Project Factor Research and Development Results

J.M. Fitch | jfitch@hawk.iit.edu

Illinois Institute of Technology

Abstract

We convened a group to build a system to automatically generate text transcripts from IIT course videos. My research was focused on media conversion and processing, and quantification of the quality of transcript output. This essay describes the methods and results of this research.

Project Factor Research and Development Results

"Project Factor" was initiated to build a suite of applications that would automatically generate text transcripts from IIT course videos using open-source software tools. There were myriad problems to solve, but responsibilities were ultimately distributed based upon each member's interests and skills. I have some background in media production and business analytics, so my research was focused on three main areas:

1. extraction and processing of audio from video source files for input into the speech recognition engine,
2. quantification of transcript quality, and
3. conversion of source video into a common format for our "Window Pane" application (which will be described in greater detail).

Varying degrees of success were achieved. This essay will describe the progress that was made, and will also include suggestions for continued research and development.

Environment note: while the software tools utilized in this research are all cross-platform, development was performed in a Windows 7 environment; thus, any script examples are written in MS-DOS.

Part 1: Audio Extraction and Processing

The speech recognition engine chosen for our research was Carnegie-Mellon University's Sphinx Speech Recognition Toolkit[1]. This engine requires that the format of audio input files be "16kHz (or 8kHz, depending on the training data) 16bit Mono (= single channel) Little-Endian" (Frequenty Asked Questions (FAQ) - CMUSphinx Wiki, 2013). Also, the Sphinx documentation specifically states that "zero silence regions in audio files decoded from mp3 break the decoder" (Frequenty Asked Questions (FAQ) - CMUSphinx Wiki, 2013). In order to avoid this error, we decided to use lossless encoding in a WAV container (lossless encoding is that which is able to reproduce the original data in an unaltered state; BMP image files, FLAC audio files, and WAV audio files are all encoded using lossless algorithms (Salomon)). This also has the benefit of not introducing additional compression artifacts to the converted audio stream, as most video files utilize lossy algorithms for both the video and audio streams.

Prior to conversion, files needed to be pulled from IIT's servers into the local environment; GNU Wget[2] was the tool utilized for this job. Extraction and conversion of the audio streams was accomplished using ffmpeg[3], an open-source, cross-platform media conversion tool. Finally, Dropbox[4] was used for deployment of

[1] http://cmusphinx.sourceforge.net/

[2] http://www.gnu.org/software/wget/

[3] http://www.ffmpeg.org/

[4] https://www.dropbox.com

the output files to the rest of the team.

Wget and ffmpeg can be run directly from the script directory; the Windows binaries used can be downloaded from http://gnuwin32.sourceforge.net/packages/wget.htm and http://ffmpeg.zeranoe.com/builds/, respectively. A sample script which will pull the videos, extract the audio, and publish the output files is as follows:

```
1  ::GET FILES
2  wget -nd -r -N -A mp4 http://ricelake.rice.iit.edu/spring2013
3
4  ::CONVERT AUDIO
5  for %%X in ("*.mp4") do ffmpeg -ss 60 -i "%%X" -t 30 -acodec
   pcm_s16le -ac 1 -ar 16000 "output\%%X.wav" -y
6  for %%X in ("*.mp4") do ffmpeg -ss 60 -i "%%X" -t 30 -af
   "highpass=frequency=500, lowpass=frequency=4000,
   volume=volume=12dB" -acodec pcm_s16le -ac 1 -ar
   16000 "output\%%X.eq.wav" -y
7
```

```
8   ::POST TO DROPBOX

9   move output\*.* %dropbox%\
```

Wget[5] and ffmpeg[6] each have an extensive library of command line options available, but the important sections of the above commands are as follows:

1. Line 2: `-A mp4` indicates the file extension(s) to pull; this list can be extended by adding comma-separated variants, i.e.: `-A mp4,avi,wav`.
2. Line 2: `-nd -r` commands Wget to recursively search the server and return the files without the source directory hierarchy.
3. Line 2: `-N` commands Wget to retrieve newer files only.
4. Line 5: `-ss 60` commands ffmpeg to start encoding at 60 seconds; `-t 30` commands it to stop after 30 seconds[7].
5. Line 5: `-acodec pcm_s16le -ac 1 -ar 16000` sets the output file encoding method, sampling rate, and bitrate.
6. Line 6: `-af "highpass=frequency=500, lowpass=frequency=4000, volume=volume=12dB"` applies a set of equalization parameters to the

[5] http://www.gnu.org/software/wget/manual/wget.html

[6] http://ffmpeg.org/ffmpeg.html

[7] The files created using this script were meant for experimental use, thus the truncation.

output file[8].

All files created using this method were imported and read successfully by Sphinx.

Part 2: Quantification of Transcript Quality

Using the methods outlined in part 1, we were able to generate audio sample files from all of the 115 available videos. What quickly became apparent was the dramatic variation in the quality of the audio being supplied: variations in recording quality, signal-to-noise ratio, environmental noise, volume, and speaker clarity and cadence, were easily identifiable in subjective A/B comparisons. As a result of these variations, we observed (again, subjectively) wild swings in the quality of Sphinx's output.

The Sphinx engine allows a great deal of customization, and experiments were performed to try to affect the quality of the transcript output[9]; also, preprocessing of the audio signal seemed to have an effect on output. Unfortunately, the necessity for further experimentation was not realized until near the end of the semester, so rigorous measurement techniques were not able to be developed. In order for further experimentation to be successful, these techniques *will* need to be developed.

[8] Equalization was applied for use in the experiments described in part 2.

[9] See B. Bailey's results for further detail.

First and foremost, benchmark transcripts will need to be created. Unfortunately, there is likely no simple way to do this: the most viable method of creation would require manual transcription, whereby a listener simply writes out what is said on the audio sample. This would be a time-consuming process, but by using short sample files (we used 30 second samples), the time required could be brought to a reasonable limit.

A complicating (and significantly challenging) extension of the manual transcript generation would be the addition of timestamps to the transcript. Sphinx output is customizable, but by default it is simply a text file, formatted "$word_1$ ($start_1$, end_1) $word_2$ ($start_2$, end_2) etc.", where *word* is the processed word, *start* is the start time in seconds, and *end* is the end time in seconds. In order to perform the most detailed possible analysis, these time stamps would need to be added to the benchmark transcript, which may prove to be prohibitively time-consuming.

An alternative - and less time-consuming - strategy would be to develop a subjective weighting system, wherein a user could rank the quality of a set of candidate files from best to worst. This would be simple to deploy, and would at least be capable of displaying high-level patterns of improvement as audio processing and Sphinx settings are changed. Should the project continue, this is the approach we will initially pursue.

Despite our lack of a rigorous quantitative analysis framework, it is worth

noting that the quality of the source audio stream is clearly important to producing quality transcripts. Many of the class videos were apparently recorded using a laptop microphone with a distant speaker, which is a suboptimal recording environment. A headset or lapel microphone worn by the speaker will almost certainly improve Sphinx output.

Additionally, some of our preliminary experiments introduced dramatic equalization adjustments (see part 1, note 5) in order to compensate for the high signal-to-noise ratio of these videos, but further research into this technique is required to produce a definitive result. A normalization process may be of particular interest, though ffmpeg does not have such an integrated routine ("normalization" refers to an automated volume adjustment which brings the audio stream's peaks to just under the maximum volume level). It is thought that this - and other - audio adjustment may produce higher quality Sphinx output (Stern).

Part 3: Video Conversion for Use in Window Pane

"Window Pane" was the subproject dedicated to developing a user interface for interacting with the videos and transcripts[10]. Part of this interface is a video window which will display the video parallel to the transcript. For the sake of reliability and consistency in this interface, it was determined that the source videos should be converted to a standard format, as the source files used a broad range of

[10] See M. Hoekstra's results for further detail.

encoding schemes, resolutions, and containers.

While ffmpeg is capable of performing the video conversion task, we ultimately chose HandBrake[11] - an open-source, cross-platform video conversion tool - for its performance and simplicity. The Windows command-line version can be downloaded from http://handbrake.fr/downloads.php, and like ffmpeg and Wget, the program can be run portably. A sample script which will convert all AVI files in a directory to 360p H.264/MPEG-4 MP4 files is as follows:

```
for %%X in ("*.avi") do "HandBrakeCLI.exe" -i "%%X" -o
"output\%%X.mp4" --preset="Normal" -Y 360 -X 640 --start-at
duration:60 --stop-at duration:90
```

Like ffmpeg, HandBrake has an extensive library of parameters, which can be viewed at https://trac.handbrake.fr/wiki/CLIGuide. The few customizations the sample script does apply are as follows:

1. `--preset="Normal"` commands HandBrake to use its “Normal” preset to convert the files.
2. `-Y 360 -X 640` commands HandBrake to restrict the output dimensions to 640 pixels wide by 360 pixels high.

[11] http://handbrake.fr

3. `--start-at duration:60 --stop-at duration:90` commands HandBrake to convert only the 30 seconds of video starting at one minute.

The videos output by this script are adequate, but further optimizations and tweaks would likely be added in the final environment.

Conclusion and Recommendations

The audio and video extraction and conversion tasks described were fully successful. However, further development of the automation environment will be necessary to scale up the system, as processing full-length videos will require computing resources well beyond those available in a personal working environment. The Eucalyptus system developed by J. Hajek is the obvious choice for this, so the sensible next step would be to explore deploying these tasks to it.

Transcript quality measurement is a much more significant challenge, and is of utmost importance if the project is to be successful. The quality of the output generated by Sphinx was highly inconsistent, and the sheer number of independent variables necessitates that an effective experimental environment be designed to improve this. Should the project continue, my primary focus would be on solving this problem.

References

(2013) Frequenty Asked Questions (FAQ) - CMUSphinx Wiki. *cmusphinx.sourceforge.net*. Retrieved on July 27, 2013 from http://cmusphinx.sourceforge.net/wiki/faq#qwhy_my_accuracy_is_poor

Salomon, David. (2009) *Handbook of Data Compression*. Springer-Verlag, London, England

Stern, Richard M. and Acero, Alejandro. (1987) Acoustical pre-processing for robust speech recognition. *HLT '89 Proceedings of the workshop on Speech and Natural Language*, 311-318.