TEXT PROCESS TEAM TEST SPEC

Test Strategy

Our Testing Strategy was quite simple. We wanted to implement 1 feature at a time and thoroughly test it so we wouldn't have to spend an incredible amount of time debugging at the end. By splitting each feature in our code into a separate function, we were able to easily run unit tests on each. The unit tests were designed to reach all possible inputs and cases making sure everything worked as intended. When all features were complete. We ran a final set of tests making sure the integration of all functions were working as intended. Since we already knew each component was working flawlessly individually, the only problems we ran into were due to integrating them all together which was quite easy to debug and fix.

Test Plan

Our testing plan was to make sure each of the use cases were covered and tested. We ran multiple tests per test case making sure each test case hit at least 1 use case. By formulating our tests this way we came up with 3 solid test cases that were able to hit these requirements allowing us to thoroughly test our model.

Without access to web services due to lack of time. We created our own MP4 files to run tests on. We made sure to use a few different files to make sure everything would run as intended. However, this limited our testing because we lacked integration testing with the other teams.

Traceability Matrix

	Tracibility Matrix						
UC1	Convert MP4 to Wav		UC1	UC2	UC3	UC4	UC5
UC2	Convert Wav to Text	TC1	x	x			
UC3	Analyze for Correctness	TC2			x	x	
UC4	Save to txt file	TC3	x	x	x	x	x
UC5	Send ouput string back to dialogue						

Test Cases

Name: Convert MP4 to Wav to Text

Requirement Covered: Correctly converting mp4 audio into a string of text

Purpose: Allow String to be used to analyze for correctness and stored to transcript

Input Test Data:
https://github.com/NootCode/CS4800-Final/tree/master/TestFiles/helloTest.mp4

Steps Taken:

Unit tests ran on mp4 to wav conversion

Unit test ran on wav to text conversion

Expected Result:

We expected equality between the spoken sentence (audio file) and the string it produced.

Overall Outcome: After a few tests and some limitations of the speech to text API we were able to successfully convert the mp4 files to text. We had some issues with direct spelling and slight inconsistencies but that should not be an issue for the program.

Name: Analyze Correctness/Save to txt

Requirement Covered: Analyze the correctness of an input string and save string to txt file

Purpose: Calculate the correctness score and append the string and score to a txt file

Input Test Data: String “Hello my name is Andre and I am a very hardworking person. I began CS 6 years ago and have 3 years total of work experience”

Steps Taken:

Unit tests ran on correctnessScore function

Verified txt file output to match string and score given

Expected Result:

We expected the score to be above .1 using the keywords given. We also expected to see that exact string and the produced score to be appended to the transcript file.

Overall Outcome: The outcome we got was the exact outcome we expected with not many problems. The only issue we currently have is a lack of keywords which impacts the score given. The more keywords we implement the more confident we are in the score produced.

Name: Full Integration

Requirement Covered: Test the functionality of the whole program from start to finish

Purpose: Make sure everything works properly together. Starting with MP4 input

Input Test Data: https://github.com/NootCode/CS4800-Final/tree/master/TestFiles/intro.mp4

Steps Taken:

Record an MP4 file

Run the entire program using that file

Expected Result:

We expect to see the mp4 file converted into a wav file then transcribed into text. That text should be given a correctness score then appended to a text file.

Overall Outcome: After some testing and tweaking we were able to integrate all aspects of our program together. All the functions work as intended. The only issue we have is connecting to web services.

Test Summary

Since we had quite a linear program to implement our testing strategy proved to be very effective. By implementing each function after the other and testing them all separately we were able to narrow down the bugs towards the end. By doing so we were able to save quite a bit of time testing allowing us to spend more time fine tuning our model. Unit Tests proved to be very efficient in catching any unnecessary bugs and unwanted features. The final version of the project still lacks a little functionality with the main issue being the connection to the web services, however that shouldn't be too difficult to implement and test due to the lack of bugs in our current version.