Tom Slesinger and Nathan Duffy

INFO 4700

Jason Zietz

2 May 2019

Capstone Final Report

The overarching question of our capstone project was to see if it is possible to classify the quality of a song based off of its audiovisual spectrogram. We used a program called Spek[1] which allows a user to display the visual spectrogram of a song. This program generates a frequency graph which indicates which parts of the audio codecs frequency range are being used, and at which time they are being used. In general, the higher the peaks are in the spectrogram, the higher quality the song is. The reason this algorithm is important is since some songs can be mislabelled as full quality (320 kbps) when they are encoded as a lower long. The only way to correctly identify the quality of the song is by listening to it, which is proven to be difficult, and by the frequency graph. Actually identifying the quality of a song isn't too difficult, with this guide[2], but it is monotonous and would take hours to correctly identify an entire music library. Thus, the idea sparked to make a machine do all the hard work (or most of it) to correctly identify a song. We will now go in depth into the machine learning portion of the project which can be split up into three main categories: data gathering, data labeling, and learning.

Data gathering started out as the first hurdle in this project. Since we had roughly 5000 songs to generate frequency graphs of, simply using the GUI of Spek was out of the question. Since the GUI was unfeasible, and Spek has no command line interface (CMI), we decided to write a .py script. We messed around with a few different libraries but eventually settled on the standard library *subprocess*. This library is an established library within the Python programming language. Subprocess has a function called *popen* which stands for process open. We passed the file path of the Spek executable file in argument slot 0 and the file path to the song name in argument slot 1. This one line of code opened the program Spek and loaded a song for spectrogram generation.

The next hurdle was figuring out how to make the computer wait long enough to generate the frequency graph. Since there were so many songs, it would be inefficient to set a static wait time. We ended up using the *os* library which allowed us to get the file size of each song in bytes. With some tuning, we divided the file size 1000000 and that number by 2. Which gave Spek roughly enough time to generate the frequency graph for each song. Once these two steps were completed, we just had to save the spectrogram. Again, since there is no CMI we couldn't pass a command to Spek for it to save. Instead, we used *pyautogui* to just enter key presses that would save the file and then quit the program to allow the Python script to go onto the next song. Overall, the script ran for about 5 hours and generated roughly 5000 spectrographs.

Once we had gathered all of the data, we had to start the labeling process. We had a few options for this process, one of which being Mechanical Turk (MT). However, after researching this process, our problem is far too complex for MT and would be overwhelmingly expensive. Our other option was to start to label ourselves to understand how to label, and then get a group of friends of classmates together to label pictures. After speaking with our machine learning Professor, Michael Paul, he said that we should try to aim to label 1000 pictures first instead of all 5000. This made this portion of the project more manageable.

The second step was to find a platform to host all of these images, a way to give them a label, and most importantly, a collaborative platform. At first Google Sheets seemed impossible since we could only upload one picture at a time. Luckily, there was an add-on to Google Sheets called ImageKit[3] which allows for mass upload of pictures. Once we had uploaded 1000 pictures, we thought of which meta-data we wanted to give the picture. We knew we wanted to have the given label and the new label, but struggled to figure out if we would need the song name. After some deliberation, we attempted to use os.walkpath to find the correct song titles but weren't able to correctly walk through the directory to properly give the songs a title in text. Once the Google Sheet was created, we had our INFO 4700 class help label roughly 200 pictures and labeled the last 800 ourselves. After about a week of grinding, we had 1000 pictures labeled by their spectrogram. Which brings us to the interesting part of the project: machine learning. This part of the project was easily the most involved since it involved so many steps. We had to get the data from the Google Sheet into a pandas dataframe, convert the actual pictures into a datatype that an algorithm can understand, feed that data into an algorithm and tune the hyperparameters.

The first step of the project seemed easy enough, but it took a while to figure out. Firstly, you can't give a machine learning algorithm and PNG and ask it to find patterns, that's not how it works. Secondly, pandas does not handle pictures well at all. Firstly, this meant we had to find a way to get the pictures from Google Sheets into Pandas. Luckily, due to ImageKit, each picture was actually just a Google Drive link, which could be passed right into *skimage* (a good library that can handle large amounts of data image processing). Secondly, this meant that we had to convert the pictures into some datatype before loading it into a dataframe. As a baseline, we started with the most basic encoding of a picture, following the CIFAR-10 dataset. This encoding takes each pixel and finds the corresponding RGB values (0-255). This type of encoding worked will since the CIFAR-10 dataset pictures were 32x32 pixels, but our spectrograms were 330x629. Since the *skimage* library converts the picture into numpy arrays, we were able to find a function[4] on StackOverflow to crop the pictures for us. The metadata around the picture (Hz values, timestamps, and decibel values) was not useful for the algorithm, only for humans. Once we cropped the picture, we used *skimage.imread* to generate a 3-D array of RGB pixel values. We then flattened that array to get a list of values that corresponded to the label. Once we figured

this aspect out, we simply looped through every Google Drive link and added the flattened features to a dictionary with the index of the features as a key giving us a dataset of 1000x446926. However, we were able to turn this sparse matrix into a dense matrix by removing columns that contained all 0 values giving us a 1000x8235 matrix. However, this was still too large to convert to a CSV and read back to Pandas which meant every time we restarted the notebook, we had to rebuild the database.

None the less, after we had the dense matrix of RGB encoded values we standardized the features with *StandardScaler* to normalize all the values. Once that was completed, we ran the data through a dummy classifier which does a very simple 'most frequent' strategy to give a baseline accuracy. The dummy classifier got a training accuracy of 13% and a testing accuracy of 12%. We then decided to test a multitude of different machine learning algorithms using K-Fold-Cross-validation to see which algorithm performed the best. However, every algorithm(Logistic Regression, K Nearest-Neighbors, a Multi-Layer Perceptron, a Decision Tree, and a Support Vector Machine) performed roughly the same as the dummy classifier which is a red flag that the data in its current format may be unlearnable. We decided to tune some hyperparameters across the decision tree since it was the most accurate out of all of them (13.4% testing accuracy) but didn't see any improvements. This meant that we had to find a different way to translate the pictures into numbers.

Our first thought was to use DAISY[5]: a well-established algorithm that finds descriptors within pictures. However, not only is DAISY not correct for this problem, but it also generated over 1 million features per label which gave us a memory error when trying to load it back into Pandas. This meant that we had to create a new way, with fewer features, to feature engineer these pictures. After another discussion with Professor Paul, we thought of a way to give each picture a score based on how high the shelf in the picture is. Generally, the higher the frequency bars in a frequency graph, the higher the quality. We used the same steps as before: create a dictionary, loop through the Google Drive links, and slightly crop the picture, but this time we took a different approach. Firstly, we used an argument as_gray for the *skimage.imread* feature which turned the 3-D array into a 2-D array. Secondly, by finding the shape of the picture, we could loop down the first column of pixels to find where the black pixels started and stopped. By using a for loop within a for loop (O^2 we know we should find a better way) we could achieve this. We kept a score counter at 0 for each column. In that column, if the pixel is black we subtract the running score by 1, if the pixel is anything but black, we increase the running score by 1. This should give a general idea of how many black pixels are in the frequency graph, and ideally the less black pixels the higher quality the song since the shelf is higher.

This way of feature engineering already proved to be useful since it reduced the features to the number of pixels on the X-axis instead of 8000. We passed these new features into the dummy classifier with no improvement. However, after running K-Fold-Classification we found that a support vector machine increased the testing accuracy up to 32%. The next highest were

Logistic Regression and a Multi-Layer-Perceptron. We attempted to tune the hyperparameters of the SVM model, including the kernel and the C value but didn't increase the testing accuracy. This holds true with the MLP and the Logistic Regression model. Interestingly, we were able to achieve a training accuracy of almost 99% but our testing accuracy never increased passed 32%. This means that our models are overfitting, which is most likely due to a lack of data.

This lack of data is a problem, but not one that we haven't thought about fixing. To make up for the lack of data we intended on creating a website. The machine learning model is great on its own, but what good is it if it just sits on our computer as another inaccessible file? To solve this problem we plan on making the model available to the public. This solves two problems, our lack of data and accessibility. First off, by making the model publically available we are able to collect more data on spectrograms. We plan on doing this by adding a mySQL database to our website which will house the various spectrograms and their respective audio qualities. Our accuracy is now at 32%, but with this fix, we are hoping to bring it up to something that is more easily trusted by our future consumers. I'll walk you through the process of completing this next step in our capstone process.

Tackling website creation is tricky. Do you utilize javascript, Wix, Flask, or other tool? This was quite a process, and took a while to determine. After being exposed to some entry level javascript and HTML/CSS I thought I would start here. This became tricky really fast. Tom originally used a subprocesses module within his machine learning code, so I looked for the javascript equivalent of this to utilize for the website. I found this equivalent to be child nodes, a part of the data object model within a webpage. This would theoretically allow me to access a function called exec.file() and run the machine learning model, while closing out the process promptly afterwards all within a single child node. After some time tinkering around with this process I determined that the functionality of the process wasn't going to get the job done.

This led me to my next attempt, which was the utilization of AJAX (Asynchronous JavaScript and XML). This tool is a set of web development techniques used to create asynchronous web applications. It would allow me to communicate between the javascript file and the python file. To do this I had to import a Jquery library that was inevitably deprecated, and hindered the process of the tool I was attempting to use. After some more hours of playing around with this, I came to the conclusion that it was far too difficult to try and communicate between two different web development languages. I was then led to flask, the tool that I ultimately used in the final product of the website. Flask is a light-weight web framework that is utilized in python. This tool allows me to set up a local server environment that I can host the machine learning model on. After reading up on the tool, I was able to find out that it possessed the functionality I needed for what we were trying to accomplish. First step was creating some HTML template files that gave the website it's looks. Next step was figuring out how I was going to run the machine learning model on the webpage. After some research I stumbled upon

pickle. A module in python that the serialization of files. This process allowed me to save the contents of the machine learning file, and load them onto our website along with our desired output from the model. Pickle in combination with Flask was enough to get the job done for this next step in our website. Almost. I've been playing around with it for a while, but it seems as though the model is not outputting exactly what we are wanting. It does give us a prediction and probability, but it did not vary between pictures. This could be due to an error in variable mapping, I plan on continuing to tinker with this until I find the problem. Once this is fixed, next steps would be to plan around with the database and start collecting that new data so we can improve our accuracy.

In conclusion, to answer the research question we started with, yes it is possible to verify audio quality. However, to make this algorithm viable we need much more training data, and most likely another improvement in feature engineering. That being said, we did improve the accuracy greatly. Since there are 10 labels, a person guessing would have a 10% chance of getting the label right. We started out with having our algorithm is only 3% better than guessing to 23% better than guessing. With more data and more feature engineering, this algorithm could be a viable tool for anyone who wants to know the true quality of their song.

Appendix

1. http://spek.cc/
2. https://www.reddit.com/r/xTrill/comments/6fre3q/spek_guide_2017_edition/?utm_source=reddit&utm_medium=usertext&utm_name=xTrill&utm_content=t5_386fy

3. https://chrome.google.com/webstore/detail/imagekit/cnhkaohfhpcdeomadgjonnahkkfoojoc?hl=en
4. https://stackoverflow.com/questions/39382412/crop-center-portion-of-a-numpy-image/39382475
5. https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_daisy.html