LingoPros

Final Report V2

May 9, 2018

Josh Shaffer, Luis Montes, Erik Strauss, Matt Quintana

Sponsors: Dr. Okim Kang & Dr. David O. Johnson

Mentor: Ana Paula Chaves Steinmacher

NORTHERN ARIZONA UNIVERSITY

Overview: This final report document will detail our entire project so that it can be easily understood and recreated for future developers. It will contain our exact process for developing the project, the requirements established with our client, and how we designed and implemented the software.

Table of Contents

Introduction **3**

Process Overview **4**

Requirements **5**

Architecture and Implementation **7**

- Website 7
  - Database 7
  - User Login 9
  - Server 9
- AuToBI Based Analyzer 11
  - Data Analysis 11
  - Training the Neural Net 13
  - Testing the Neural Network 14

Testing **14**

- Unit Testing 14
- Integration Testing 15
- Usability Testing 15

Project Timeline **16**

Future Work **17**

Conclusion **18**

Glossary **19**

Appendix A:
Development Environment and ToolChain **20**

- Website Setup 20
- AuToBI Setup 21

Introduction

Non-native English speakers often have a hard time articulating their speech not just in what they say, but how they say it. Proper emphasis on words or just general speech tone is something that native English speakers do not have to think about in their everyday speech as it is something that just comes naturally. For example, in a conversation between a non-native English speaker and a native speaker, the non-native speaker may not be aware of how, by not matching the tone or cadence of the native speaker, the conversation can begin to feel awkward and unnatural. Recognizing these differences in tone, pitch, stress and other prosodic features between native and non-native English speakers, is one of the key focuses of Dr. Okim Kang's research at the Applied Linguistics Speech Lab (ALSL) at NAU.

Currently, students working with Dr. Kang at the ALSL have to do all of the speech analysis by hand, listening to audio samples and recording what features they hear or recognize. This process can take hours on end even for a small 10 second audio clip, so as such, an automated process of speech analysis that is easily available to the students would drastically improve their research capabilities by saving them time and energy. Dr. Kang has already developed her own experimental speech analysis program that can recognize prosodic features, However, the model that the current program uses, a framework based on a distinct form of speech analysis posed by David Brazil called *discourse intonation* [1] is not recognized as the standard in speech analysis. A set of conventions called Tones and Break Indices (ToBI) is considered to be the standard for annotating speech features, and so the clients want to show that the her program performs just as well or better than other automatic analyzers such as AuToBI.

For our project, Dr. Kang wants us to develop a program that is able to analyze recordings of non-native English speakers and grade their English proficiency based on aspects like tone and emphasis, and secondly she want us to implement an easily accessible web portal for users to upload and analyze audio samples in one location. We have been brought on to develop a speech analysis program based on the ToBI model as well as implement the web portal that will house the Praat Scripts that will help speed up our clients work with analyzing speech.

We will develop a machine learning program to score the proficiency of speakers using the output of the AuToBI program as an input. The reason to use machine learning is so it can better calculate a more precise score that is consistent with human scoring. We will also be developing the web application to house Praat scripts which will help our client so that her and her students can access it from anywhere, assuming the client allows them to have full access to the site.

In this document, we will cover our exact plan of implementing the AuToBI prosodic labeler and how we created the website that runs the Praat scripts. We will talk about specific technologies and software that we will use to accomplish that as well as the architectural designs of our programs. By the end, there should be a clear understanding on how we accomplished our tasks and how the programs will run not only now but for future developers as well.

Process Overview

Initially, we had a pretty clear plan as to how we were going to advance. Since our project is essentially two projects, we needed to work on both parts concurrently. To do this, we decided to split the team in half: half of us would work on the website and server while the other half would work on the AuToBI toolchain. With that in mind, we were able to efficiently create both sides of the project.

Website

Luis and Erik were tasked with designing the website and server. To keep work consistent, Github was used as version control. Before any work on the site started, different technologies had to be chosen and agreed on. First, the site needed a database, and the ultimate choice for the database system was MongoDB. Second, the site needed to be hosted online somewhere for people to access from anywhere. Initially, the first goal was to have NAU ITS host the site, but we ran into major problems and roadblocks following that path. After scrapping the ITS idea, Digital Ocean was chosen as a host.

Once the technologies were figured out, actual development began. Development followed a schedule that was created to keep us on track and moving. It started with basic web page implementation and adding basic functionality, like uploading a file onto the site. The main roadblock encountered here was trying to put the client's MATLAB code on the server. A good portion of development was stuck trying to figure this part out, but eventually we just flowed around the problem and continued with other development tasks. The user login page was developed and finalized towards the end of our development cycle and we eventually got speech analysis working completely on the server.

AuToBI

Josh and Matt were tasked with developing the AuToBI toolchain. Github was also used for collaboration and version control. Again, research was done before any real implementation began we read about ToBI and AuToBI . Then downloaded AuToBI and began using it to see how it worked and what kind of format the output file was. Next, we needed a machine learning API to select features from the AuToBI output and create/train a neural network in order to calculate the speakers proficiency. A Java API named Weka was what was ultimately chosen.

The core of this program is to use a third-party software named AuToBI, which is an automatic speech analyzer that uses the tones and break indies model (ToBI). AuToBI produces raw data about the speech analysis in the form of Attribute-Relation File Format (.arff) files which will contain a series of attributes and a large set of data records, where each entry in the record corresponds with a value for a specific attribute. This arff file will be passed into the Machine Learning API Weka so that it can perform an attribute selection process to determine which attributes are the most important for a particular analysis. From the most important attributes the program will find the mean value of those attributes from each file and pass those numeric values into our neural network. The neural network will process the inputs, and then make a guess at the proficiency level of the speaker. If incorrect, the weights on the network will be adjusted and the input will be processed again. We need the network to progressively learn what values constitute proficient English so that it may correctly estimate a speaker's proficiency in the language after training. Overall, the AuToBI program will label the speech file with different prosodic features, and the machine learning software will analyze those features and score the speech files accordingly.

Requirements

Before we started doing any work on the projects, we needed to acquire clear requirements from our client. We needed to know exactly what she wanted before doing anything. To do this, we had some bi-weekly meetings with her the first semester of capstone. After these meetings, we were able to understand clearly what she wanted in terms of functionality and security. Below are the requirements for the two pieces of our project.

Website

The most obvious requirement that we established was that Dr. Kang wants a website that can perform speech analysis on an audio file completely online, no download required. With that information, we established that we would need a page where users can upload an audio file from their computer. We also needed a page where once the user has uploaded the file, they simply just need to click a button to perform the analysis. The results from the analysis then need to be displayed back to the user, so we needed to make another page for that. So we needed to make three pages for all the analysis functionality.

One thing we established rather quickly with Dr. Kang was that she didn't want just anyone to be able to use her website. She wanted it limited to her students and other people that she could approve herself. So we established that we need a user login page, where users need prior approval by Dr. Kang to be able to create an account. In summary, the website requirements are as follows:

- Upload audio file to server
- Analysis is run through the server
- Results displayed back to the user
- User login page
- Create account page
- Admin page for Dr. Kang to easily approve new users

AuToBI

The main requirement for our AuToBI toolchain was that it obviously needed to integrate AuToBI. AuToBI is used to identify features in a speech file, so then our program needed to take the attributes identified by AuToBI and send them off to the next part of the program. Our client suggested that we use a machine learning software on the information gathered by AuToBI and use it for feature selection (identifying the most important features extracted by AuToBI). Once the features and their scores are all selected, we then needed to pass it on to our neural network, which would make a final guess on the proficiency of the speaker. This portion of the project did not require any kind of GUI, so it is all just command line interface.
In summary, the requirements for the AuToBI portion are as follows:

- Use AuToBI for feature extraction on uploaded audio file
- Pass the information into a machine learning program to select important features
- Machine learning also gathers the mean score of each chosen attribute
- Machine learning information passed onto neural net
- No GUI required

Once we acquired all these requirements, we started research on different possible technologies to use to solve our problems.

Architecture and Implementation

In order to be efficient in our development phase, we needed to have a clear understanding of how exactly we were going to implement all the different parts of each project. We developed a clear architecture for both parts of the projects that effectively demonstrate exactly how all these moving parts would work together to yield the result that we anticipated.

Website

To quickly recap the requirements, the web application needs to allow users to login to the site, upload an audio file, perform an analysis completely on the server, and display the results of the analysis back to the user. Dr. Kang also needs to be able to control who exactly can create an account on her site. With these requirements in mind, we were able to create a clear and concise architectural outline of the web application portion of the project.

Database

This part of the web application will store users that Dr. Kang has approved to be able to use the analysis on our system. The plan is that first Dr. Kang will insert access codes that user accounts can be created with. She will have access to a special admin page (that only she has access to) where she can create and manage access codes for the users who want to create an account on the site. A person who is seeking to make an account will input simple profile information such as a username, password, and the aforementioned access code. If the inputted access code exists on our database and a unique username was chosen: the user account will be. Upon account creation, the access code is deleted from our database of acceptable access codes to avoid one access code allowing multiple accounts. This database gets checked from the server anytime the server takes in a user login attempt to see if a user is allowed to analyze their audio file with our web application by having an account in this database. Now a user can log in when they desire to run an analysis using our site.

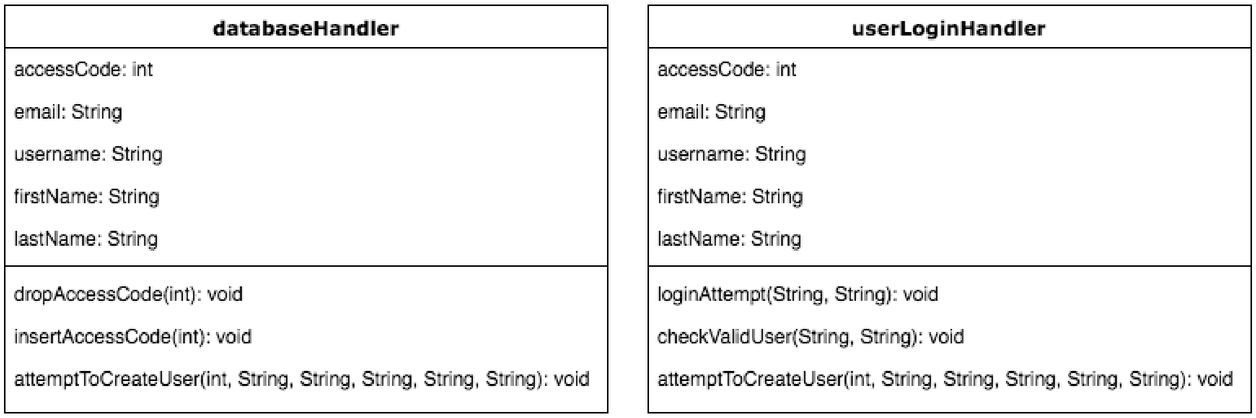
databaseHandler
accessCode: int
email: String
username: String
firstName: String
lastName: String
dropAccessCode(int): void
insertAccessCode(int): void
attemptToCreateUser(int, String, String, String, String, String): void
userLoginHandler
accessCode: int
email: String
username: String
firstName: String
lastName: String
loginAttempt(String, String): void
checkValidUser(String, String): void
attemptToCreateUser(int, String, String, String, String, String): void

Figure A - UML Diagrams for the web toolchain. Database handler functions are ran to allow new users to be potentially made access codes that Dr. Okim chooses. User Login handler functions are then used for user to try to make a new account and try to login in with that account.

The following functions are used for our database system. These functions are conceptual representations of what is exactly happening on the main javascript file of our application, namely "app.js", which has the the input handlers and most of the computation functions of the entire web tool chain:

insertAccessCode(int)

Passes an input number from an admin user (Dr. Kang) from the website into our database. It will then allow for creation of a new account should the user creating the account know the code that the admin has put into the database.

dropAccessCode(int)

Deletes an access code from the database if the number that was entered exists in the database. This happens automatically after someone successfully creates an account using a valid access code. This is meant to stop one user from sharing the code they have been given with other non-approved users or to make multiple accounts.

attemptToCreateUser(int, String, String, String, String, String)

Attempts to create a user for our application. When the server finds that a valid access code has been input, first and last names are not empty, and that a username is not already in use, the server inserts the new account into the list of created accounts in our database and tells the user that an account has been created for them. The dropAccessCode is called here if the function had inputs that lead to successful account creation. If any of the parameters could not be used to create an account (e.g. duplicated username, empty first/last name fields, invalid access code), the user is then prompted to try again from this function.

User Login

The following functions are unique to the login module and while not word for word the same, conceptually the functions look like the following:

attemptToCreateUser(int, String, String, String, String, String)

This function, though previously mentioned, is also associated with this module thus it needs to be included in this section as well. This is because when the user login input correct arguments to create a new account, this will be called if the server found no problems with his or her attempt and thus was successful.

loginAttempt(String, String)

This function gets called when a user tries to log in with the fields in our login page. It contains proper sanitization for the fields so as to avoid SQL injections or other potentially malicious attacks on our system. We use the JavaScript library "body-parser" to accomplish this. After sanitization, checkValidUser is called with the same input, but it is now sanitized.

checkValidUser(String, String)

The server here checks to see if a user logged in with valid input. Valid input would mean that the email and password inputs were found in our database and are associated with some profile. If found to be valid input, the user is alerted that his or her login was successful. If found to be invalid input, the user is prompted to try again or ask Dr. Kang for an access code to create a new account. Upon successful login, the user may now input audio files to be analyzed by the hosted application.

Server

This part of the program is truly the workhorse of our application. Here the server has the job of interacting between the user and database, taking in audio files from logged in users, running the speech analysis, and formatting results in a readable way to the user.

The following functions are unique to the server module and while not word for word the same, conceptually the functions look like the following:

storeFile(file inputAudioFile)

Here the server takes in an alleged audio file from a logged in user, checks if the file has a valid .wav file extension, and if so stores this in the DigitalOcean's local storage. This storage is fine as the .wav files do not have to be hidden to outsiders should DigitalOcean ever get an attack that leaks server documents.

runAnalysis(file inputAudioFile)

This is where the server starts the analysis on the given audio file. This calls on the praat script that exists on the server and all the analysis is performed directly on the server.

generateResultsPage()

Here is just a simple call to stylizeAnalyzedData(file analysisOutput), but for purposes of good code modularization, we nest it in here to keep to convention of "generate" + some page (or file) consistent.

stylizeAnalyzedData(file analysisOutput)

We input analysisOutput to this function so that we can organize and present the analyzed results in a clear way with JavaScript because of the view library PUG. After following PUG syntax, we'll have a web page to show our user the analyzed data in a concise and organized way.

cleanServerAnalysis()

This function call happens after a user logs in successfully. The server deletes from its local storage any previously input audio file and output file. This makes room in local storage for our server for any more input, and allows for a new user user to analyze the same file again, or analyze a file that happens to be named the same as a previous entry.

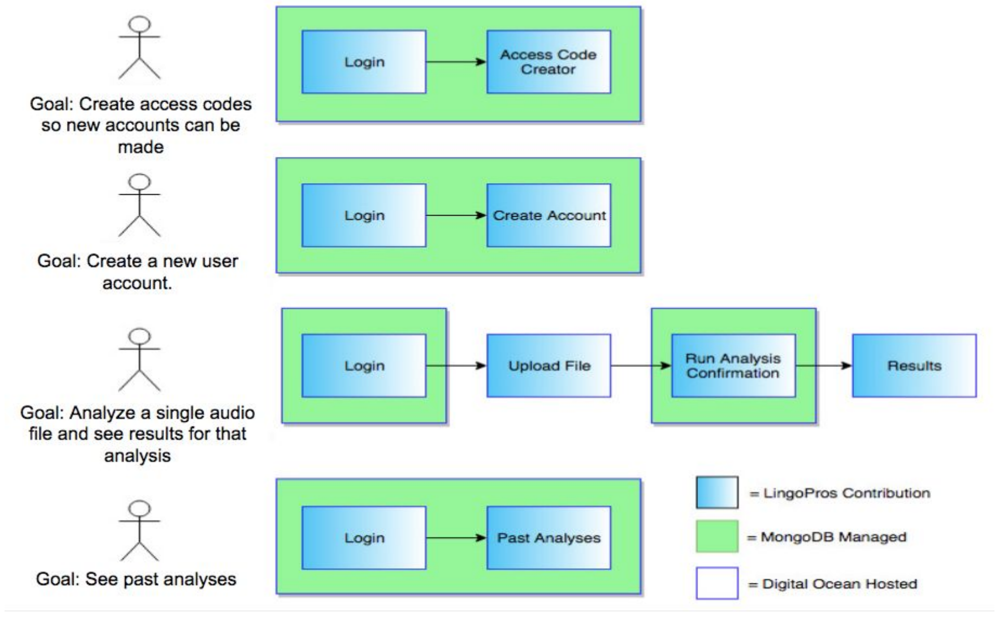
Goal: Create access codes so new accounts can be made
Login
Access Code Creator
Goal: Create a new user account.
Login
Create Account
Goal: Analyze a single audio file and see results for that analysis
Login
Upload File
Run Analysis Confirmation
Results
Goal: See past analyses
Login
Past Analyses
= LingoPros Contribution
= MongoDB Managed
= Digital Ocean Hosted

Figure B - A simple diagram showing the flow of a user using the site for different goals.

All the previous information is our product as-built, but these were not all the initial plan. Perhaps the biggest thing that changed in our development cycle was the decision to not use

MATLAB. The original speech analysis software that Dr. Kang wanted on the website was written in MATLAB. However, our team could not figure out how to successfully install MATLAB on the Digital Ocean server. We proposed to drop the MATLAB completely in favor of the Praat scripts and that decision was made later in the development cycle, but most of the architecture stayed the same. The only real thing that changed was that we didn't need a start script that kicked off a MATLAB instance on the server.

AuToBI Based Analyzer

For the second piece of our project, our team has to develop a speech analysis program that processes speech samples using AuToBI which then determines the speakers proficiency in English based on measurements it obtains through the speech analysis. Initially the neural network knows nothing so it must be trained first in order to calculate the proficiency. After the network is trained it will run the analysis quickly to guess the proficiency score as shown in Figure C.

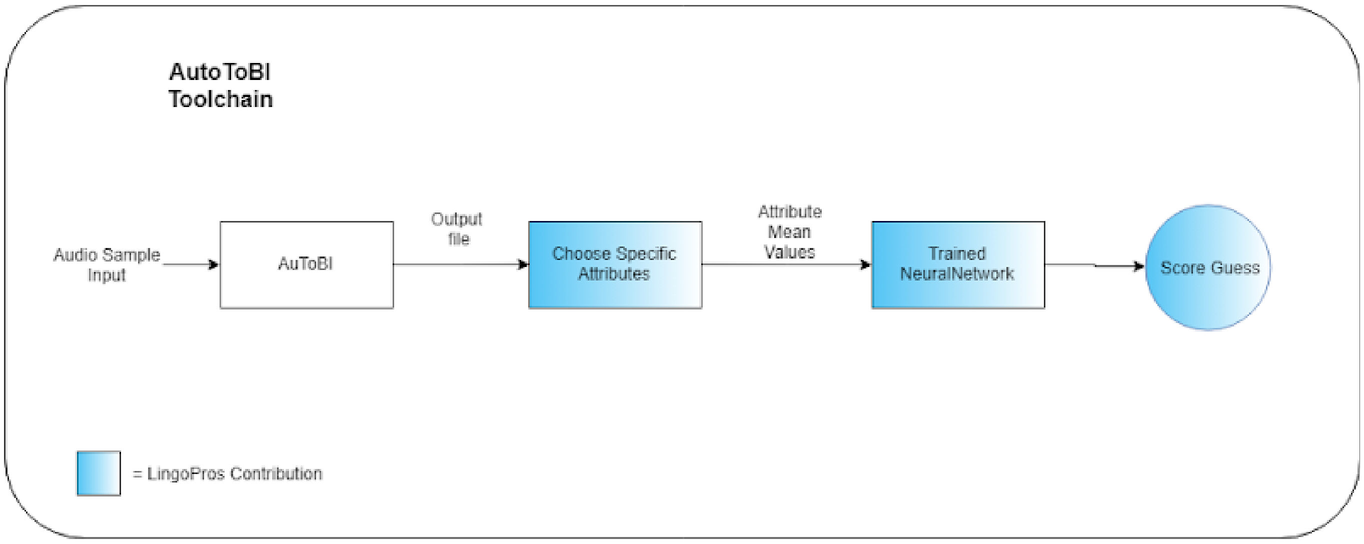

Figure C - A diagram of the basic architecture of the AuToBI Toolchain

Data Analysis

This part of the program will first run analysis using AuToBI inputting an audio file and then outputting the attribute file. After this then we will then take the output and identify which features are chosen the most by Weka as being key features of an analysis. A set of arff (or converted csv) files will be passed through the Weka machine learning suite in which it will perform a feature selection and return the indices of the key features in the arff file. This is challenging because we are not professional linguists so we will need to consult with out client to make sure that the program is selecting the correct features and is working how they intended it to. In Weka there is a method for automatically selecting attributes that seem important, we will be iterating over this process multiple times and analysing the data and using this to determine which features are deemed most important by Weka. Once we select these features we can choose a certain amount of them to begin training the neural net. After training is completed we can then pass in test arff files that the analyzer hasn't processed before and have it make a guess on the speakers proficiency.

Class: autobiRunner

Function: RunAuToBIAnalysis(FilePath...)

Runs AuToBI using the command line arguments of input audio files, model, type of analysis, and the output arff file to be analyzed.

Class: wekaRunner

Function: useLowLevel(Instances)

Uses Weka library to perform an attribute selection on a specific data source, aka on a single arff or csv file. It returns the indices of the attributes selected as a list of integers. This will be called from the countFrequency() function in order to as it loops through all of its input files. This function returns a list of indices that correspond to specific attributes in the arff file.

Function: countFrequency(File[])

Takes in a list of arff files and passes each one to the useLowLevel function that selects important attributes and returns a list of the attributes selected. Once each file is passed through the useLowLevel function, countFrequency counts how many times each attribute was chosen by useLowLevel and returns a sorted, descending order, list of the attributes with the highest frequency. This function will be called during the training session

Function: buildMeanArray(File, int[])

This function will take in an arff or csv file and an array of integers which represent the set of attributes at certain indexes. From these specified attributes, the mean value of each attribute from the data generated will be extracted and placed into an array of double values to be returned. The array of mean values will be passed into the neural network to determine the proficiency.

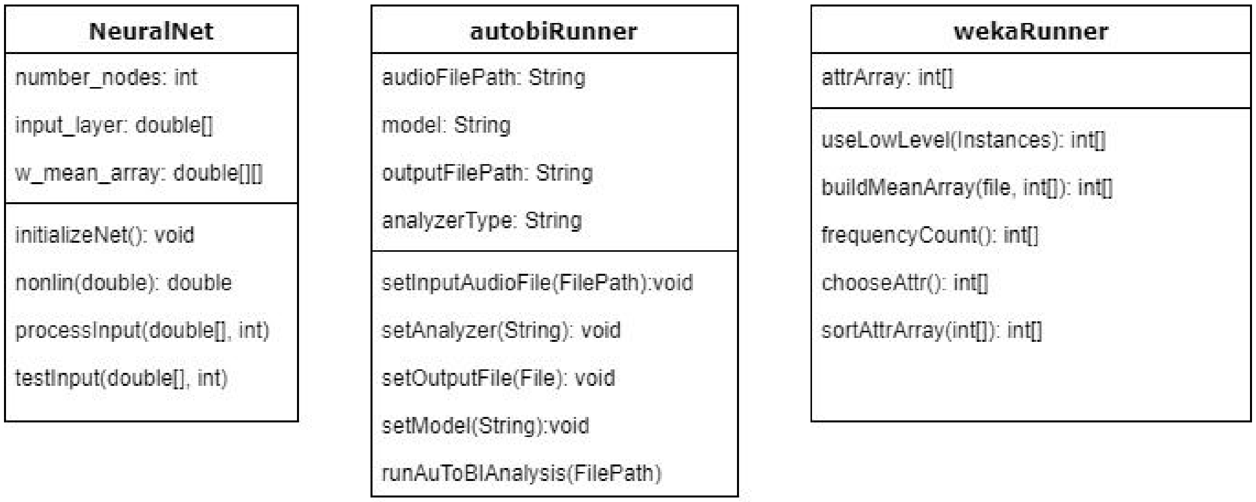

Figure D - UML diagram of different classes used in the AuToBI analyzer. AuToBIRunner is ran first and that output is passed into the wekaRunner which selects the attributes from the arff file which then passes the respective attributes and their data into the NeuralNet.

Training the Neural Net

This part of the program will serve to train the neural network to recognize what measurements of features determine the English proficiency of a speaker. The mean values of several different attributes are passed into the neural network and based on the weights of the graph, it will make a guess at the English proficiency of the speaker. If the guess is wrong, then the weights of the network will be adjusted and a new guess will be made until the network arrives at a correct guess. After several training files with different proficiency levels are passed in, the network should have adjusted itself to correctly determine what values for each attribute accurately determine a speaker's proficiency.

Class: NeuralNet

Function: initialize_net()

Creates a new neural network and randomizes the values of the weights of each node. This will allow us to start with an unbiased neural network for training. This function will be called when a neural network object is instantiated.

Function: nonlin(double)

Normalizes a value between 0 and 1. Will be used to normalize the weights on the neural network after passing values through the net. This function will be called after each pass of data through the network.

Function: processInput(double[])

Passes the input data through the neural net, and makes a guess at the English proficiency. If the guess is wrong, the function checks the difference in error and adjusts the weights of the net so that on the next guess, it can have a better chance at making the correct guess.

Testing the Neural Network

This part of the program will be to test our trained neural network with analysis results that it has not seen before. If the network guesses correctly on a high enough percentage of the test data, then we can confidently say that it is prepared for further samples.

Function: testInput(double[])

Passes the input data through the neural net, and makes a guess at the English proficiency. This function will not make any adjustments to the network.

Testing

Below is our overall testing strategy for the three different types of tests we ran to validate implementation. The first being unit tests to insure that invalid or malicious input was not allowed anywhere in our program. The second being integration testing where we ensure that each piece of the program are interacting together correctly and everything runs smoothly. Finally we have usability testing which only concerns the website portion of the project because the AuToBI portion of our project is used through the command line.

Unit Testing

Website

- Sanitize inputs to prevent SQL injections or malicious input
- Make sure all inputs are valid and correct type.
- Display any error messages letting the user know what went wrong and how to fix it.

AuToBI

- Make sure all inputs are valid and correct file types are used.
- Output arff file correctly edited to be handled by Weka.

Integration Testing

Website

- Ensure client and server are communicating.
- Ensure application and database are communicating.

AuToBI

- Neural Network accuracy is satisfactory.
- Output data transfers correctly from AuToBI to Weka.

Usability Testing

For the website portion of our project we sent the website link to our client and asked her to send it to some of her students so they could test it out. We asked that they answer three questions regarding the website and give any feedback for the website. The first question was to rate on a scale of 1-10(10 being most intuitive and 1 being least) how easy was it for them to understand how to use each of the website pages. The second was to rate on a scale of 1-10(10 most stylish and 1 being least) how good the websites interface looked to them. The last one was to rate on a scale of 1-10(10 being best/correct flow of operations and 1 being least/wrong) How would they rate the flow of the website and the linking of pages. Once they tested and we received their feedback and began to implement some of their suggested fixes. Some of the small fixes included small typo's, or upgrading buttons to be more descriptive. The main problem we found from this testing was that our project could not handle audio files that had spaces in the filename. This is because it passes the filename to the server which runs a command using that name so if it has spaces it thinks that after the first space is a new argument to the command. To fix this we prevented users from entering spaced file names and displayed an error message saying that the filename could not include spaces.

Project Timeline

Our basic overall project timeline started back in September of 2017, when we were introduced to our project and the client. Over the next couple of months we had bi-weekly meetings to establish the project requirements with our client and fully understand their wants/needs. Once we knew more about the project we began to research the two main pieces the website and AuToBI. After researching we began to develop a plan in order to create efficient solutions for our clients problems. We executed this plan by deciding on the best technologies to handle the needs of our requirements and designing our program architecture. Once these blueprints were created and we had a good proof of concept we began implementation. We had several issues come up during this phase that hindered our progress but we found ways to work around them and offered alternative solutions to the problem. The next step after completing our minimum viable product demo was to test and get feedback then improve the software accordingly. Finally, we presented our finished product to our client and they approved of what we had created. Although they were happy with the result there are still some minor things still left to do such as transferring server admin privileges but these things will be handled immediately. Below in Figure E an in depth visualization of our implementation schedule:

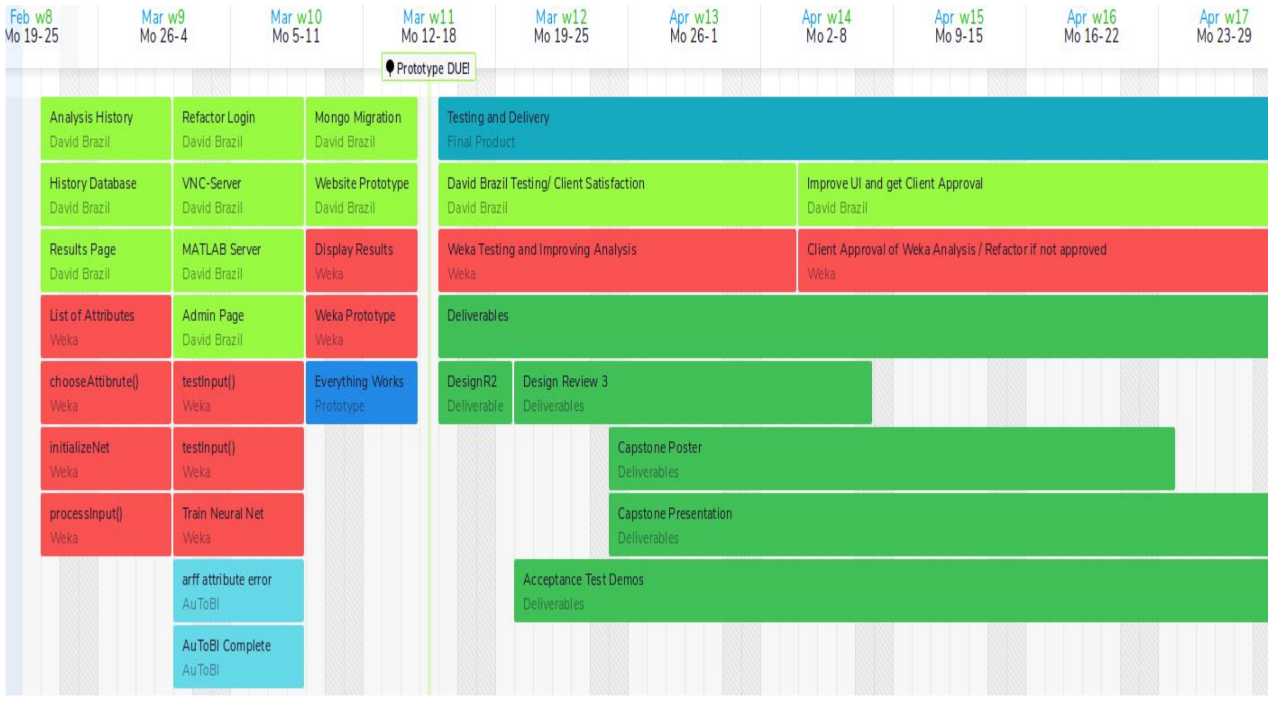

Figure E - A Gantt chart showing the workflow on the implementation of each module and their respective methods. The colors are coordinated by pieces of our project, for example dark green is deliverables and light green is the website portion.

Work Assignments				
Project Details	**Matt Quintana**	**Josh Shaffer**	**Luis Montes**	**Erik Strauss**
AuToBI Weka Analysis	✔	✔		
David Brazil Server			✔	
Web Portal			✔	✔
Team Website	✔	✔	✔	✔
Testing	✔	✔	✔	✔
Quality Assurance	✔	✔	✔	✔

Figure F - A table describing how work was delegated among team members.

Future Work

Working on any project will generate ideas that are outside of the scope of the project. Due to time constraints, it is not always possible to implement all the ideas that are thought up, but they can be outlined for anyone that may work on the project in the future. Our team has thought of some potential implementation tasks that would improve the overall quality of our project. The AuToBI portion is all command line, so we did not think of a whole lot of meaningful graphical additions for that side of the project. However, we did come up with quite a few ideas for the website that could be implemented in the future.

First of all, we did come up with one thing for the AuToBI portion of the project. We toyed with the idea of putting the AuToBI analysis on the website alongside the Praat script so that user could have a choice of what kind of analysis to run. Both analyses provide different results, so depending on what the user wants, they could get a more specialized analysis of their audio file. Another idea of similar concept was to put more than just one Praat script on the server. Again, users could pick and choose what script or analysis they want to run so that they could get different results according to what they want.

Another idea was to reimplement the client's original program without using MATLAB. Perhaps the biggest challenge for us was trying to host MATLAB on a server and we never really quite got there, so we proposed that Praat solution and got rid of the MATLAB entirely. We think that if we had more time, we could have potentially remade the MATLAB using the Weka API so it would be much easier to host on a server.

Those were the major ideas we came up with, but we also had a few minor ideas that would improve the quality of life on the site. One such feature would be a chat system where users could either chat with each other or with Dr. Kang. We also thought that a comment feature would be useful, as users could note any observations on an analysis that they ran. Lastly, a system that allows user to report bugs would also be useful as it would help the website stay running efficiently.

The AuToBI toolchain can also still be improved upon in terms of effectiveness in performing guesses. The improvement of the guesses would have to come from an improvement of the neural network class, and how it runs the adjustment of the map. This portion can be found in the processInput of the NeuralNetwork class, specifically in the processInput function. Currently the program takes the array of mean values and multiplies it by one matrix of weighted values, and then another matrix of weighted values to produce the final output guesses. The 1x10 input array is dot multiplied by a 10x4 weighted array which is then dot multiplied by a 4x4 weighted array to produce a 1x4 array of values, where each entry corresponds to the probability of a specific proficiency being chosen. With the probabilities available, the program takes the highest probability one and uses that as the guess for proficiency. On a wrong guess, the numerical difference between the correct and the wrong guess is back-propagated through the network weights of the 10x4 and the 4x4 matrix.

With this current process, the neural network does not produce completely accurate results and will usually give all proficiencies an equal chance of being chosen. As an improvement, future work can focus on adjusting how error values are back-propagated through the network. It is possible that just checking the numerical difference between the wrong and the correct guess is not enough to effectively adjust the weights of the matrices, so other forms of evaluating error could be researched by future teams.

With these ideas in mind,

Conclusion

Dr. Kang and the Applied Linguistics Speech Lab perform all kinds of linguistics research here at NAU. A lot of this research includes performing manual speech analysis on recordings. This is very time consuming and tedious, so they tried to develop their own program to perform these analyses for them. The problem with this is that they don’t know if the results they get back from their program will be valid or accepted by the community as it is not the standard method of speech analysis.

We built two things to help alleviate their problems. First was a website that Dr. Kang and her students can visit to quickly have an analysis performed on a file that they upload. Second is a speech analysis toolchain that uses the standard method of speech analysis so that

Dr. Kang can compare the results of their model with the standard and see if they are valid or not. The website includes:

- User Login
- Ability for Dr. Kang to control who can make an account
- Ability to upload an audio file into the server
- File analysis that is performed completely on the server
- Intuitive user interface

The AuToBI toolchain includes:

- Command line interface
- Overall proficiency score of audio file

We believe that our project will greatly help Dr. Kang and her students in their research. The website will allow them to analyze speech much faster than if they had to do it all by hand, and the AuToBI toolchain will help them improve the program they have been working on. Hopefully, as their research continues, the field of linguistics will develop a better understanding of human speech and communication, which in turn will improve things like automatic speech recognition.

The project has been an experience to work on and all our members have gained some valuable experience from working on it as well. We feel that this Capstone class has helped us better understand the development process of a project in industry. We are all coming out of this as better developers.

Glossary

1. Discourse Intonation- an approach to the teaching and analysis of everyday speech. It consists of four components: a theory, a set of categories & realisations, a notation, and transcription practice.
2. ToBI: Tones and Break indices is just a method used for identifying features in speech.
3. AuToBI: A free software that runs a ToBI analysis on an audio file.
4. Weka: Java API machine learning framework.
5. MongoDB: Database system that we used

Appendix A:
Development Environment and ToolChain

Website Setup

Hardware:

The website was mainly developed in a Mac environment. A typical Macbook was used, so the system specs are not too high end but specifically has an i7 Intel processor, 8 GB DDR3 memory, and is running OSX 10.13.3 ("High Sierra"). Though we developed the site using a Mac, there is no reason it would not work in either a linux or windows environment as well. Especially since the final product is a website that presents text content, there are very non-intensive requirements for hardware.

Toolchain:

Several tools were used in the implementation of the web application on the website "www.lingoInspect.com:3000" and they can be classified as services, editor, runtime environment, and programs.

The main service used in the project was DigitalOcean and their purchasable virtual machines. We bought (and later transferred billing responsibility to Dr. Okim) a virtual machine that was GUI-less and only command line usable, 100% in the cloud, and ran the lightweight operating system image of Ubuntu 16.04. The virtual machine allowed for us to place our Node.js application and any other command line programs that our web toolchain part of the project needed to interact with. The second service was a domain name registry, which we used www.enom.com and bought the domain "LingoInspect" for a year at $13.95. We bought and applied a domain name at Dr. Okim's agreement so users would not have to enter in a lengthy and hard to remember IP address to navigate to our website where our web application lived. Applying the domain name meant changing DNS settings on both DigitalOcean's virtual machine information and Enom registered domain information.

For editing code throughout the development of the web application, we used the text editor known as Vim. Vim is a very moddable text editor and accessible from the command line in most UNIX systems. Our particular instance of Vim on the Mac machine used to build the project had many plugins that improved speed and efficiency of writing the application like "NERDTree" to see the file directory inside of Vim, "JSLint" to produce quality JavaScript which used yellow to highlight the code's bug suspicious sections and red to highlight the code's runtime error creating sections, and the plugin bundler "Pathogen" to easily add or remove other plugins from the Mac machine's instance of Vim. Being that Vim is accessible from the command line, it was our goal to get knowledgeable and comfortable with Vim because the virtual machine (or "VM" for short) we chose to host the final product on is a GUI-less command-line-only image of Ubuntu 16.04 so small changes there could not be made in a more intuitive GUI text editor like Sublime or Atom. Vim on that VM, though plugin-less, was available and small changes were easy to make even there rather than having to SCP edited files back and forth from the Mac machine to the cloud.

The workhorse of the application in our VM is the Node.js runtime environment. As of 2009, JavaScript (or "JS" for short) was available to be used server side provided it stayed in the Node.js runtime environment. Hosted servers accomplish server-side JS login, like our own project, by installing Node.js on the server (following this tutorial provided by DigitalOcean: https://www.digitalocean.com/community/tutorials/how-to-install-node-js-on-ubuntu-16-04) using "sudo apt-get install nodejs" and running the particular JS application (in our case "app.js") using shell scripts on that server to keep the JS application running. We did exactly that and made it even easier for ourselves using the Node.js library "pm2" which wrote the necessary check status, stop, and start shell scripts for us. We added Node.js libraries using the official Node.js package manager aptly called "npm" which is short for "Node package manager". Adding Node.js libraries to the project was as easy referencing them in app.js as a "var <someRefrenceName> = require('<someNPMLibrary>')" and in the application directory (~/capstonePraatInterface/) running "npm install <someNPMLibrary>". Maybe there is a non npm involving way to install Node.js libraries, but we found npm usage extremely easy.

Programs that needed to be included on our VM were MongoDB a JavaScript based database for data storage like analysis history and account information, and GUI-less Praat: the actual application that interprets a ".praat" script file with a sound file. Instructions for installing and running both could be found at:

1. https://www.digitalocean.com/community/tutorials/how-to-install-mongodb-on-ubuntu-16-04 for MongoDB
2. http://www.fon.hum.uva.nl/praat/download_linux.html for GUI-less Praat.

Setup:

Setting up and running the service locally is discouraged due to the many steps a person must go through to accomplish this goal and can simply use www.lingoInspect.com after a request to Dr. Okim Kang. That said, to the curious or those seeking to implement new features and catch new runtime bugs, here is how to set up the web application locally.

1. Install Node.js on your machine for whichever operating system you use following this link's steps: https://nodejs.org/en/download/
2. Open a terminal and run the commands:

```
sudo git clone https://github.com/louiemontes/capstonePraatInterface.git
```

```
cd capstonePraatInterface
npm install
```

3. Install MongoDB using whatever your system specifications are here https://docs.mongodb.com/manual/installation/
4. Start MongoDB using whatever command line start command that MongoDB website says to use per your system (on the Mac it is “sudo mongod” and on the VM its “sudo service mongod start”).
5. Enter the Database typing in:
   ```
   mongo
   ```
6. You are now in the MongoDB shell and can make the application database by entering in:
   ```
   use dbm
   ```
7. Now create the tables by running the commands:
   ```
   db.createCollection(“users”)
   db.createCollection(“accessCodes”)
   db.createCollection(“fileHistory”)
   ```
8. Open a new tab in your terminal.
9. In this tab navigate to the folder that cloned over from Github in step 2.
10. Now install Praat in this folder by following instructions in http://www.fon.hum.uva.nl/praat/ and navigate to anyone of the “Download Praat” headed links in the top left corner. Use GUI-Less Praat if possible so that the interface will not open every time you analyze a file when using your local copy of our web application.
11. Once Praat is successfully installed, find out your operating system’s command line command to run it. On our VM (GUI-less Ubuntu 16.04) it is “Praat --run” but on Mac (OSX 10.13.13) it is “Praat.app/Contents/MacOS/Praat --run”. Once found, open app.js in any text editor you would like and change the “praatStartCommand” variable (specifically “let praatStartCommand = ‘<PraatRunCommand>’”) to equal whichever your system specific command is.
12. Run in the cloned folder:
    ```
    npm start
    ```
13. Open a web browser to “localhost:3000”
14. Enjoy a personal copy of our service*.

*Disclaimer: We make no hints at maintenance past the Spring 2018 term but would happily accept feature requests or bug reporting from anyone to be displayed on the repo. Feel free the fork the repository or clone it without needing to request permission.

Production Cycle:

The application is being hosted on a virtual machine through Digital Ocean. Dr. Kang has a Digital Ocean account, from which the virtual machine can be SSH'd to. To make any edits to the application, you will need to have her give you access to her account and set up proper SSH keys between your local machine and the server which can only be done through her DigitalOcean interface and a little personal command line maneuvering, tutorials for this can be found at:

1. https://www.digitalocean.com/community/tutorials/how-to-use-ssh-keys-with-putty-on-digitalocean-droplets-windows-users for Windows.
2. https://www.digitalocean.com/community/tutorials/how-to-use-ssh-keys-with-digitalocean-droplets for Mac.

Once you have SSH'd to the server, here are the steps to halt the server to change some features:

- You can check the status of the application by typing "pm2 status www".
- You can also use :pm2 stop www" to halt the application.
- Now you can edit the files as you need
    - Edit front end pages by changing the files in "/capstonePraatInterface/views/"
    - Edit logic of the application in "/capstonePraatInterface/app.js"
- Once you are finished with edits, type "pm2 start www" to restart the application.

The application should be up and running again and ready to resume normal use.

AuToBI Setup

Hardware:

We developed on two separate operating systems linux and windows and believe it should all work on any computer as long as they can run Java programs. Both machines used are basic laptops with average processors and RAM so any modern laptop should be capable of doing everything we did as long as they follow the setup instructions below.

Toolchain:

On the linux machine everything was done on command line with java, ant, and jars. Whereas on the windows machine we used the Eclipse IDE for all development. The program has a dependency on the Weka Machine Learning suite in Java and the AuToBI program developed by Andrew Rosenberg.

Setup:

The setup varies depending on what platform you chose above but the below steps are general guidelines on the process.

Install the AuToBI Runner:
Step 1: Install Apache Ant by using the following guide:
http://ant.apache.org/manual/install.html
Alternatively you could use an IDE with ant projects built in but we used command line ant.
Step 2: Make sure that you have java installed and can be run on command line.
Step 3: Download our programs files from our github at:
https://github.com/jls865/CapstoneAuToBI/tree/master/CapstoneA
Step 4: Make sure you cd into the correct directory where you placed our program files.

Install the WekaRunner & Neural Network Same Requirements as Above:
Step 1: Download our program files from the github at:
https://github.com/MattQuintana/Capstone-Labeler-Demo
Step 2: Make sure you cd into the correct directory where you placed our program files.
Step 3: Ensure that the Weka dependencies are included in the root folder of the project, or just installed on your machine.

Production Cycle:

As previously stated we mainly used command line with a little bit of IDE for our development but the below cycle focuses on command line.

Running AuToBI

Once you have everything setup and are in the correct directory enter the following commands int the terminal:
Step 1: The first command is just: ant after typing ant hit the enter key
Step 2: You will be prompted to enter in four valid inputs they are as follows(Be sure to hit the enter key for every command).
Step 3: The first input is the input audio file which you want to analyze must be of type .wav.
Step 4: The second input is the output .arff file where you want the results to be stored.
Step 5: The third input is the model that you want to use but as of now there is only one available model so just put a 0.
Step 6: The fourth input is the type of analysis you would like to run the following inputs are valid and their value is explained to the right:

- 0 = This command is the boundary tone classifier
- 1 = This command is the intermediate phrase boundary detector
- 2 = This command is the intonational phrase boundary detector
- 3 = This command is the phrase accent classifier
- 4 = This command is the pitch accent detector

- 5 = This command is the pitch accent classifier

An example of exactly what you would type is below:

```
ant
test.wav
out.arff
0
3
```

This command will run AuToBI analysis on the test.wav file and it will place the results into the output file out.arff. The analysis will be ran using the BURNC model (0) and the type of analysis would be phrase accent classifier (3). The output file will be placed into the lib folder of the project after analysis is complete.

Fixing AuToBI data mismatch issues

In order to fix the data mismatch issues that AuToBI generates, an external python script must be used. The script to do replacements is located in the utils folder of the project and it is called replacer.py.

It requires four arguments when called, the filename of the file with the issue, the name of the attribute that has the error, the text that needs to be replaced, and text that needs to replace it.

From the command line, the command would be:

python text_replacer.py "filename" "attribute name" "text to replace" "text to replace with"

The program searches for the specific attribute that it needs to fix in the file, and then replaces the text in the line with the replacement text.

Weka and Neural Network

In order to run Weka and the Neural Network you will need to do the following steps:
Step 1: Make sure that you have an arff output file from AuToBI to input into the program.
Step 2: Navigate to the bin directory of the project folder.
Step 3: To call the general tester of the arff file, call the following command from the command line.

java capstone_demo.neuralNetAnalysis [location of arff file to pass]

The program should output a guess about the proficiency score of the file input.