigns biases and weights to various features in the image, and classifies one from another. The role of a CNN is

to transform the images into a convoluted form to avoid the loss of important features. As a result, CNNs can give good predictions on test images (Saha, 2018). They also act as extensions to multilayer perceptrons, but the difference is that CNNs consist of convolution layers and pooling layers in addition to multilayer perceptrons (Piczak, 2015).

A convolutional layer is responsible for extracting features such as edges, shapes and colors from an input image. Usually the first convolutional layer will start by extracting low-level features. Gradually, the high-level features will be captured by the following convolutional layers. This convolutional layer works by maintaining hidden units. Each hidden unit processes only a tiny part of the complete two-dimensional input image data. The weights of each hidden unit create a convolutional kernel or filter, resulting in a convoluted form or a feature map (Piczak, 2015).

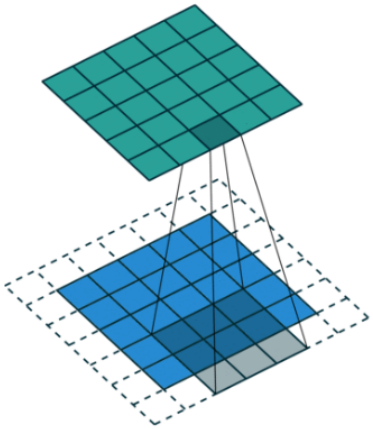


Figure 23: Operation of a Convolutional Layer (Saha, 2018)

A pooling layer is responsible for decreasing the dimensions of the feature maps obtained from the convolutional layer. This layer also reduces the computational power required to process the data (Saha, 2018). There are two types of pooling commonly used: max pooling and average pooling. Max pooling only picks up a maximum value from each feature map, while average pooling obtains a value by calculating the mean of each feature map.

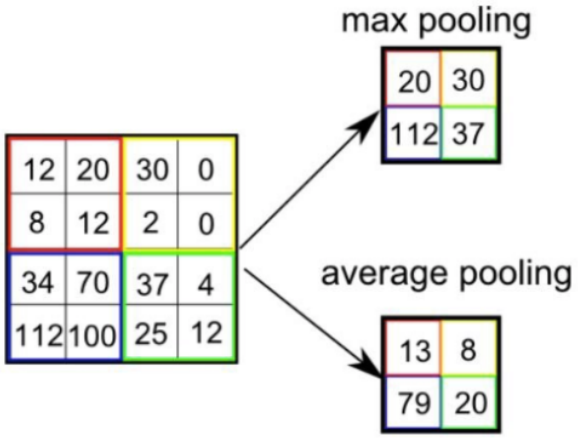


Figure 24: Operation of Max Pooling and Average Pooling Layers (Saha, 2018)

Finally, after pooling, the resulting output is flattened and sent into a fully-connected multilayered perceptron with feed-forward neural network and back-propagation. After training for a certain number of epochs, the model will be able to perform classification.

2.8.2 Sequence Modelling: Recurrent Neural Networks (RNN)

Sequence modelling is a type of data modelling that gives a computer the ability to predict or generate any type of sequential data. Typically, this includes text, speech or other audio, and video. Sequence modelling can be extremely powerful at predicting or generating complex sequences and has been applied to a wide variety of problem areas, including: music/text/image/speech generation, language translation, image captioning, stock trading, chatbots, sentiment analysis, and many others. Most modern sequence modelling techniques utilize deep learning. A common implementation of sequential modelling using deep learning is the Recurrent Neural Network (RNN).

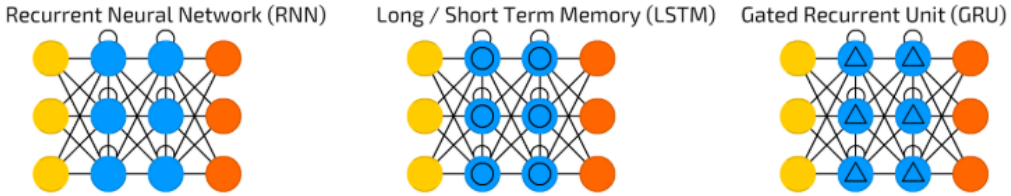
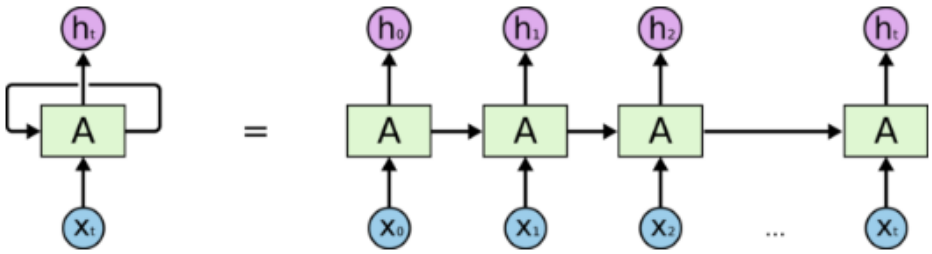


Figure 25: Topologies of RNN, LSTM, and GRU (van Veen, 2017)

RNN's are differentiated from regular neural networks because of their ability to store persistent information. Using this information as a sort of "context," RNN's can use the computations done on a previous element to influence the computations they preform on the current element (Banerjee, 2018).



An unrolled recurrent neural network.

Figure 26: An unrolled RNN (Banerjee, 2018)

As shown in Figure 26, an RNN can be represented by a loop that takes some input. The loop allows for data to be passed from one step to the next (Banerjee, 2018). By examining the unrolled RNN, it can be shown how each segment of the network, represented by A, takes one input element, outputs a result ht, then passes relevant information on to the next segment of the network (Banerjee, 2018).

Each cell in an RNN is connected not only through layers, but also time (van Veen, 2017). They store their previous values, and contain an extra weight connected by the previous values of the cells (van Veen, 2017). These weights between current and previous values have a certain "state" and can

vanish if not "fed" with new information after some time (van Veen, 2017). By working in this loop architecture, an RNN can operate over a sequence of input vectors, and output a sequence of vectors (Banerjee, 2018). This is a critical difference between RNN's and CNN's, which accept only a fixed size vector and output a fixed size vector.

In theory, an RNN should be able to remember context over any desired gap of time. However, in practice, this is not the case. Because the previous values are passed through an activation function, each updates passes this activated value through the activation function again (van Veen, 2017). This causes continuous information loss, and often times all the information can be lost after just four to five iterations (van Veen, 2017). This necessitates the use of other kinds of neural networks.

2.8.3 Sequence Modelling: Long Short Term Memory (LSTM) Models and Gated Recurrent Units (GRU)

Long short term memory models are a type of RNN that can remember relevant information longer than a regular RNN (Olah, 2015). As an example, while trying to predict the last word in the sentence "The clouds are in the *sky*," an RNN would be able to remember adequate context to predict "sky." However, predicting the last word of the sentence "I grew up in France, I speak fluent *French*," requires remembering context across a larger gap in time (or input) (Olah, 2015) which a vanilla RNN can struggle with. This is where LSTMs excel.

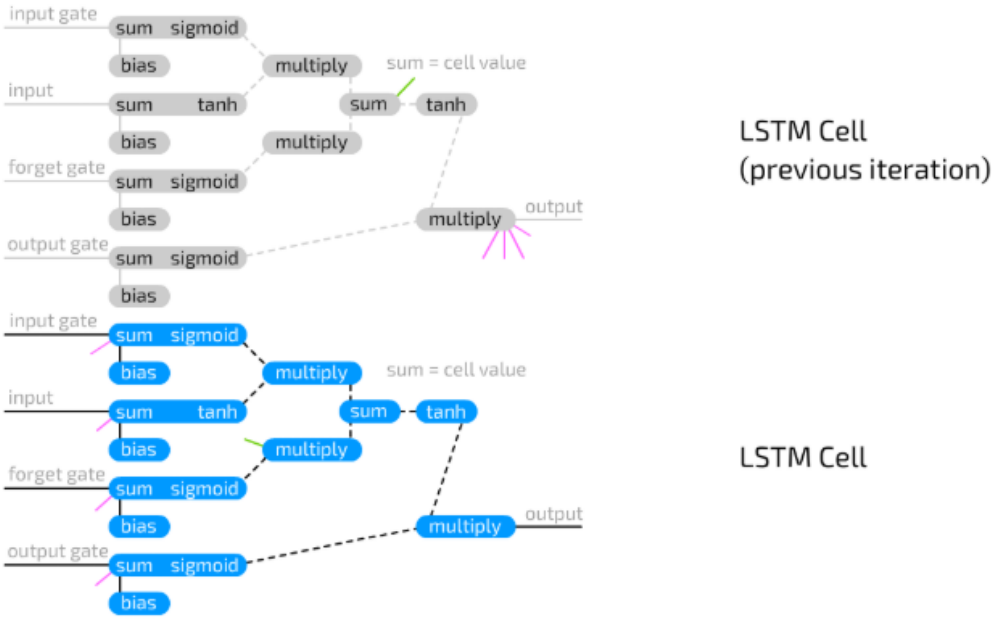


Figure 27: Inside an LSTM cell (van Veen, 2017)

LSTM cells are modelled after computer memory cells. A cell state runs through the entire LSTM chain, carrying information along (Olah, 2015). LSTM cell gates can let information through to the cell state. An LSTM cell has three gates: input, output, and forget (van Veen, 2017). These gates are composed of a sigmoid neural net layer and a pointwise multiplication operation (Olah, 2015). The sigmoid layer outputs a value from 0 to 1, determining how much information should be let through (Olah, 2015). Incoming information is multiplied by this value, and the input gate determines how much of the input will be let through (van Veen, 2017). The output gate determines how much of the output value can be seen by the rest of the network, and the forget gate determines how much of the last memory cell state to retain (van Veen, 2017). The forget gate is not connected to the output, and therefor far less information loss occurs than with an RNN (van Veen, 2017).

Gated recurrent units (GRU) are a form of LSTM cells. Using only an update and a reset gate, GRUs trade expressiveness with speed (van Veen,

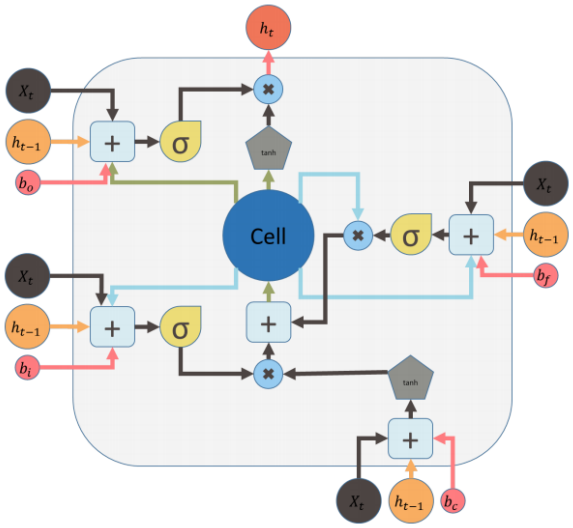


Figure 28: An LSTM cell (Ma et al., 2016)

2017). By abandoning the protected cell state and combining LSTM input and forget gates into the update gate, GRUs offer a slightly faster alternative to LSTMs (van Veen, 2017). LSTMs and GRUs are both valid options in most cases, and choosing between the two is often on a case-by-case scenario (Nguyen, 2018).

2.8.4 Feeding Audio Data into Neural Networks

The kinds of neural networks we considered using are clear, but we also needed to consider how to feed audio data into those neural networks. Convolutional neural networks are primarily used for processing images, but the study by Karol Piczak showed that CNNs can be used for audio classification purposes. In that study, the audio data were fed into neural network by means converting them into log-scaled mel-spectrogram. The mel-spectrogram is a spectrogram with Mel scale applied to its vertical axis (Piczak, 2015). However, using mel-spectrograms are not the only technique and, it was worth exploring other techniques that might have been useful.

The simplest way to feed audio data into neural networks is by transforming the raw wave form of an image into a one-dimensional array and utilizing that array for one-dimensional convolution. One drawback from using this

technique is that concepts of features such as edges and shapes are not present in this raw wave form. The study by Yuan Gong and Christian Poellabauer mentioned that it is unclear how neural networks learn by using raw audio wave (Gong and Poellabauer, 2018). Figure 29 shows how raw audio waves are processed by convolutional neural networks.

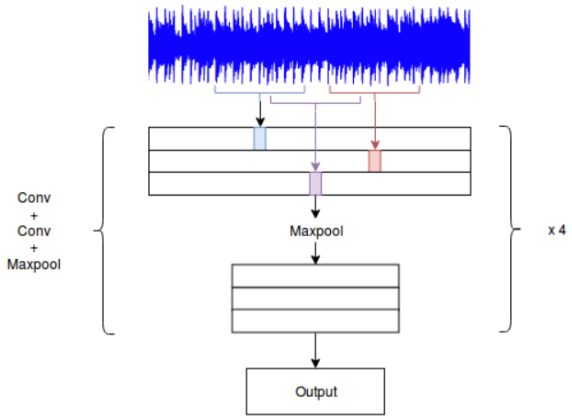


Figure 29: Processing raw audio wave (Mansar, 2018)

Another method of feeding audio data into neural networks is by using spectrograms. Spectrograms are visual representations of the spectrum of frequencies within signals, as they vary with time. Unlike raw audio waves, spectrograms can capture features such as edges and shapes. They can be fed into neural networks by representing them as a two-dimensional image. Log-scaled mel-spectrograms, as used in the study by Karol Piczak, are an advanced form of spectrogram, meant to simulate the capabilities of the human auditory system (Piczak, 2015). The main way it differs from normal or linear spectrograms are that they produce images within the range of frequencies that human ear can hear thereby eliminating noises present in the audio data. Figure 30 explains how mel-spectrograms can be used for deep learning models.

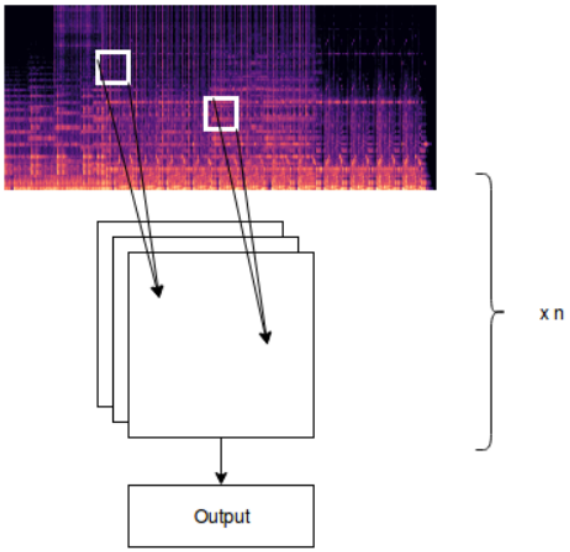


Figure 30: Processing mel-spectrogram (Mansar, 2018)

2.9 Machine Learning and Deep Learning Frameworks

Since our hypothesis involves using machine learning and deep learning techniques to classify depression, it was necessary for us to understand which machine learning frameworks were available, to aid us in implementation and experimentation. Our team consisted of undergraduate students who were proficient in coding Python, so we decided to explore top deep learning frameworks that use mainly Python. According to the article by (Sayantini, 2019), among such frameworks, Keras, TensorFlow and PyTorch are most preferred by data scientists and students working in the fields of machine learning and artificial intelligence. Figure 31 compares Keras, TensorFlow, and Pytorch based on their popularity.

2.9.1 Pytorch

Pytorch is an open-source deep learning framework backed by Facebook, inspired by Torch, another deep learning framework based on Lua (Vu, 2019). Pytorch offers fast runtimes and high performance, as it runs on Python without using high level application interfaces. Furthermore, researchers can debug training models with existing Python debuggers such as PyCharm,

pdb, ipdb and even conventional "print()" statement according to the article by (Vu, 2019). Pytorch can handle large data sets and it is customizable along the implementation of training models. Despite its advantages, Pytorch is a very young framework compared to TensorFlow. As such, it has a smaller community than Tensorflow.

2.9.2 TensorFlow

TensorFlow is backed by Google. As a mature framework, it offers a stronger community than Pytorch. Unlike Pytorch, TensorFlow has both high level and low level programmable interfaces. It also offers high performance with superior tensor visualization. One thing that TensorFlow fails to provide is debugging. The article by (Sayantini, 2019) also mentioned that many programmers found Tensorflow to be difficult to debug as they trained the models.

2.9.3 Keras

Keras is a high-level programmable interface which simplifies the complex architecture of low-level deep learning frameworks. It also runs on top of other frameworks such as TensorFlow, Microsoft CNTK and Theano according to the article by Kevin Vu (Vu, 2019). As it provides simple architecture with which to build a model, it is used primarily for educational and rapid-prototyping purposes. As simple as it is to use, Keras lacks the capability of handling large data sets, and has considerably slower performance than the other two deep learning frameworks mentioned above (Sayantini, 2019).

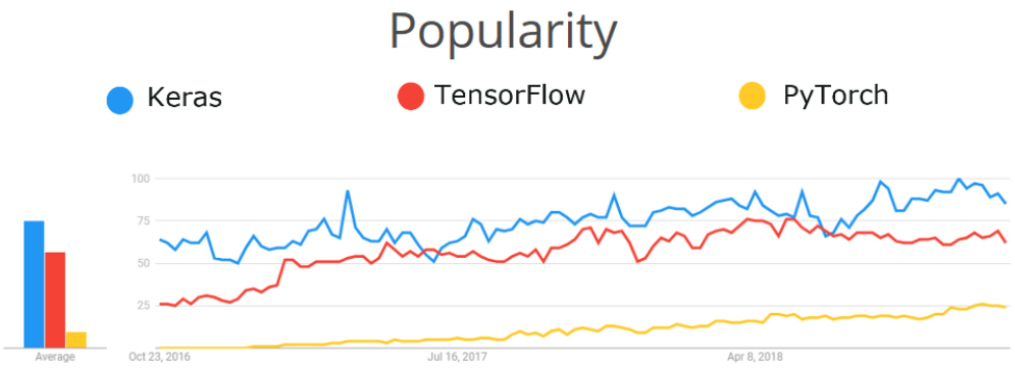


Figure 31: Popularity of three frameworks among researchers (Sayantini, 2019)

2.10 Topological Data Analysis

2.10.1 Definition

Topological Data Analysis, also know as TDA, is the practice of looking at topological features such as holes of a data set to help analyze the data. This is done by using persistent homology (see section 2.10.3) to extract and categorize these features. Categorization is done via transforming data into simplicial complexes, then finding the persistence of topological features via filtration. This has mostly been used for analyzing three dimensional sets of data, but has in recent years been used to analyze time series (Meryll, 2018).

2.10.2 Simplicial Complex

A simplicial complex is defined by a pair $K(V, S)$ where V is a finite set of points called *vertices* of K, and S is a set of non-empty subsets of V that satisfy the following conditions (Boissonnat and Maria, 2014):

1. $p \in V \Rightarrow \{p\} \in S$
2. $\sigma \in S, \tau \subset \sigma \Rightarrow \tau \in S$

Each $\sigma \in S$ is known as a *simplex* of K, thus the entire structure being known as a *simplicial complex*. A polytope is a generalized term for a n-dimensional shape. A simplex is a general term for the smallest regular

polytope in a given dimension, such as a triangle or a regular tetrahedron. For simplicial complexes, we look at the connections between these simplicies. An example of a simplicial complex can be seen in Figure 32, in which simplicies are created via considering the intersections of circles around each point. In this particular example, each circle has the same radius, however this is not required to form a simplicial complex.

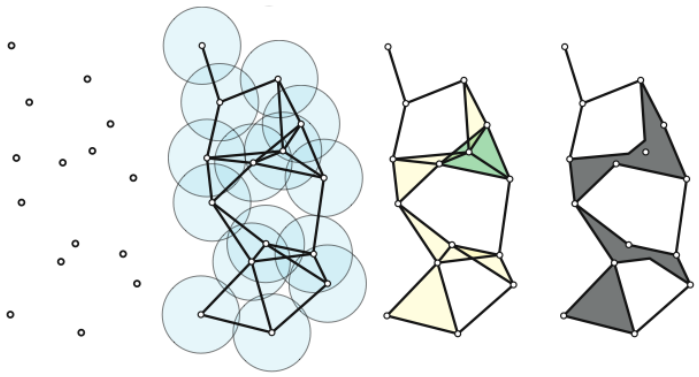


Figure 32: Example of constructing a simplicial complex (Meryll, 2018)

Filtered Simplicial Complex A filtered simplicial complex, or filtered complex, is a nested sequence of simplicial complexes such that

$$K_0 \subset K_1 \subset K_2 \subset \cdots \subset K_n$$

For any K_i, i is known as the *filtration level* of K_i. Notably, a subset of a simplicial complex can contain the same number of points as the original complex, but different connections between points. An example of a filtered complex can be seen in Figure 33.

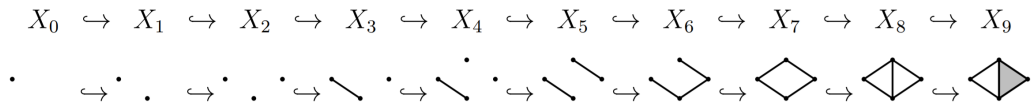


Figure 33: Example of a Filtered Complex (Bubenik, 2017)

2.10.3 Persistent Homology

Persistent homology observes how the topological features appear and disappear over a filtered complex. The persistence of a feature is defined by a pair (i, j) such that i is the filtration level at which the feature appears and j is the filtration level at which the feature disappears. These are referred to as the *birth time* and *death time* of the feature respectively. An example of a feature that could be tracked is whether a component in the complex is connected to the rest of the components in a complex. In this case, a feature would start when a component is added, and end when the component is connected to another component. This type of feature is known as a 0-dimensional hole.

2.10.4 Persistence Barcodes And Persistence Diagram

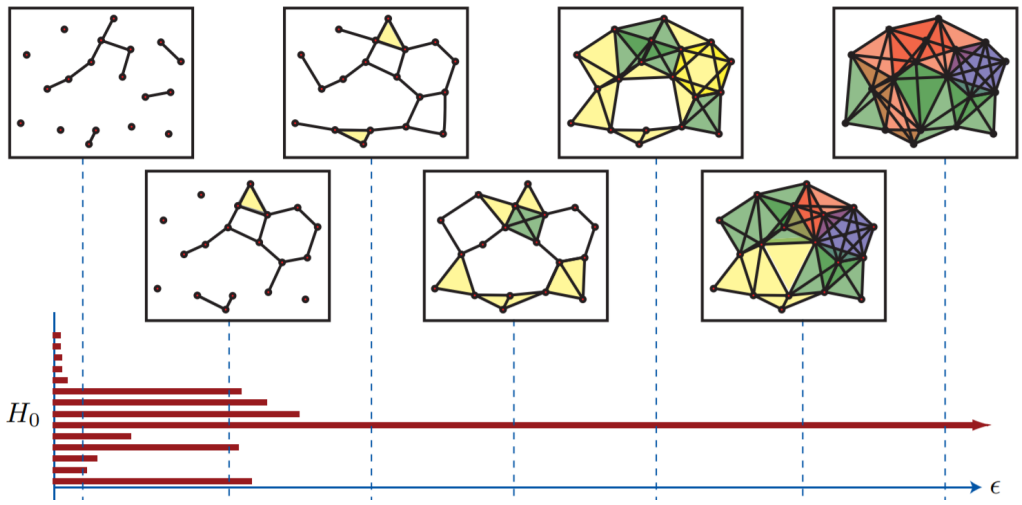


Figure 34: Example of a Persistence Barcode of a Filtered Complex. H_0 is 0-d holes, as explained in the above paragraph. Multiple steps of a filtered complex are show, and at each picture the number of lines in the barcode is equal to the number of 0-d holes in the shown complex (Ghrist, 2008)

The persistent homology of a filtered complex can be represented as either a persistence barcode or a persistence diagram. Both consider the "birth" and "death" date of the topological features tracked. A persistence barcode

(Figure 34) represents the distance between a birth and death time as a line between filtration levels. This represents the persistence of each feature as a distinct interval over filtration levels, after which each interval is displayed over filtration levels.

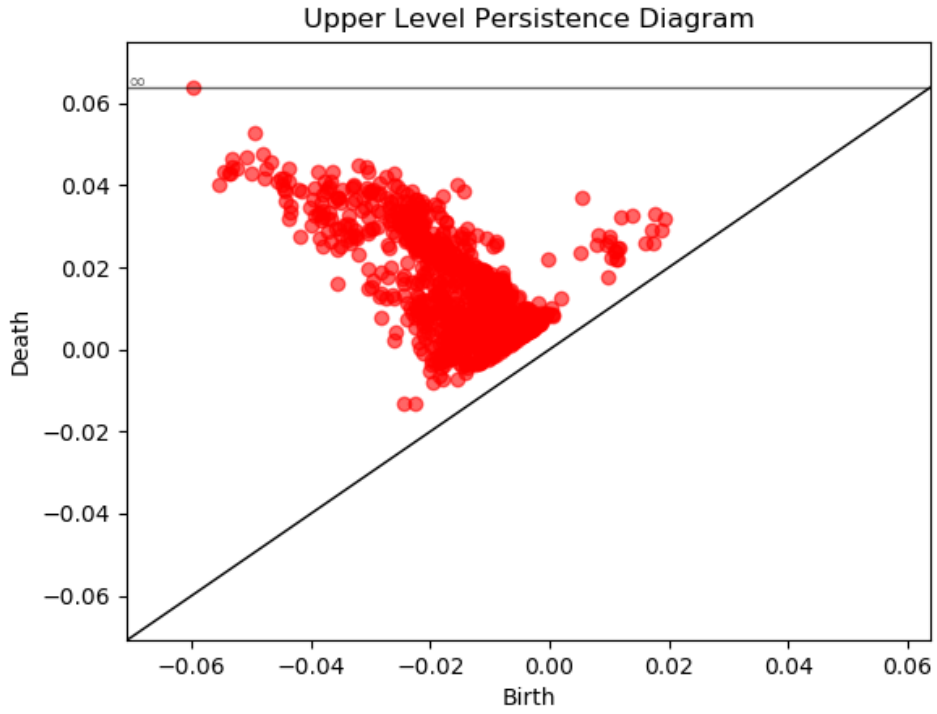


Figure 35: Persistence Diagram of a Filtered Complex Formed from a Sound Clip. This particular clip was from the DAIC-WOZ data set (see section 2.3.1. Each point corresponds to the birth and death date of a feature in the sound clip

A persistence diagram represents a birth date as a x coordinate and the death date as a y coordinate on a graph (Figure 35). Both persistence barcodes and diagrams of these provide useful information on the shape of the data, however this information is much more useful to a human than a machine. Disjoint intervals or points representing the persistence of features do not provide much information on how these features interact or compare to

one another, but as humans we can discern this information visually. Thus, we strive to combine all of these features into a single data source, so it can easily be interpreted by a machine.

2.10.5 Interpreting Topological Data

To interpret the persistence of features as useful data, we used Betti Curves. This method transforms the persistence barcode of a filtered complex into a single line, which can easily be represented as a 1-dimensional array in data, thus easily forming a feature set for machine learning.

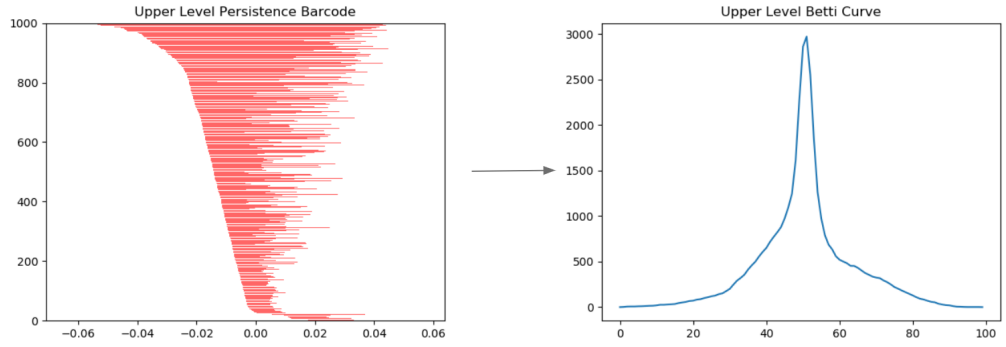


Figure 36: Example of constructing a Betti curve with 100 components from a persistence barcode. Unlike Figure 34, the features tracked have varying birth dates

A Betti curve represents the sum of lines in a persistence barcode over filtration levels. To do this, each line in the barcode is considered as a one if active or a zero if not. Then, the barcode is sampled over the filtration levels at n equally spaced points. At each point, the number of active lines in the barcode is totaled, and added to the curve. This n defines the number of components in the Betti curve. This leads to a curve which provides a good linear representation for the original barcode, which has the distinct advantage of being easier to construct compared to other methods. These curves were first defined in a 2017 paper which explored classifying time series using TDA (Umeda, 2017). In 2019, another group used Betti curves for classification of arrhythmias from heartbeats, and found favorable results as mentioned in Section 1.2.5. Because we will be using TDA to classify sound, which is a form of time series, we decided to use this method.

2.11 Generative Adversarial Networks

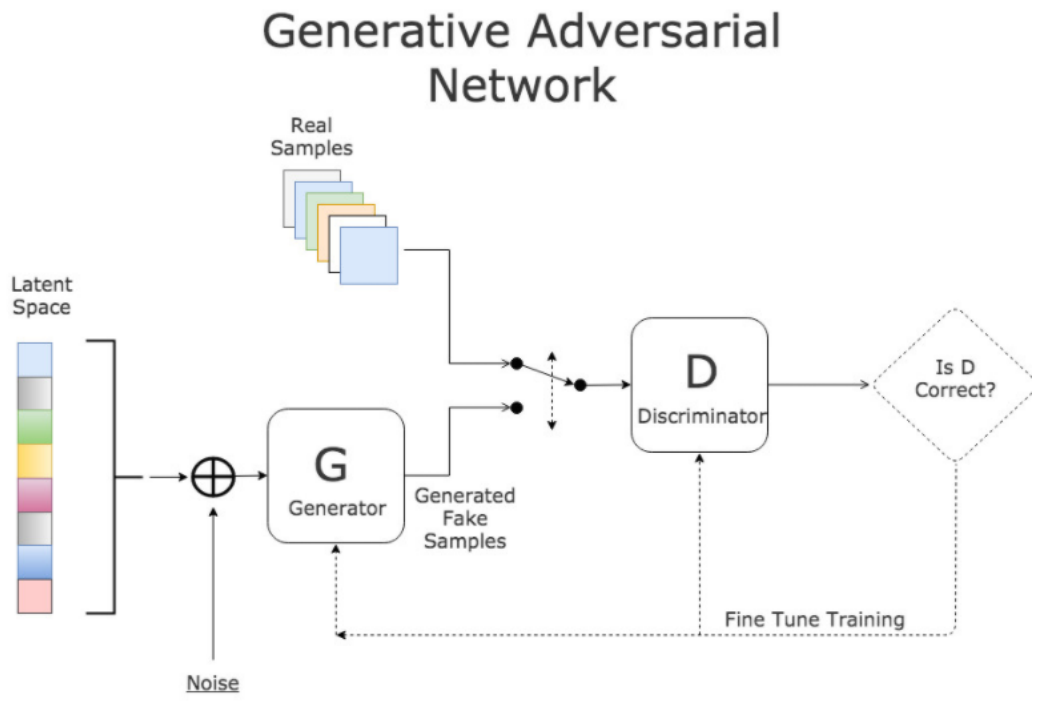


Figure 37: Diagram of GAN Process (Goodfellow et al., 2014)

Generative Adversarial Networks (GANs) are generative machine learning systems. GANs work by training two distinct networks, generator G, which learns to generate samples from a given data set, and discriminator D, which tries to discern fake samples generated by G from real samples. Fundamentally, the two are playing a minimax game, where G is trying to maximize the number of generated samples that D thinks is real, while D is trying to minimize the number of generated samples that are chosen. Figure 37 shows the general flow of how GANs work. With enough iterations, G will eventually converge and produce samples that D cannot distinguish from the real samples.

2.11.1 Sequence GAN

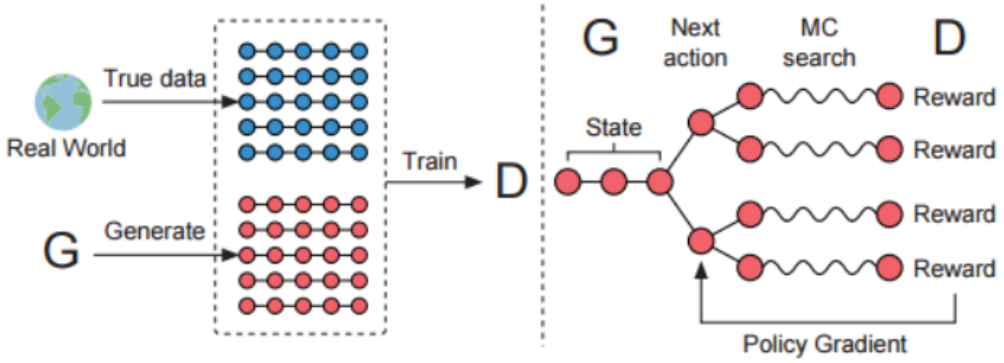


Figure 38: Sequence GAN Diagram (Yu et al., 2016)

GANs are designed specifically for real, continuous data sets, but are not optimal for generating sequences of discrete tokens like texts. One of the main reasons why is because normal GAN methods have no way of evaluating partially generated sequences, allowing the generator to estimate how well a partially complete sequence will end up. Sequence GAN, or SeqGAN, is a framework that allows for the generation of sequences using GAN techniques. The generator is able to use feedback from the discriminator to evaluate partially generated sequences, avoiding the issue that GANs normally have with discrete sequences. SeqGAN uses a RNN for generation, and a CNN for discrimination, as these neural network types have been shown to be the most optimal for text generation and text classification, respectively. Figure 38 shows the training process of generative net G and discriminator net D in SeqGAN. Before running adversarial training, however, SeqGAN uses maximum-likelihood estimation (MLE) to pre-train the generative network for a number of epochs, to accelerate the generation of more meaningful samples (Yu et al., 2016).

2.11.2 Sequence GAN Variants

Variations of SeqGAN exist that use different techniques to improve text generation capabilities. One such variation is RankGAN, which uses a very different discrimination method to provide better feedback to the generator. Instead of using a binary discriminator, where samples are identified as real

or fake, RankGAN gives the discriminator a list of samples, which the discriminator then ranks. This feedback contains considerably more information to learn from, allowing the generator to learn more from fewer epochs (Lin et al., 2017).

Another variation is LeakGAN, designed specifically to work with longer texts. In normal SeqGAN, the discriminator only provides binary feedback after the entire sample has been generated. LeakGAN allows the discriminator to leak the kinds of features it looks for to the generator, which the generator uses at all steps of generation to produce better, longer samples. Figure 39 shows the flow of LeakGAN, showing how the discriminator shows its internal state to the generator (Guo et al., 2017).

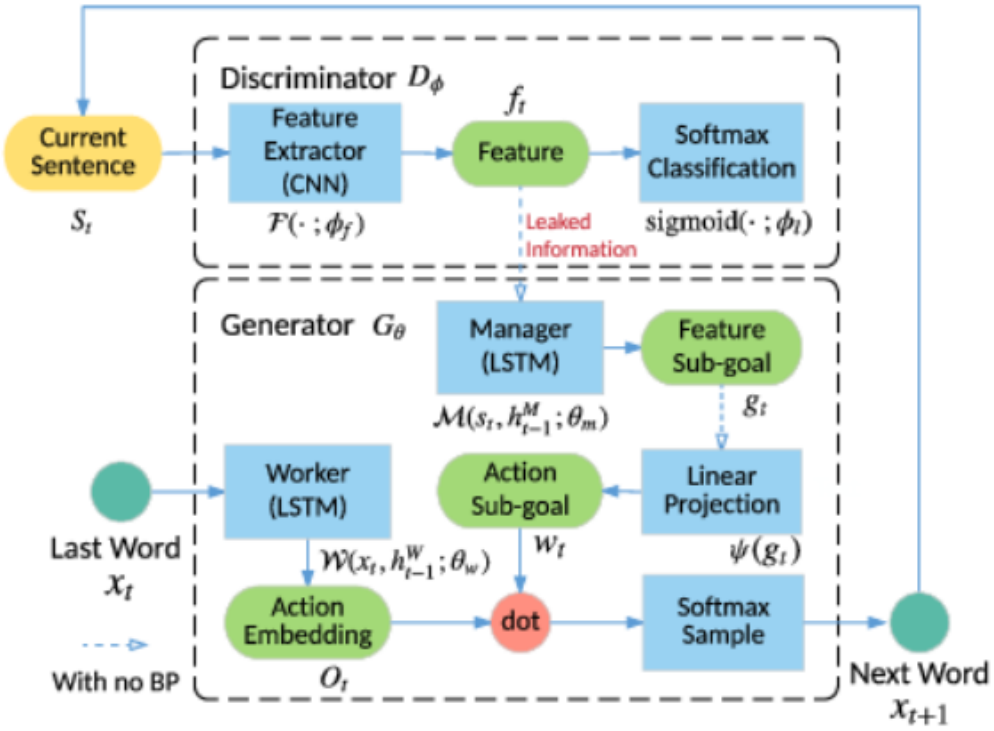


Figure 39: LeakGAN Flow Diagram (Guo et al., 2017)

RelGAN is a recently-developed sequence GAN method that builds upon the work of previous previous sequence GAN studies. RelGAN uses relational memory in the generator, instead of the traditional RNN or LSTM, to allow for a more expressive text generator. Additionally, RelGAN uses less demanding reinforcement learning heuristics for training, and multiple representations for each generated sequence in the discriminator. All of this has shown that RelGAN outperforms other sequence GAN methods consistently (Nie et al., 2019).

2.11.3 Sequence GAN Evaluation Metrics

Different metrics are used to evaluate the performance of sequence GANs. One metric specifically developed in the SeqGAN paper is NLL-oracle loss (Negative Log Likeliness). NLL-oracle loss randomly initializes an LSTM to be treated as the oracle, which is compared with the generator to produce a value that the GAN minimizes as it runs through multiple epochs. Figure 40 shows the NLL metric, where G(theta) is the generator, and G(oracle) indicates the oracle model. For NLL, lower values are considered better, as NLL represents the entropy to be minimized in the model (Yu et al., 2016).

$$\mathrm{NLL}_{\mathrm{oracle}} = -\mathbb{E}_{Y_{1:T}\sim G_\theta}[\sum_{t=1}^{T} \log(G_{\mathrm{oracle}}(y_t|Y_{1:t-1}))]$$

Figure 40: NLL-Loss (Yu et al., 2016)

BLEU (Bilingual Evaluation Understudy) score is another metric that can be used, specifically used for evaluating texts generated by SeqGAN variants. BLEU score was originally developed as a metric for evaluating machine translation, but has become the standard metric for evaluating all machine-generated text. BLEU is a "translation closeness" metric, which compares a generated sample to a corpus of group of good samples. For machine learning purposes, this corpus is usually the training data. BLEU scores range from 1.0, which means that the generated sample exactly resembles one of the samples in the corpus, to 0.0, which means that the generated sample does not resemble one of the original samples at all. Since a score of 1.0 indicates that the neural network has just replicated one of the given real

samples, BLEU scores in the 0.6-0.9 range are considered better (Papineni et al., 2002).

3 Feature Extraction

3.1 Text Feature Extraction for Machine Learning

To extract features, we started from a script that was made for the paper "Screening for depression with retrospectively harvested private versus public text". This script made use of "word category frequencies, part of speech tag frequencies, sentiment, and volume" (Monica Tlachac, 2020) features. This script generated features using the tools: Empath and Textblob. The script makes use of TextBlob's Part of Speech tags, as well as custom sentiment features from the TextBlob output. We modified the provided script to make it to work for the DAIC-WOZ data.

The script reads in a file with three columns: ID, which contains the participant ID, and question number, Content, which contains what the participant replied, and PHQ-8 Score, which is simply the participant's PHQ-8 score. The script produces a .csv file with a numbered first column, a second column with ID(participant ID and question number), score(the PHQ-8 score), followed by 245 columns of features that were extracted.

3.2 Audio Feature Extraction for Machine Learning

3.2.1 Pratt

To extract the audio features from DAIC-WOZ, we decided to use the Pratt script from the 2019 Major Qualifying Project to extract pause time features. We realized that some of the clips from our processed DAIC-WOZ data set included numerous empty clips, So we modified the script to prevent if from if there was no audio in the clip.

On each run through a directory, the script uses for parameters that determine how the features are extracted. The first is silence threshold, which relates to the dB value needed for the program to classify the clip as a pause in the conversation. The second parameter is minimum dip between peaks.

The third parameter is minimum pause duration, which refers to how long there has to be silence at the silence threshold for it to be considered a pause.

The program outputs the file ID, the number of syllables, the number of pauses, the duration of the audio file (in seconds), the vocalization time (in seconds), the pause time (in seconds), the speech rate (number of syllables divided by the duration of the audio clip), the articulation rate (the number of syllables divided by the vocalization time), and the speaking time, divided by the number of syllables. The program outputs it to the Pratt interface, which is then copied into a .csv file.

For the first trial, we run the Pratt script on 2-5 second audio clips from DAIC-WOZ. We set the parameters as follows: silence threshold of -25dB, minimum dip between peaks of 2dB and minimum pause duration of 0.3 secs.

Due to the high volume of data that it would need Praat feature extraction, we modified it to work on the cluster.

3.2.2 openSMILE

openSMILE is an audio extraction software used for signal-based features, unlike Pratt which is used for pause-time features. The feature extraction on openSMILE is done using the 2010 Embrosa configuration and is being used to generate 1600 features per audio clip.

3.3 Topological Data Analysis

To use data in topological data analysis, it much first be transformed into a filtered simplicial complex (see section 2.10.2). To do this, we use the Gudhi python library[2] using simplicial trees. Simplicial trees are a data structure that efficiently represents simplicial complexes in data using a tree structure, and is what Gudhi uses to represent simplicial complexes (Boissonnat and Maria, 2014). Gudhi is a python library dedicated to performing topological data analysis, and was the most simple to use due to being used in previous efforts in implementing Betti curves.

[2]https://gudhi.inria.fr/python/latest/

3.3.1 Constructing Filtered Complex From Sound Waves

To construct a filtered complex from sound waves, we consider each segment of the wave as a path between two vertices, then link these paths together to create the full wave. Then we assign filtration levels to each segment such that where 0-d holes appear and disappear are at local maxima and minima of the wave. This is done by considering the magnitude of the sound wave at a particular point as its filtration level. Thus, the holes appear and disappear at minima and maxima. This can be seen by observing the following series of diagrams:

First, we can see the wave as a hole, before filtration is applied.

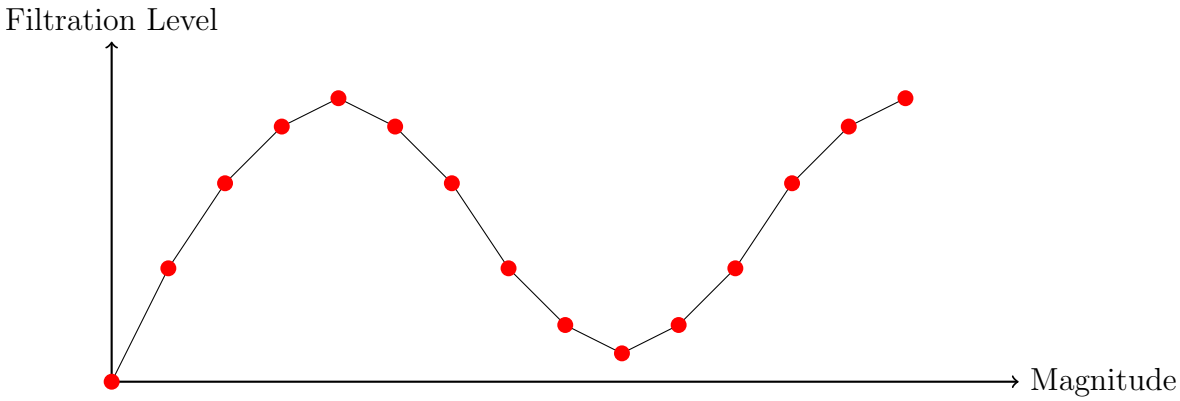


Figure 41: A Simplicial complex from a simplified sound wave

Next, the smallest values are added.

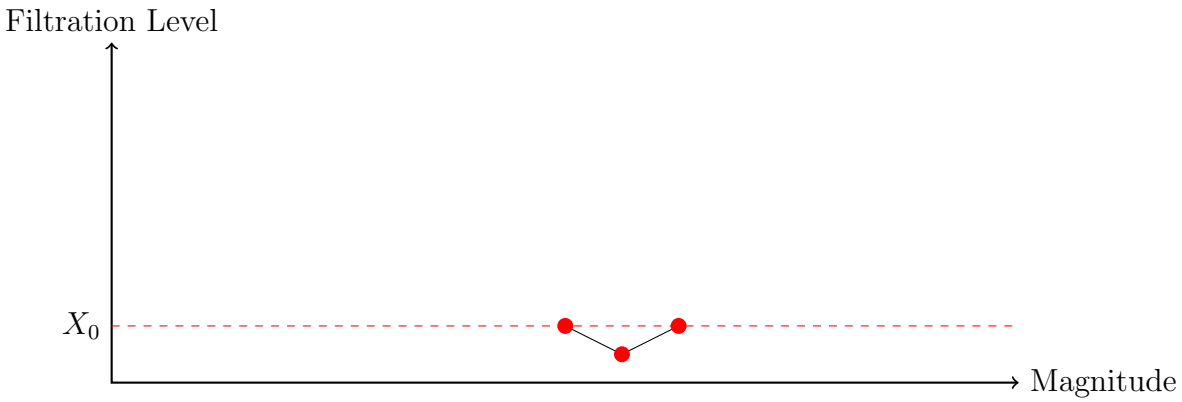


Figure 42: First Filtration Level of Wave Filtered Complex, X_0. Each segment with a filtration level less than X_0 is added

Then, the first segment on the left is created, making the first hole.

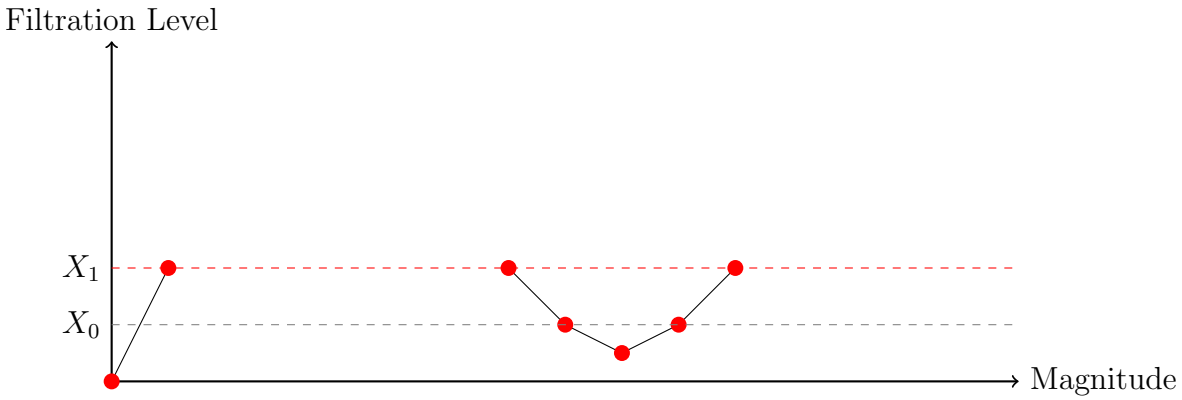


Figure 43: Second Filtration Level of Wave Filtered Complex, X_1. Each segment with a filtration level less than X_1 is added

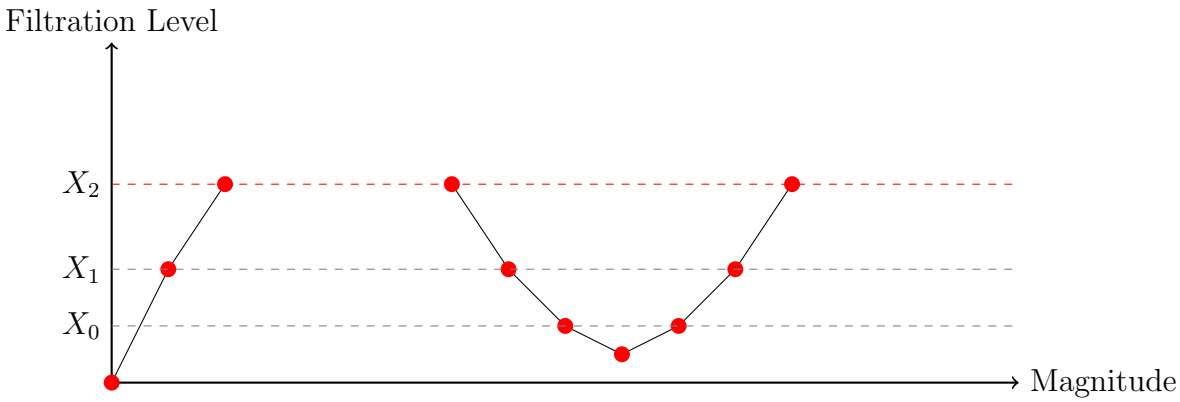


Figure 44: Third Filtration Level of Wave Filtered Complex, X_2. Each segment with a filtration level less than X_2 is added

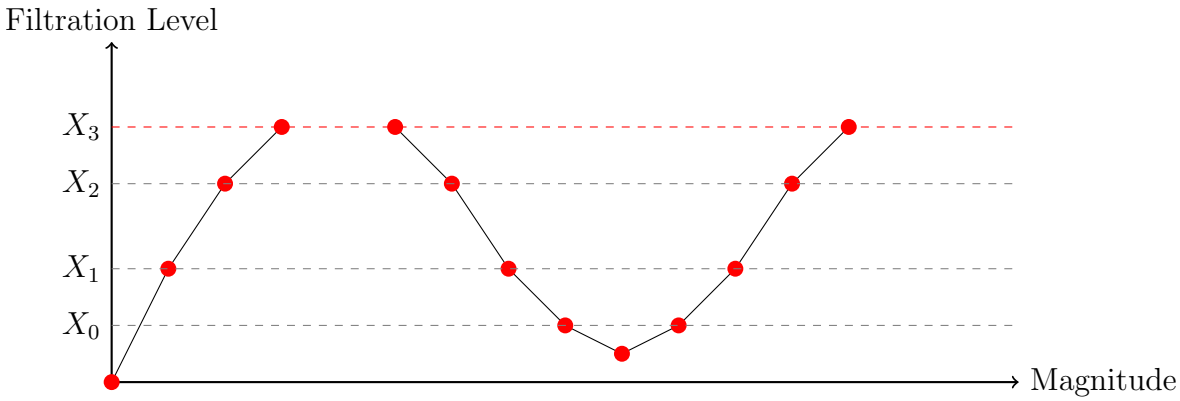


Figure 45: Fourth Filtration Level of Wave Filtered Complex, X_3. Each segment with a filtration level less than X_3 is added

This hole is continued over several filtration levels, until finally closed when the maxima is added at X_4.

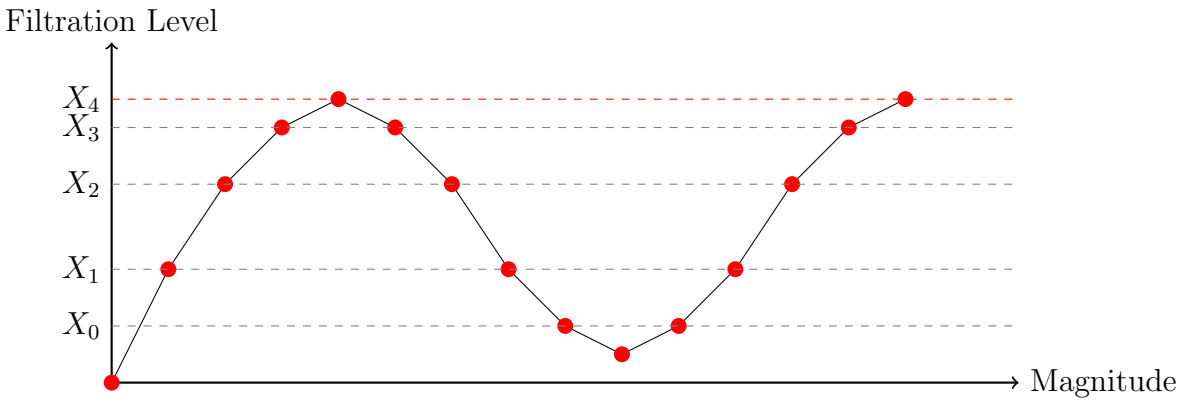


Figure 46: Fifth Filtration Level of Wave Filtered Complex, X_4. Each segment with a filtration level less than X_4 is added

This is how our filtered complexes were constructed, insuring that persistent homology captures the critical points of the sound wave. We also consider filtration from maximum value to minimum value, and differentiate the two by calling minimum to maximum "upper level" and maximum to minimum "sub level". We do both of these as the persistence captured can vary slightly based on when features appear and disappear. Next, we convert these into barcodes, followed by Betti curves, whose construction is covered in Section 2.10.5. This is the final state of our topological features extracted from sound waves, after which they can be inserted into a machine learning model.

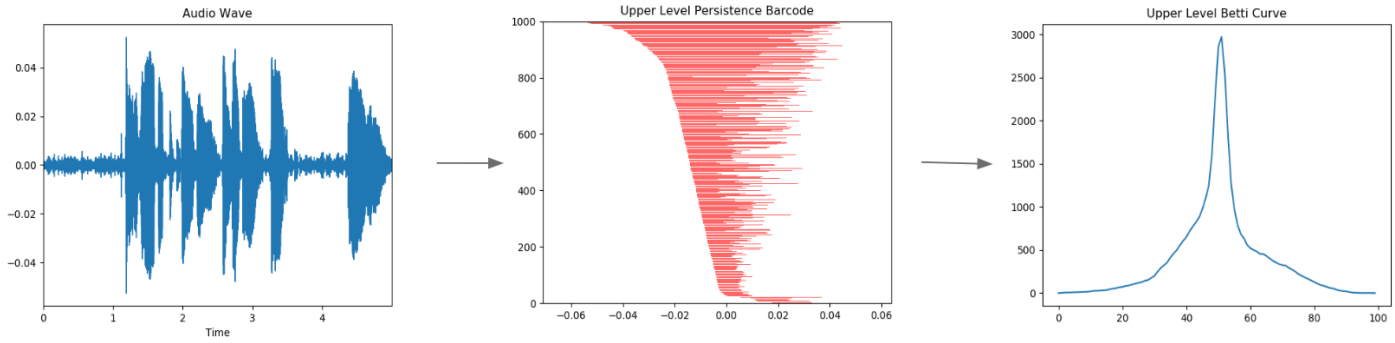


Figure 47: TDA Feature Extraction Pipeline

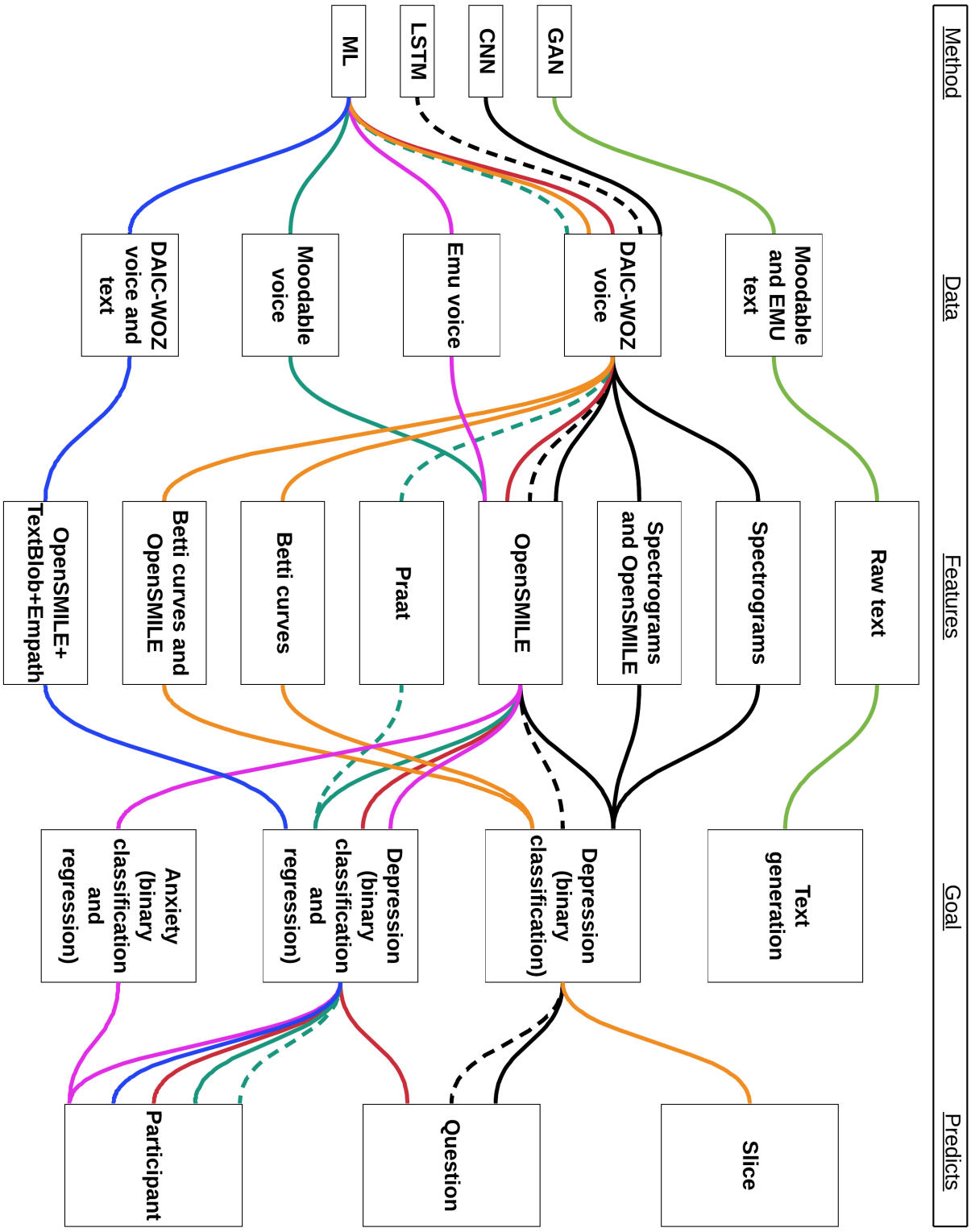


Figure 48: All combinations of method, data set, feature type, goal, and prediction type in this paper

3.4 Generative Adversarial Networks

The SeqGAN variants used only raw text data from EMU and Moodable, without extracting specific features, as the goal was to expand that data set and extract features from the texts generated by it.

4 Methods

4.1 Machine Learning

In this project, we implemented regression and classification methods for six models: Support Vector Machine, Random Forest, XGBoost, Adaptive Boosting, k-Nearest Neighbours, and Multilayer Perceptron. We conducted various tests for each method and model variation to find best performing ones. Through out the project, we were able to build an Machine Learning efficient pipeline to conduct experiments on three different data sets: EMU, Moodable, and DAIC-WOZ. Although most of our experiments were targeted at predicting PHQ-8 labels, we had experiments that predicted GAD-7 Labels for EMU data set. Overall, our goals within this sub project were comparing Machine Learning method performances across different datasets' Experimentation.

4.1.1 Machine Learning Pipeline

For our Machine Learning methods, we first sliced the audio data into sub-clips. Further, we supplemented the audio clips with Sub-Clip Boosting (which was shown to improve scores (Toto et al., 2019)) implementing overlaps between the sub clips. After successfully slicing our audio data, we generated features using openSMILE and Praat tools. Resulting feature files were cleaned and labelled with corresponding labels: PHQ-8 or GAD-7. After generating fully labelled clean feature files, we performed feature selection to get the optimal number of features for our experiments. Next, we split feature files into their respective folds. Splitting is done by using a json file that contains participant IDs used to determine whether participants are allocated in either train or test sets. These splits were kept the same throughout all experiments. After participants within feature files were split into train/test sets, we used a grid search to tune optimal parameters for the models. Some models didn't compute grid search within computational

time thus random set of parameters were tested on and the best performing ones were selected. Next, using parameters generated from grid search, we fit classifier and regressor models. After models are fitted with training data set, we use these fitted models to predict our testing data set to get our results. The generated results contain binary classifications and regression predictions. In addition, they contain prediction probabilities.

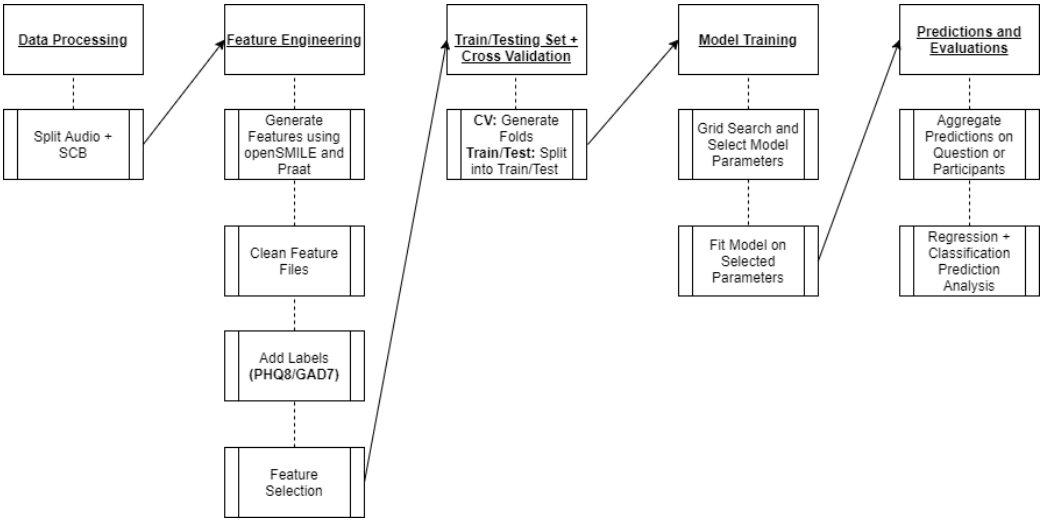


Figure 49: Machine Learning Pipeline

Figure 49 illustrates our pipeline process. Bold and underlined steps represent main processes, while the boxes below main processes represent sub processes.

4.1.2 Machine Learning Method Configurations

Data Processing: DAIC-WOZ Audio For DAIC-WOZ audio, we first split participant audio data into question segments. We further broke the question splits into Sub-Clips based on clip duration. Intervals of 3, 4, 5, 6, 7, 8, 9, and, 10 seconds were used. Once data were split into sub clips, we implemented SCB (Toto et al., 2019) to enhance our experiment scope. For SCB, overlapping of 0%, 25%, 50%, and 75% were used to boost clips.

Data Processing: DAIC-WOZ Transcript+ Audio For DAIC-WOZ audio, we first split participant audio data into question segments. We then

organized the transcripts into question segments and made sure the same naming conventions as the audio data were used.

Data Processing: EMU For EMU data, we split participant audio data into Sub-Clip splits. Sub-Clips with 2,3,4,5,6,7 segment splits were used. Once data were split into sub clips, we implemented SCB to enhance our experiment scope. For SCB, we used the standard overlapping intervals as the DAIC-WOZ processing technique.

Data Processing: Moodable We used the exact same technique used to process EMU data for processing Moodable data.

Feature Selection: openSMILE Feature selection was implemented for openSMILE features due to large number of features. Firstly, we used Extra Trees to determine the features' importance. Next, we kept features that had feature importance higher than the mean feature importance. We used scikit libary's ((Buitinck et al., 2013) ExtraTreesClassifier() tool. We invoked the resultsing model's attribute feature_importance to get feature importance values. Resulting dataset had a reduced set of 176 features.

Participant Splits Participant splits were mainly done using a script that generates 5 folds and randomly allocating participants within those folds. These folds were stored in a json file and maintained throughout experiments. For example, a fold split for EMU GAD-7 dataset would be used across all EMU GAD-7 experiments. This approach was done for Moodable, DAIC-WOZ and EMU PHQ-8 experiments. However, we added two extra folds for DAIC-WOZ dataset. This was done to enable result comparisons with other researches using DAIC-WOZ dataset and amongst our Deep Learning and LSTM experiment results. The particiapnts were split into train and test splits for this part instead of maintaining cross validation approach mentioned earlier. For comparing with other researches we used the DAIC-WOZ official splits: train split and dev splits. Dev splits were interpreted as our testing set and train split was used as training set. To compare with our Deep Learning and LSTM models, we used a train and testing split determined randomly. Data set was split into 70 to 30 ratio in favor of training set. In addition the Depressed and Non-Depressed participants had same ratio in both splits. We used a standard cutoff of 10 as a

binary cutoff.

Training We fitted the data on the Machine Learning methods mentioned earlier. We used grid search to determine optimal parameters for our models when possible. However, since this was computationaly expensive we opted to select optimal parameters out of a random set of parameters for our models.

For Random Forest, we found the best parameter selections to be: 400 estimators, maximum depth of 7 for the trees, minimum samples split of 5, minimum sample leaf of 1. These parameters were used in both regressors and classifier models.

For K-Nearest Neighbours, we found best parameter selections for number of neighbours and leaf size to be 9 and 1 respectively.

For XGBoost, we selected inputs for the parameters learning rate, gamma, maximum depth of tree, minimum child weight, subsample, colsample by tree; Values for these parameters were 0.01, 5, 4, 1, 0.7, 0.6 respectively.

For Support Vector Machines, we were only able to use linear kernels due to computational issues. Thus, we weren't attain optimal parameters for the model.

For Adaptive Boosting, we used a Random Forest with the previously mentioned parameters as a base estimator. In addition, we used 400 trees as estimators.

For Multilayer Perceptron model, we used an L2 penalty of 1, maximum number of iterations of 50, solver was stochastic gradient descent, a learning rate that is adaptive (keeps the learning rate constant to initial learning rate as long as training loss keeps decreasing), initial learning rate.

Predictions Our Machine Learning experiment framework outputs test predictions in a csv file format. The files contain binary predictions, binary prediction probablities, and regression predictions for each participant Sub-Clip features. These are later aggregated to have predictions for participants. Note, in the case of DAIC-WOZ dataset experiments aggregations

were done based on questions as well. Once aggregated we evaluate predictions by computing TN, TP, FP, FN, accuracy, sensitivity, precision, F1 score, specificity, MAE, RMSE, R2, MSE. We also determined a cutoff for our regression predictions in order to compute F1 score and compare with a classification approach. Grouping of predictions were done based on different aggregating methods: median, mean, max and voting. We also used a weight that was altered during experiments. The participant was elected to be depressed if the binary prediction probability of one (Depressed) multiplied by the determined weight was higher than the binary prediction probability of zero (Non-Depressed). In addition, we used F1 score as the standard metric to compare different experiments.

4.2 Deep Learning

In order to use Deep Learning for mental health sensing, we first needed to implement data-preprocessing on our available data set. We primarily intended to use DAIC-WOZ data set so that we could produce results comparable to other techniques like machine learning and DepAudioNet. For the purpose of building and debugging our first deep learning pipeline, we started with combined EMU and Moodable data set since testing on them is less resource-expensive than DAIC-WOZ data set. At the same time, we also built data-preprocessing on DAIC-WOZ data set. Once we made sure our deep learning pipeline is free of syntax and compilation error, we began finding an efficient model by gradually testing out various hyperparameters such as number of layers, learning rate and kernel sizes. Then, we analyzed each of our deep learning models by confusion matrix, convergence of train loss and test loss curves, F1 score and accuracy.

4.2.1 CNN Experimental Setup

Our goal here was to see if we could build our own convolutional neural network that was able to train on one of the available data sets. For the EMU/Moodable data set, data preprocessing was already done prior to working with those data sets, so we were able to begin by start by defining configurations and laying out python source code to fit our new model. The setup we built here is similar the one in Figure 22, except that we tweaked the parameters to suit our data. We started with minimal amount of configurations to see the changes and compare them as we tune them.

- Hyper parameters: Batch-size was set to 1 and would gradually increase up to 16. Number of epochs was kept at 60 since more would take a longer time to finish. The optimizer used here is Adam with learning rate set up to 1.5e-7.

- CNN structure: Input was a 2D array of spectrogram data with respective labels of PHQ-8 scores. These PHQ-8 scores were taken in as non-depressed with scores below 10 and depressed with scores between 10 and 24. The output in this case was binary classification with depressed as 1 and non-depressed as 0. Therefore the loss function used here was cross entropy loss for binary classification. The model had two two-dimensional convolutional layers and two max pooling layers. The first convolutional layer had (3x3) kernel, and step size or stride of 1. The first max pooling layer used (4x3) kernel, and with a stride of 3. The second convolutional layer had (1x3) kernel, and stride of 1. The second max pooling layer uses (1x3) kernel, and stride of 3. The test data set of 30 percent was from the original data sets.

- Feed-Forward network: the output we have from CNN was flattened as a one-dimensional array and then fed into this multilayered perceptron. Here we had two more hidden layers: one with flattened array size and another with 512 nodes. The drop out layer with 0.15 was added to reduce over-fitting. The output layer matched two number of classes which were depressed and non-depressed.

4.2.2 LSTM Experimental Setup

The goal of our sequence-based model was to exploit the ability of an LSTM network to remember information and use context. There are a few different ways in which our data can be treated sequentially. We briefly explored inputting spectrogram images of DAIC-WOZ data, before focusing on features generated with openSMILE. Our first batch of experiments was using a set of 25 openSMILE (os25) features, selected as the most impactful. After getting better results using a more verbose set of 176 features (os176) we ran the rest of the experiments on this set. The final model was built using PyTorch.

- Structure: The data was first fed through an LSTM layer, the output of which was then passed through a linear layer and finally through

a sigmoid output layer. The loss function was binary cross-entropy, and we experimented between AdaDelta, RMSProp, and Adam for optimizers.

- Data: The openSMILE features were stored in a .csv file. Each row represented the features for one sound clip. We used 3 second clips with 0.75 overlap between clips. The data was grouped by question and given a label, meaning that each time step was a sound clip for the question. Predicting label for an entire question gave the best results. The model predicted binary label, giving a cutoff of 11 and above as depressed, below as non-depressed. During experiments, the original data (in order by participant and question) was split at 70% for the training set, and the remaining 30% as validation. To compare results with other models in this project, the best models were re-trained with the same split as the rest of the deep learning and machine learning solutions.

- Hyperparameters and settings: Learning rate was {1e-5, 1e-6, 5e-7, 1e-7, 1e-8, 1e-9}, hidden nodes were {50, 100, 150, 250}, and number of layers was {1, 2, 4, 6}. Dropout rate was {0, 0.1, 0.2, 0.5, 0.8, 0.9}.

4.2.3 Deep Learning Pipelines

CNN The first thing that needed to be done before other steps was data processing. This step included collecting audio files, limiting question numbers per participant, and extracting PHQ scores and gender labels. After that, the whole data set was divided into training set and testing set. While splitting into two separate data sets, data balancing was also performed on the training set to avoid failing to recognize the minority class. The training process began when we had balanced training set and testing set. During the training process, we started off with CNN layer configurations as stated above in Experimental Setup. However, hyper parameters such as kernel sizes and number of dense layers were subject to changed as we were trying to find a decent model through multiple experiments. Gender label were also added into fully connected network after receiving feature map through convolutional process. Once the training process was done, the model was predicted with testing set and the confusion matrix was collected.

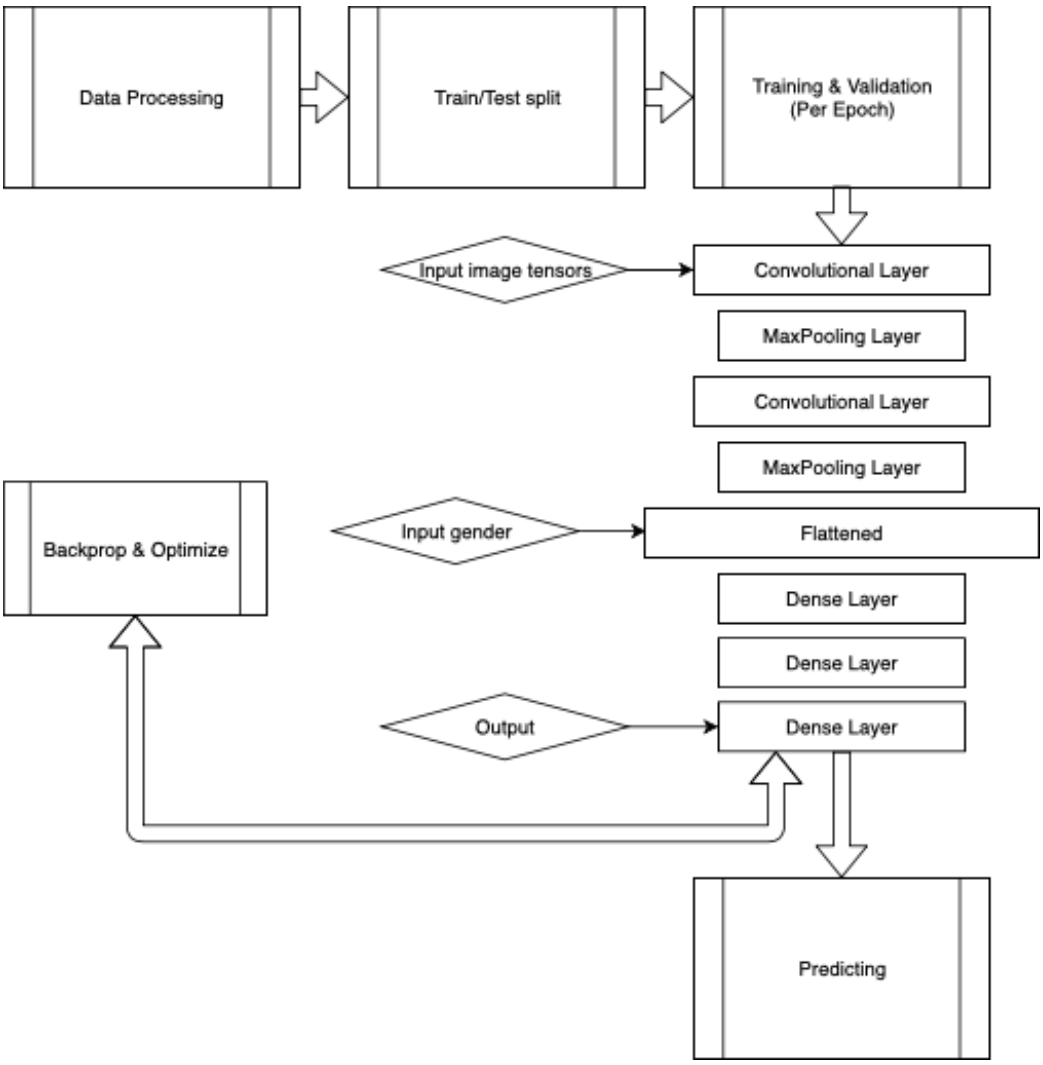


Figure 50: Convolutional Neural Network pipeline

LSTM As with the CNN, data preprocessing was done first as described in Section 4.3. The LSTM model mostly used the 3 second clip length with 75% overlap. openSMILE was used to extract features for each clip. At this point, the fill data set was a .csv file with each row containing the feature set for one clip. Here, the data was split into a train and a test set. Then, the clips were grouped by individual question response and paired with the depression label of the participant answering (for the sequence model we used a PHQ-8

score of equal to or less than 10 to mean non-depressed, and higher than 10 as depressed). We then trained our model on this data, validating at the end of each epoch with the test set. After every epoch we collected metrics about the performance. These included the hyperparameters and settings used, the loss and accuracy of the train and test sets, and the confusion matrix values for the test set which was used to calculate precision, recall and F1 score. Both in real-time and after training has completed, these metrics could be passed to a custom performance analytics and visualization tool in a Jupyter notebook to examine how the model was doing.

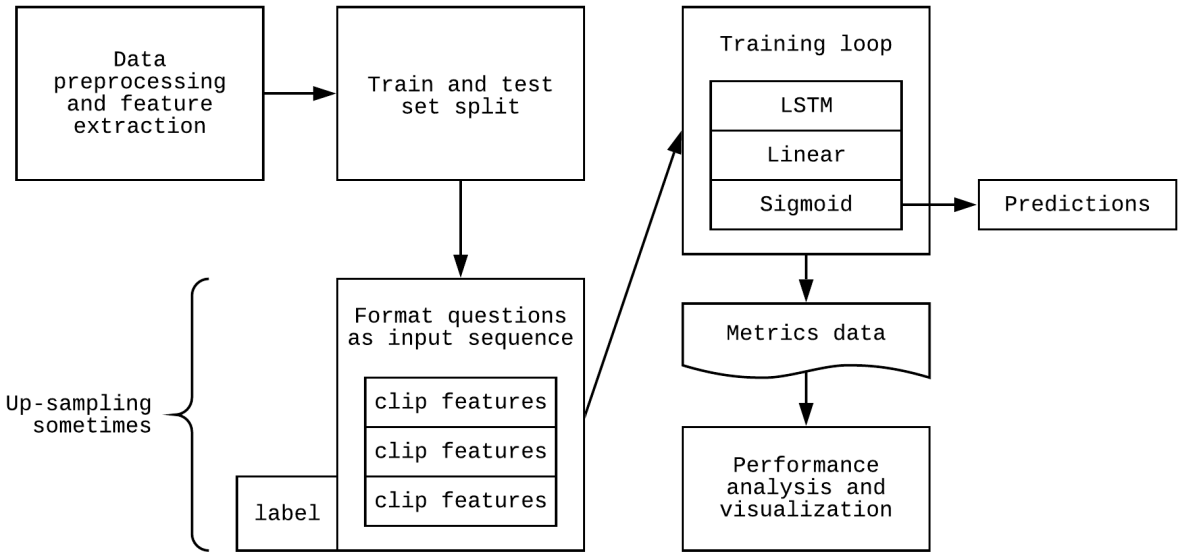


Figure 51: Sequence Model (LSTM) pipeline

4.2.4 Experimental Planning

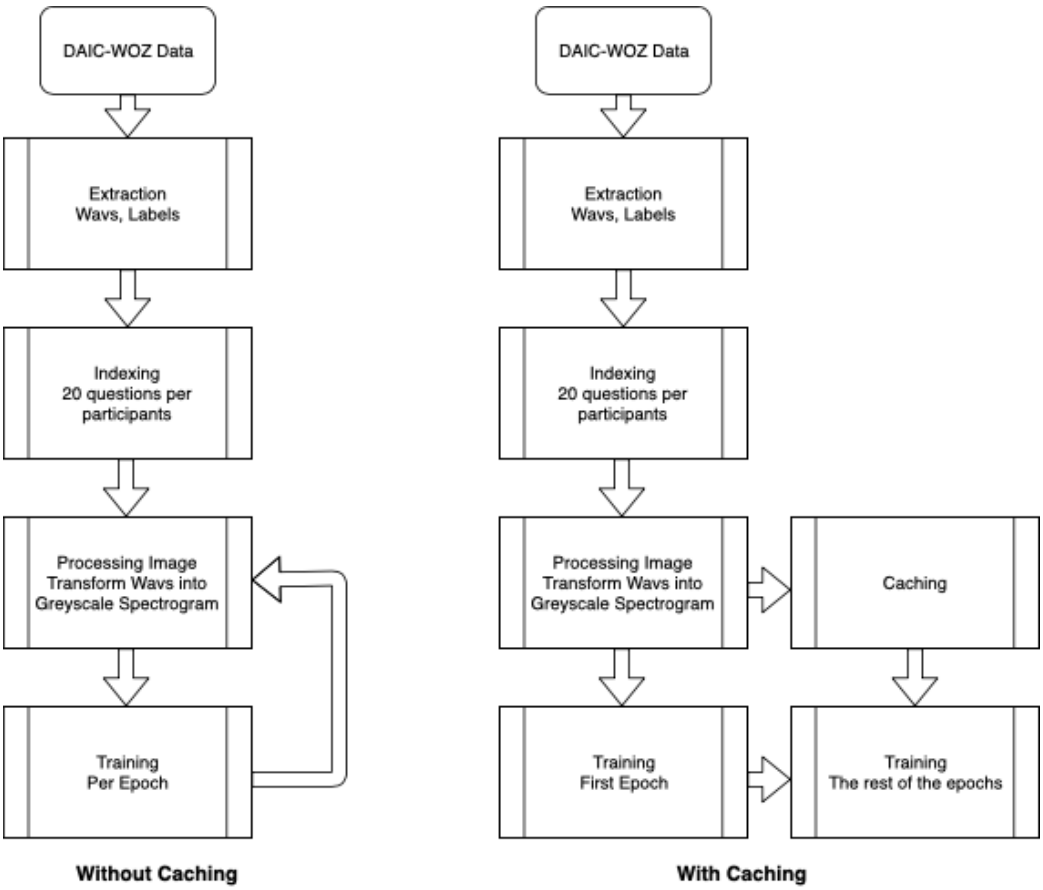


Figure 52: Workflow between Image Caching and without Caching

Reducing Training Time The difference in size of EMU+Moodable data sets and DAIC-WOZ data set is vast. While the combined EMU and Moodable only has 290 audio files, DAIC-WOZ contains 10570 audio files. The current setup we have would work on EMU and Moodable for a considerable amount of time, it could take over a week to process and train raw audio files from DAIC-WOZ data sets. One way was to implement a caching system that would boost up the speed of the experiments using the same configurations for generating the spectrograms. For example, if we decided the spectrogram to have one dimensional tensor of 256 by 64, 1024 slices and 1000 overlaps, the other experiments that used the same spectrogram configurations would not need to generate extra cache. This would allow us to train the DAIC-WOZ data sets over a day with more epochs.

One promising way to reduce training time for both CNN and LSTM models is node or weight pruning. The concept here is that by ordering neurons or weights by how much they contribute, the lowest-ranking can be removed (Le Cun et al., 1990). The paper by (Molchanov et al., 2016) details an iterative process for pruning CNNs by ordering nodes, removing the last, testing the affects, then continuing on to the next if needed. There are also methods of removing entire filters from the CNN instead of single weights or nodes (Li et al., 2016). Similar concepts can be applied to LSTM-based models, to see similar results.

LSTM Workflow Running experiments, changing parameters, generating and comparing results, and editing code can all make developing deep learning models very time-consuming. We have created some tools to automate this process. For the LSTM model, hyperparameters and other options like the optimizer, learning rate, number of hidden nodes and layers, and dropout rate are passed from the command line. Metrics such as loss and true and false negatives and positives, are collected after each epoch and saved to a file. When running a test, a bash script is run to submit the job to the job management system on our research cluster, Slurm. In this way, when we have an experiment to run, we can write a script to submit all the related jobs with their individual hyperparameters in parallel. Once the experiment has finished, we can pass the directory where we saved the metrics for each run to a metrics utility. This utility graphs the loss over time, as well as accuracy, precision, recall, and F1. It also displays the confusion matrix for the last epoch. This same file doubles as a log of experiments, keeping track of their hyperparameters, settings, Slurm job ID's, and additional notes.

In addition to saving metrics, the states of the model and optimizer after each epoch (weights and parameters) were saved to disk as a .pt file with PyTorch's built-in method. If a model showed that loss, accuracy, or F1 has not yet stabilized, we could submit a new job that would load the model back in and continue training from where we left off. Some later experiments were run up to 800 epochs in 200 epoch chunks until metrics stabilized, or it became apparent that the model was overfitting. This was only necessary due to the computation time limit on the research cluster.

CNN Since we could not know which CNN model was going to perform best for detecting depression, we needed additional planning and multiple ex-

periments to compare the results. Here, we could proceed in two approaches. One was to simplify the CNN structure to 1 convolution and 1 max pool layer and figure out the number of layers and kernel sizes that might work for our data sets. The advantage of this plan was that once we figured out the right setup, we could continue training and predicting other data sets which are similar to DAIC-WOZ or EMU and Moodable data sets. The downside was that we would have to continue testing each setup until we find out a proper model. Another one was to test with other features such as openSMILE along with convoluted features. With this approach, we could tune the feed forward network properly and observe if openSMILE improved model learning or not.

Convolution & openSMILE Experimenting with CNN alone on DAIC-WOZ data sets was quite challenging and unpredictable. Since convolution layers filtered out the features from each spectrogram by kernels and max pooling, the feature set we could get from convolution may not be as distinguished as other feature sets such as openSMILE. Thus, we could also speed up our experiments by combining CNN with other feature set. For example, we could determine if a model improved predicting by combining feature set generated from Convolution layers and respective OpenSmile Features and then fed them together into the same Feed Forward Network.

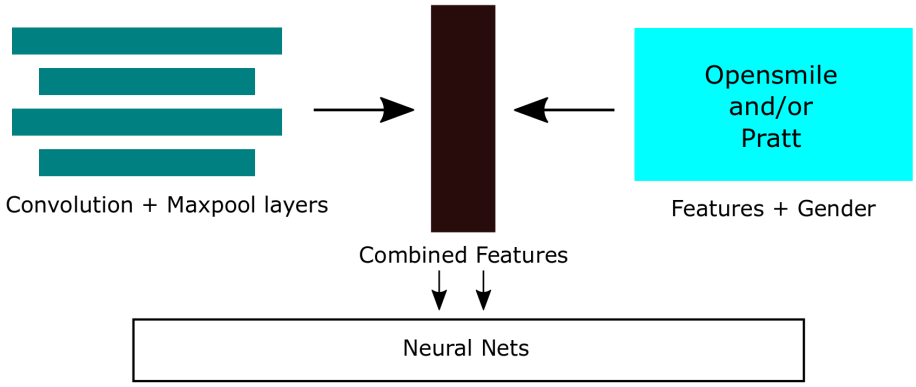


Figure 53: Combining Convolution and openSMILE

openSMILE From the above premise, we could also derive experiments to test openSMILE features and Feed Forward Network only. By doing so, we could find significant parameters which would substantially impact our models inside Feed Forward Network. Since these experiments would not consist of Convolution layers, another advantage was that they would require less computational resources and could complete in a shorter amount of times compared to CNN experiments.

Parameter tuning Parameter tuning would be done in all experiments but there were specific parameters we considered might have big impact in improving our learning models. Besides from convolution and feed forward layers, parameters such as learning rates and decays played a huge part in improving our models. Learning rate referred to the amount at which the weights were updated every epoch. Learning rate that was too large could lead the gradient descent to step over the global minimum and too small could make it stick at the local minimum (Brownlee, 2019). Decay rate as a multiplicative factor would be applied to learning rate. The reason using decay rate was that we wanted to allow the model to converge faster at the earlier epochs and slower towards the end so that it would avoid local min-

ima and had a better chance of finding a global minimum. Mathematically, learning rate at every epoch is as follow with "t" being the current epoch.

$$LearningRate_t = LearningRate_{t-1} * DecayRate \quad (1)$$

4.3 Tests Run on Audio Data

We divided the DAIC-WOZ audio files by question response, with a folder containing the response for each question by each participant. We then proceeded to divide each file by three seconds. When the split was done, any response less than three seconds was disregarded. Audio files that were more than three seconds were split into full three second clips and the reminder was also disregarded. This was then repeated for four,five,six and seven seconds. All splitting and processing was done by scripts on the cluster.

4.4 Tests Run on Transcript

DAIC-WOZ came in two parts: audio data and the transcript data. We processed the transcript data to organize replies by participant and questions. We extracted features from this using Textblob and Empath.

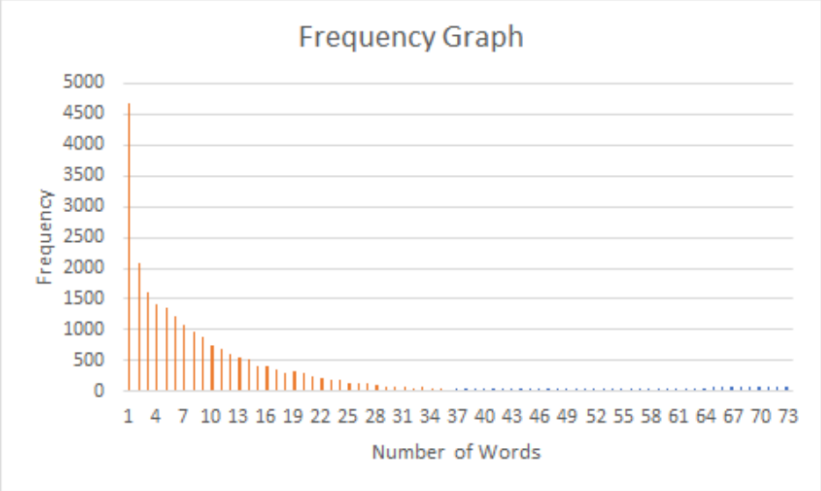


Figure 54: Frequency Graph for Transcript Data (Ali, 2018)

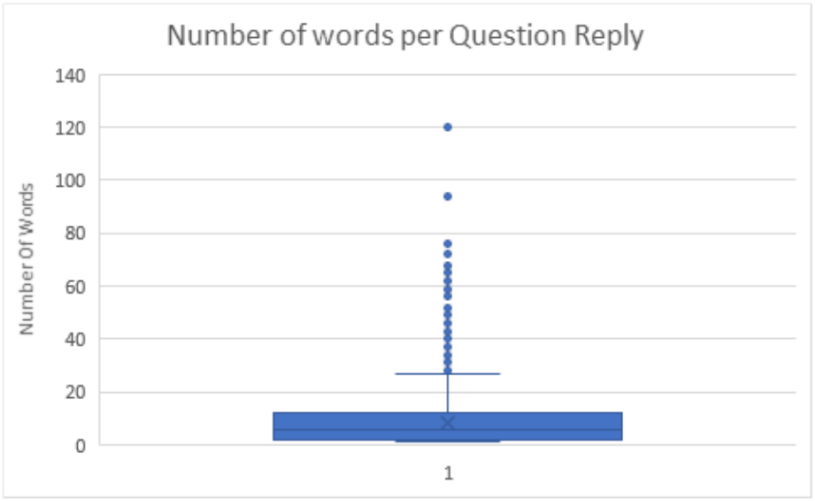


Figure 55: Histogram of Transcript Data (Ali, 2018)

As you can see in the above figures, the number of words in a question response varied. The number of words ranged from one, all the way up to 120 words. We set up a script to feed the pre-processed transcript data(a .csv file) and number for the minimum number of words allowed. For example, when we fed it the file and the number four, it made sure all responses that were under four words was filtered out. This is good because responses with few words would give less features and yield lower lower results. We experimented with 4,7,and 10 word minimums.

4.4.1 Tests Run on Audio and Transcript

The next step was to combine both audio and transcripts to attempt to get the best results. It was unrealistic to combine the sliced audio and get the transcript for that without doing it manually for tens of thousands of clips. So, the reasonable process was to do it question by question. So, we ran the pipeline through each question response audio clip. We then selected the text features for the transcript data and used a script with the merge command to merge the two. The result was that each row would contain both the audio features and transcript features for a question reply.

4.5 Topological Data Analysis

To test the viability of Betti curves (see section 2.10.5), we used machine learning methods similar to that of Section 4.1. Our data sets used all stemmed from five second DAIC-WOZ clips with no overlap, with different selections from these clips being used in training. To prepare, upper level and sub level Betti curves for each participant were extracted into separate folders, then several data sets were constructed by pulling specific curves from each participant. These included random 5 curves, the first curve for each question, and the first 5 curves. Another data set was also made which combined the openSMILE features of the first 5 clips from each participant with the first 5 curves from each participant. This is done using a truncated version of openSMILE with only 176 features. For each data set, both a upper level and sub level set is formed separately. In total, this leaves us with 8 unique data sets. After each data set is formed, they are fed through dimensionality reduction using PCA, kPCA, and Chi^2 (see section 2.4). Chi^2 chooses up to the total number of features, leaving at least one data set with all features. kPCA and PCA both choose up to 15 features. After this is complete, each data set is fed through a series of machine learning models: Naive Bayes, Linear Regression, k Nearest Neighbor with 3 neighbors (kNN3), k Nearest Neighbor with 5 neighbors, Support Vector Classification, XGBoost with 2 levels, XGBoost with 3 levels, XGBoost with 4 levels, and ADABoost. In total, we end up with 216 separate combinations of data set, dimensionality reduction, and machine learning model. The overview of the TDA machine learning pipeline can be seen in Figure 56.

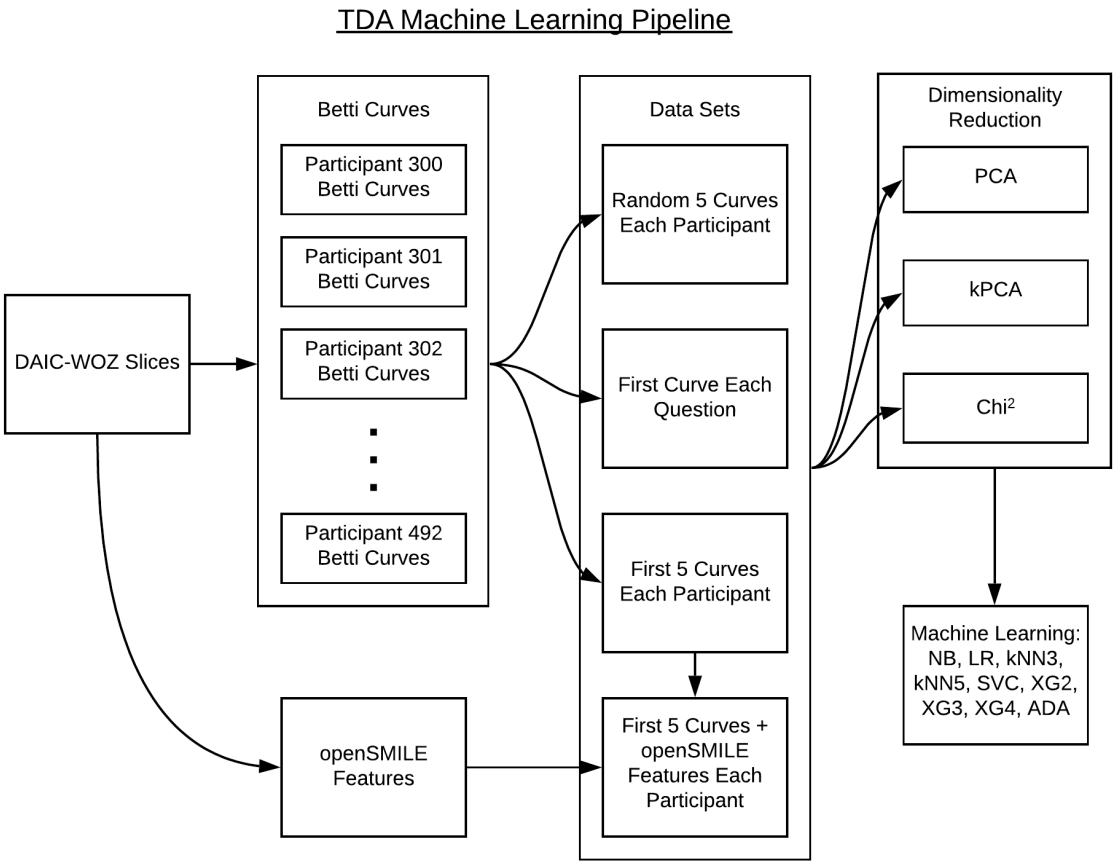


Figure 56: Pipeline for Machine Learning Using Betti Curves

4.6 Generative Adversarial Networks

We used a number of methods for generating text based on our existing data set. We first focused on using the Moodable data set to generate new texts using Texygen, an open source framework designed for easily running and comparing text generation methods (Zhu et al., 2018). We created a pipeline for running sequence GAN tests. Two sets of experiments were run: one on the ACE cluster, and one on the Turing cluster, as it had more resources available for experiments.

4.6.1 ACE Experiments

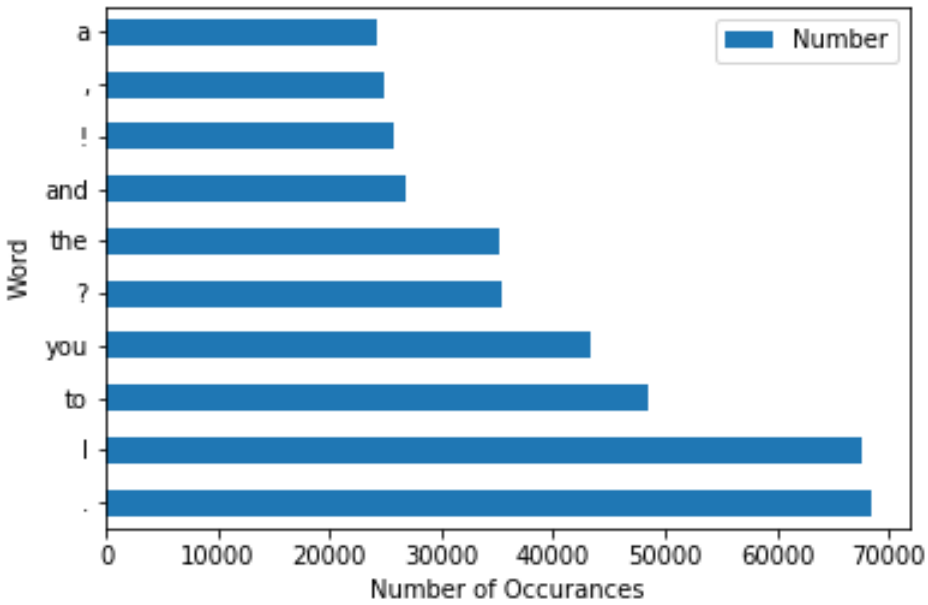


Figure 57: Moodable: Ten Most Frequent Words

Before the text generation methods could be run, the text messages had to be pre-processed appropriately. Text message data was provided in a CSV file, which indicated the user ID the text was from, and the text itself. This CSV was loaded using a python package called Pandas, which allowed for easier handling and visualization of data. Some statistics were gathered on the data set, before breaking it into subsections based on user ID. Figure 57 shows the top ten words that occur most frequently in the Moodable data set, while Figure 58 shows the participants with the ten highest text counts in the Moodable data set.

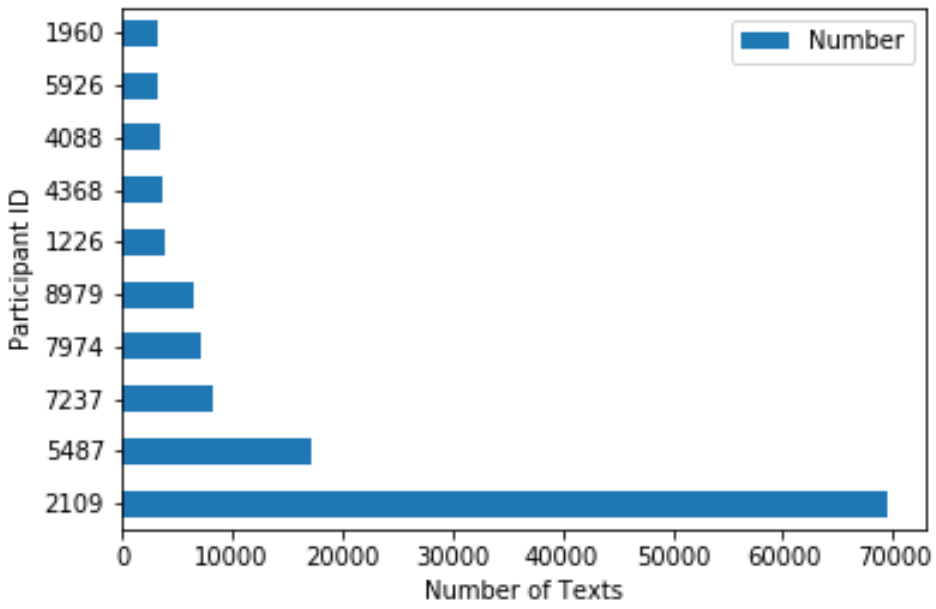


Figure 58: Moodable: Ten Participants with the Most Texts

Texygen requires input files that have a single text on each line, with all of the relevant data for each run in a single file. To simplify the data that the text generation techniques would have to learn from, the natural language toolkit (NLTK) was used to find the frequencies of each word within data subsets, and replace words that occurred less frequently than a given threshold with $< UNK >$. For the tests we ran this time, the frequency threshold was 5.

The first experiment was run using SeqGAN, using every text from Moodable as training data. This experiment ran out of time after twelve hours, the maximum amount of time allowed by the ACE cluster we were working on. The data set was then simplified, using only ten-thousand texts from a single participant, instead of using every text, but this also ran out of time. The number of epochs was then reduced from Texygen's default number of 80 pre-training epochs and 100 adversarial training epochs to only 20 pre-training epochs and 30 adversarial training epochs. The data set was further

simplified, only using 1074 texts from a single participant. With these parameters, the experiment successfully finished within the time limit. Further experiments were run using these values, for MLE training on its own (as a control group), RankGAN (as an improvement over SeqGAN), and LeakGAN (as it performed better with longer texts). As a result of the CPU specific requirements, GPU acceleration was not possible on the cluster at this point.

After running these experiments, we used matplotlib to derive and plot the NLL-Loss and BLEU scores. Other statistics were automatically generated by Texygen itself, such as NLL-loss, but these were somewhat problematic to use, as the output CSVs generated by each text generation method were formatted inconsistently.

4.6.2 Turing Experiments

After ACE proved not to be optimal for running Tensorflow and RelGAN, we gained access to the Turing cluster, which allowed for far more epochs to be run using GPU acceleration. Some further improvements were made on the experimental process, after the rests run on ACE.

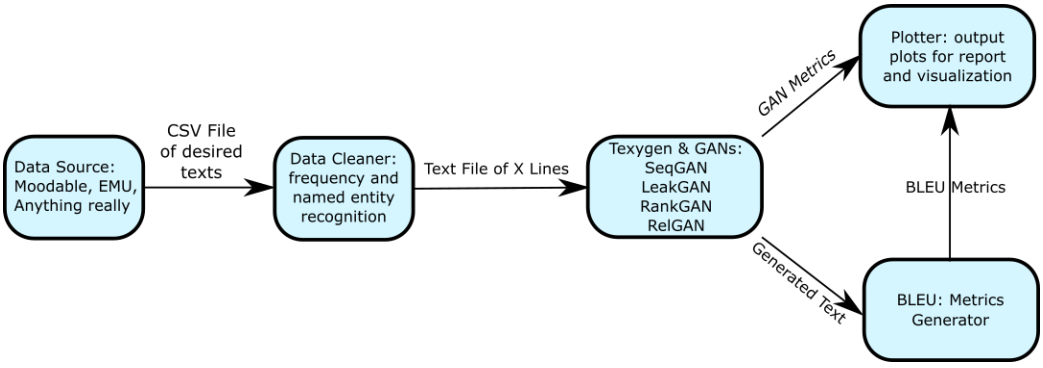


Figure 59: Text Generation Pipeline

Process Improvements The overall process was improved by modifying the existing frameworks. Texygen was forked and modified to accept a range of new command line arguments to facilitate much faster iteration of experiments. Arguments were added to control output file path, pre-training

epochs, adversarial training epochs, and CSV output path.[3] Furthermore, we developed a script that would automatically generate batch files for submitting new jobs to the Turing cluster. The pipeline created for text processing during this stage can be seen in Figure 59.

RelGAN Additionally, RelGAN, which was not included in Texygen, was set up to run with our data set. RelGAN does not accept most data sets by default, so it was also expanded to accept multiple data sets.[4] Due to the different requirements needed to run RelGAN, a separate batch file generator was created for RelGAN jobs.

5 Results

5.1 Machine Learning

5.1.1 GAD-7 Experiments on EMU

Although most experiments we did throughout this project used PHQ labels, this sub-project used GAD-7 labels to make predictions. GAD-7 have shown to be reliable predictors for anxiety, which is generally easier to predict than depression and have more consistent symptoms (**?**). Generally, in our GAD-7 experiments we found that tree based models and cutoff performed better. In addition, there were better results when using up-sampling compared to down-sampling. Tree based models have been shown to perform better in previous studies as well (Yang et al., 2016). We attribute the experiment results being skewed towards binary cutoff of 5 due to the EMU dataset's dispersion; most participants have a GAD-7 score of less than 10. The best performing test runs for our GAD-7 Experiments can be seen in Figure 60. As mentioned earlier, the top 5 test runs used either XGB or ADA methods. In addition, all top test runs used a binary cutoff of 5. Models also performed best when using our up-sampling technique; it is also evident that SCB works best for these experiments.

[3]This modified code can be found at https://github.com/Nicolas-Pingal/Texygen
[4]This modified code can be seen at https://github.com/Nicolas-Pingal/RelGAN

Model	PHQ Binary Cutoff	Resampling Technique	Audio Split	Overlap	Aggregation Method	F1 Score
XGB	5	Up-sampling	2 Clips	50%	Voting	0.828
XGB	5	Up-sampling	2 Clips	75%	Voting	0.82
ADA	5	Up-sampling	5 Clips	50%	Median	0.811
ADA	5	Up-sampling	5 Clips	50%	Voting	0.806
ADA	5	Up-sampling	3 Clips	50%	Mean	0.805

Figure 60: Classification: GAD-7 Experiments Top F1 Scores

As seen in Figure 61, even though the F1 scores were significantly lower, most assertions about tree based methods performing better still hold true. However, in this section of test runs, down-sampling performed better. Down-sampling works better for binary cutoff of 10 as a result of the dataset becoming more skewed towards depressed participants when using this cutoff value. Furthermore, SCB still seemed to improve model's performance.

Model	GAD7 Binary Cutoff	Resampling Technique	Audio Split	Overlap	Aggregation Method	F1 Score
RF	10	Down-sampling	2 Clips	50%	Max	0.667
XGB	10	Down-sampling	2 Clips	75%	Max	0.667
RF	10	Down-sampling	2 Clips	50%	Max	0.652
kNN	10	Down-sampling	3 Clips	25%	Mean	0.632
ADA	10	Down-sampling	3 Clips	25%	Median	0.629

Figure 61: Classification: Best Test runs for Cutoff=10

We converted the regression predictions into binary predictions, to compare them with a direct binary prediction. The results were slightly lower, so it is hard to deduce any meaningful conclusions.

Model	GAD7 Binary Cutoff	Resampling Technique	Audio Split	Overlap	Aggregation Method	F1 Score
XGB	10	Down-sampling	2 Clips	75%	Voting	0.811
XGB	10	Down-sampling	3 Clips	25%	Voting	0.791
kNN	5	Up-sampling	3 Clips	75%	Voting	0.771
XGB	10	Down-sampling	2 Clips	0%	Voting	0.769
RF	5	Up-sampling	4 Clips	25%	Voting	0.767

Figure 62: Regression: GAD-7 Experiments Top F1 Scores

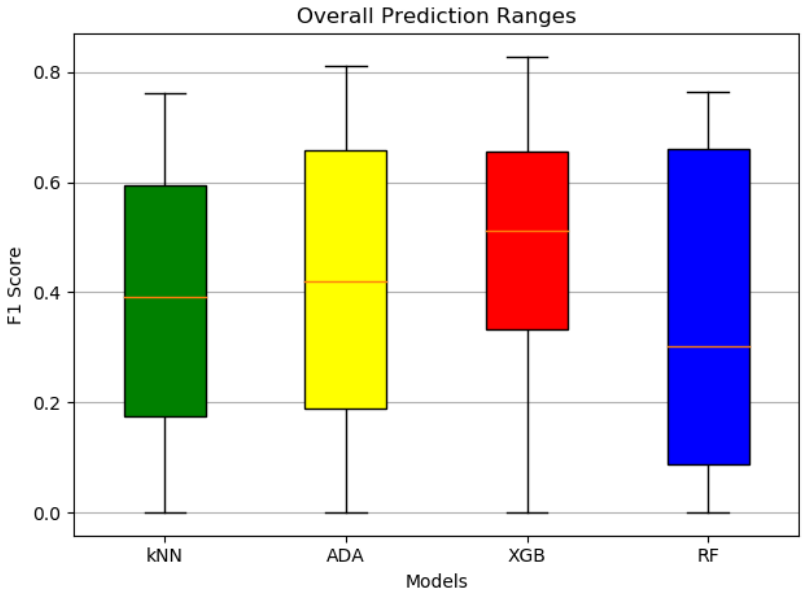


Figure 63: F1 Score distribution across Models

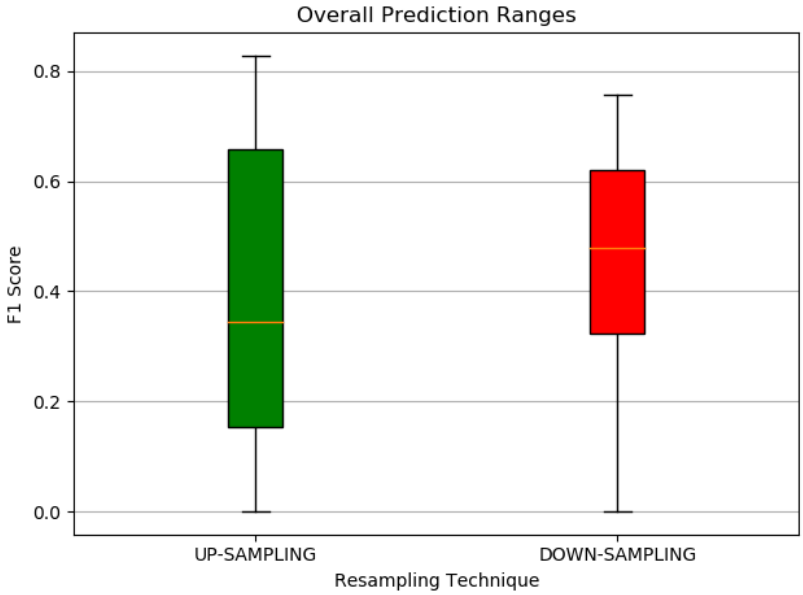


Figure 64: F1 Score distributions among Resampling Techniques

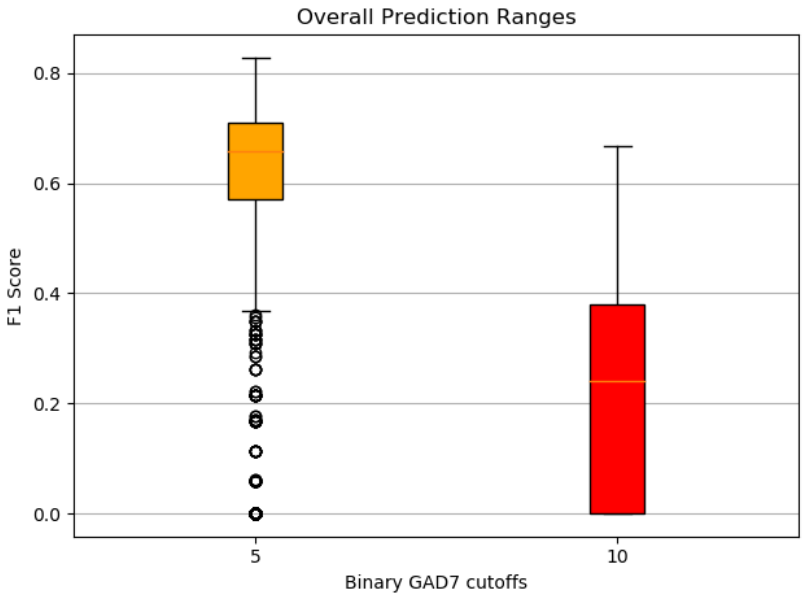


Figure 65: F1 Score distributions among Binary Cutoffs

Original dataset experiments We were able to do an experiment using the original data without doing any splits or overlaps. The results were relatively poor compared to SCB and split audio experiment results. The highest F1 score was 0.528 with RF model and a cutoff of 10 using up sampling.

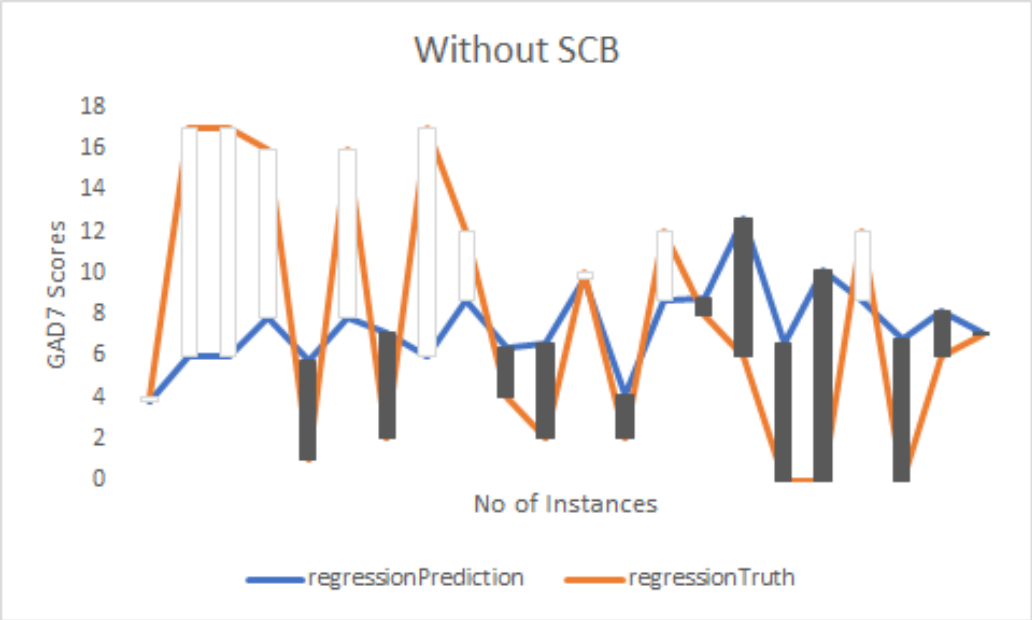


Figure 66: Predictions for un-split and no-overlap

5.1.2 PHQ-8 Experiments with EMU

Our experiments on the EMU dataset using PHQ-8 performed slightly worse than our experiments using GAD-7. However, compared to previous MQP's experiments using audio features from EMU, we found significantly higher results as depicted in figure 67 and figure 68. Figure 67 shows box-plots of all model experiments. Highest F1 score across all model tests doesn't reach a F1 score higher than 0.7. Result improvements can be mainly attributed to the increase in experiments and incorporation of SCB technique.

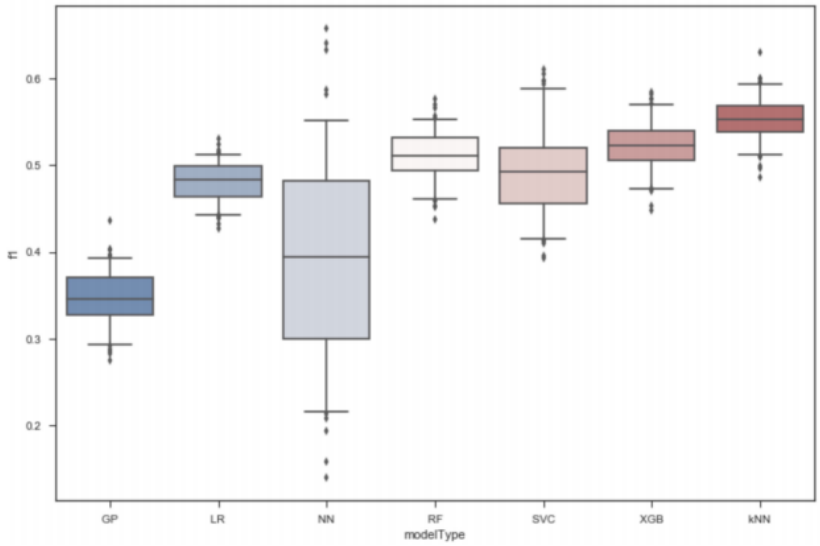


Figure 67: Previous MQP EMU dataset experiment results (Resom et al., 2019)

Model	PHQ Binary Cutoff	Resampling Technique	Audio Split	Overlap	Aggregation Method	F1 Score
ADA	5	Up-sampling	2 Clips	0%	Voting	0.813
ADA	5	Up-sampling	7 Clips	50%	Mean	0.809
ADA	5	Up-sampling	7 Clips	75%	Mean	0.805
ADA	5	Up-sampling	5 Clips	75%	Max	0.800
ADA	5	Up-sampling	6 Clips	0%	Median	0.791

Figure 68: EMU PHQ Experiments Best Results

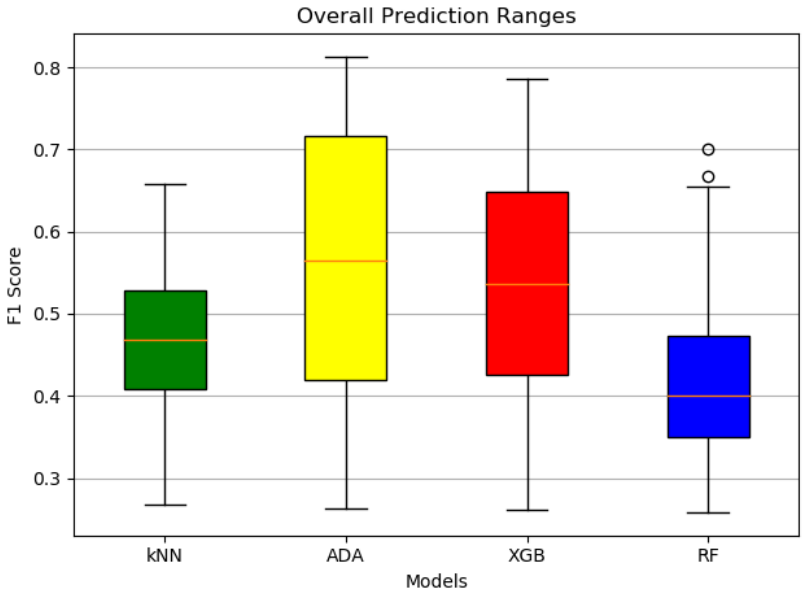


Figure 69: F1 Score distribution across Models

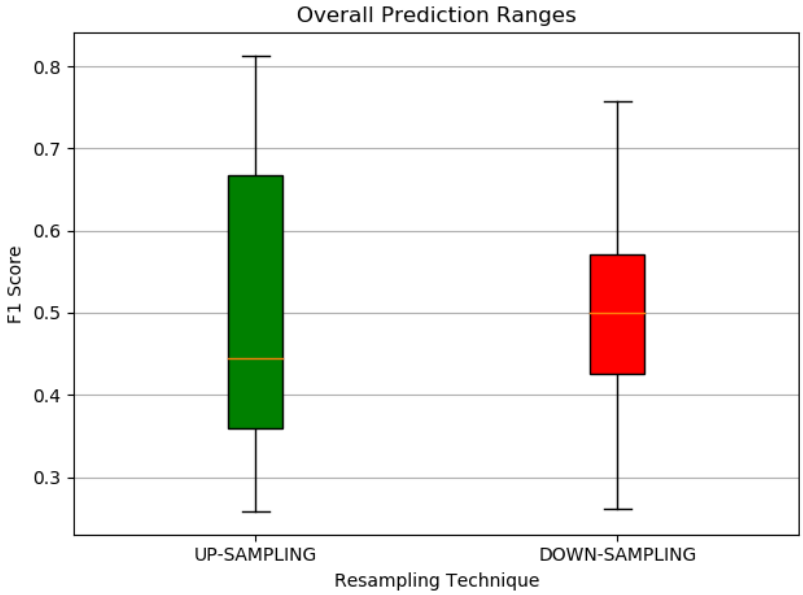


Figure 70: F1 Score distributions among Resampling Techniques

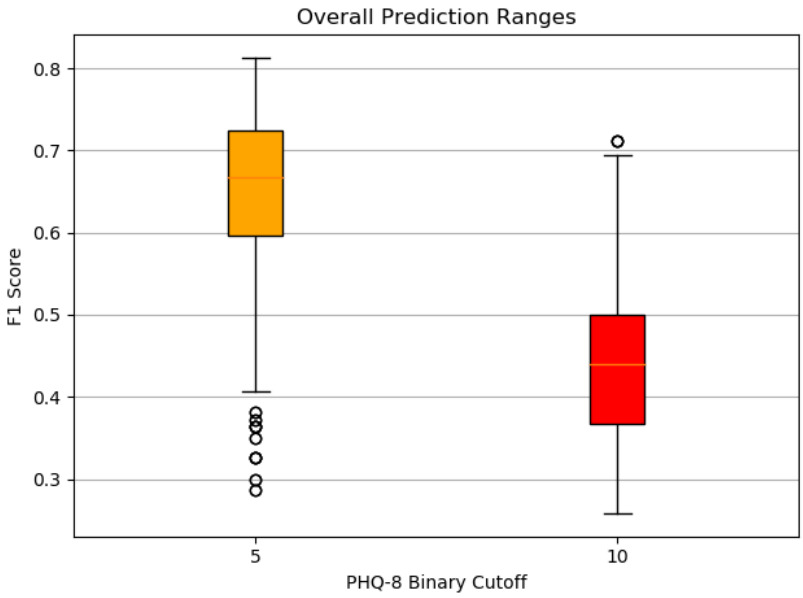


Figure 71: F1 Score distributions among Binary Cutoffs

5.1.3 PHQ-8 Experiments with EMU+Moodable

Our Machine Learning experiments have encompassed using Moodable as well. Specifically, we were able to experiment the effectivness of using the combined dataset of Moodable and EMU. We completed this phase of the experiment early on in our research project. Consequently, we used only three models to experiment with this dataset: kNN, SVM and RF.

As a result of the limited variation test parameters, the combination of EMU and Moodable has performed below EMU dataset separately.

Model	PHQ-8 Binary Cutoff	Resampling Technique	Audio Split	Overlap	Aggregation Method	F1 Score
kNN	10	Up-sampling	2 Sec	75%	Max	0.692
kNN	10	Up-sampling	3 Sec	50%	Max	0.675
kNN	10	Up-sampling	2 Sec	75%	Mean	0.663
SVM	10	Up-sampling	3 Sec	0%	Median	0.659
SVM	10	Up-sampling	3 Sec	50%	Max	0.657

Figure 72: EMU+Moodable PHQ Experiments Best Results

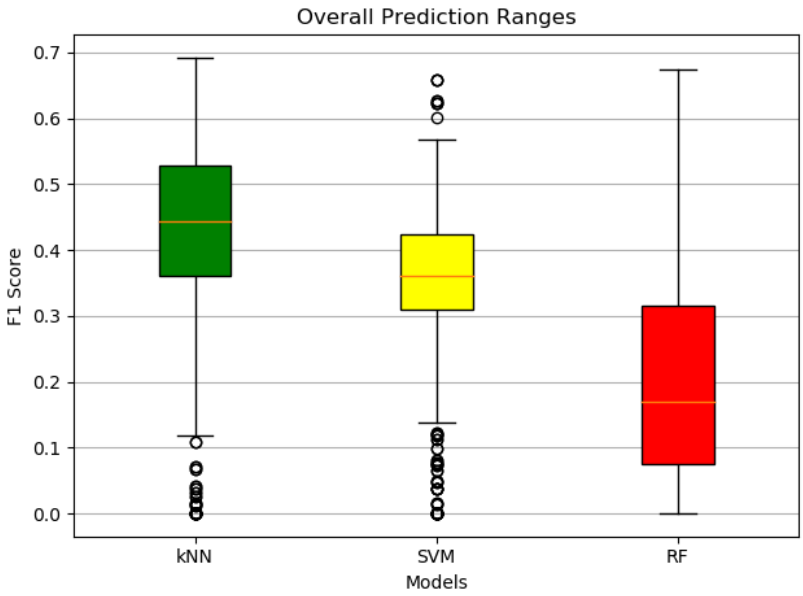


Figure 73: F1 Score distributions for EMU+Moodable

5.1.4 DAIC-WOZ Audio Experiments

We used DAIC-WOZ data for the majority of our Machine Learning experiment scope. We used this dataset to predict PHQ-8 labels. Since this dataset is readily accesible to future researchers, experiment results acquired using DAIC-WOZ can be compared with other works and aid as building blocks for future experiments.

Using DAIC-WOZ data, we were able to analyze different factors affecting a Machine Learning method's effectiveness in predicting depression from audio features. We also ran the experiments across different participant splits. For this section of experiments, we were able to run tests for ADA, XGB, kNN, and RF models. SVM and MLP models weren't able to ran within our allocated cluster computing time.

Random Generated Train-Test Split Since both our Deep Learning sub project and Machine Learning were under performing using official DAIC-WOZ split, we opted to generate a random split. The random split we used had a 70/30 ratio in favor of training split. In addition, we maintained the same distribution of depressed and non-depressed participants across both

splits. As seen in figure 74, we obtained a F1 score of 0.7.

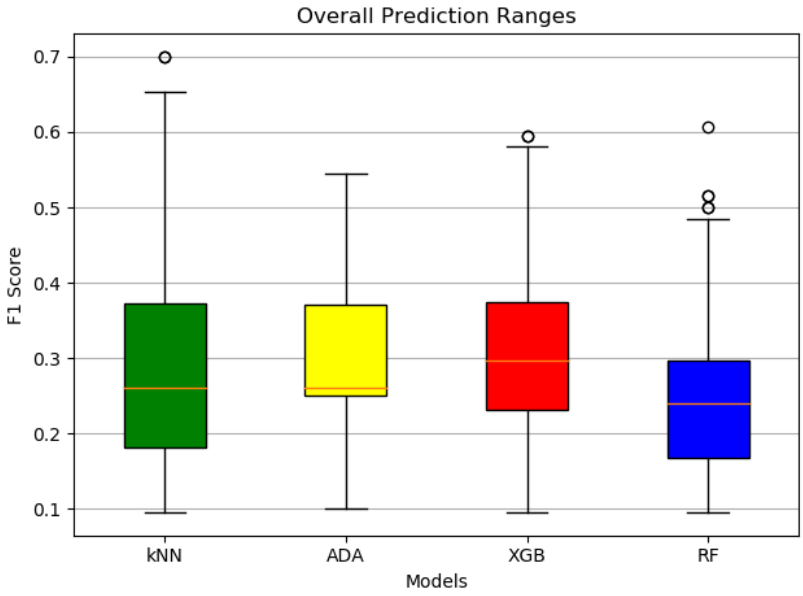


Figure 74: F1 Score distribution for Participant Grouping

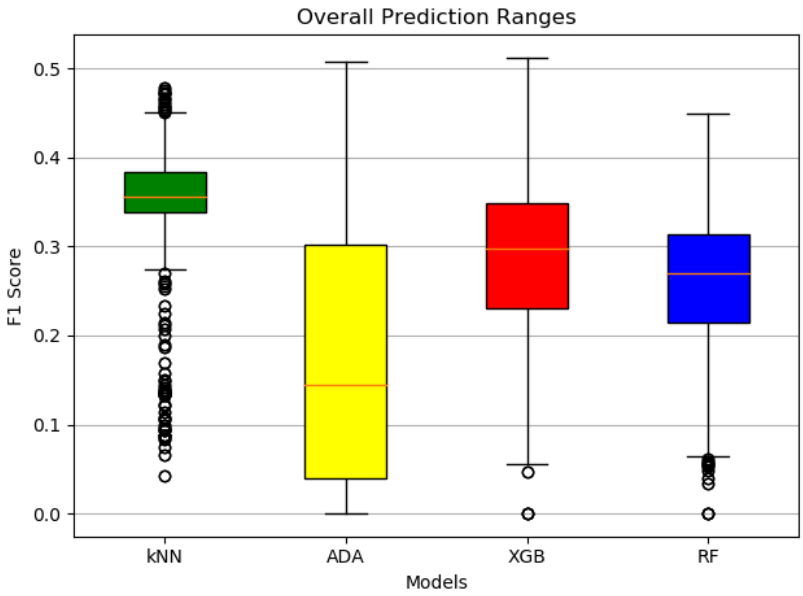


Figure 75: F1 Score distribution for Participant + Question Grouping

DAIC-WOZ Official Split We run experiments using official DAIC-WOZ dataset splits: train_split.csv, dev_split.csv, and test_split.csv. However, since the test split didn't contain labels we opted to treat dev split as test set. This left us with 107 participants in train and 35 participants in test set. In general, results weren't promising as we didn't obtain F1 scores that were better 0.5 range. Since we were able to reach upto a F1 score of 0.7 using our generated splits, this can be attributed to the imbalanced nature of the given splits.

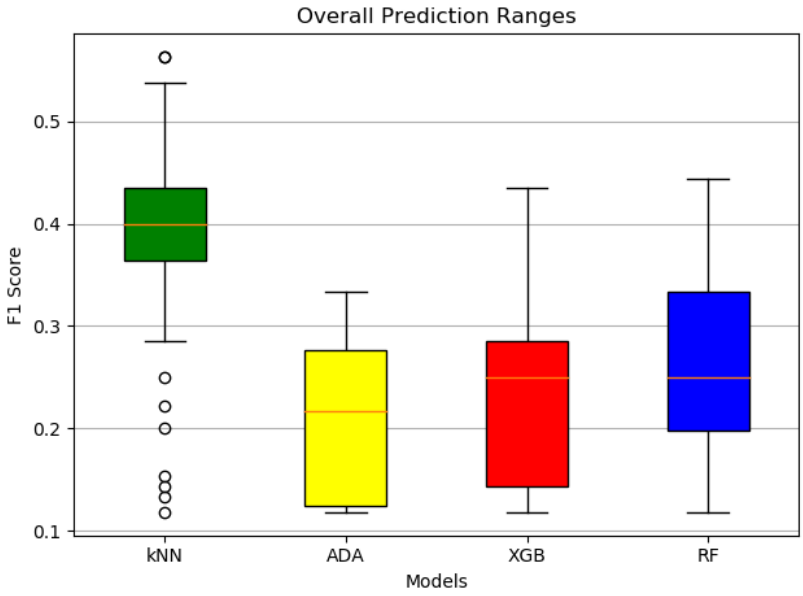


Figure 76: F1 Score distribution for Participant Grouping

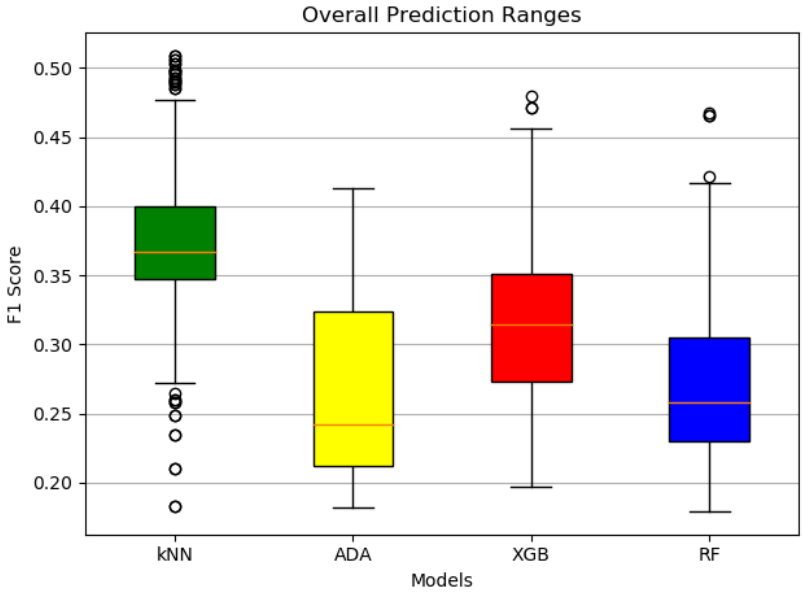


Figure 77: F1 Score distribution for Participant + Question Grouping

5.1.5 DAIC-WOZ Transcript+Audio Experiments

DIAC-WOZ came with a transcript that corresponded to the audio files. We decided to try and attain higher F1 scores for predicting depression by combining both the text and audio data. We decided to do it based on question replies, and not clips of a specific duration, as done in our audio-only experiments.

We extracted openSMILE features for each question reply from the audio data. We then extracted Textblob and Empath features for each response from the transcript data and matched them. When we ran the machine learning models on them, and found that none were able to yield above 0.5 F1 scores. In order to improve the results, we added filters. These filters would remove all question responses in which the participant has used less than X words.

5.1.6 Audio vs Transcript + Audio Predictions Analysis

Audio results were based on question responses divided up into clips which allowed higher precision. On the other hand, transcript + audio predictions limited us to using only response per question and didn't allow us to delve deeper.

Our initial tests with all transcript data yielded low data, with almost all results below 0.5. Looking at the word count distribution, it shows that there are a lot of replies with only one or two words. The text extraction feature would not be able to get a lot of data from one or two word replies, and as a result, it caused the results to be worse.

Our results for when we did only replies with four or more words, seven or more words, and ten or more words, yielded better results. Without the smaller responses messing up the data, the machine learning models were able to attain higher F1 scores. Surprisingly, the best results were when replies with four or more words were used. The best result for four or more word replies was 0.62, while that for seven or more words was 0.602 and for

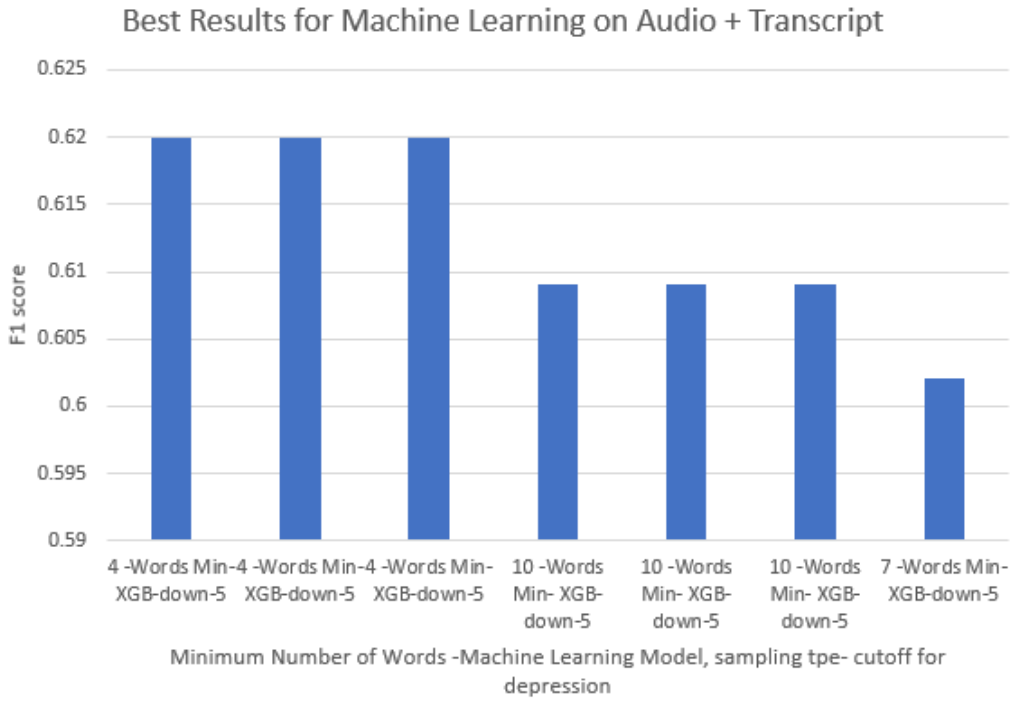


Figure 78: Best Performers: DAIC-WOZ Audio + Transcript

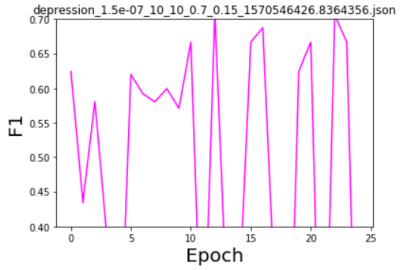
10 or more words was 0.609 Unfortunately, the highest F1 scores from the combined data were not as good as audio only.

5.2 Deep Learning

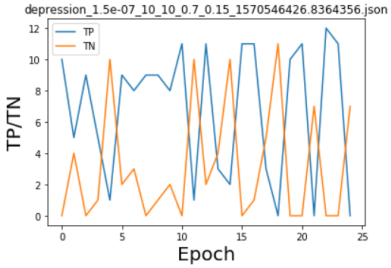
5.2.1 EMU/Moodable: CNN

After setting up and laying out the source code to build our model, we started testing the combined data sets. The experiment on EMU and Moodable was carried out during our first term. The results from the test run show that our model was not learning anything. Figure 79 shows the results of our trained CNN model. The F1-score plots indicates that the model was not validating good enough on each epoch and it did not improve over 60 epochs. The accuracy was also fluctuating between 0.3 and 0.5 and it also did not improve over time. The test loss and the train loss did not converge at any epoch and the test loss was still higher than train loss. Therefore,

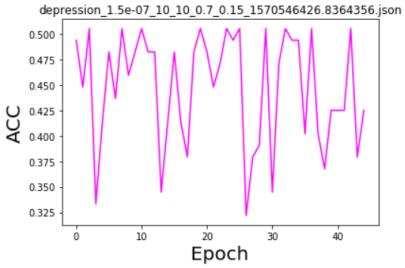
we concluded that our model was not learning enough. There were more experiments we could do to find out a better way to fit our model and obtain good results.



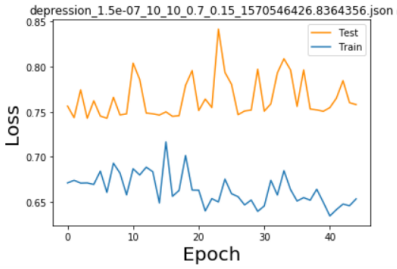
(a) F1-score against epochs



(b) True Positive/True Negative against epochs



(c) Accuracy against epochs



(d) Loss against epochs

Figure 79: Results from running CNN on EMU and Moodable Data

5.2.2 DAIC-WOZ: CNN

We started testing our default CNN models with the DAIC-WOZ data set in our second term. First of all, we found out that running CNN on the whole DAIC-WOZ data set was out of our computational limitation. The whole DAIC-WOZ data set contains total questions counts of 7938 with available PHQ scores regardless of participants IDs. Training our default CNN model the first time took more than 12 hours to complete, which exceeded the time limit of our cluster. Thus, in order to save a significant amount of training time, image caching was implemented. Even so, the number of epochs we could set is no higher than 50 depending on other parameters as well. Thus, for test running, we set the question limits to 20 for each participants and

used random split ratio of 70 percent train set and 30 percent test set. The resulting data set then contained question counts of 2840. This regenerated data set would be used for experimenting until we found a proper learning CNN model.

Testing number of layers In the first approach we planned to experiment, we started off simplifying model with 1 convolution, 1 max pool, 1 dense and 1 output layer with binary classes. Then, another experiment would be increased layers in convolution which results in 2 convolutions, 2 max pools, 1 dense and 1 output. These both experiments and the upcoming experiments also included Gender label. The gender label was added after convolution and into the fully-connected layers. The parameters that we used for both experiments are as follow:

- Number of Epochs: 150
- Image size: 256x128
- Image Channel: 1
- Learning rate: 0.0000005
- Dropout: 0.3
- Optimizer: Stochastic Gradient Descent
- Loss function: Cross Entropy
- Train/Test split: 70/30 ratio

After running both experiments, we found out from the table 1 that both experiments ran poorly. Adding one more layer into convolution did not seem to improve the model. The table below compares the confusion matrices of both experiments. The first table shows an experiment with 1 convolution, 1 max pool and the second one shows an experiment with 2 convolutions and 2 max pools.

We can see here that in both experiments only a few people were identified as being depressed and the F1 score could not achieve beyond 0.2. Thus, adding more layers did not seem to improve the model.

Actual—Prediction	Depressed	Non-depressed
Depressed	27	160
Non-depressed	66	534
F1 score:	0.19285714285714287	
Actual—Prediction	Depressed	Non-depressed
Depressed	17	170
Non-depressed	21	579
F1 score:	0.15111111111111114	

Table 1: Confusion Matrix from testing layers

Testing Kernel Sizes The next step we took to improve our model was to try to find a kernel size that would work well for DAIC-WOZ data. The kernel size was important for finding features inside the image spectrograms. If the kernel size was too big, the features would become less distinct, but if the kernel size was too small, it would lose the bigger picture of the important features. Thus, to find the features in the image spectrograms, we planned out multiple experiments with varying kernel sizes. The overall structure of our model in those experiments was 2 convolutions, 2 max pools, 1 dense and 1 output. The list below contains the parameters and kernel sizes we adjusted to run those experiments.

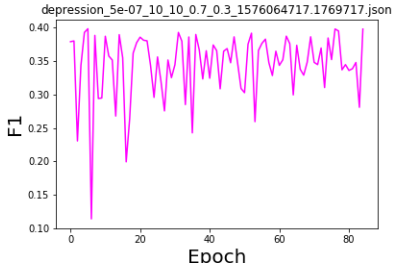
- Number of Epochs: 100
- Image size: 256x128
- Image Channel: 1
- Learning rate: 0.0000005
- Dropout: 0.3
- Optimizer: Stochastic Gradient Descent
- Loss function: Cross Entropy
- Train/Test split: 70/30 ratio
- Convolution kernel size: 3x3 to 10x10
- Max pool kernel size: 3x3 to 6x6

One set of experiments included testing with varying convolution kernel sizes from 3x3 to 10x10 with one specific max pool sizes. Since we were going to test 4 max pool sizes up to 6, we had total of 32 experiments to find the kernel sizes. After testing all those 32 experiments, the results were not too impressive but not all of those experiments were performing as poorly as before. We picked out the best performing model from each set of experiments. The results are as follow.

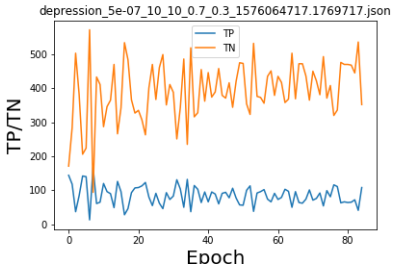
Model	Kernel	TP	FN	FP	TN	Accuracy	F1
2conv,2max	6x6,3x3	120	67	305	295	0.5273189327	0.3921568627
2conv,2max	4x4,4x4	101	86	236	364	0.5908513342	0.3854961832
2conv,2max	7x7,5x5	108	79	248	352	0.584498094	0.3977900552
2conv,2max	7x7,6x6	80	107	180	420	0.6353240152	0.3579418345

Figure 80: The confusion matrix data for four best performing experiments

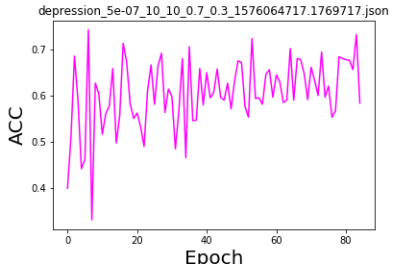
The two highlighted runs had the best F1 scores out of all our experiments. The results were still not impressive but we have seen some improvements. The F1 scores exceeded 0.2 and became almost close to 0.4. After all, we had not yet seen a single experiment that gave us F1 score which exceeds 0.4. To find out what had happened during the experiment, we visualized the best performing experiment which was with 7x7 convolution kernel size and 5x5 max pool kernel size.



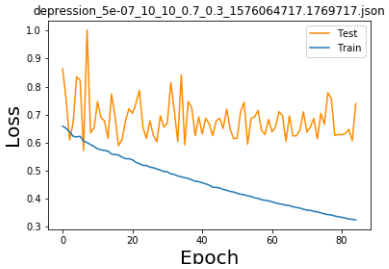
(a) F1-score against epochs



(b) True Positive/True Negative against epochs



(c) Accuracy against epochs



(d) Loss against epochs

Figure 81: Results from running 7x7 convolution and 5x5 max pool

From the figure 81, we can see that F1 score did not exceed 0.4 but the pattern shows that F1 score becomes more stable towards more epochs. The True Positive/True Negative plot does not show that the model was learning, as ideally, the true positive curve should rise and the true negative curve should drop. Accuracy seemed to rise eventually with more epochs. Train loss curve eventually dropped but the test loss curve was fluctuating and could not go down and converge towards train loss.

CNN & openSMILE Another approach we proposed earlier was to combine the convoluted feature set with other distinctive feature sets such as openSMILE. Continuing with the CNN alone was unpredictable at this point, since we had other parameters such as learning rates and decays to test, and the number of experiments we needed to test was exponentially larger if CNN alone would not predict well after those experiments. Thus, we decided to see how much F1 and Accuracy we could get by combining convoluted and openSMILE feature sets. We used the same structure and parameters from

Model	Epochs	Accuracy	F1
CNN+openSMILE25	100	0.6454	0.4324
CNN+openSMILE176	150	0.7455	0.0094

Table 2: Results from Running CNN and openSMILE

the previous runs.

- Number of Epochs: 100, 150
- Image size: 256x128
- Image Channel: 1
- Learning rate: 0.0000005
- Dropout: 0.3
- Optimizer: Stochastic Gradient Descent
- Loss function: Cross Entropy
- Train/Test split: 70/30 ratio
- Convolution kernel size: 3x3 to 10x10
- Max pool kernel size: 3x3 to 6x6

Table 2 represents the two trial experiments we tested with CNN and openSMILE together. CNN with openSMILE did not seem to be performing bad on 25 features of openSMILE, with an F1 score of 0.43, higher than those we got from previous runs. Another experiment with 176 openSMILE features showed us a better accuracy of 74%, but had a mediocre F1 score of 0.0094. From this experiment, we could adjust our feed forward network and other parameters like optimizer for our final term.

openSMILE & Neural Net For the next step, we adjusted the layers of feed forward network. The input data was 176 openSMILE features, from the entire DAIC-WOZ data set, using both up-sampling and down-sampling. Up-sampling was usable in this case, due to each experiment only running for a few hours. At the same time, while we were planning the experiments, we also tried to determine if different learning rates would give better results. The following list include the layers and parameters we adjusted.

- Number of Epochs: 50
- Input Layer: 176
- Dense(Hidden) Layer 1: 256
- Dense(Hidden) Layer 2: 128
- Output Layer: 2
- Optimizer: Adam
- Decay Rate: 0.8
- Dropout: 0.3
- Optimizer: Stochastic Gradient Descent
- Loss function: Cross Entropy
- Train/Test split: 70/30 ratio
- Convolution kernel size: 3x3 to 10x10
- Max pool kernel size: 3x3 to 6x6

Optimizing our feed forward network gave us a few good results as shown in table 3. Most of them could achieve above an 0.6 F1 score, but their accuracy of below 0.5 left much to be desired. We tested these experiments using the same decay rate of 0.8. openSMILE with down-sampling and learning rate of 1e-5 already gives us a good result. This model would be used as a base model for further testing on CNN and openSMILE176 combined experiments.

Model	Sampling	Learning Rate	Accuracy	F1
openSMILE176	Down	1e-5	0.6512	0.6514
openSMILE176	Down	1e-6	0.6112	0.5945
openSMILE176	Down	1e-7	0.5008	0.5228
openSMILE176	Down	1e-8	0.4833	0.6094
openSMILE176	Up	1e-5	0.5000	0.0033
openSMILE176	Up	1e-6	0.5000	0.6654
openSMILE176	Up	1e-7	0.4989	0.6640
openSMILE176	Up	1e-8	0.5003	0.6667

Table 3: Results from testing openSMILE 176 features with different learning rates

Model	Sampling	Learning Rate	Accuracy	F1
openSMILE176	Down	1e-5	0.6572	0.6514
CNN+openSMILE176	Down	1e-5	0.6664	0.6678

Table 4: Comparison with and without CNN

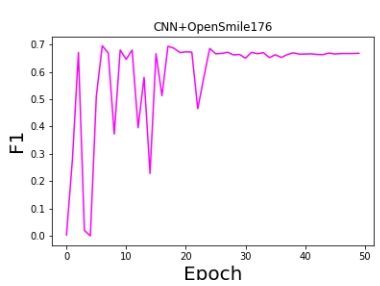
CNN & openSMILE with Adjusted Layers After optimizing our feed forward network, we tested CNN and openSMILE combined models using the layers and parameters from the optimized feed forward network, as we wanted to figure out which part of the CNN, either convolutional part, or feed forward network part, was performing worse. Table 4 shows the comparison between an experiment with only the openSMILE feature set, and another with both the CNN and openSMILE feature sets combined. The results indicate that optimizing the feed forward network improved our combined model even more than using the openSMILE feature set alone.

Testing Decay Rates It was also essential to test decay rates, after figuring out the proper learning rates for the model. Since decay rate is a multiplicative factor applied to learning rates at each epoch, higher decay rates would slowly reduce the learning rate. Table 5 shows how different decay rates affected our model. By comparing these decay rates, the results did not show any significant improvement.

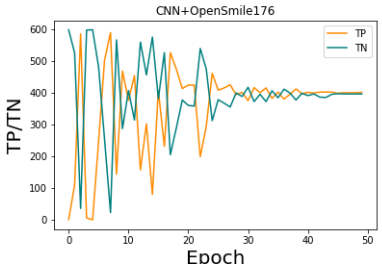
Model	Sampling	Learning Rate	Decay	Accuracy	F1
CNN+openSMILE176	Down	1e-5	0.6	0.6513	0.6499
CNN+openSMILE176	Down	1e-5	0.7	0.6519	0.6471
CNN+openSMILE176	Down	1e-5	0.8	0.6664	0.6678
CNN+openSMILE176	Down	1e-5	0.9	0.6488	0.6209

Table 5: Comparison of Different Decay Rates

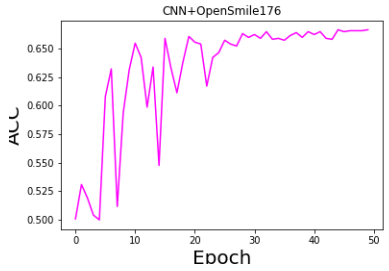
After all of these experiments, we saw a lot of improvements in accuracy and F1 score. We were able to achieve an accuracy and F1 score of 0.65 above. This can be seen in Figure 82, which shows that accuracy curve and F1 score curve eventually rose. Train loss and test loss also converged eventually. The amounts of True Positive went up and True Negative went down till they converge. These plots indicate that our model was learning and predicting well, but it was still far from perfect. Moreover, since we had worked with our own generated folds for experiments, we had yet to see how our model would perform on official DAIC-WOZ folds.



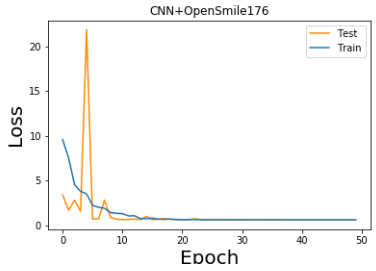
(a) F1-score against epochs



(b) True Positive/True Negative against epochs



(c) Accuracy against epochs



(d) Loss against epochs

Figure 82: Best Results from running CNN with openSMILE176

CNN & openSMILE with Adjusted Parameters and DAIC-WOZ Official Splits Since we had been testing on our own random folds with 70:30 splits, our results were not directly comparable with other researches. DAIC-WOZ data set had its own official train and test split. Thus, we tested our adjusted parameters again with the official train and test split. The table 6 shows the results from testing with DAIC-WOZ official splits. As we can see, the models did not perform as good as they did on random folds. The highest F1 score we could achieve was 0.5506 with respective accuracy of 0.5811. From the research by Chen et al. (Ma et al., 2016), DepAudioNet could achieve 0.52 F1 score. One reason why our models did not perform well on official splits was that, the official splits is grouped by participants and not by audio clips. Thus, this created a different split ratio when we re-grouped the official splits by audio clip.

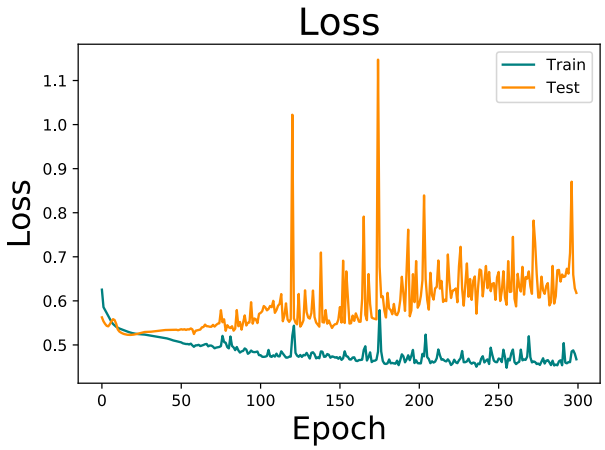
Model	Sampling	Learning Rate	Decay	Accuracy	F1
CNN+openSMILE176	Down	1e-5	0.2	0.5488	0.5147
CNN+openSMILE176	Down	1e-5	0.5	0.5149	0.4841
CNN+openSMILE176	Down	1e-5	0.8	0.5041	0.4647
CNN+openSMILE176	Down	1e-6	0.2	0.5811	0.5506
CNN+openSMILE176	Down	1e-6	0.5	0.5232	0.5232
CNN+openSMILE176	Down	1e-6	0.8	0.4909	0.5028
DepAudioNet					0.52

Table 6: Results from testing with DAIC-WOZ official splits

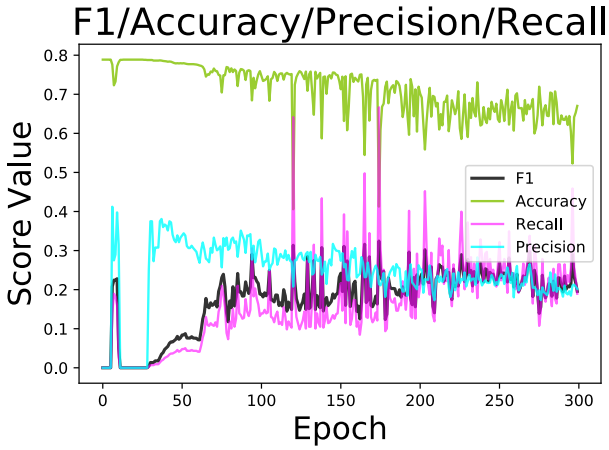
5.2.3 DAIC-WOZ: Sequential Model

Experiments were run on a model made using PyTorch as described in Sections 4.2.2 and 4.2.3. Batch size and output dimension were 1, while the optimizer type, learning rate, number of hidden nodes, number of layers, and dropout rate differed between experiments.

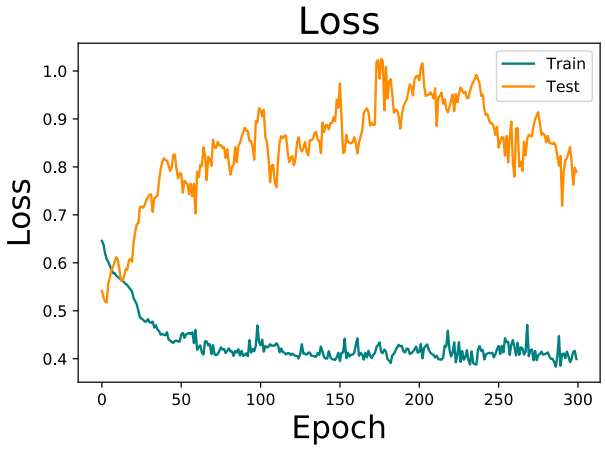
We first compared optimizer performance. We tried AdaDelta, RMSProp, and Adam with a learning rate of 1e-5, 128 hidden nodes, 1 layer, and no dropout. With these settings, we found that Adam performed the best, which was expected (see Figure 83). RMSProp performed slightly worse, however was able to perform better with a very small learning rate and a small number of hidden nodes. This will be discussed later. The model using Ada failed to learn anything with these hyperparamters and classified every example as non-depressed. We continued to use Adam for the majority of experiments. Our next experiment was with learning rate. We found that a value of 1e-5 was the most effective given the assumed 128 nodes and 1 layer.



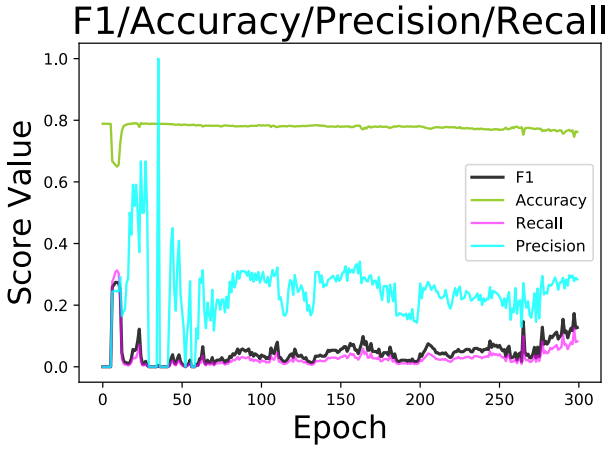
(a) Adam model



(b) Adam model



(c) RMSProp model



(d) RMSProp model

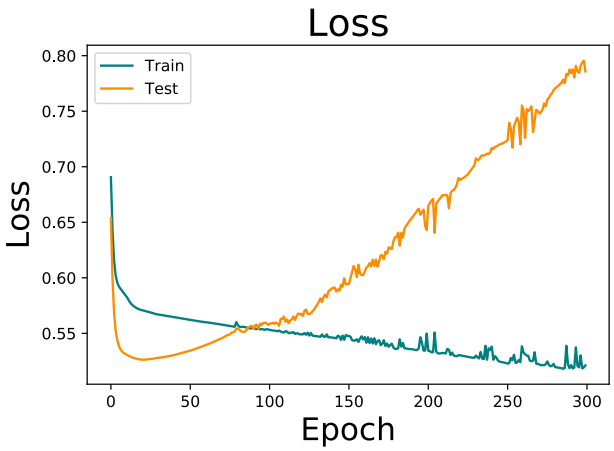
	TP	TN	FP	FN	ACC	PRE	REC	F1
Adam	83.0	1303.0	328.0	355.0	0.67	0.202	0.189	0.196
RMSProp	36.0	1540.0	91.0	402.0	0.762	0.283	0.082	0.127

(e) Metrics for each model. Actual TP/TN is 437/1632

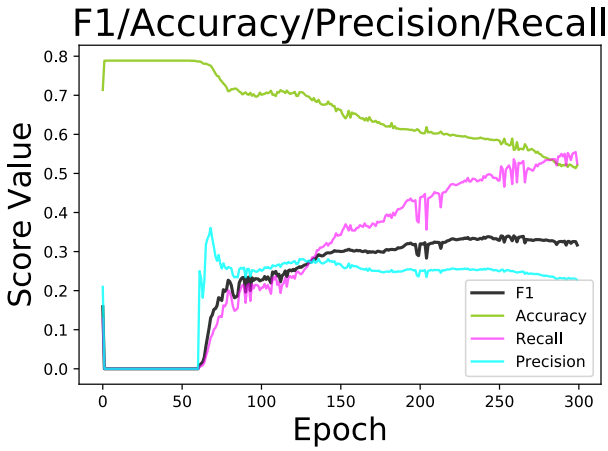
Figure 83: A comparison of Adam vs RMSProp with a learning rate of 1e-5, 128 nodes, and 1 layer

Random resampling of just the training set was introduced next. Because of the complexity of the input, random sampling made more sense than using SMOTE. As each input was a variable-length sequence of lists, SMOTE was more difficult to implement. Continuing to use Adam with a learning rate of 1e-5, we ran a set of experiments on different combinations of {128, 256} hidden nodes, {0, 0.2, 0.4} dropout, and {1, 2, 3} layers. These models quickly showed training and testing loss diverging and a very high variance in accuracy and F1 between epochs. Overfitting increased with an increase in hidden nodes and layers. Higher dropout values appeared to hinder model learning and, contrary to expectations, promote overfitting. This may be explained by the relatively small data set and small majority class within (21% of the data set). Losing any of the tiny amount information predicting true positives could have been harmful to the model. Noticing the overfitting prompted us to experiment next on the number of hidden nodes.

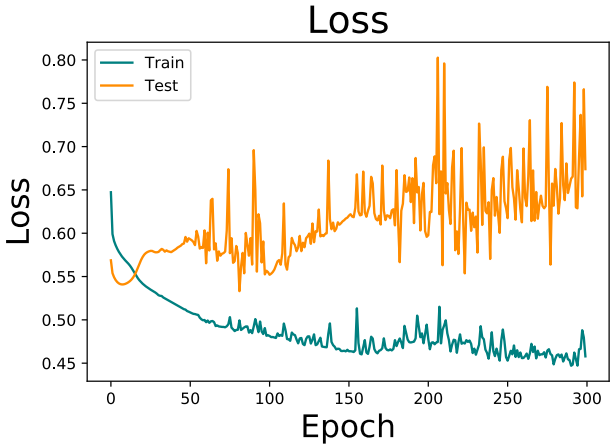
Using Adam with a learning rate of 1e-5 and 1 layer, we tested {10, 30, 50, 75, 100} hidden nodes. The results of these trials are shown in Figure 84. As the number of nodes increased, we saw less divergence in train vs test loss and accuracy. There was also more variance in the curves. Accuracy increased significantly, from roughly 0.5 with 10 nodes to around 0.65 with 100 nodes. Loss also improved. However, as the nodes increased the model became more overfit. With 10 nodes, the true positives were closest to the actual, but false positives were also the highest.



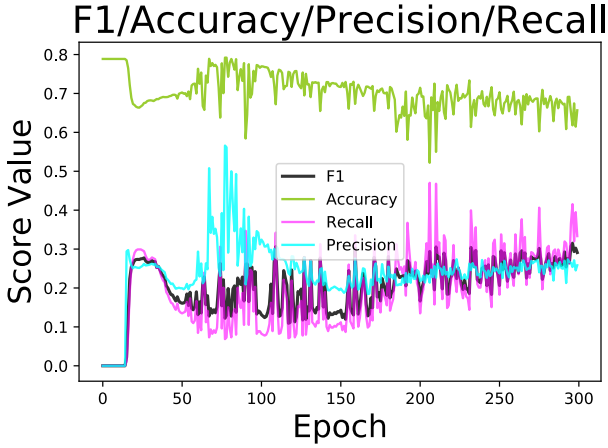
(a) 10 hidden nodes



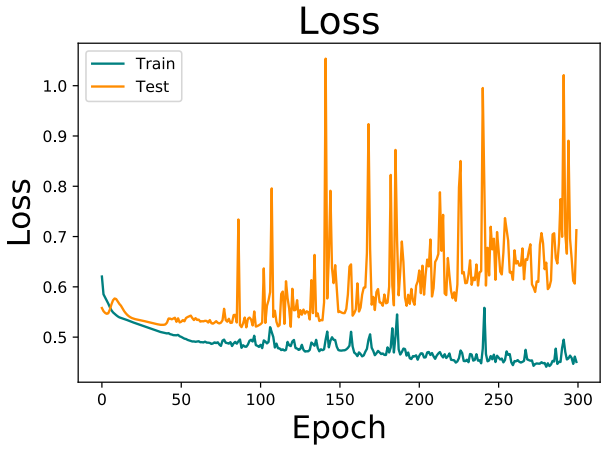
(b) 10 hidden nodes



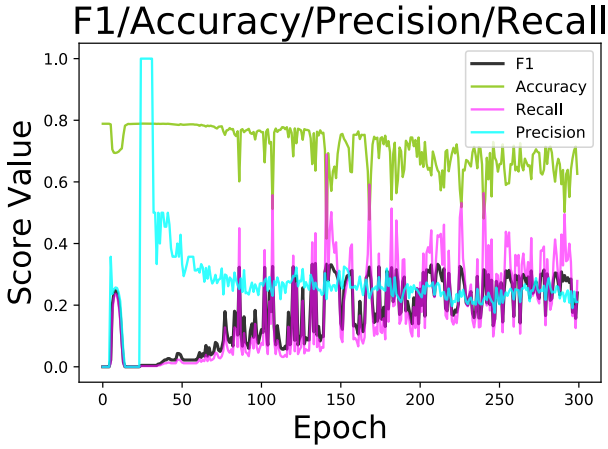
(c) 50 hidden nodes



(d) 50 hidden nodes



(e) 100 hidden nodes



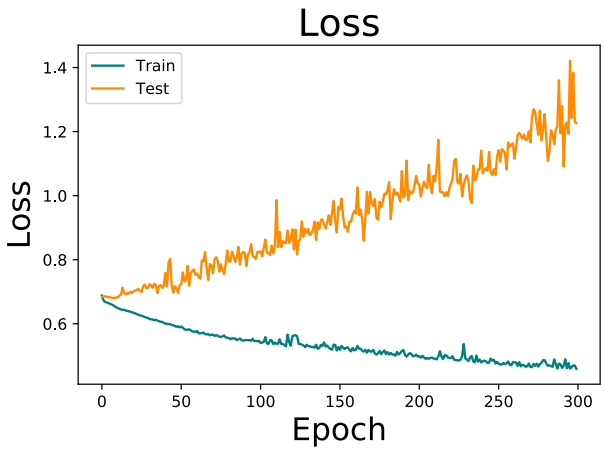
(f) 100 hidden nodes

	TP	TN	FP	FN	ACC	PRE	REC	F1
10 nodes	230.0	847.0	784.0	208.0	0.521	0.227	0.525	0.317
30 nodes	156.0	1240.0	391.0	282.0	0.675	0.285	0.356	0.317
50 nodes	146.0	1212.0	419.0	292.0	0.657	0.258	0.333	0.291
75 nodes	130.0	1148.0	483.0	308.0	0.618	0.212	0.297	0.247
100 nodes	122.0	1173.0	458.0	316.0	0.626	0.21	0.279	0.24

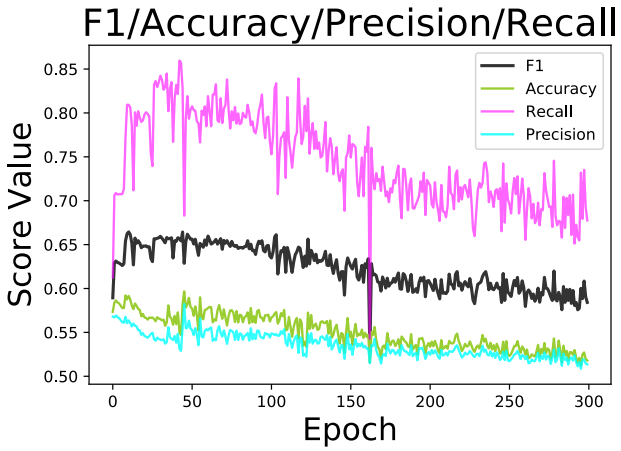
Figure 84: A comparison of different hidden node amounts. Actual TP/TN is 437/1631.

The next set of experiments was done to determine if it was more effective to train with both the test and train sets upsampled, instead of just the training set. We also simultaneously ran experiments using the os25 features and the os176 features, testing upsampling test on both data sets. We found that the models performed far better when both test and training sets were upsampled as shown in Figure 85. Both models were run using Adam with a 1e-5 learning rate, 128 nodes and 1 layer. Although the models both suffer from diverging loss and decreasing F1 score, they remain comparable. In Figures 85c and 85d, the metrics point to worse than random with an accuracy of 0.45 and an F1 of 0.32. The untouched test set had 437 questions in the minority class (depressed) and 1631 in the majority. Random resampling of the minority class up to 1631 examples yielded better results. As seen in 85a and 85b, the overall fitness is better with an accuracy of 0.52 and an F1 of 0.59. However, the test accuracy shows a decreasing trend, while the model with the untouched test set shows a fairly constant test accuracy.

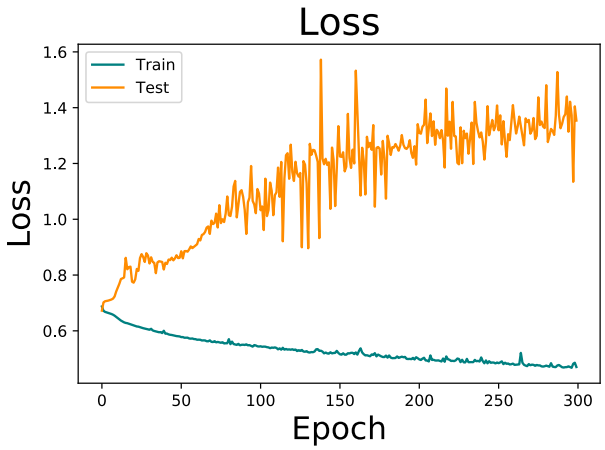
Regarding the features, results were unclear at the beginning. We have no direct comparisons between the os25 and os176 features, although we witnessed a general trend towards better results and less overfitting in the confusion matrices of models using the os176 features. We switched to using those for all remaining experiments.



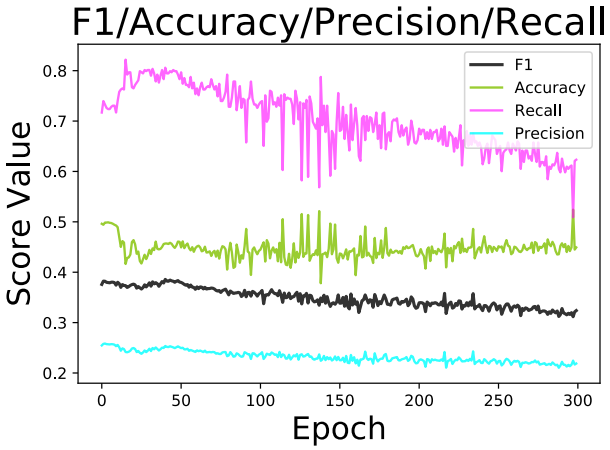
(a) Both upsampled



(b) Both upsampled



(c) Only test upsampled



(d) Only test upsampled

	TP	TN	FP	FN	ACC	PRE	REC	F1	Actual TP/TN
Both upsampled	1105	584	1047	526	0.518	0.513	0.677	0.584	1631/1631
Test upsampled	273	656	975	165	0.449	0.219	0.623	0.324	437/1631

Figure 85: A comparison of upsampling both train and test sets vs. just the test set

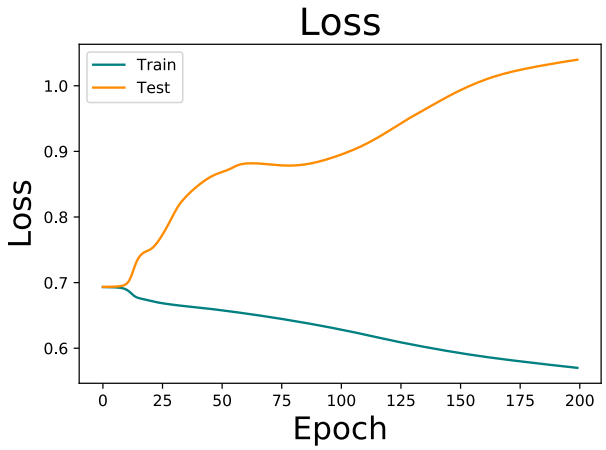
After deciding to continue training models with the os176 feature set, we ran a range of experiments with different combinations of learning rate and hidden nodes as shown in Figure 86. All experiments used the Adam optimizer. These showed better results with smaller learning rates. Nodes between 200-250 appeared to perform the best, although the same trends as seen in the node size experiment were present. More nodes led to higher true and false positives.

We then decided to test another hypothesis about the features. All previous experiments has been using 3 second audio clips that overlapped by 75%.

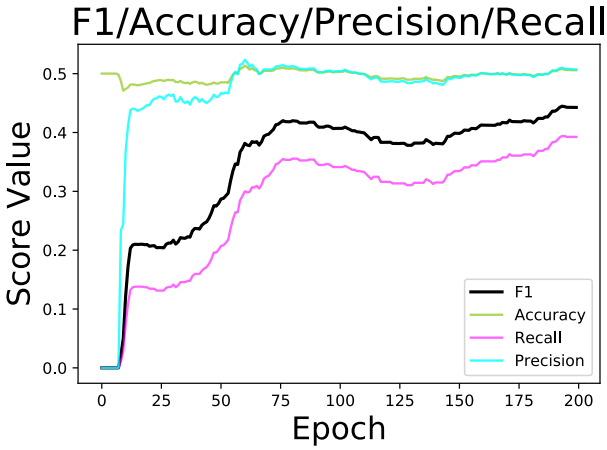
LR	Nodes	TP	TN	FP	FN	ACC	PRE	REC	F1
1e-06	100	1190	639	992	441	0.561	0.545	0.73	0.624
1e-06	250	1447	318	1313	184	0.541	0.524	0.887	0.659
1e-06	250	1462	375	1256	169	0.563	0.538	0.896	0.672
1e-06	400	1421	309	1322	210	0.531	0.518	0.871	0.65
1e-05	200	1001	575	1056	630	0.483	0.487	0.614	0.543
1e-06	400	1019	791	840	612	0.555	0.548	0.625	0.584
1e-06	200	1378	465	1166	253	0.565	0.542	0.845	0.66
1e-07	200	1149	704	927	482	0.568	0.553	0.704	0.62

Figure 86: Hyperparameters and metrics for models trained on os176 features using Adam. Actual TP/TN is 1631/1631

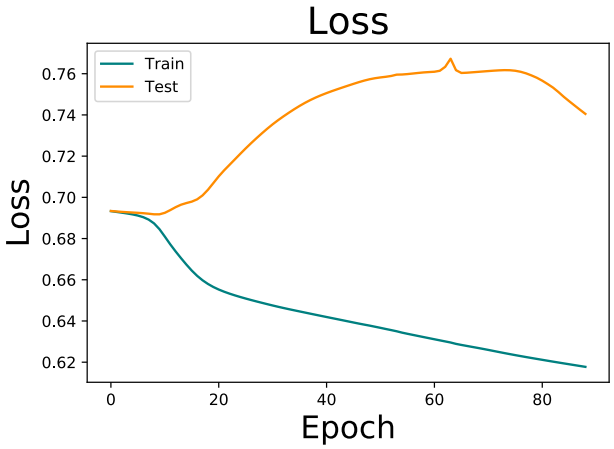
Our hypothesis was that using the 3 second clips with no overlap could combat the overfitting that had been pervasive throughout our models. Unfortunately, the metrics showed our models performing slightly worse, averaging lower accuracy and F1. Both of the models in Figure 87 used 250 nodes and 4 layers with a learning rate of 5e-7. One thing to note in the model trained with the 0 overlap features: Although F1 and recall trend upward and have not settled, they are still far below that of the model trained on overlapping features and probably would not have reached such high values regardless of extra training.



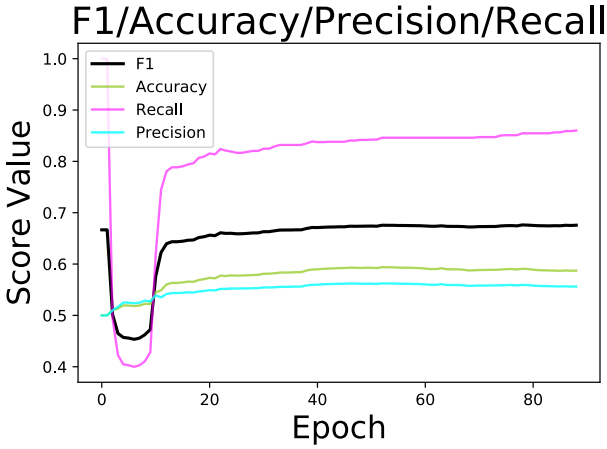
(a) No overlap



(b) No overlap



(c) 0.75 overlap



(d) 0.75 overlap

	TP	TN	FP	FN	ACC	PRE	REC	F1
No overlap	765.0	1206.0	743.0	1184.0	0.506	0.507	0.393	0.443
0.75 overlap	1403.0	512.0	1119.0	228.0	0.587	0.556	0.86	0.676

(e)

Figure 87: A comparison of using 3 second Sub-Clips with no overlap and 75% overlap between clips

In order to generate results comparable with other studies, we switched over to using the official data splits from DAIC-WOZ. Because we were not using a three-way split, the 'validation' set was not used. This resulted in a training set around 60% instead of the previous 70%. All the data in the 'validation' set was not used, reducing the overall total amount of data. Random resampling was still used to upsample both the train and test sets.

At this point, we had trained most recent models with around 200 hidden nodes and a learning rate around 1e-6, and between 1 and 4 layers. Given our past experimentation, these appeared to be close to the optimal hyperparameters. We trained models with these settings on the previously defined split and got a mix of complete overfitting (all examples classified in majority class) and other odd results that did not show learning. Given that the size of the data set was significantly smaller, we hypothesized that the hyperparamters and other settings were no longer optimal.

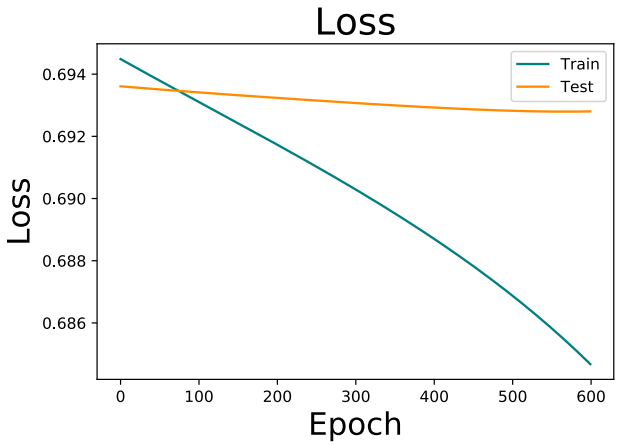
In response, we decided to test models with a smaller node count and learning rate. With these hyperparameters, using the Adam optimizer tended to show less learning, and could even fail to predict anything in the target class. We had previously noticed that RMSProp performance would slightly increase with lower node counts and learning rates, so we switched to RMSProp for these experiments. Two experiments were run with 75 hidden nodes, 1 layer and a learning rate of 1e-7, with dropout rates of 0 and 0.5. Both models showed diverging train and test losses. Although F1 was trending upwards, precision was constant and only recall was increasing. This is indicative of overfitting as well, as a higher recall can mean more false positives. The same model with 75 hidden nodes and 1 layer was run with a learning rate of 1e-8 and showed very promising metrics. This was one of the first experiments in which we were able to see both train and test loss decrease. It is not quite optimal, however, as the loss curves appear relatively linear with different slopes. Although test loss decreases, it will never converge with the train loss as the difference increases over time. Because train and test accuracy, as well as both precision and recall, increased over time and had not yet stabilized, we continued to train the model. Originally being trained for only 200 epochs, we trained for an additional 400 epochs. This model was able to achieve an accuracy of 0.51 with an F1 score of 0.56, recall of 0.63, and precision of 0.51.

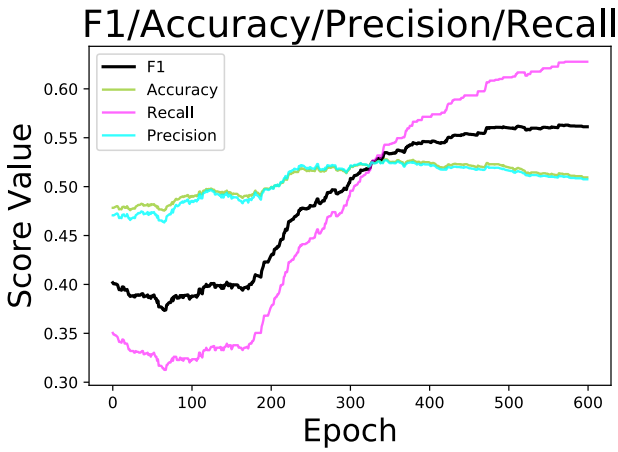
The best sequence models come from a few different areas of experimentation. Our best model trained on the DAIC-WOZ split, although lacking in numbers compared to some others below, is the best that shows test set loss decreasing over time. Although all others show a divergence of train and test loss over time, the models themselves are fit quite well to the data and can provide classification significantly better than random. All following models (excluding baselines) were trained on the os176 openSMILE feature set of the three second 75% overlap clips of DAIC-WOZ audio, using random resampling to up-sample both train and test sets as described above. In the "Split" column of Figure 88, "Internal" refers to the sequential 70/30 split and " DAIC" refers to the official DAIC-WOZ split, both as described earlier in this section.

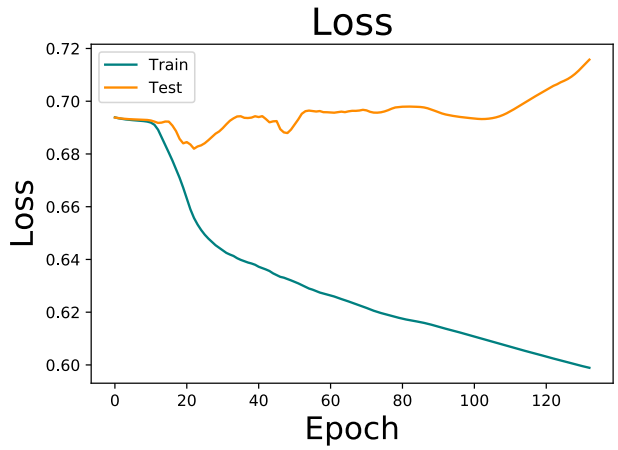
5.3 Topological Data Analysis

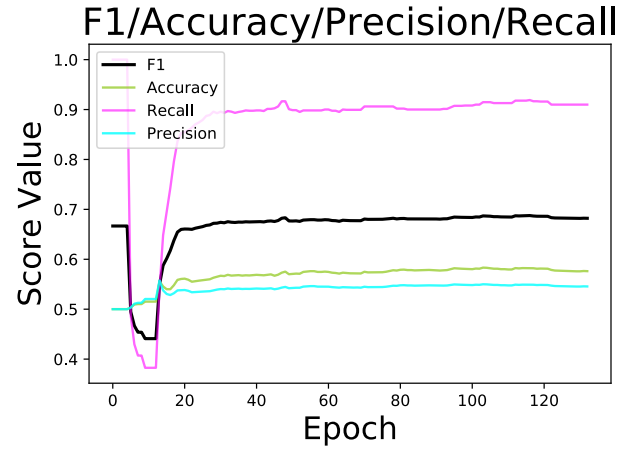
Data Set	Model	DR	Features	F1	Accuracy	AUC
ULBC, First 5	XG3	Chi^2	100	0.59	0.60	0.59
SLBC, First 5	NB	PCA	13	0.62	0.54	0.55
ULBC, Random 5	XG2	PCA	10	0.59	0.57	0.55
SLBC, Random 5	NB	PCA	10	0.60	0.56	0.54
ULBC, First Clip	XG4	Chi^2	100	0.55	0.55	0.55
SLBC, First Clip	NB	kPCA	11	0.55	0.53	0.54
OS176 + ULBC First 5	XG3	Chi^2	80	0.70	0.69	0.72
OS176 + SLBC First 5	XG3	Chi^2	90	0.71	0.71	0.75
OS176	XG4	Chi^2	90	0.71	0.70	0.75

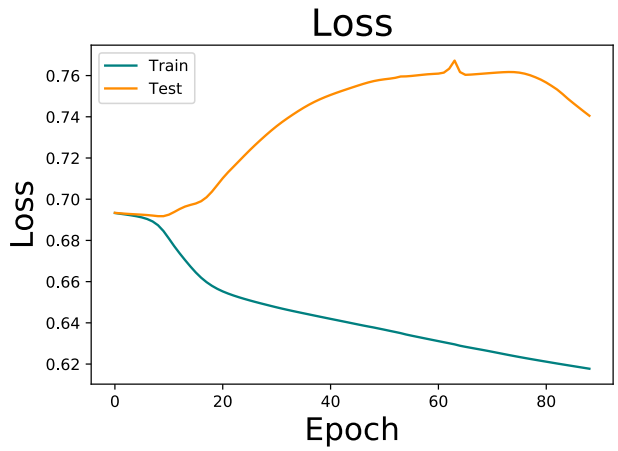
Table 7: Results From Best Experiments. A PHQ-8 Score cutoff of 10 was used for all tests. ULBC stands for Upper Level Betti Curve, SLBC stands for Lower Level Betti Curve, OS Stands for openSMILE, and DR stands for Dimensionality Reduction.

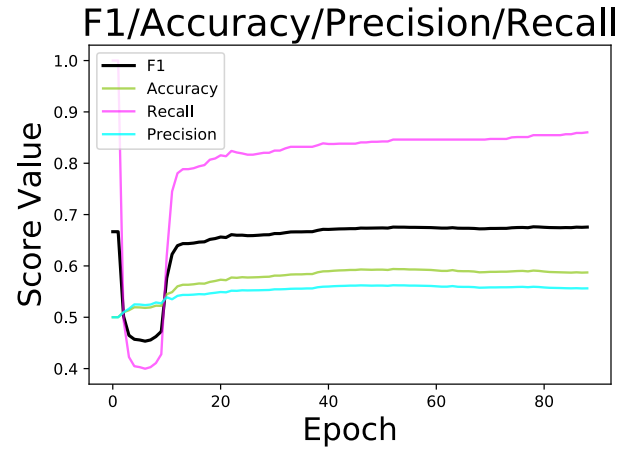

Loss
Train
Test
Loss
Epoch
0.694
0.692
0.690
0.688
0.686
0
100
200
300
400
500
600

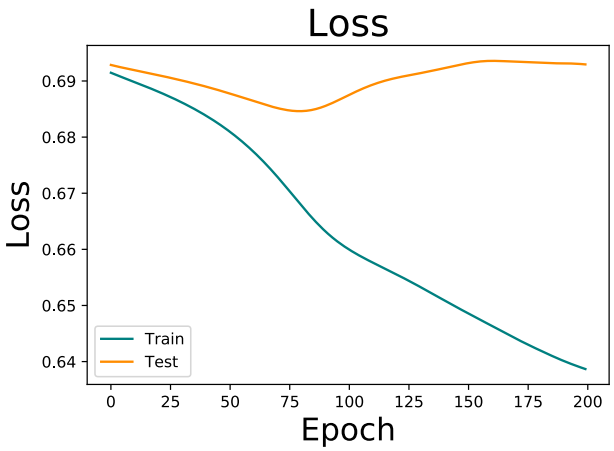

F1/Accuracy/Precision/Recall
F1
Accuracy
Recall
Precision
Score Value
Epoch
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0
100
200
300
400
500
600

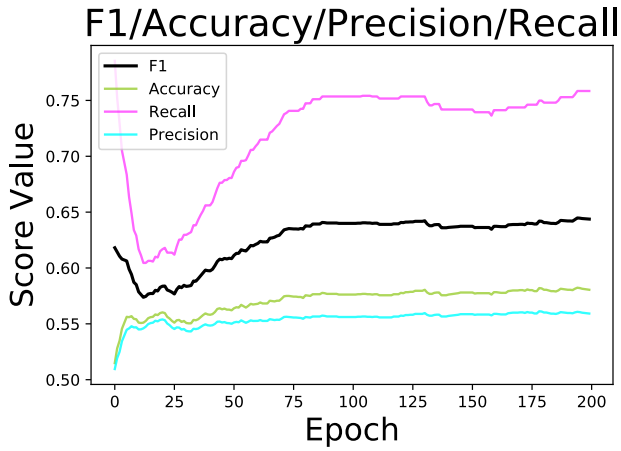

Loss
Train
Test
Loss
Epoch
0.72
0.70
0.68
0.66
0.64
0.62
0.60
0
20
40
60
80
100
120


F1/Accuracy/Precision/Recall
F1
Accuracy
Recall
Precision
Score Value
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
20
40
60
80
100
120


Loss
Train
Test
Loss
Epoch
0.76
0.74
0.72
0.70
0.68
0.66
0.64
0.62
0
20
40
60
80


F1/Accuracy/Precision/Recall
F1
Accuracy
Recall
Precision
Score Value
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
20
40
60
80





	F1	PRE	REC	ACC	Split	Optimizer	LR	Nodes	Layers	Dropout
Our Approaches	0.561	0.507	0.628	0.509	DAIC	RMSProp	1e-8	75	1	0
	0.682	0.546	**0.910**	0.576	Internal	Adam	1e-6	100	6	0.8
	0.644	**0.559**	0.758	0.580	Internal	RMSProp	1e-7	150	1	0
	0.676	0.556	0.860	**0.587**	Internal	Adam	5e-7	250	4	0
DAIC Baseline	0.460	0.320	0.860							
Ma et al. (2016)	0.520	0.350	**1.000**							
Al Hanai et al. (2018)	**0.630**	**0.710**	0.560							

Figure 88: Our best LSTM-based sequence models over the course of this project, compared to baselines

For our data sets, we only used 5 second clips with no overlap. Across the board, we found the best results with Betti curves when using Naive Bayes or XGBoost for models. These models consistently performed better than random for the majority of data. Our best results for Betti curves on their own were found when using the first 5 clips from each participant, versus random 5 or the first clip from each question. When using the first clip from each question, our best results actually may be outliers, with most results lying close to random or worse. This may be due to the fact that the first couple clips from each participants are generally answering very similar questions from the interviewer, which may prompt similar responses across participants. Sub level Betti curves provided better results than upper level Betti curves in some scenarios, usually with a Naive Bayes model. When combined with the openSMILE 176 feature set, much better results were found with both upper level and sub level Betti curves. It is somewhat doubtful that the Betti curve features are actually being used in these results however, as PCA was applied and the openSMILE features on their own performed extremely similarly.

5.4 Generative Adversarial Networks

5.4.1 ACE Experiments

These experiments were run after Texygen was first set up on the ACE cluster. These are the following parameters for these experiments:

Parameter	Value
Total Epochs	50
Pre-Training Epochs	20
Adversarial Training Epochs	30
Input Texts	1074

Though RankGAN had a higher average BLEU score, all of the BLEU scores from this experiment were exceptionally low. We hypothesise that this was due to the inconsistent nature of text data, so the GANs were not able to find consistent patterns from the texts to recreate. The number of epochs was also reduced, from the 80 pre-training and 100 adversarial training epochs, used in previous studies, to only 20 pre-training and 30 adversarial training epochs, due to the limitations of the ACE cluster.

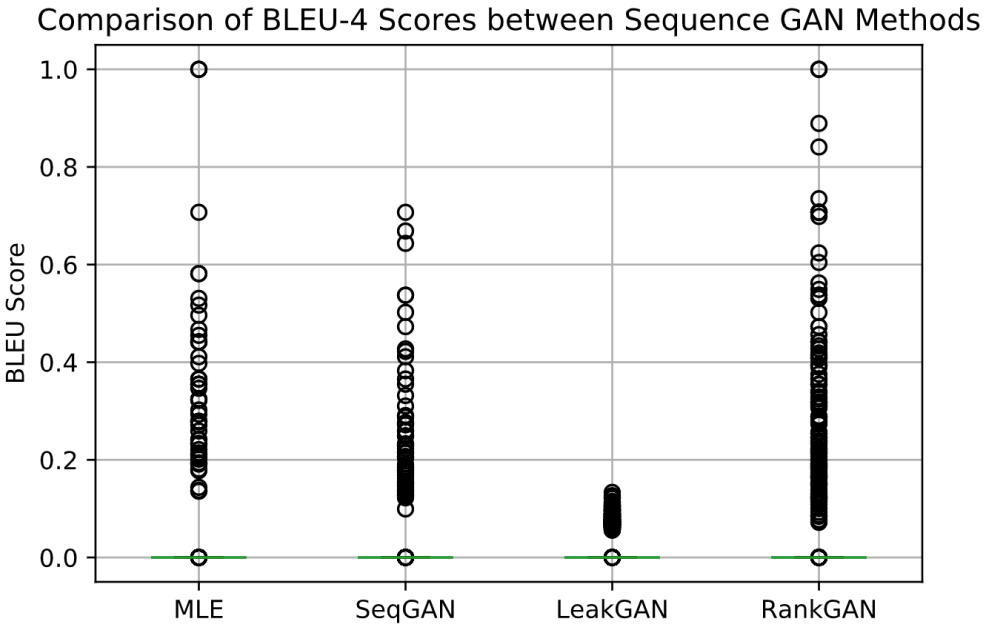


Figure 89: Comparison of BLEU-4 Scores: ACE Experiments

GAN Method	Average BLEU-4 Score
MLE	0.001950947597504571
SeqGAN	0.0015532267864985627
RankGAN	0.0043830571838256085
LeakGAN	0.002125034325184055

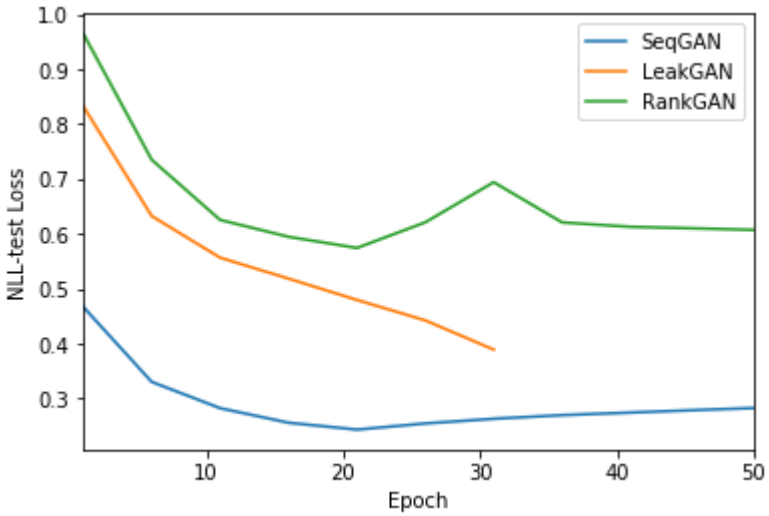


Figure 90: NLL-test Loss Over 50 Epochs

The NLL-test loss in Figure 90 shows that it did not converge, nor did it go notably low for LeakGAN or RankGAN. Once again, this was expected, due to the low number of epochs run from this current experiment. Despite the very low results compared to the studies that implemented these GAN methods, we can consider this phase of experiments a success, due to showing that these GAN methods can successfully be run on our SMS message data.

5.4.2 Turing Experiments

The second round of experiments performed on the Turing cluster was more to prove the functionality of the improved framework than to test the viability of individual text generation methods. The following parameters were used for the experiments on Turing:

Parameter	Value
Total Epochs	60
Pre-Training Epochs	30
Adversarial Training Epochs	30
Input Texts	2000

Each of these SMS texts was from a single participant, and no word frequency filtering or filtering by length was done. A distribution of BLEU

scores among Sequence GAN methods run on Turing can be seen in Figure 92:

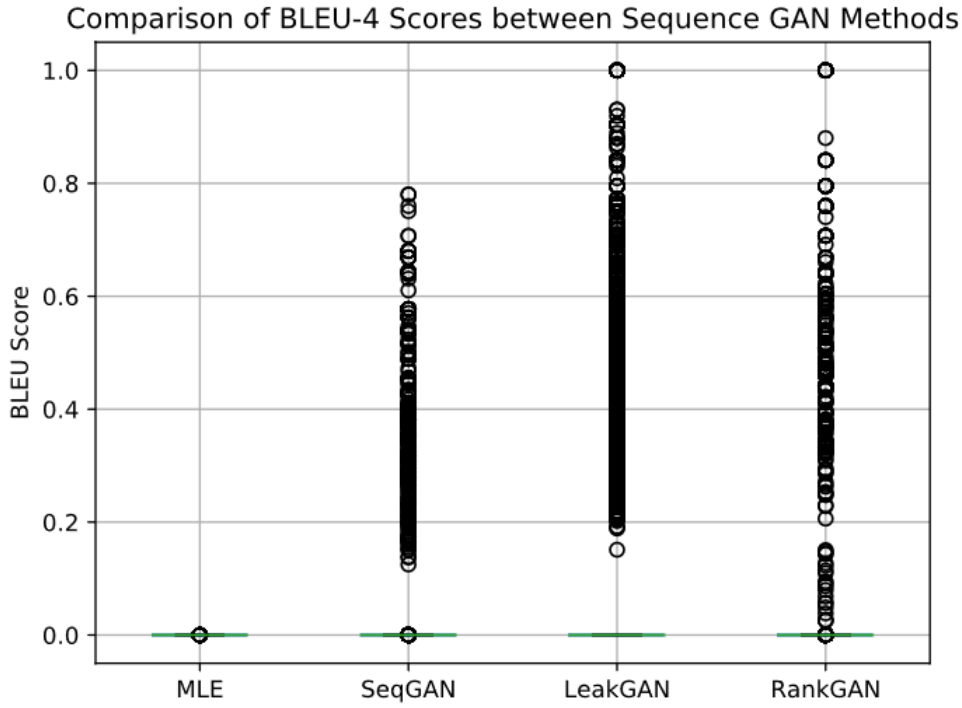


Figure 91: Comparison of BLEU-4 Scores: Turing Experiments

GAN Method	Average BLEU-4 Score
MLE	4.126752767215812e-156
SeqGAN	0.010481158991399186
RankGAN	0.012939276614873981
LeakGAN	0.09086621413976959

These experiments all finished within twelve hours, despite having more epochs, due to the increased resources of the Turing cluster. As a result of the increased number of epochs, methods had average BLEU-4 scores a full exponent greater than the previous experiments. LeakGAN's average BLEU score are considerably worse, due to LeakGAN having a bug and crashing after fifteen epochs. Unusually, MLE performed considerably worse than the previous tests, presumably due to running only the thirty pre-training epochs

instead of the full sixty.

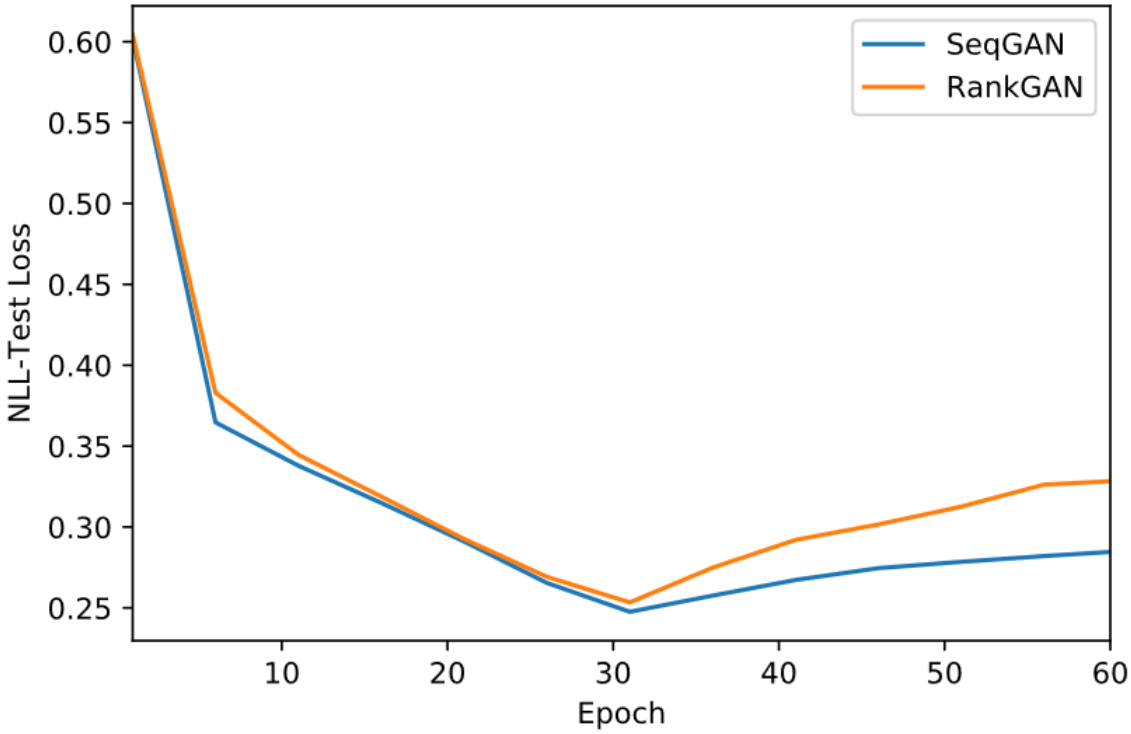


Figure 92: Comparison of NLL-Test Scores: Turing Experiments

Figure 92 shows the NLL-test loss scores over all sixty epochs of this round of experiments. MLE and LeakGAN NLL-test loss scores are not displayed here due to output issues with both methods. As seen in the previous round of experiments, NLL-test loss fell rapidly during pre-training, and slowly rose back up

6 Conclusion

Overall, we have been able to reliably demonstrate the ability of our implemented machine learning and deep learning methods to detect depression

and anxiety using voice audio and accompanying transcripts. This technology could open new opportunities for how patients of depression or anxiety are treated. It could also help many who lack access to mental health resources or who are undiagnosed find the help they need. Reliable detection of depression can also be used in sentiment analysis. The ethical concerns of this are significant and the technology can easily be abused. All necessary precautions should be taken to ensure that private or government use of this type of depression detection is monitored and ethical in order to avoid another study like (Kramer et al., 2014), in which almost 700,000 Facebook users were emotionally manipulated without consent.

In Machine Learning scope of research, we have seen that SCB has improved our prediction results. Using regression hasn't altered our results significantly. We were also able to perform slightly better when detecting anxiety through GAD-7 labels. In addition, tree based ensemble methods like XGB and ADA have performed better overall. Future improvements can be made by adding feature selection methods like PCA, Chi^2, etc. SVMs could also be used with Gaussian kernel, polynomial kernel, and Sigmoid kernels to improve predictions (Valstar et al., 2016). Using CNN for detecting depression has proven to be effective as we were able to achieve an F1 score close to DepAudioNet's. Some of the models we tested on random folds were able to perform better. Another step to continue improving this model would be adjusting our input to be grouped by participants instead of audio clips. Using LSTM-based sequence models has also proven effective, reaching an F1 score of as high as 0.68. Different data processing can be implemented to improve results further, but training a sequence model with both audio and the respective transcript is the most promising path forward. A larger data set would also be impactful. Using Betti Curves with topological data analysis has been shown to provide better than random results. Sequence GAN methods have been fully set up to experiment with artificial participant generation in future studies.

6.1 Machine Learning

We started out this research by experimenting with Machine Learning methods on DAIC-WOZ data set. Initial phase of experimentation didn't include grid search, up-sampling, and ADA method. In addition, we were

using only Praat features due to computational restrictions - research experiments were mainly conducted locally at this point. No GAD-7 based experiments were also done during this phase. SCB and Regression weren't integrated into our Machine Learning pipeline as well. As a result, our experiments weren't fruitful and we obtained a highest F1 score of 0.63 and scores ranged between 0.48 - 0.63.

We were able to move our experiments to ACE cluster. This enabled us to compute more experiments and do computationally expensive tests. We started using openSMILE features, added regression approach to all methods we had been using, started sub-clip boosting, expanded dataset. These led to better overall results. Specifically, up-sampling, SCB, openSMILE features all improved our model's performance significantly. We also found our overall best results when using EMU dataset with GAD-7 labels: highest result was a test run with an XGB method using a cutoff of 5 which resulted in a F1 score of 0.828. We also outperformed previous MQP's EMU audio feature test results; they produced a highest F1 score of 0.7, while we were able to reach up to a F1 score of 0.813. Our pipeline didn't perform well with DAIC-WOZ when using official train and test split. The highest F1 score for this section of experiments was 0.513. This could be to not using other feature selection techniques: we used only one type of feature selection approach outlined in section 4.1.2. In addition, we also weren't able to use SVM unless we set the kernel to linear. Other related works have generated higher results using SVMs with other kernels (Valstar et al., 2016).

6.2 Deep Learning

Since we already have the EMU and Moodable data sets available from previous MQP, we used it to implement our first deep learning pipeline while we figured out the data-preprocessing for DAIC-WOZ data set. Running first on EMU and Moodable data sets was not able to give us good results but it gave us a foundation of knowledge in deep learning aspect of detecting depression. When all the data were prepared ready for DAIC-WOZ data set, we started our process of finding an efficient model which will help predict depression.

Neural Networks We started off testing with a base model and changed one step at a time to see any noticeable difference. Through testing different layers, kernel sizes, combination of feature sets, learning rates and decays, our final model was able to achieve a F1 score of above 0.6678 with accuracy of 0.6664 on random folds. Then, we tested this model with official folds to compare against DepAudioNet (Ma et al., 2016). With Official folds, our model barely performs as good as DepAudioNet since our model achieved a F1 score of 0.5506 whereas the best result of DepAudioNet is 0.52. Testing with random folds and testing with official folds give us a significantly different F1 scores while the former performs better than the latter. Our conclusion on this difference of F1 scores is that we were using the splits which were grouped by audio clips instead of participants although a single participant was not cross-included in both of train and test set. The official splits were grouped by participants and when we re-grouped them into audio clips, we have different ratio splits.

Since the features extracted from audio play a huge part in finding an efficient model, combining convoluted feature sets with other feature sets such as PRAAT or Betti curves could produce potential results. Another approach to improve our model would be using multimodal approach. In this approach, not only audio data but also other available data such as text messages and transcripts should be considered for feeding into each subnetwork (which could a be CNN or LSTM).

Sequence Model Using LSTM-based neural networks has proved an effective method for categorical depression prediction using voice samples alone. Our best model was able to show an F1 of 0.68, precision of 0.55, recall of 0.91, and accuracy of 0.58. As compared to the best baseline (Al Hanai et al., 2018), we have +0.05 F1, -0.16 precision, and +0.35 recall. However, there is still room for improvement. All of our models suffered from overfitting. This is most likely because of the size of the data set. Sequence models typically perform better when they are fed a large and diverse data set, and DAIC-WOZ may simply not be expansive enough. Other studies (Al Hanai et al., 2018) have addressed this issue by training two parallel sequence models, one trained on audio data and the other on the transcripts of the same audio. Combining their output has shown to significantly increase accuracy and F1 score.

Using larger data set opens up some more possibilities for data preparation. At one point, we removed all inputs from the LSTM training data that had a size of 1, i.e. removed all inputs where the question response was only one clip long. This reduced the data set by too much and exasperated the issues we experienced that were caused by small data set size. This technique could be applied to a far larger data set for more accurate results.

Additionally, making the model bidirectional allows it to consider relevant 'memory' generated from both past and future events in the sequence. This additional context could reduce the number of false positives. The benefits of this will be compounded when using a larger data set. Another interesting possibility is using raw spectrogram data, instead of features, as input for the LSTM model. Slices of a Sub-Clip's spectrogram can be treated sequentially.

Compared to other attempts to use sequence modelling with DAIC-WOZ audio, our models tend to perform as well, if not better in many cases (Al Hanai et al., 2018). We believe that combining our current model with a model trained on transcript data would significantly improve performance, possibly exceeding an F1 score of 0.8. As shown in Al Hanai et al. (2018), their multimodal model demonstrates an increase of +0.14 in F1 and +0.27 in recall over their model trained on just audio.

Another technique to implement could be the SMOTE algorithm. Random resampling had been used to upsample the minority class for many of the experiments. But it can still promote overfitting with it's inability to produce new data. Random resampling could still be used to generate more instances of the depressed class, but inside each resampled input sequence, SMOTE can be applied to replace the sequences of features with new generated arrays. Synthetically expanding the amount of unique data may lead to improvements on many of the issues outlined with our sequence modelling.

6.3 Topological Data Analysis

Our main goal with using Topological Data Analysis was to determine if Betti curves generated from sound waves are viable in machine learning. Betti curves, as stated in Section 2.10.5, are a general method for interpreting persistence barcodes as data. However, our construction of a filtered complex was unique (Section 4.5) and was only previously used in a study which used

Betti curve to analyze heart beats to categorize heart arrhythmias (Dindin et al., 2019). From our results, we can see that Betti curves were indeed able to provide useful information, such that our models produced better than random results. When combined with a known good form of feature extraction, openSMILE, even better results were found. However, it seems that the openSMILE features may be responsible for the majority of these good results. Regardless, our goal was to determine if Betti curves were viable at all as a form of feature extraction on sound waves, and we conclude that they are.

Several things can easily be done for future work on this topic. For one, varying clip lengths and looking at overlapping clips would be a good idea, as this could determine what length of clips work best with Betti curves. Overlap on waves was used in a previous study using Betti curves as well (Dindin et al., 2019), so better results may be found using this method. Using a larger data set may provide better results as well, as we used a truncated data set in all of our testing due to the limitations of our cluster computing. Using Betti curves as a feature in deep learning would be an interesting avenue to approach as well, as previous studies used this approach (Dindin et al., 2019).

6.4 Generative Adversarial Networks

The goal of this branch of the project was more to develop an environment for future studies than to show the feasibility of any particular text generation method. One clear conclusion that can be drawn, however, is that the data set we have been using for testing text generation is very inconsistent and difficult to train a sequence GAN with. Other studies that used sequence GAN methods filtered their data sets to texts that are all around the same length (Nie et al., 2019). This allows the GAN to learn a smaller number of extra features and generate more consistent texts. As a result, we suspect that text data sets of a more consistent length, such as Twitter messages, would work considerably better for generating consistent new data with sequence GANs.

Despite our best efforts, there were some definite limitations of our experiments with sequence GANs. LeakGAN has been problematic since the very beginning, and despite multiple attempts to debug it, we could not get it to run more than 15 epochs, so we do not have complete data on it.

The framework we have built for testing sequence GAN methods will be useful for future studies to test the feasibility of artificial participant generation. This can be proven by selecting SMS messages from real participants with similar PHQ-8 scores, and generating an artificial participant with their data. If the deep learning or machine learning method can correctly classify the new participant, then the feasibility of artificial participant generation will have been proven. Furthermore, it would be highly beneficial for future studies to standardize the output files from each GAN method in Texygen, to allow for much easier comparing of GAN methods. The output files used for this round of experiments still required manual reformatting to allow them to be plotted against each other.

A Tables of Accomplishments

A.1 A Term

Name	Accomplishments
Nick	Team Leader Set up communication logistics Downloaded/Organized DAIC-WOZ Wrote initial DAIC-WOZ audio processing code Depression and machine learning lit review Wrote report outline Major report editor
Adam	Team Scribe Literature review on DAIC-WOZ DAIC-WOZ Audio Processing: Slicing into questions and X-second segments, organization DAIC-WOZ Audio Processing Statistics Mental Health + Patient Health Questionnaire Literature Review Set up LaTex Document and Bibtex Citations Minor editing in report Wrote Sections on DAIC-WOZ, Patient Health Questionnaire, Depression Statistics, and Data Processing
Myo	Deep Learning specifically CNN, and RNN Set up clusters and Python environments for team members Researched suitable Frameworks to build Neural Networks Built and experiment CNN models using Pytorch and Python

Name	Accomplishments
Joe	Set up Git repository Researched different types of neural networks to determine best fit for our task Researched different ways to format audio data to feed into a neural net Researched how to evaluate deep learning models Learned how to Jupyter and research cluster Started to experiment with CNN model, testing different hyperparamters with EMU and DAIC Learned LaTeX, Python, and PyTorch Lent lots of assistance to group members with Git issues
Yosias	Made the following modifications to analyzer script with Yared: including regression, fixing train/test method, generating predictions, worked on script to make it run through different parameters, worked on getting it to run on the cluster. Did audio slicing for 3-7 seconds on local machine and for overlap. Did Praat and openSMILE extraction for 3-7 seconds DIAC-WOZ on local machine Worked with Yared to modify openSMILE to make it work on cluster. Worked with Joe to make Praat work on cluster.
Yared	DAIC-WOZ slices organization Appended PHQ-8 Scores Literature Review on Data Balancing and Evaluation Models Researched Metrics (Confusion Matrix, F1, precision, recall) Further processed audio slices for pratt extraction compatibility Run initial experimental tests in Machine Learning models Analysis of test results Wrote XGBoost, Random Forest and Decision Tree and Artificial Neural Networks Wrote Data Balancing Section Wrote Evaluation Metrics

A.2 B Term

Name	Accomplishments
Nick	Sequence GAN sub-project Set up Texygen Debugged Texygen Processed Moodable text data Ran SeqGAN experiments Wrote code to process experiment results Major editor for the report Fixed bibliography errors
Adam	Developed code for generating topological data from sound waves Created data set of DAIC-WOZ 5 second Betti curves Performed in-depth research on TDA Research on PCA and NMF Wrote sections on dimensionality reduction techniques Wrote TDA background section Wrote TDA implementation section Wrote TDA Analysis Helped debug LaTex code Added new DAIC-WOZ boxplots
Myo	Deep Learning specifically CNN, and RNN Set up clusters and Python environments for team members Researched suitable Frameworks to build Neural Networks Built and experiment CNN models using Pytorch and Python Reduced Training time by image caching Added Up-sampling on DAIC-WOZ data Implemented experiments regarding to kernel sizes and layers

Name	Accomplishments
Joe	Created and implemented an LSTM model in PyTorch and experimented heavily on data sources and formats, hyperparameters, internal settings, sampling techniques Attempted an LSTM model with spectrogram input Implemented LSTM model in Keras based on paper, experimented with our data (mostly learning experience, recorded on weekly slides but no results in paper) Continued experimentation on CNN with DAIC, including learning rate decay Researched ways to combine audio and text inputs Initiated and followed through on managing use and access of large data sets in Git and on the research cluster Created starter documents for DL and ML testing workflows and Git usage Wrote script framework for managing LSTM experiment submission and logging Started an experiments log with a suite of robust performance analysis tools Modified Praat feature generation script, wrote wrappers for feature extraction job management Assisted with Praat section in instruction manual Lent lots of assistance to group members with Git issues
Yosias	Made the following modifications to analayzer script with Yared: including regression, fixing train/test method, generating predictions, worked on script to make it run through different parameters, worked on getting it to run on the cluster. Did audio slicing for 3-7 seconds on local machine and for overlap. Did Praat and openSMILE extraction for 3-7 seconds DIAC-WOZ on local machine Worked with Yared to modify openSMILE to make it work on cluster. Worked with Joe to make Praat work on cluster.
Yared	Organize Test/Train sets and set Included Regression Models and added model parameter options Added prediction outputs for sub-clips Ran experiments for Classification models over multiple variations Start setting up Machine Learning pipeline on cluster Wrote Machine learning results and methods (with Yosias) Worked with Ermal for incorporating Machine Learning process on cluster openSMILE feature generation for all sub-clip variations on EMU Set up instruction manual and organized the outline Was scribe for the term Upload all data to a workspace on Ace cluster Wrote script to determine number of features to use

A.3 C Term

Name	Accomplishments
Nick	Sequence GAN Sub-Project Debugged Texygen Modified Texygen with parameter arguments Got RelGAN working Modified RelGAN to accept other data sets Made Texygen job file generator Made RelGAN job file generator Ran Texygen and RelGAN experiments Wrote code to process experiment results Re-organized report Edited every section in the report Wrote all Generative Adversarial Network sections
Adam	Rewrote and expanded all TDA sections In depth research into topology and persistent homology 216+ unique machine learning experiments with Betti curves Generated all truncated Betti curve data sets Generated combined openSMILE/Betti curve data sets Modified code for running batch machine learning experiments Wrote code for truncated and combined data set generation Debugged LaTeX Added design improvements to report
Myo	Deep Learning specifically CNN, and RNN Team Leader Built and experiment CNN models using Pytorch and Python Developed code for multiple deep learning models, and combining CNN with openSMILE Tested various hyperparameters for CNN Added Upsampling and Downsampling on DAIC-WOZ data

Name	Accomplishments
Joe	Continued expansive experimentation on main LSTM model Brief experimentation on Keras LSTM model for binary classification Heavily expanded on logging and performance analysis/visualization tools to include more automated data collection and higher quality/more flexible visualizations Significant time and effort debugging PyTorch, issues on Ace cluster Squashed bugs in the Praat feature generation pipeline Focused lots of work on the final report Researched adding CNN layers for feature extraction on top of LSTM Lent lots of assistance to group members with Git issues Served as scribe for the term
Yosias	Worked on the Transcript component of DAIC-WOZ Did pre-processing work on transcripts and set up scripts for it Worked on modifying text extraction code to function for DAIC-WOZ transcripts Extracted Textblob and Empath features Worked on doing tests for transcript and audio features combined worked on setting up pipeline for transcript +audio
Yared	Continue and finalize regression experiments Finalize EMU, Moodable and DAIC-WOZ dataset experiments Incorporate GAD into Machine Learning experiment scope Tested SCB with audio files Generate Official DAIC-WOZ folds Generate in-team DAIC WOZ fold Research GAD-7 anxiety background and related works Wrote Machine Learning Methods (section 4.1) Wrote Machine Learning Results (section 5.1.1, 5.1.2, 5.1.3 and, 5.1.4) GAD-7 vs PHQ-8 analysis SCB impact analysis Code debugging and assisting teammates in using Machine Learning pipeline

B Table of Authorship

Section	Primary Author	Primary Editor
Abstract	Joe, Nicolas	Joe
1: Introduction	Nicolas, Adam	Joe
1.1: Related Works	All	Joe
1.2 Our Approach	All	Joe
2.1: Previous MQPs	Yosias	
2.2: Depression Statistics	Adam, Yared	
2.3: Data Sources	Adam, Yosias	
2.4: Dimensionality Reduction Techniques	Adam	
2.5: Data Balancing	Yared	
2.6: Evaluation Metrics	Yared	Joe, Adam
2.7: Machine Learning	Yared, Yosias	Yared
2.8: Deep Learning	Myo, Joe	Joe, Adam
2.9: Machine Learning and Deep Learning Frameworks	Myo	Joe
2.10: Topological Data Analysis	Adam	Nicolas
2.11: Generative Adversarial Networks	Nicolas	Adam
3.1: Text Feature Extraction for Machine Learning	Yosias	
3.2: Audio Feature Extraction for Machine Learning	Yosias	
3.3: Topological Data Analysis	Adam	Prof. Paffenroth
3.4: Generative Adversarial Networks	Nicolas	Adam
4.1: Machine Learning	Yared, Yosias	
4.2: Deep Learning	Myo, Joe	Joe
4.3: Tests Run on Audio Data	Yosias	
4.4: Tests Run on Transcript	Yosias	
4.5: Topological Data Analysis	Adam	Nicolas, Prof. Paffenroth
4.6: Generative Adversarial Networks	Nicolas	Adam
5.1: Machine Learning	Yared, Yosias	Yared
5.2: Deep Learning	Myo, Joe	Joe, Adam
5.3: Topological Data Analysis	Adam	Prof. Paffenroth
5.4: General Adversarial Networks	Nicolas	Adam
6: Conclusions	Joe	
6.1: Machine Learning	Yared, Yosias	
6.2: Deep Learning	Myo, Joe	Joe
6.3: Topological Data Analysis	Adam	Nicolas, Prof. Paffenroth
6.4: General Adversarial Networks	Nicolas	

References

Abu-Rmileh, A. (2019). The multiple faces of 'feature importance' in xgboost. https://towardsdatascience.com/be-careful-when-interpreting-your-features-importance-in-xgboost-6e16132588e7.

Ahire (2018). The artificial neural networks handbook: Part 1.

Al Hanai, T., Ghassemi, M., and Glass, J. (2018). Detecting depression with audio/text sequence modeling of interviews. In *Proc. Interspeech 2018*, pages 1716–1720.

Al Hanai, T., Ghassemi, M. M., and Glass, J. R. (2018). Detecting depression with audio/text sequence modeling of interviews. In *Interspeech*, pages 1716–1720.

Alghowinem, S., Goecke, R., Wagner, M., Epps, J., Breakspear, M., and Parker, G. (2013). Detecting depression: a comparison between spontaneous and read speech. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7547–7551. IEEE.

Ali, A. (2018). K-nearest neighbor with practical implementation. https://medium.com/machine-learning-researcher/k-nearest-neighbors-in-machine-learning-e794014abd2a.

Anderson, M. (2019). Mobile technology and home broadband 2019. *Pew Research Center: Internet, Science & Tech*.

Banerjee, S. (2018). An introduction to recurrent neural networks.

Batista, G., Prati, R., and Monard, M.-C. (2004). A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explorations*, 6:20–29.

Berry, E., Chen, Y.-C., Cisewski-Kehe, J., and Fasy, B. T. (2018). Functional summaries of persistence diagrams. *arXiv preprint arXiv:1804.01618*.

Bhumika Gupta, Akshay Jain, N. D. A. R. A. A. (2017). Analysis of various decision tree algorithms for classification in data mining.

Binieli, M. (2018). Machine learning: an introduction to mean squared error and regression lines. https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-c7dde9a26b93/.

Boissonnat, J.-D. and Maria, C. (2014). The simplex tree: An efficient data structure for general simplicial complexes. *Algorithmica*, 70(3):406–427.

Brems, M. (2019). A one-stop shop for principal component analysis. https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c.

Brid, R. S. (2018). Decision trees–a simple way to visualize a decision. https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb.

Brownlee, J. (2019). Understand the impact of learning rate on neural network performance. https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/.

Bubenik, P. (2017). Introduction to topological data analysis.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122.

Carlsson, G. (2009). Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308.

Chawla, Bowyer, H. K. (2002). Smote: Synthetic minority over-sampling technique. AI Access Foundation and Morgan Kaufmann Publishers.

Chepushtanova, S., Emerson, T., Hanson, E. M., Kirby, M., Motta, F. C., Neville, R., Peterson, C., Shipman, P. D., and Ziegelmeier, L. (2015). Persistence images: An alternative persistent homology representation. *CoRR*, abs/1507.06217.

Darji, H. (2019). An approach to solve the bird counting problem.

Demir, N. (2015). Ensemble methods: Elegant techniques to produce improved machine learning results. https://www.toptal.com/machine-learning/ensemble-methods-machine-learning.

Dham, S., Sharma, A., and Dhall, A. (2017). Depression scale recognition from audio, visual and text analysis. *CoRR*, abs/1709.05865.

Dindin, M., Umeda, Y., and Chazal, F. (2019). Topological Data Analysis for Arrhythmia Detection through Modular Neural Networks. 7 pages, 4 figures.

Dogrucu, A., Perucic, A., Isaro, A., and Ball, D. (2018). Sensing depression.

Drakos, G. (2018). How to select the right evaluation metric for machine learning models: Part 1 regression metrics. https://towardsdatascience.com/how-to-select-the-right-evaluation-metric-for-machine-learning-models-part-1-regrression-metrics-3606e25beae0.

Edelsbrunner, H. and Harer, J. (2008). Persistent homology-a survey. *Contemporary mathematics*, 453:257–282.

Fast, E., Chen, B., and Bernstein, M. S. (2016). Empath. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*.

Flach, P. A. (2003). The geometry of roc space: Understanding machine learning metrics through roc isometrics. https://www.aaai.org/Papers/ICML/2003/ICML03-028.pdf.

Fortuner, B. (2017). Logistic regression. https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html.

Furr, S. R., Westefeld, J. S., McConnell, G. N., and Jenkins, J. M. (2001). Suicide and depression among college students: A decade later. *Professional Psychology: Research and Practice*, 32(1):97–100.

Gandhi, R. (2018). Support vector machine — introduction to machine learning algorithms. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

Ghrist, R. (2008). Barcodes: the persistent topology of data. *Bulletin of the American Mathematical Society*, 45(1):61–75.

Gong, Y. and Poellabauer, C. (2018). How do deep convolutional neural networks learn from raw audio waveforms?

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc.

Guillaume Lemaˆıtre, Fernando Nogueira, C. K. A. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning.

Guntuku, S. C., Yaden, D. B., Kern, M. L., Ungar, L. H., and Eichstaedt, J. C. (2017). Detecting depression and mental illness on social media: an integrative review. *Current Opinion in Behavioral Sciences*, 18:43 – 49. Big data in the behavioural sciences.

Guo, J., Lu, S., Cai, H., Zhang, W., Yu, Y., and Wang, J. (2017). Long text generation via adversarial training with leaked information. *CoRR*, abs/1709.08624.

Harrison, O. (2018). Machine learning basics with the k-nearest neighbors algorithm. `https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761`.

Homans, Doosje, V. V. (2012). The validity of the phq-9 and the gad-7 for screening depressive and anxiety disorders in sick-listed workers. *Master thesis Clinical and Health Psychology*.

Hossin, M. and Sulaiman, M. (2015). A review on evaluation metrics for data classification evaluations. `https://pdfs.semanticscholar.org/6174/3124c2a4b4e550731ac39508c7d18e520979.pdf`.

Hunt, J. and Eisenberg, D. (2010). Mental health problems and help-seeking behavior among college students. *Journal of Adolescent Health*, 46(1):3 – 10.

Kramer, A. D. I., Guillory, J. E., and Hancock, J. T. (2014). Experimental evidence of massive-scale emotional contagion through social networks. *Proceedings of the National Academy of Sciences*, 111(24):8788–8790.

Kroenke, K., Spitzer, R. L., and Williams, J. B. (2001). The phq-9: validity of a brief depression severity measure. *Journal of general internal medicine*, 16(9):606–613. 11556941[pmid].

Kroenke, K., Strine, T. W., Spitzer, R. L., Williams, J. B., Berry, J. T., and Mokdad, A. H. (2009). The phq-8 as a measure of current depression in the general population. *Journal of Affective Disorders*, 114(1):163 – 173.

Le Cun, Y., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2016). Pruning filters for efficient convnets. *CoRR*, abs/1608.08710.

Librosa Development Team, L. (2019). librosa.decompose.decompose. `https://librosa.github.io/librosa/generated/librosa.decompose.decompose.html`.

Lin, K., Li, D., He, X., Zhang, Z., and Sun, M. (2017). Adversarial ranking for language generation. *CoRR*, abs/1705.11001.

Liu, J., Jeng, S., and Yang, Y. (2016). Applying topological persistence in convolutional neural network for music audio signals. *CoRR*, abs/1608.07373.

Loria, S. (2018). Textblob: Simplified text processing. `https://textblob.readthedocs.io/en/dev/index.html`.

Ludeña-Choez, J., Quispe-Soncco, R., and Gallardo-Antolín, A. (2017). Bird sound spectrogram decomposition through non-negative matrix factorization for the acoustic classification of bird species. *PLOS ONE*, 12(6):1–20.

Ma, X., Yang, H., Chen, Q., Huang, D., and Wang, Y. (2016). Depaudionet: An efficient deep model for audio based depression classification. In *Proceedings of the 6th International Workshop on Audio/Visual Emotion Challenge*, AVEC '16, pages 35–42, New York, NY, USA. ACM.

Mansar, Y. (2018). Audio classification : A convolutional neural network approach. https://medium.com/@CVxTz/audio-classification-a-convolutional-neural-network-approach-b0a4fce8f6c.

Marchese, A. (2017). Data analysis methods using persistence diagrams. *University of Tennessee*.

Marcus, M., Yasamy, M. T., Ommeren, M., Chisholm, D., and Saxena, S. (2012). Depression: A global public health concern. *World Health Organization Paper on Depression*, pages 6–8.

Martínez-Trinidad, J. H. A. C.-O. F. (2013). An empirical study of oversampling and undersampling for instance selection methods on imbalance datasets.

Meryll, D. (2018). From topological data analysis to deep learning: No pain no gain. `https://towardsdatascience.com/from-tda-to-dl-d06f234f51d`.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. (2016). Pruning convolutional neural networks for resource efficient transfer learning. *CoRR*, abs/1611.06440.

Monica Tlachac, E. R. (2020). Screening for depression with retrospectively harvested private versus public text. `https://ieeexplore.ieee.org/document/9049136/keywords`.

Narkhede, S. (2018). Understanding auc - roc curve. `https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5`.

Nguyen, M. (2018). Illustrated guide to lstm's and gru's: A step by step explanation. `https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21`.

Nie, W., Narodytska, N., and Patel, A. B. (2019). Relgan: Relational generative adversarial networks for text generation. In *ICLR*.

Olah, C. (2015). Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

Papineni, K., Roukos, S., Ward, T., and jing Zhu, W. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318.

Piccinelli, M. and Wilkinson, G. (2000). Gender differences in depression: Critical review. *British Journal of Psychiatry*, 177(6):486–492.

Piczak, K. J. (2015). Environmental sound classification with convolutional neural networks. In *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE.

Raschka, S. (2015). Principal component analysis. `http://sebastianraschka.com/Articles/2015_pca_in_3_steps.html`, journal=Dr. Sebastian Raschka.

Resnik, P., Armstrong, W., Claudino, L., Nguyen, T., Nguyen, V.-A., and Boyd-Graber, J. (2015). Beyond lda: Exploring supervised topic modeling for depression-related language in twitter. In *Proceedings of the 2nd Workshop on Computational Linguistics and Clinical Psychology: From Linguistic Signal to Clinical Reality*, pages 99–107, Denver, Colorado. Association for Computational Linguistics.

Resom, A., Assan, J., Flannery, M., Gao, Y., and Yuxin, W. (2019). Machine learning for mental health detection.

Ringeval, F., Schuller, B., Valstar, M., Gratch, J., Cowie, R., Scherer, S., Mozgai, S., Cummins, N., Schmitt, M., and Pantic, M. (2017). Avec 2017: Real-life depression, and affect recognition workshop and challenge. In *Proceedings of the 7th Annual Workshop on Audio/Visual Emotion Challenge*, AVEC '17, pages 3–9, New York, NY, USA. ACM.

Saha, S. (2018). A comprehensive guide to convolutional neural networks — the eli5 way. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

Sayantini (2019). Keras vs tensorflow vs pytorch : Comparison of the deep learning frameworks. https://www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch/.

Shiruru, K. (2016). An introduction to artificial neural network. volume 1, pages 27–30.

Spitzer, R. L., Kroenke, K., Williams, J. B., and Löwe, B. (2006). A brief measure for assessing generalized anxiety disorder: the gad-7. *Archives of internal medicine*, 166(10):1092–1097.

Toto, E., Foley, B. J., and Rundensteiner, E. A. (2019). Improving emotion detection with sub-clip boosting. In Brefeld, U., Curry, E., Daly, E., MacNamee, B., Marascu, A., Pinelli, F., Berlingerio, M., and Hurley, N., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 37–52, Cham. Springer International Publishing.

Tsugawa, S., Kikuchi, Y., Kishino, F., Nakajima, K., Itoh, Y., and Ohsaki, H. (2015). Recognizing depression from twitter activity. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pages 3187–3196. ACM.

Umeda, Y. (2017). Time series classification via topological data analysis. *Information and Media Technologies*, 12:228–239.

Valstar, M., Gratch, J., Schuller, B., Ringeval, F., Lalanne, D., Torres Torres, M., Scherer, S., Stratou, G., Cowie, R., and Pantic, M. (2016). Avec 2016: Depression, mood, and emotion recognition workshop and challenge. In *Proceedings of the 6th International Workshop on Audio/Visual Emotion Challenge*, AVEC '16, pages 3–10, New York, NY, USA. ACM.

van Veen, F. (2017). Neural network zoo prequel: Cells and layers. `https://www.asimovinstitute.org/author/fjodorvanveen/`.

Varghese, D. (2018). Comparative study on classic machine learning algorithms. `https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222`.

Vu, K. (2019). Tensorflow vs. pytorch vs. keras for nlp. https://dzone.com/articles/tensorflow-vs-pytorch-vs-keras-for-nlp.

WHO (2018). Depression fact sheet. *World Health Organization*.

Yang, L., Jiang, D., He, L., Pei, E., Oveneke, M. C., and Sahli, H. (2016). Decision tree based depression classification from audio video and language information. In *Proceedings of the 6th International Workshop on Audio/Visual Emotion Challenge*, AVEC '16, page 89–96, New York, NY, USA. Association for Computing Machinery.

Yang, L., Jiang, D., Xia, X., Pei, E., Oveneke, M. C., and Sahli, H. (2017). Multimodal measurement of depression using deep learning models. In

Proceedings of the 7th Annual Workshop on Audio/Visual Emotion Challenge, AVEC '17, pages 53–59, New York, NY, USA. ACM.

Yiu, T. (2018). Understanding random forest. https://towardsdatascience.com/understanding-random-forest-58381e0602d2.

Yu, L., Zhang, W., Wang, J., and Yu, Y. (2016). Seqgan: Sequence generative adversarial nets with policy gradient. *CoRR*, abs/1609.05473.

ZhiYong Lin, ZhiFeng Hao, X. Y. X. L. (2009). Several svm ensemble methods integrated with under-sampling for imbalanced data learning.

Zhu, Y., Lu, S., Zheng, L., Guo, J., Zhang, W., Wang, J., and Yu, Y. (2018). Texygen: A benchmarking platform for text generation models. *CoRR*, abs/1802.01886.