A Survey Of Open-Source Python Speech Recognition Libraries and Music Lyric Transcription

Isaac Wines

Governor's School for Science and Technology

Senior Research Project

I have read this document and approve the content

Amndeep Singh Mann

The MITRE Corporation

I have followed the Academic Integrity policy of the Governor's School for Science & Technology. I have neither given nor received unauthorized assistance in the preparation of this document.

Isaac Wines

Abstract

Speech recognition libraries for developing speech recognition programs are readily available for any developer to get their hands on. Each library's pre-trained models, architecture, and capabilities differ in training quality, code quality, and ease of use. We test libraries such as Whisper, Google-Cloud-Speech, Deepgram, Vosk, PocketSphinx, and Wav2letter to give developers the ability to choose which library best suits their needs. As an interesting problem, we test these libraries with “noisy” environments, music, so as to infer their ability to accurately transcribe busy audio. We used five different genres of music, each consisting of fifteen selections, to check the accuracy of the transcription. In testing these libraries, we collect an accuracy measurement by utilizing the Levenshtein distance statistical model to calculate how many changes would be necessary to make the transcription completely accurate to the lyrics of each song. This survey reports on these metrics as well as several other subjective and objective measurements. A full repository of the created graphs and code can be found here: Speech Recognition Mass Survey Tool.

Introduction

This research survey aims to analyze different open-source speech recognition libraries' abilities to transcribe accurately in busy or noisy audio environments. These libraries are written in or have bindings in Python. We test each library's ability to perform under this stress through the testing of 5 different genres of songs, each with varying levels of noisy environments or difficult speech patterns (faster singing or singing that is more screaming). Testing these variations allows for the inference of how the libraries could transcribe real-life audio environments. The libraries are built off of a statistical model called the Hidden Markov Model

(Rabiner, 1986), which allows an algorithm to determine the most likely word or character from changes in the vocal patterns of audio files. The speech recognition models use this to train, as the more data is fed into the algorithm, the more accurately the model can predict the possible outputs. A major part of the project was the design and development of the survey tool that allowed for the automatic downloading of songs, management of the audio and official lyrics, transcription of the songs, distance analysis, and the output of a formatted data file all performed automatically from a single command line input. The distance value is created from comparing the official lyrics and the transcribed lyrics using the Levenshtein distance statistical model (A Normalized Levenshtein Distance Metric, 2007). It is a statistical model that takes two strings of characters and outputs how many changes it would take to make one string equal to the other. Using this metric, we can determine the accuracy of a transcription by finding how many characters it would take to match the official lyrics. Through the automatic testing tool created for this survey, we can mass analyze all of our songs through every genre, then parse and automatically create graphs for later data analysis of which library, song, and genres are the most accurate.

Materials

Developmental

The development of the survey tool had two components: Github Codespaces and Poetry. Github Codespaces allowed for the active development of the project to occur across multiple computers and platforms at any location. Poetry is a dependency manager for the project, creating a file that stores the data for each library and its necessary components to work across Windows and Linux.

Dataset

The dataset of songs used for this project was a collection of 5 genres, each with 15 songs. Songs were selected based on current popularity and length, with some songs only being a minute and a half while others were upwards of 7 minutes. These songs were formatted into a data structure file called a JSON file. Python natively supports this file type and allows the songs to be easily linked to their full names, artist names, genres, and YouTube links. The structure created in the dataset allowed for any additions or changes made to a specific song to be automatically recognized and used in the program. For example, the survey tool takes each one of the songs, sorts it by genre, and then transcribes each one based on the selected libraries; the JSON file feeds in the necessary information to the survey tool, so it knows what songs are what genre, where on Youtube to download the audio file from, and what the real artist name and song name is to gather the official lyrics for that song.

Methods

Library Selection

The selection of each speech recognition library was based on multiple qualitative points: the level of documentation, the current development, the cost of the library API usage, and the needed computer hardware specifications to run said library properly. After researching many different libraries written in Python or with Python bindings, I decided to utilize Vosk, Google Cloud Speech, Whisper, Deepgram, and Wav2Vec.

Vosk is a free, lightweight program with well-written documentation that goes in-depth into the possibilities and how to use the library to transcribe audio. I chose this library because of its free open-source nature and the few hardware resources it used. The downside of this library was the difficulty in actually coding the components to get the library to transcribe properly. Google Cloud Speech was the next selection; being supported by Google and written for multiple languages (with strong Python support) through the Google CLI tool, anyone can easily set up a transcription tool utilizing their incredibly powerful models. The downside of this library is that a fee is charged every time the program calls the API, and the fee varies based on the length of the audio file. The third selection was Open-AI's Whisper; this library is backed by its powerful speech recognition models and impressive ability to transcribe quickly and straight off the local computer without needing Wi-Fi. The Whisper API is free to use, has well-written documentation, and is easy to set up. Deepgram was the fourth selection as a paid API; it was selected to be a comparison to Google Cloud Speech's paid API. Deepgram is a small company with a well-trained model supplied to the developer. It is actively being developed and changed consistently. Facebook's Wav2Vec was the final choice because it is a free model with interesting features such as auto chunking (splitting audio files into smaller files before transcribing to allow for faster and more accurate transcription). Its biggest downside is the poor documentation and the lack of support/development.

Other libraries, such as Pocket Sphinx, Speechbrain, DeepSpeech, and Speech Recognition were attempted to be integrated into the program but not used due to depreciated dependencies, difficulties setting them up, and removal of support.

Testing Framework

The design of the testing framework was split into different sections: the audio tools module, the data folder, the lists module, the main survey tool itself, and a module for each specific library. The audio tools module consists of the data ingest tools that allow for the download, normalizing, and pre-processing of local or online audio files to ensure the ability of each library to properly transcribe each song. The survey tool utilizes the dataset created with YouTube links, song names, and artist names to automatically download each song and official song lyrics to the machine. The data folder is where the final output JSON file from the survey tool and each song file are stored. The lists module consists of where the dataset dictionary is and where the official lyric JSON is stored for the survey tool to quickly access.

The Survey Tool

The survey tool is also known as a wrapper program; it takes in multiple other Python modules (coded separately to ensure there was no overly cluttered and unorganized code) and ties them all together. After running the tool, it checks that the necessary data folders are there and that every song from the dataset is properly ingested and each official lyric is downloaded. After the checks, the program processes a command line input from the user so it knows exactly what song, library, and genre to transcribe and output. The tool knows this through the available flags that the command line input can accept; through this feature, the program can take custom song inputs for processing. A sample command input would be: `python survey_tool.py -s alone -l whisper -o alone_whisper_transcription` (see Appendix Methods Figure 1). After this command, the survey tool will utilize the `Automate` class, a class written to process the flags that were used in the command and alter the automatic transcription for only the song/songs selected (which in the command above's case would be “Alone and

Forsaken" by Hank Williams) and the library/libraries selected (in this case only Whisper was selected) once the class processes these flags it then calculates and records the Levenshtein distance, time, genre, transcribed lyrics, and official lyrics for each song selected into the final output file (see Appendix Methods Figure 2).

Analysis of Data

Due to its native support, the final outputted JSON file can then be fed back into a separately coded data analysis tool in Python; in JSON files, we can easily interpret the quantitative data given, organize it by song, genre, or library, and output it automatically into graphs for later result interpretation.

Results

Song and Genre Analysis

The file created bar plots for each song's combined data across every library, a barplot to display the average accuracy for every song, and a scatterplot for the accuracy vs time of every song. By analyzing the songs, we can determine the accuracy of the libraries when transcribing to different environments. On average, the country genre songs have a less noisy environment, with slower speaking and fewer instrumental verses throughout most pieces. One particular song from this genre is "Alone and Forsaken" (see Appendix Results Figure 1). We can see from this graph a representation of the genre as a whole, with a high average accuracy of 83.2 percent and a relatively normal transcription time. This is, on average, the result for the country genre being the most accurate genre for the libraries across the board (see Appendix Results Figure 8). Through the "Alone and Forsaken" graph and the genre summary graph, we can conclude that

the country genre is the most accurate and the fastest to transcribe due to the shorter length of the songs and the libraries' innate ability to transcribe less noisy environments faster.

The rock genre tests each library more intensely than the country genre due to its longer song lengths and singing verses being muddled with louder instruments. The average data recordings for this genre consist of a time to transcribe that is about the same as country songs; however, they have a lower accuracy score. This is represented by the song “Vida La Viva” by Coldplay (see Appendix Results figure 2) and the overall accuracy (see Appendix Results figure 8), which is significantly lower than that of its country counterpart.

The heavy metal genre is one of the worst performing genres, with about the same length of songs as the rock genre but with much harsher, more erratic singing or even just screaming. This is why the genre, as seen in the song “Bark at the Moon” by Ozzy Osbourne (see Appendix Results figure 3), on average, performs significantly lower than the rock or country genre yet takes much longer to transcribe. We can infer that most songs are too noisy and have too many non-speaking yet verbal verses for the libraries to accurately transcribe. This genre consistently performs poorly, with the song “Twist” by Korn even becoming a major outlier with negative accuracy values due to the incredibly poor transcriptions created (see Appendix Results Figure 6).

The pop genre was varying in accuracy and time as it consisted of a highly diverse group of songs. The lengths and noisiness of the environments have created a lower accuracy value when compared to the country genre, but it still places itself second in the graph (see Appendix Results Figure 8).

The hip-hop genre was the lowest-performing genre due to the incredibly noisy environment and the way the singers spoke. This genre mainly consisted of rap songs, which proved to be a challenge for the libraries as they could not accurately grasp the fast-speaking verses of the songs, creating bad transcription outputs and very long transcription times (see Appendix Results figure 8).

Library Analysis

We can determine through the graphs (see Appendix Results figures 7 and 11) that the two best-performing libraries are Deepgram and Whisper. Across all the genres, both libraries constantly had the highest accuracy while having the lowest time. Deepgram performs slightly better than Whisper, with an overall 42.61 percent accuracy rate, while Whisper has an overall accuracy rate of 41.62 percent (see Appendix Results figures 12 and 13). Deepgram took significantly less time, which resulted in a better accuracy vs time rating. Vosk being the worst accurate library was surprising due to the size of the pre-trained model, while Google Speech Recognition scoring low was also unexpected due to the support the library has from Google.

Discussion and Conclusion

The quantitative results of this survey can be interpreted in many different ways, concluding the ability of the libraries to transcribe noisy conditions while also providing interesting results of how each song interacted with the libraries. We deduced that the country genre was the easiest genre for the libraries to transcribe based on the nature of country songs. This tells us that the libraries can distinguish the clearer speaking with some noise in the environment, while the results from genres like hip hop and heavy metal tell us that the libraries struggle with faster speaking or louder screaming paired with heavier instrumentals. The noise of

these environments contributes mainly to the transcription time and accuracy of the transcription significantly more than the actual length of each song does.

Hip hop made the libraries struggle due to the fast speaking with fewer instrumentals than heavy metal did with more scream-like singing and heavy instrumentals while singing. We can interpret that, on average, the libraries would perform significantly better under controlled speaking with minimal background noise, yet they can transcribe some accurate rock and pop songs, with almost no readable transcriptions coming from heavy metal or hip-hop.

Deepgram and Whisper proved to be the most resilient libraries, being able to transcribe some songs in heavy metal and hip hop with some accuracy. This tells us that for future use cases, transcribing more songs and audio files with heavy background noise or transcribing speaking from audio files with muddled speaking, Deepgram, or Whisper would be the best choices. With specifically trained models for noisier environments, these libraries would be the best choices for most use cases.

The qualitative data collected on each library can also show us different sides of each library. For example, despite Deepgram and Whisper having similar accuracy rates, the developer would have to make a choice on if it is worth using a quicker but paid service or a slower but free service, respectively. The qualitative data supports Vosk's last place ranking due to the low-quality documentation and difficulty setting up even with its large model. Google’s poorer performance relative to Deepgram and Whisper was especially unexpected because it is a paid API backed by Google’s Cloud Services.

The outliers present in the graphs also prove interesting results, especially the song “Twist” by Korn, which shows an example of a song that every library did poorly on to the point

of a negative accuracy value. The transcription created this value, by creating more words from the song than was actually in the song, creating a longer transcription than the official lyrics. This longer transcription length resulted in the negative value because of the equation used for calculating the average accuracies: `(official lyric length - transcription length) / official lyric length`. Other outliers, like “Bark at the Moon” by Ozzy Osbourne, display an accuracy score almost equal to the time it took to transcribe the song, which is unusual for most of the songs in the dataset. These heavy metal outliers show a low level of accuracy with high time value compared to hip hop, which had a similarly low accuracy but faster time values.

We see a trend of lower transcription quality or an increase in transcription time when either or both the song is longer or has a noisier background behind the vocals. Through this survey, we have found that objectively, Deepgram and Whisper performed accurately through the majority of the stress testing, giving us reasonable evidence to conclude that they would be the best choices for transcription of busy audio environments, fast speaking, or normal, clean, environments.

Acknowledgments

I thank my mentor, Mr. Mann, for his wealth of knowledge and guidance. I would also like to thank Ms. Vobrack for giving me access to MITRE and this huge opportunity to learn.

Literature Cited

L. Yujian and L. Bo (2007). A Normalized Levenshtein Distance Metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(6), 1091-1095. doi: 10.1109/TPAMI.2007.1078.

L. Rabiner and B. Juang (1986). An introduction to hidden Markov models. *IEEE ASSP Magazine, 3*(1), 4-16. doi: 10.1109/MASSP.1986.1165342.

Appendix Methods

```
@click.command()
@click.option("--everything", "-e", is_flag = True, help="Will run everything.")
@click.option("--specific", "-s", multiple = True, help='Enter the short name of the
specific song or genre you want transcribed. Run -o to see the list of songs.')
@click.option("--library", "-l", multiple = True, help='Enter the specific library you
want to use to transcribe your songs, if not in use all libraries will run.')
@click.option("--output", "-o", help='Enter the name you want your result file to be.
Do not use this flag to autocreate file name. The program will create a numbered result
file if the file name already exists or if flag has not been called.')
@click.option("--arguments", "-a", is_flag=True, help="Shows a list of the songs.
Displays: ['short name', 'real name', 'artist name', 'genre']")
@click.option("--custom", "-c", help="Allows for the use of custom songs to be
transcribed. Enter '-c [youtubelink],[desiredsongfilename(No Spaces)],[fullartistname
(Spaces Seperated By -)],[fullsongname(Spaces Seperated By -)]\n -c https://www.youtube.
com/watch?v=dBN86y30Ufc,willie,Willie-Nelson,On-The-Road-Again")
```

Figure 1: The code for each Click CLI command as explained above

```
def automate(self):
    temp_song_result_dict = generate_library_data_setup(self.library_choice)
    song_result_dict = {}

    try:
        for song in self.lyrics:
            try:
                lyric = audio_tools.normalize.normalize(self._lyric_dict[song])
            except:
                lyric = "no lyric found"

            for lib in temp_song_result_dict:

                print(f"\n{lib}", f"_{song}_")
                print(self._testing_paths[song], "\n")
                library_string = {
                    whisper_folder.whisper_demo: "whisper_demo",
                    wav2_folder.wav2letter_demo: "wav2letter_demo",
                    vosk_folder.vosk_demo: "vosk_demo",
                    google_folder.google_speech_demo: "google_speech_demo",
                    deepgram_folder.deepgram_local_demo: "deepgram_local_demo",
                    #pocketsphinx_folder.pocketsphinx_demo: "pocketsphinx_demo"
                }
                results, time_calc = run(self._testing_paths[song], lib)
                try:
                    temp = {song: {f"{library_string[lib]}": {"genre": f"{self.
                    _testing_paths[song]['genre']}", "results": f"{audio_tools.
                    normalize.normalize(results)}", "lyrics": f"{lyric}",
                    "distance": f"{distance_c(audio_tools.normalize.normalize
                    (results),lyric)}", "time" : f"{time_calc}"}}}
                    print(temp)
                    song_result_dict.update(temp)
```

Figure 2: The `automate` function from the survey tool.

Advanced Methods

Library Selection

The selection of each speech recognition library was based on multiple qualitative points: the level of documentation, the current development, the cost of the library API usage, and the needed computer hardware specifications to properly run said library. After researching many different libraries that were written in Python or had Python bindings, I decided to utilize Vosk, Google Cloud Speech, Whisper, Deepgram, and Wav2Vec. Vosk is a free, lightweight program with well-written documentation that goes in-depth into the possibilities and how to use the library to transcribe audio. I chose this library for its completely free open-source nature and how little hardware resources it used. The downside of this library was the difficulty in actually coding the components to get the library to transcribe properly. Google Cloud Speech was the next selection, being supported by Google and written for multiple languages (with strong Python support) through the Google CLI tool, anyone can easily set-up a transcription tool utilizing their incredibly powerful models. The downside of this library is that a fee is charged every time the program calls the API, the fee varying based on the length of the audio file. The third selection was Open-AI's Whisper, this library is backed by Open-AI's powerful speech recognition models and its impressive ability to transcribe quickly and straight off of the local computer without the need for Wi-Fi. This library API use is completely free and with the well-written documentation is easy to set up. Deepgram was the fourth selection, being a paid API it was selected to be a comparison to Google Cloud Speech's paid API. Deepgram is a small company with a well-trained model supplied to the developer and is actively being developed and changed consistently. Facebook's Wav2Vec was the final choice for it is a free model that

had interesting features such as auto chunking (the process of splitting audio files into smaller files before transcribing to allow for faster and more accurate transcription). Its biggest downside is the poor documentation and the support/development not being active anymore. Each one of these libraries was selected and used in the creation of the survey tool, each one having files that the survey tool was able to easily utilize in the automatic transcription process. Other libraries were chosen but not used due to depreciated dependencies, difficulties setting them up, and support being removed for the models themselves. Pocket sphinx, Speechbrain, DeepSpeech, and Speech Recognition were selected at the beginning but as development furthered they were removed.

Testing Framework

The design of the testing framework was split into different sections, the audio tools module, the data folder, the lists module, the main survey tool itself, and a module for each specific library. The audio tool file consists of the YouTube downloader file, the audio manipulator file, the lyric web scraper file, and the lyric normalizer file. The Youtube downloader file was coded using a library called Pytube, this file was created with the idea to take the Youtube links supplied in the dataset for each song, download the videos from YouTube, and then use FFMPEG to convert the video files into usable audio files. The converted audio files were then uploaded to a Google Cloud Storage Bucket for the Google Cloud Speech library to be able to access the files properly (This was necessary only for the Google Cloud Speech library, the rest of the libraries utilized locally downloaded files). All of this was done automatically for each song, then downloaded to the appropriate genre folder inside of the data folder. The audio manipulator file taken in each song that the YouTuber downloaded changed the sampling rate to 44.1kHz and changed the channel to mono to ensure a uniform testing

environment for each song quality. The lyric web scraper used the GENIUS API tool to automatically take the real song name and artist name from the dataset and output the official lyrics of each song into another JSON file so that the survey tool could take each song's lyrics for comparison to the song being transcribed. The lyric normalizer file took the official lyrics and the transcribed lyrics and outputted each one separately but without any punctuation or capitalization to ensure the most accurate transcription. The data folder is where the final output JSON file from the survey tool is stored and where each song is stored. The lists module consists of where the dataset dictionary is and where the official lyric JSON is stored. Each library program file is stored in its respective module to ensure an organized file structure. Finally, the survey tool is stored on the root of the project so that it can access each file in every module/folder whenever needed, as it is the tool that automatically takes all the data in, organizes it, and outputs it into a readable and organized JSON file for post data processing.

The Survey Tool

The survey tool is also known as a wrapper program, this is a program that takes in multiple other Python modules (which were coded separately to ensure there was no overly cluttered and unorganized code) and ties them all together. In our case, the survey tool utilizes Python relative importing to import every file from their respective modules to initialize everything needed. It then checks to make sure that the necessary data folders are there and that every song from the dataset is downloaded, converted, uploaded to the Google Storage Bucket, and placed in the correct folder by genre. The survey tool was written to ensure that it checked the lyric list to make sure that each song existed and was included in the lyric_listJSON file. After this pre-processing and checks were complete, the survey tool was written utilizing the Python Click CLI tool library to take a command line input from the user and output it to the

survey tool program so it knew exactly what song, library, and genre to transcribe and output. The tool also allowed for flags (identifiers written as options to be used in the command to the program) to be added that allowed the user to change the desired final_file output name to be, a help option to show the user how to use the command line tool to create a command, all the possible songs to choose from to transcribe based off of what is currently in the dataset, and the option for a custom youtube link, song file name, real song name, and artist name to be inputted into the program. This custom flag lets the user take any song not included in the dataset and automatically transcribe it and analyze the accuracy of the song per each selected library. An example command for this tool would be `python survey_tool.py -s alone -l whisper -o alone_whisper_transcription` (see figure 1). After this command, the survey tool will utilize the `Automate` class, a class written to process the flags that were used in the command and alter the automatic transcription for only the song selected (which in the command above's case would be Alone and Forsaken by Hank Williams) and the library/libraries selected (in this case only whisper was selected). Once these flags are processed by the class it then runs the actual `automate` function. The first two lines set up a temporary data dictionary for the storage of the transcribed lyrics; the time it took to transcribe them, the library used to transcribe them, the official lyrics, and the distance value calculated based on the transcribed lyrics and official lyrics (see figure 2). The function runs the specific song through the specific library in this case (more than one song and library or even whole genres or every song and library can be run at once), and then stores the data into the above-mentioned data dictionary. This is updated to a final dictionary and outputted to a JSON file as the final output file that includes all necessary accuracy and song/library information.

Analysis of Data

The final outputted JSON file can then be fed back into a separately coded data analysis tool in Python due to its native support, JSON files we can easily interpret the quantitative data given, organize it by song, genre, or library, and output them automatically into graphs for later result interpretation.

Appendix Results

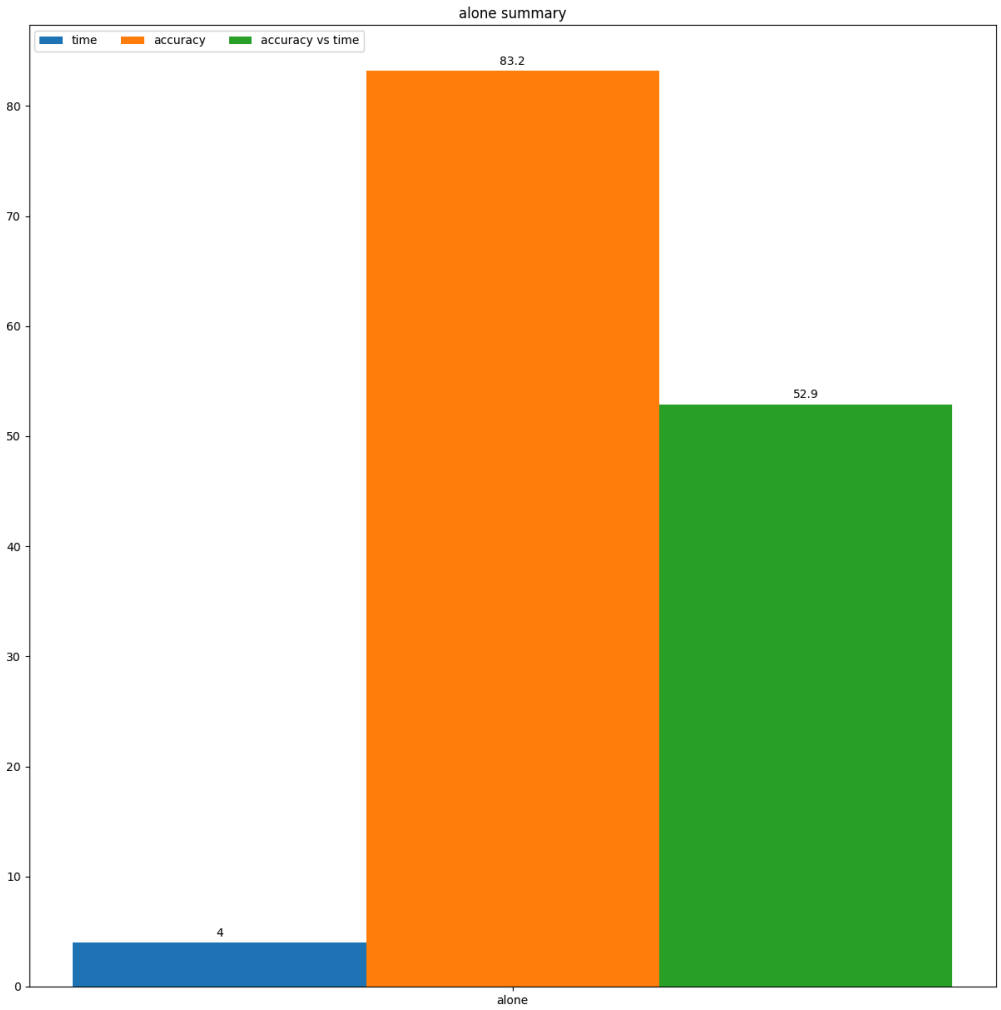

Figure 1: The summed data of Alone and Forsaken - Hank Williams

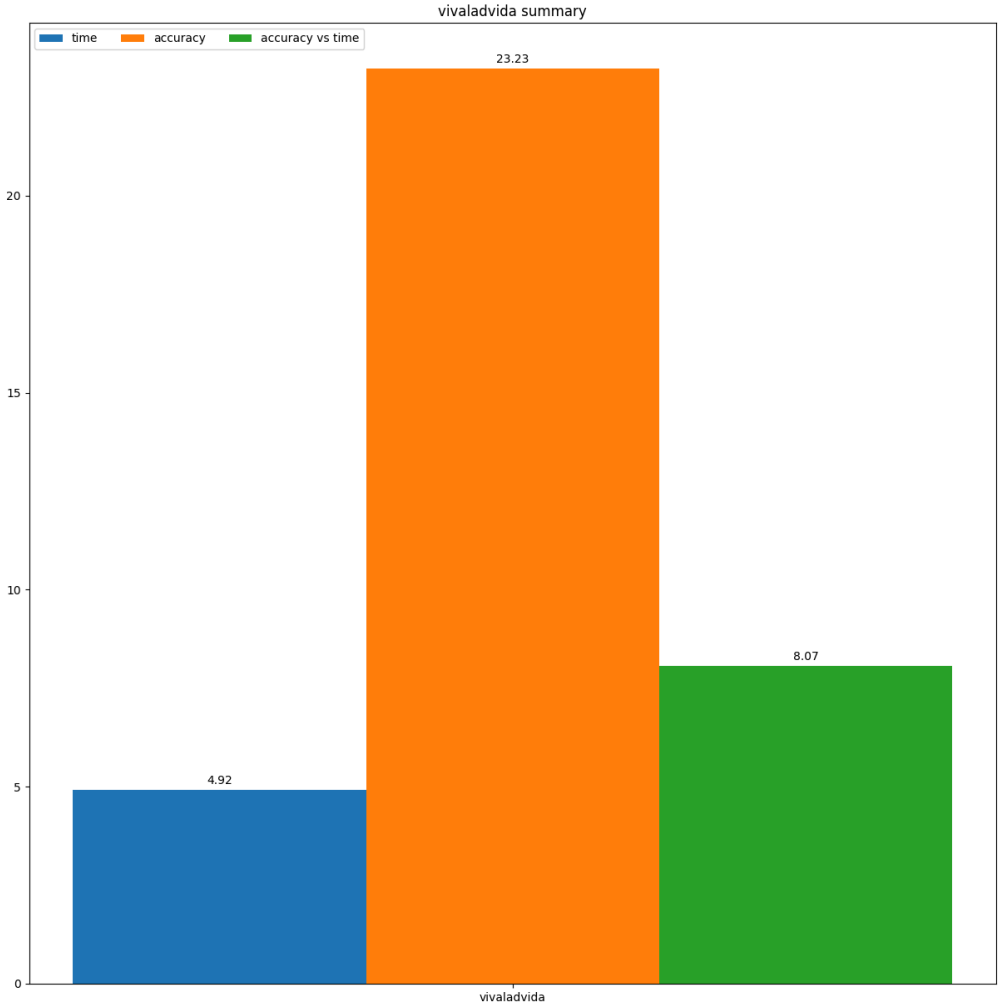

Figure 2: The summed data of Viva La Vida - Coldplay

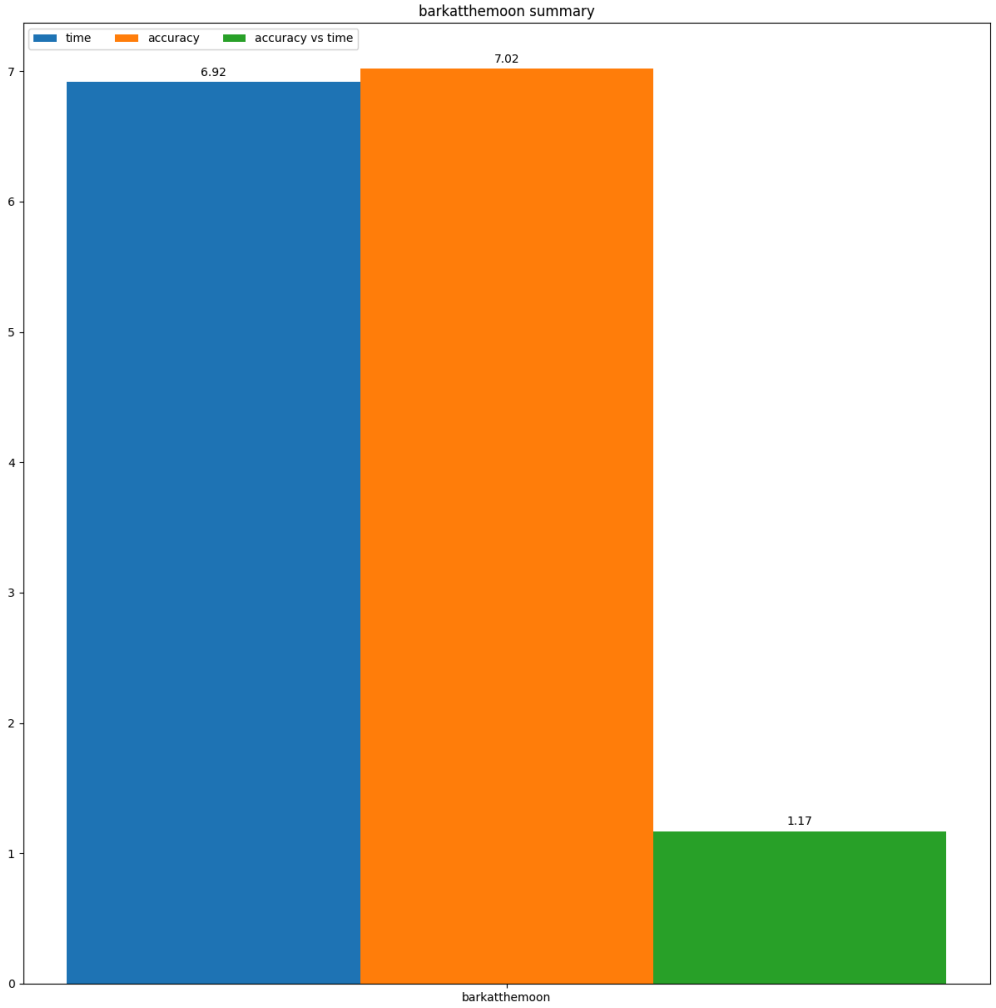

Figure 3: The summed data of Bark at the Moon - Ozzy Osbourne

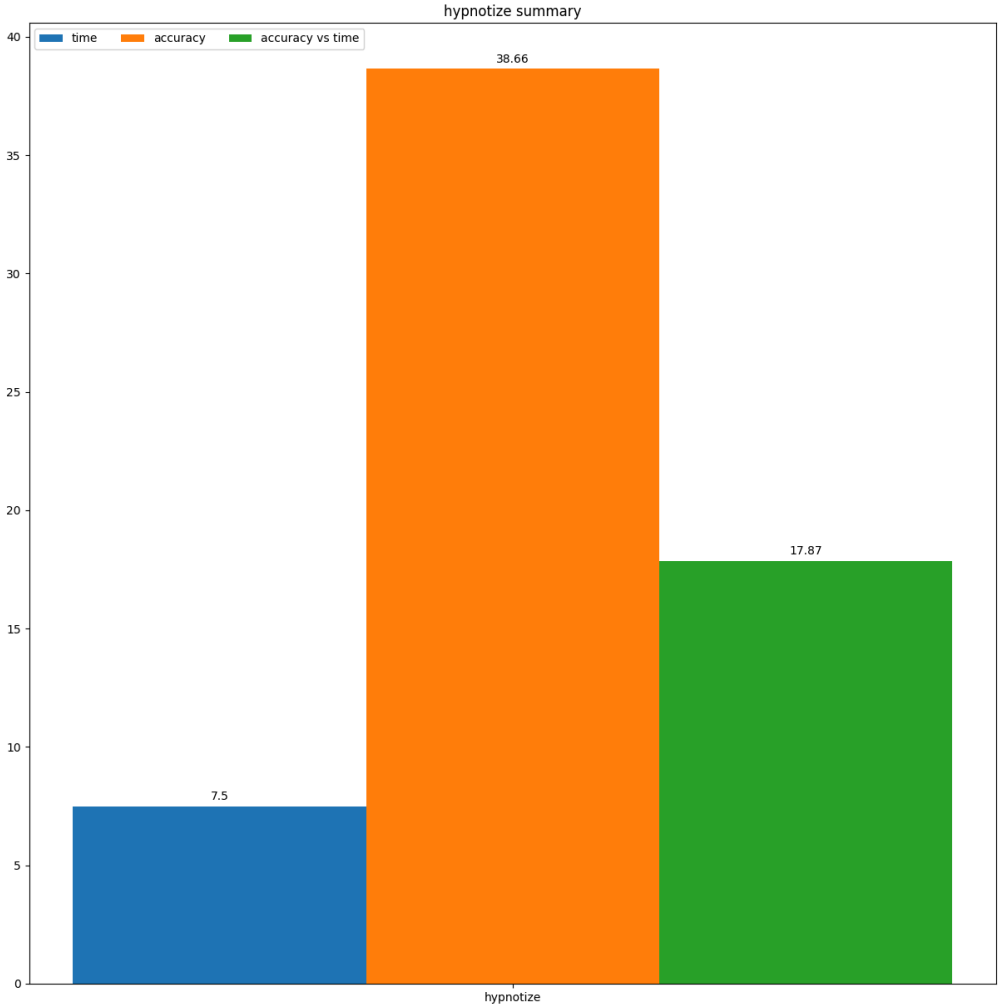

Figure 4: The summed data of Hypnotize - The Notorious B.I.G.

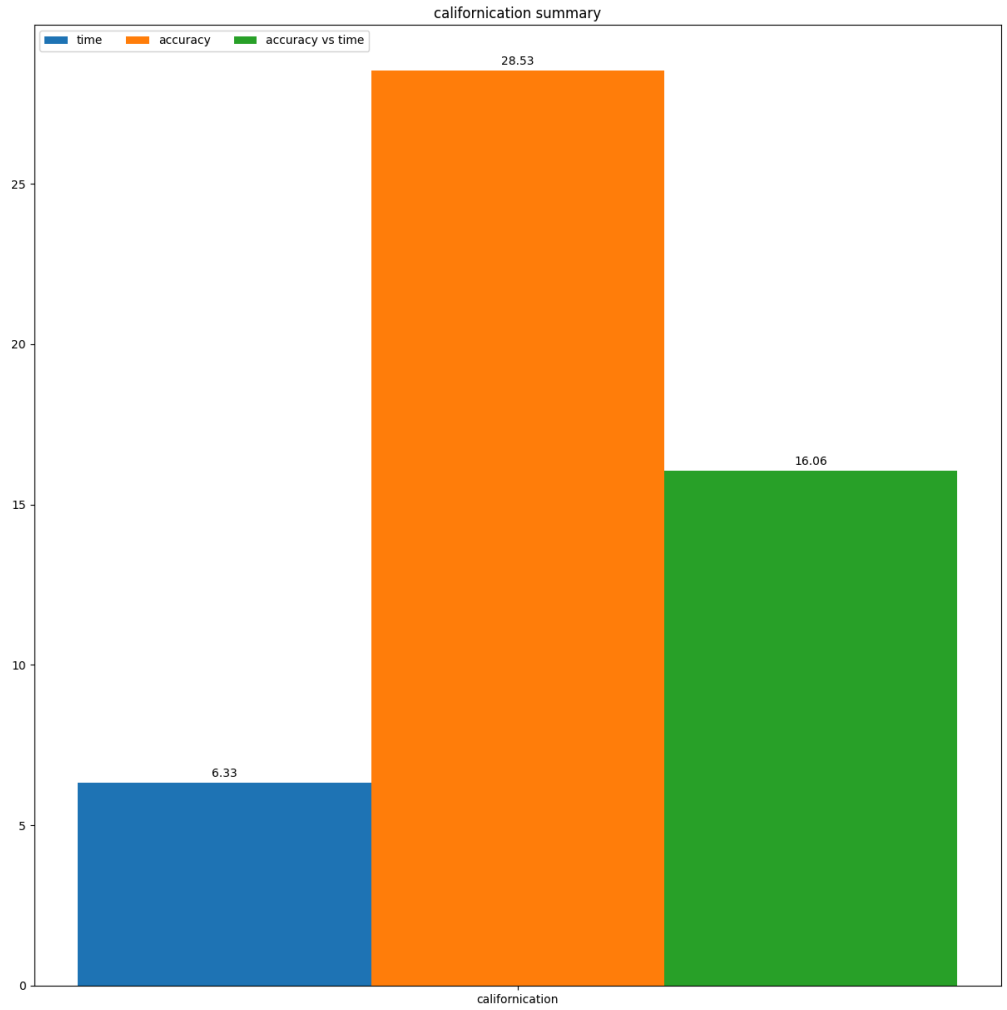

Figure 5: The summed data of Californication - Red Hot Chili Peppers

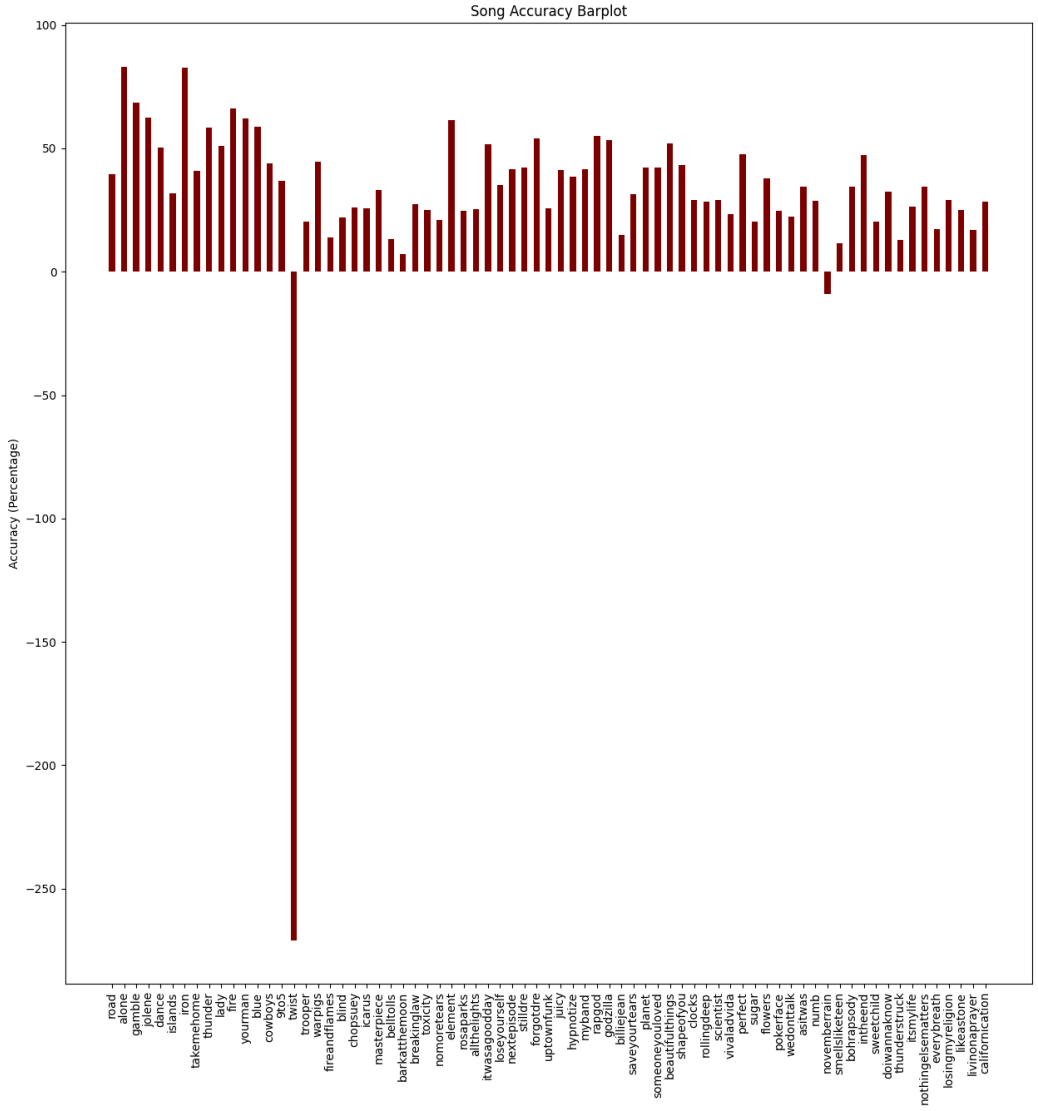

Figure 6: The summed song accuracies

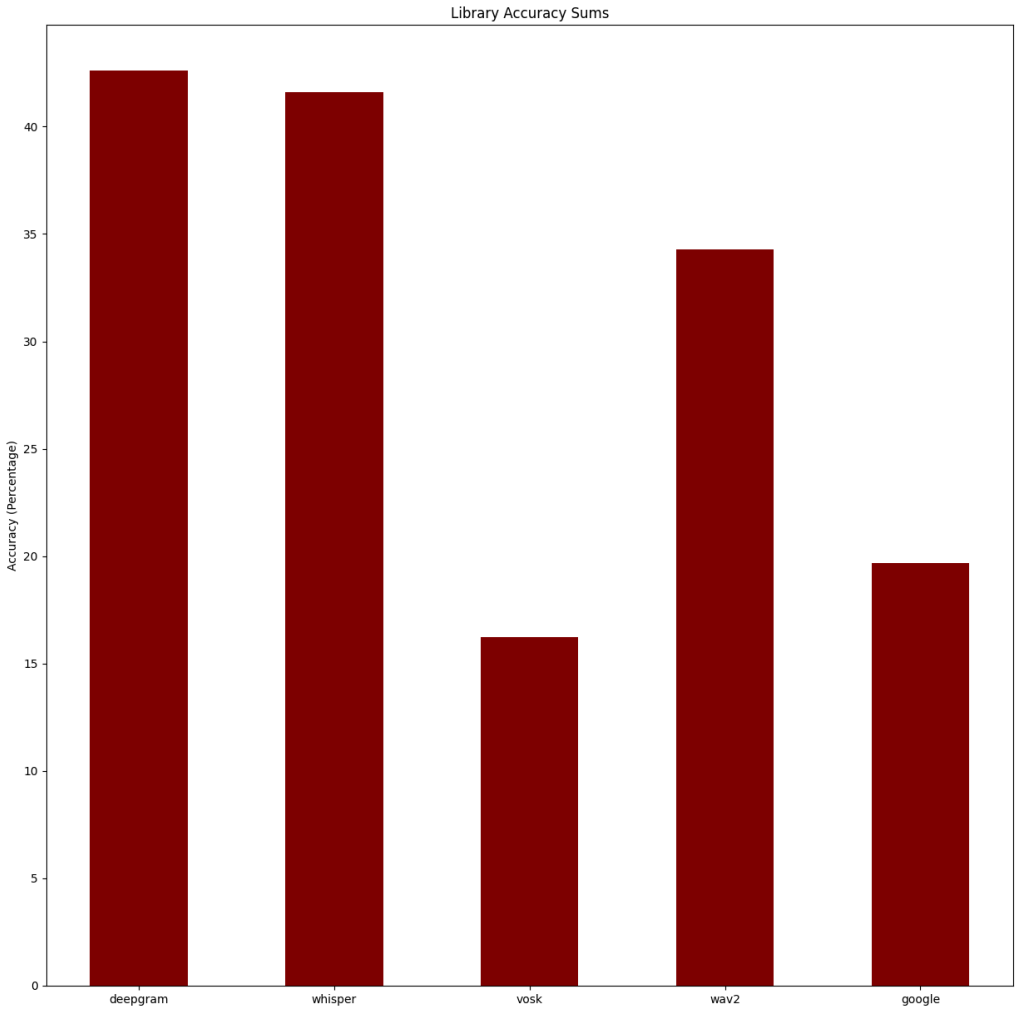

Figure 7: The summed library accuracies

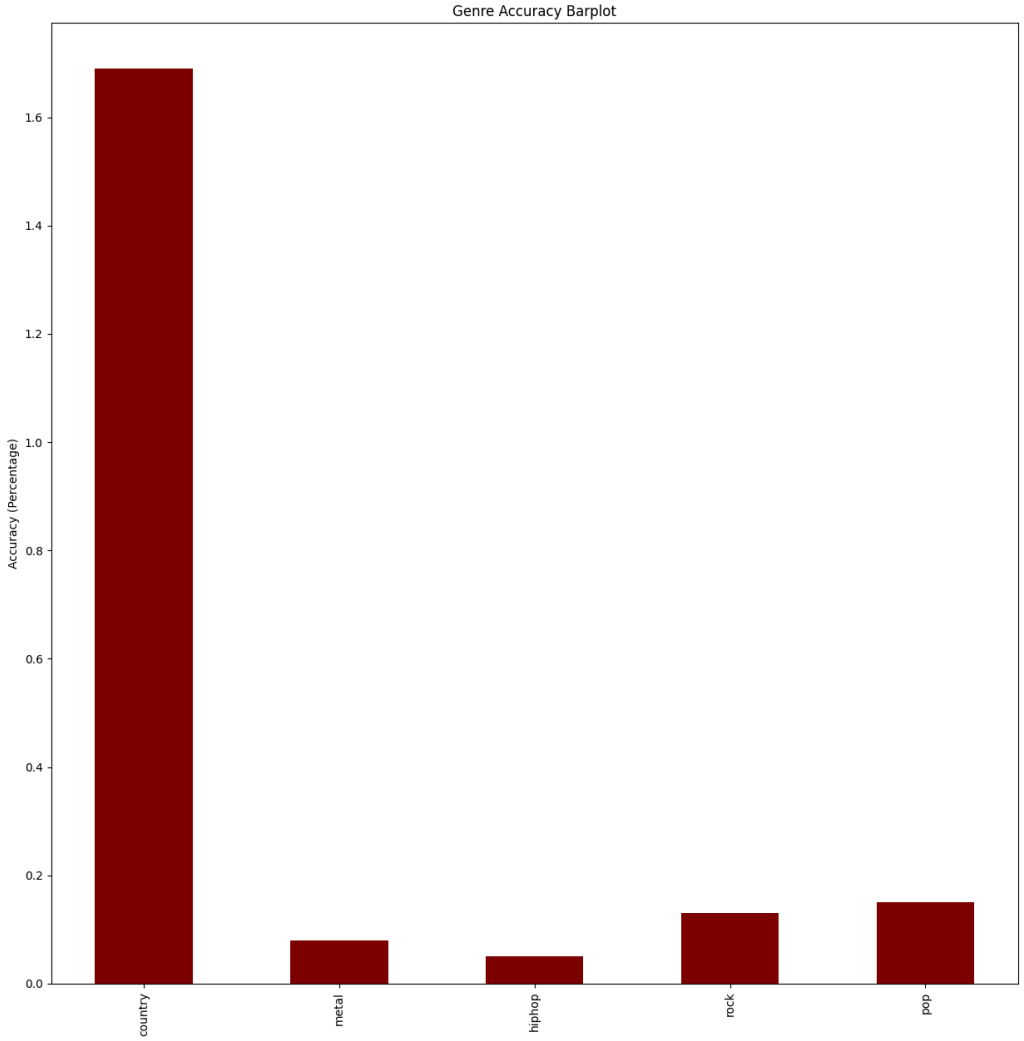

Figure 8: The summed genre accuracies

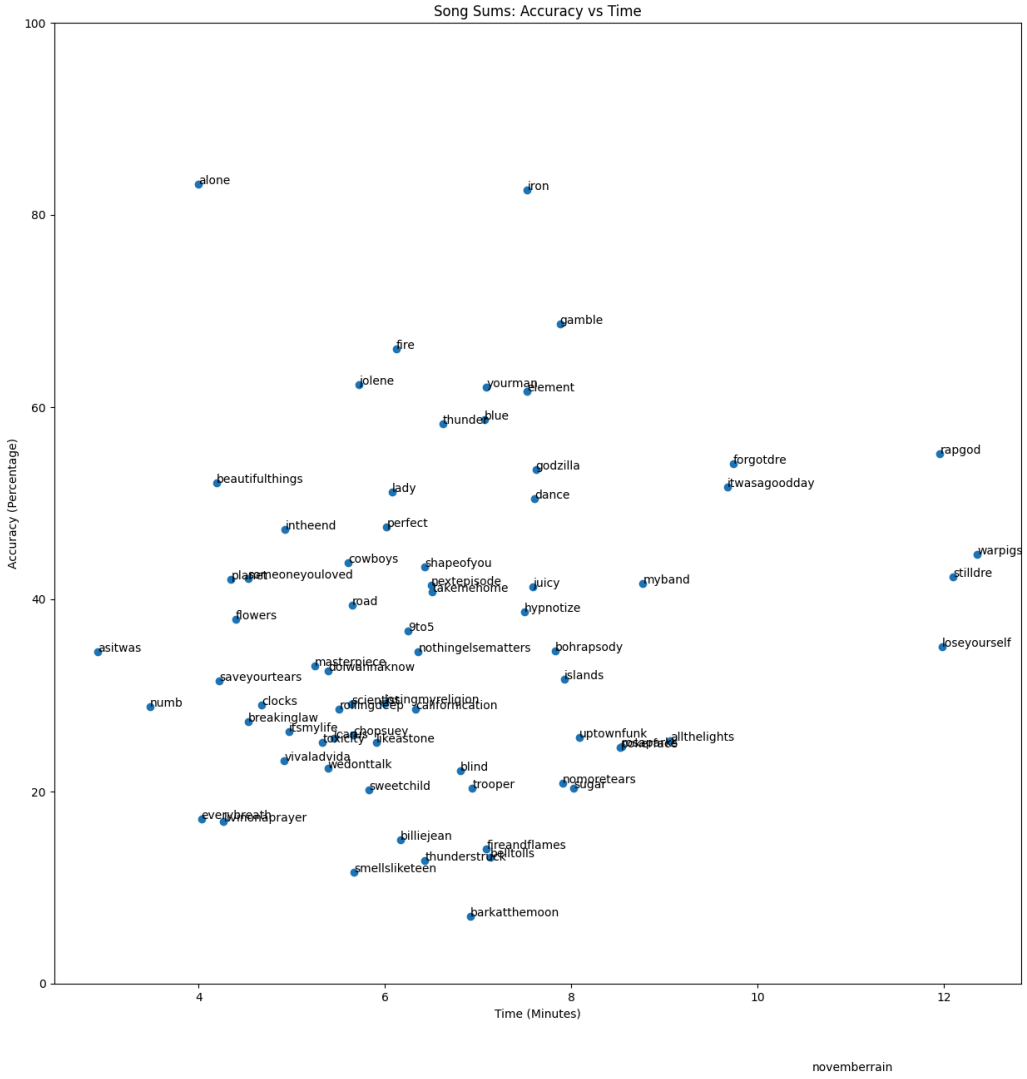

Figure 9: The summed song accuracies vs times

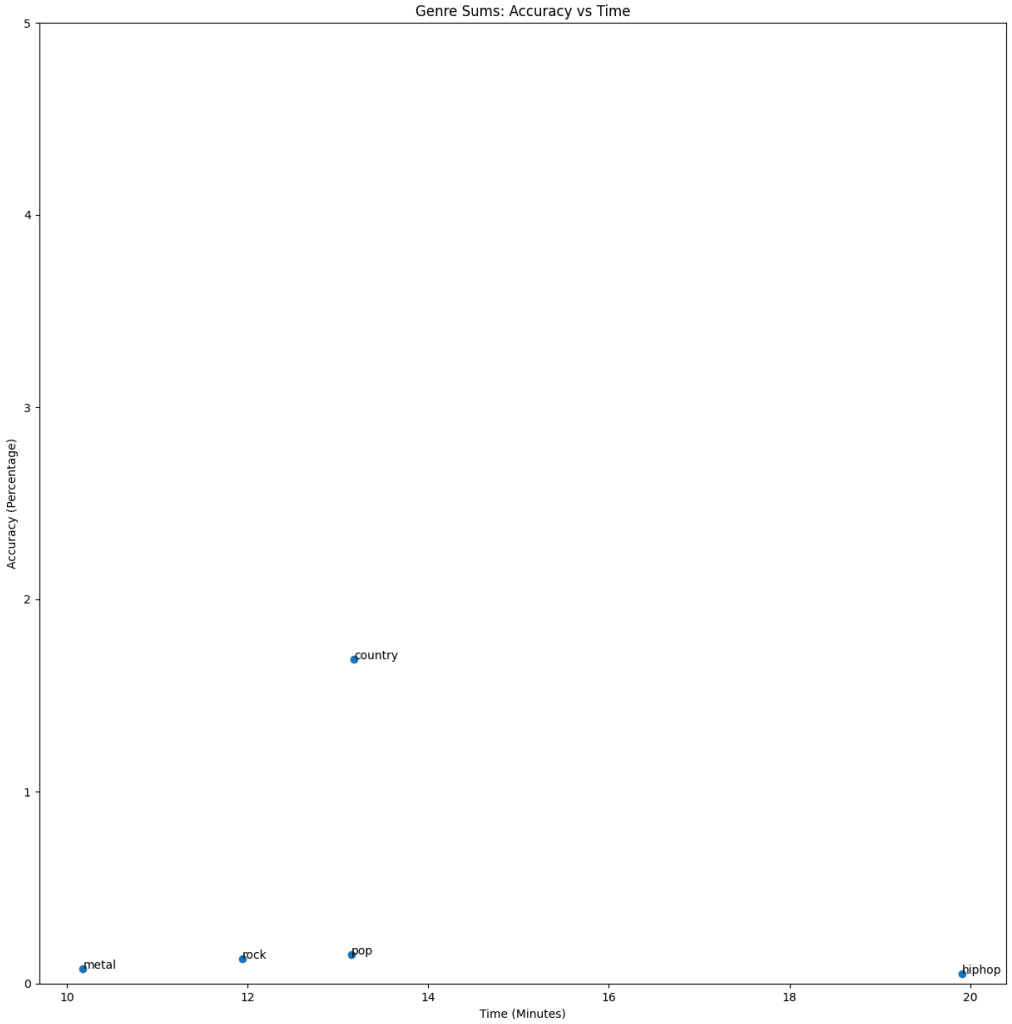

Figure 10: The summed genre accuracies vs times

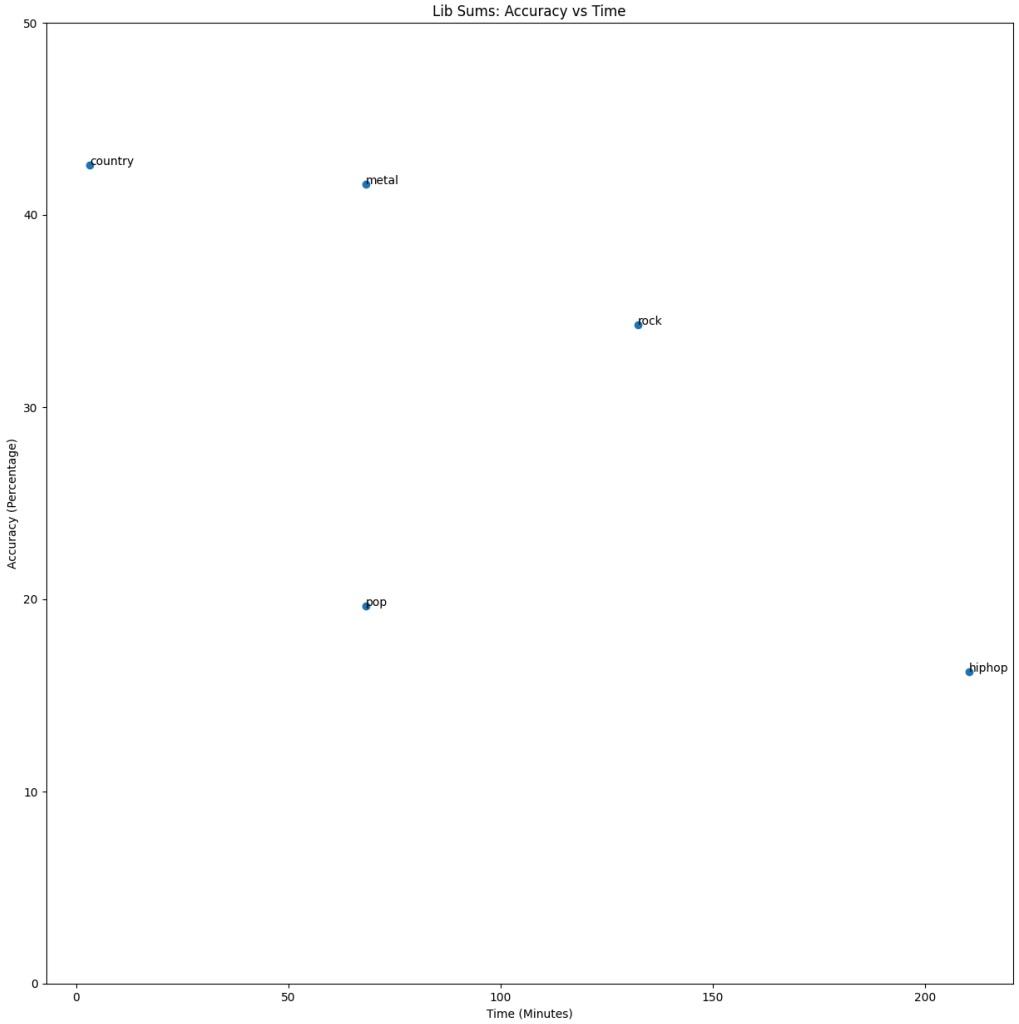

Figure 11: The summed library accuracies vs times

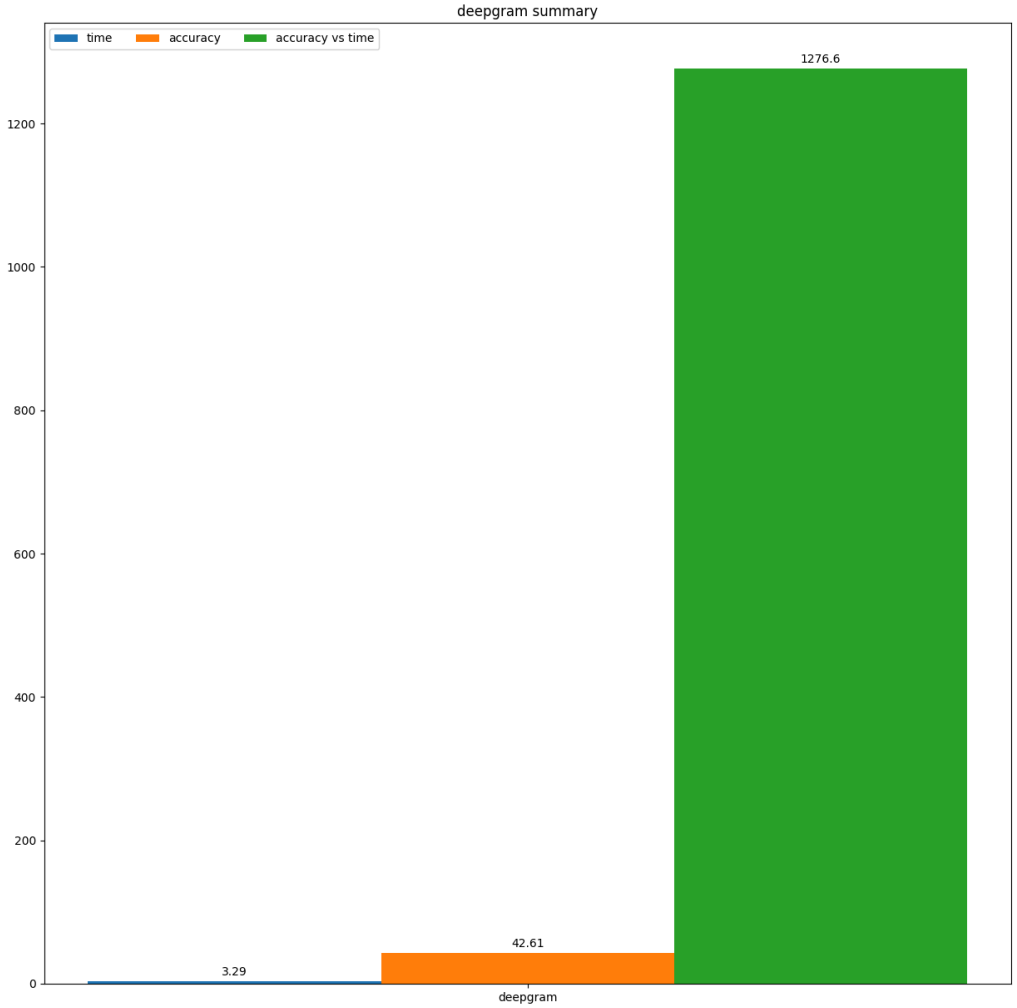

Figure 12: The summed Deepgram data

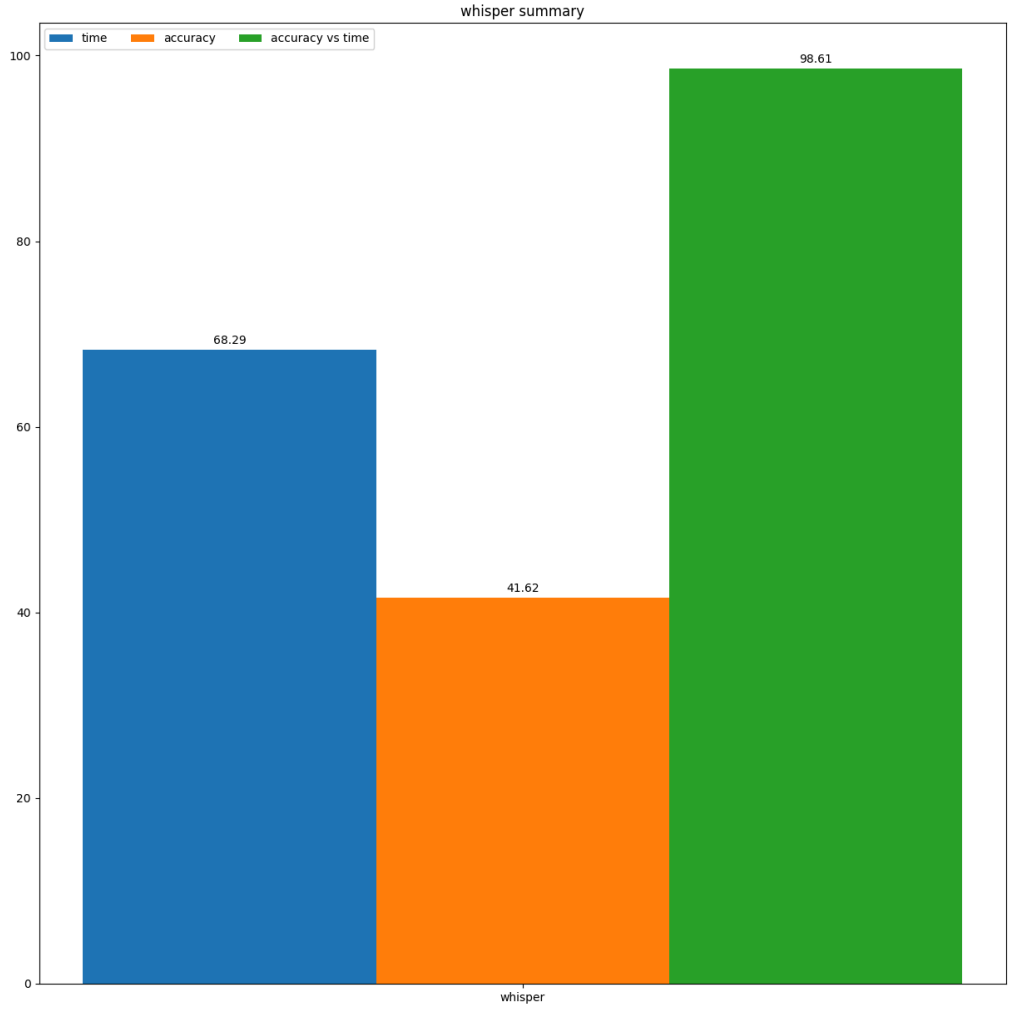

Figure 13: The summed Whisper data